Pushing Analytics to the Edge

Goutham Kamath  Pavan Agnihotri  Maria Valero  Krishanu Sarker  Wen-Zhan Song
Department of Computer Science
Georgia State University
{gkamath1,pagnihotri2,mvalerodelemente1,ksarker1}@student.gsu.edu, wsong@gsu.edu

Abstract—Edge or Fog computing is emerging as a new computing paradigm where the data processing, networking, storage and analytics are performed closer to the devices (IoT) and applications. The edge of a network plays an important role in the IoT system. It is an optimal site for off-loading bandwidth hungry IoT data. In order to generate business value out of the large volume of data on the edge, we need decentralized machine learning (ML) algorithms. These algorithms must enable edge devices to communicate autonomously and deliver information seamlessly to the decision makers. In this paper, we present EdgeSGD a decentralized stochastic gradient descent method to solve a large linear regression problem on the edge network. The solution is applied to the seismic imaging use-case and evaluated using an edge computing testbed. The proposed algorithm avoids sending raw data to the cloud, and offers faster and balanced computation.

I. INTRODUCTION

In the era of Internet of Things (IoT), high data rate sensors such as video camera, cell phones are becoming ubiquitous. Today, most high volume data obtained from these sensors are stored close to the point of capture, and only few are transferred to the cloud. In future, sending all the data from billions of IoT devices to the cloud can easily overwhelm the existing infrastructure. To overcome these issues, a new paradigm called Edge or Fog computing is emerging [13]. This paradigm brings the data processing, networking, storage and analytics closer to the devices (IoT) and applications. Edge computing aims at placing small data center’s called cloudlets at the edge of the internet, in close proximity to mobile devices, sensors and end users. The new paradigm counters the theme of consolidation and massive data centers and opens up the door for further research in decentralized computing and analytics.

The edge of a network plays an important role in the IoT system. It is an optimal site for off-loading bandwidth hungry sensor data. To generate business value out of the large volume of data on the edge, we need decentralized machine learning (ML) algorithms. These algorithms must enable edge devices to communicate autonomously and deliver information seamlessly to the decision makers. Unlike the cloud infrastructure, edge network exhibits following properties. i) heterogeneous hardware ii) unreliable low bandwidth communication network iii) limited on-board memory and processing power. Moreover, the edge analytics algorithm should not rely on any central coordinator and must be fault-tolerant as node/link failures are a common occurrence. Existing decentralized machine learning platform such as MapReduce [18] is not fault-tolerant and requires a coordinator node to perform reduce operation.

In this paper, we present EdgeSGD, a decentralized stochastic gradient descent algorithm suitable for machine learning and analytics on edge network. This method can be applied to wide range of problems arising in decentralized machine learning [1]. As a case study, we solve a fundamental machine learning problem of linear regression, where the objective is to estimate the feature vector \( \theta \in \mathbb{R}^n \) using \( m \) training example \( \{(x_1,y_1),(x_2,y_2),\ldots,(x_m,y_m)\} \) given by,

\[
\min_{\theta \in \mathbb{R}^n} \frac{1}{2m} \sum_{i=1}^{m} \|y_i - x_i^T \theta\|^2 + \lambda \|	heta\|^2, \tag{1}
\]

where, \( \lambda \) is the regularization parameter. As a real-world use-case, we apply EdgeSGD to predict subsurface seismic anomaly via real-time imaging [5]. As far as we know, we are the first to explore the potentials of such algorithms on edge computing framework. The key contributions in the paper are:

- We present a novel decentralized stochastic gradient descent algorithm for solving linear regression problem on the edge node.
- The proposed algorithm is applied to learning/predicting seismic anomalies via real-time imaging.
- We evaluate the algorithm on edge computing testbed, using both synthetic and real seismic dataset.
- We compare the proposed solution with other existing decentralized methods such as MapReduce, DGD and EXTRA, and examine in particular the effects node/link failure and communication cost.

The rest of the paper is structured as follows: In Section II, we provide seismic imaging background and the problem formulation. In Section III, we describe previous works related to decentralized machine learning. We provide a detailed design of EdgeSGD in Section IV. In Section V, we evaluate the performance of our algorithm and provide experimental results. We conclude the paper in Section VI.
II. Preliminaries

A. Seismic Image Background

The current subsurface imaging technology to visualize oil/gas reservoir or fault/magma movement lacks the capabilities of obtaining real-time information. The principles of seismic imaging consists of three main steps as illustrated in the Fig. 1(a). i) Seismic sensors (green triangles on the surface) measure the vibration after the occurrence of an earthquake, calculate the arrival time of the p-wave [7] and obtain the measurement $y$. ii) Next, the earthquake location and time is estimated using origin and time using prior estimates. iii) Lastly, the rays are traced from earthquake to node $i$ (blue rays) to form data set $x_i$. The feature vector $\theta$ represents the image, where the components $\{\theta_1, \ldots, \theta_n\}$ represent the value of $n$ pixels.

Today, thousands of dense arrays of seismic sensors (geophones) are deployed to monitor subsurface anomalies such as earthquake etc. These geophones passively listen to the ground vibration and record them to a local storage. These acoustic signals are sampled at 16-24 bit 100-500 Hz which accumulates to few GB/day per node. Transferring large volume of data to a central repository (cloud) using low-power radio is often infeasible due to energy and bandwidth limitation [13]. For this reason, we propose an edge computing framework to perform real-time seismic imaging. Here the edge nodes form a layer between geophones and the cloud (Fig. 1(b)). Several geophones are connected to each edge node, and since they are located close to the sensors, seismic signals are transferred to them in real-time. The edge nodes form a mesh network, and our goal is to design a decentralized algorithm to obtain $\theta$ (image) by collaboratively optimizing the objective function Eq. (1) over the edge network.

![Fig. 1. Illustrating the process of real-time seismic imaging on an edge network. (a) Principle of travel-time seismic imaging. (b) Edge computing architecture for seismic imaging.](image)

B. Problem Formulation

In this paper, we consider $N$ edge nodes connected to form an arbitrary topology given by an undirected graph $G(V, E)$, with node set $V = \{1, \ldots, N\}$ and edge set $E$ that contains set of links in the network. We have $\{i, j\} \in E$ if node $i$ and node $j$ can communicate with each other. $N_i$ denotes the neighbor set of node $i$. Each edge node $i$ is connected to one or more seismic sensors and have $m_i$ training samples given by, $\{(x_1, y_1), (x_2, y_2), \ldots, (x_{m_i}, y_{m_i})\}$, where $\sum_{i=1}^{N} m_i = m$.

Now using Eq. (1), we can form a local loss function at node $i$ given by $f_i(x) = \frac{1}{2m_i} \sum_{i=1}^{m_i} \|y_i - x_i^T \theta\|_2^2 + \lambda \|\theta\|_2^2$.

Now combining all to a single equation we get the following optimization problem,

$$\min_{\theta \in \mathbb{R}^n} \frac{1}{2N} \sum_{i=1}^{N} \|Y_i - X_i^T \theta\|_2^2 + \lambda \|\theta\|_2^2,$$

where, $X_i \in \mathbb{R}^{m_i \times n}$; $Y_i \in \mathbb{R}^{m_i}$ denote the $m_i$ rows training set at node $i$. Here, the regularization parameter $\lambda$ is a positive number that controls the weight between $\|Y_i - X_i^T \theta\|_2^2$ (goodness fit measure) and $\|\theta\|_2^2$ (regularity measure). We represent $\theta^k$ feature estimated at the iteration $k$. The goal now is to develop an algorithm to solve regularized regression problem Eq. (2) in a decentralized and asynchronous way on the edge network.

III. Related Work

Distributed machine learning algorithms can be classified based on 1) Shared Memory and 2) Distributed Memory architectures. In a shared memory architecture, the data resides in a location accessible to all the compute nodes e.g. GPU computing. Some of the recent ML algorithms for such architectures are [8], [10]. These algorithms perform distributed stochastic gradient descent using several processors. Since, compute nodes have access to shared memory, ML algorithms designed for such architectures are not suitable for edge analytics.

In a distributed memory architecture, each compute node assumes data to be available locally. We can further classify this into Client-Server and Peer-to-peer P2P architecture. MapReduce is one of the most popular machine learning algorithm for client-server architecture. Authors in [18] describe a distributed stochastic gradient descent method suitable for large multi-core platforms. This method incorporates the online learning principle of [9]. These methods scale linearly with the data size and log-linearly with number of compute nodes. Similar method for medical imaging was studied by [4]. In MapReduce, although the data and computation is distributed, the reduce operation takes place at a central coordinator and we show from our experiment that these schemes are not suitable for edge network.

In recent years, several P2P based optimization methods have been developed for solving decentralized machine learning [11], [16]. In these methods, computation nodes do not require any central coordinator and perform optimization by communicating only with their immediate neighbors [2], [3], [6]. Some of the recent advancement include decentralized gradient descent (DGD) [16] and EXTRA [11]. Methods that have both local computation and communication are suited for edge computing platform. The convergence of such algorithms depends both on the training set and the network connectivity. In this paper, we attempt to combine the benefits of the online learning model [18] with the asynchronous average strategy as proposed by [12] and design a robust edge analytics algorithm.
IV. ALGORITHM DESIGN

To design machine learning algorithm for a specific application, we assume that the feature vectors are already known, and the incoming data is normalized. However, the other two important characteristics of big data i.e Volume and Velocity are still an issue while dealing with a specific application. For instance in seismic imaging, the row size $m_i$ of the samples $\{x_i, y_i\}$ at node $i$ increases with the occurrence of vibration. Therefore, the computation algorithm should be iterative in nature, which processes single row or multiple rows at a time. Since the computation is carried out on the edge, the algorithm should also be lightweight and robust. The optimization algorithm suitable for these scenarios are based on the method of online learning [1] and the Stochastic Gradient Descent (SGD) is the most prominent among them.

A. Stochastic Gradient Descent

Unlike, Conjugate Gradient and batch gradient descent, SGD does not require knowledge of a complete training set. This method performs a random projection onto a set of available training set (hyperplanes) until convergence. The method starts with an arbitrary initial vector $\theta^0$ and at every iteration $k$, it randomly selects a row $i(k) \in \{1, \ldots, m\}$ of training set. It then uses the sampled training set to calculate the gradient based on the local loss function i.e $\partial f_i(x_i, y_i)$. The parameter $\theta^k$ is updated by moving a small step size $\alpha$ along the negative gradient as shown in Algorithm 1.

Algorithm 1: Stochastic Gradient Descent (SGD)

1: Initialize: $\alpha$, iterations $T$, $\theta^0 \leftarrow 0$
2: for $k \leftarrow 0$ until convergence or maximum iteration $T$ do
3: \hspace{1em} draw $i \in \{1, \ldots, m\}$
4: \hspace{1em} update $\theta^{k+1} = \theta^k - \alpha \partial f_i(x_i, y_i)$
5: end
6: return $\theta$

B. MapReduce SGD

Due to inherent sequential nature, SGD algorithm does not scale very well and is hard to parallelize [10]. With the influx of bigdata, there has been renewed interest in large scale machine learning methods [18] especially for MapReduce and more recently Spark architectures. The main idea behind these frameworks are to partition data and distribute them onto the compute nodes (MAP) followed by joining or aggregation of the partial results from each node (REDUCE). This process is repeated until convergence. Algorithm 2 gives a brief description the MapReduce scheme and the application of this on seismic imaging problem can be seen here [5].

Line 5 of the Algorithm 2 essentially performs multiple rounds of SGD in parallel on node $i \in \{1, \ldots, N\}$ using the initial parameter $\theta^{(k)}$ obtain from the $k$-th iteration. Line 7 denotes the REDUCE operation that computes $\theta^{(k+1)}$ by averaging the partial solutions from all the nodes. The reduce operation requires synchronization and aggregation. The communication cost of this method is of the order $2kNn$ where $k, N, n$ represents iteration, the number of nodes and size of $\theta$ respectively. These methods are suitable for GPU, Multi-core or cloud-based architecture which can facilitate a scatter-gather operation via a coordinator node. Data collection and dissemination on a large scale mesh network can become challenging and can limit the systems scalability. Our previous research [5] shows that applying MapReduce scheme on an edge network can create bottleneck near the coordinator node and also decreases the performance with increase in the node size. The multihop communication increases the communication overhead, packet loss, and slows the convergence. These practical limitations motivated us to develop another class of algorithm to solve Eq. (2) that limits the communication only with the neighbors.

C. Edge Stochastic Gradient Descent

To overcome the drawbacks of MapReduce, we introduce a novel algorithm called EdgeSGD, which is completely decentralized (no coordinator) and does not require synchronization. Unlike a single coordinator based reduce strategy [18] used in MapReduce, we propose a decentralized reduce operation based on Gossip Averaging [2]. Gossip methods are emerging as a new communication paradigm for large-scale distributed systems [14]. Some of the features that makes gossip methods attractive are: 1) absence of central entity or coordinator node 2) high fault tolerance and robustness 3) self healing or error recovery mechanism [15] 4) efficient message exchange due to only neighbor communication 5) provision for asynchronous communication. These interesting characteristics make them suitable for edge nodes to carry out decentralized computation.

In the gossip reduce, when a node $i$ activates at $k$-th iteration, the following set of events occur: i) Node $i$ sends its current parameter estimation $\theta^k_i$ to its neighboring node $j$. ii) Node $j$ receives $\theta^k_i$ and updates it in the following way: $\theta^{k+1}_j = \beta \theta^{k+1}_i + (1 - \beta) \theta^k_j$ where, $\beta \in (0, 1)$. iii) Node $j$ sends $\theta^{k+1}_i$ to $i$, where it updates $\theta^{k+1}_i = \theta_{i}^{k+1}$. In order to avoid race condition, we must ensure that no two nodes update their neighbors value at the same time. Notice that gossip reduce operation performs average only with its neighbor. Next, we will show how this can be used to design EdgeSGD algorithm.

We assume that node $i$ contains $m_i$ samples of training set $\{(x_1, y_1), (x_2, y_2), \ldots, (x_{m_i}, y_{m_i})\}$. The row size increases with the occurrence of events (seismic vibration) and gets appended to the existing set without discarding the previous data. Since the algorithm is decentralized and asynchronous we allow nodes to wake up at anytime. In practice, we can control the wake up cycle based on the events or the energy available. Once a node $i$ wakes up, it performs Gossip Reduce.

Algorithm 2: MapReduce SGD

1: Initialize: $\alpha$, iterations $T$, Node $N$, $\theta^0 \leftarrow 0$
2: Map data each nodes
3: while not converged do
4: for all $i \in \{1, \ldots, N\}$ parallel do
5: \hspace{1em} $\theta^k_i = \text{SGD}(x_i, y_i, T, \alpha, \theta^0)$
6: end
7: $\theta^{k+1} = \frac{1}{N} \sum_{i=1}^{N} \theta^k_i$
8: end
with one of its neighbors $j$ given by, $\tilde{\theta}^k_{ij} = \beta \theta^k_i + (1 - \beta) \theta^k_j$. Next, if $||\theta^k_i - \theta^k_j|| > \epsilon$, $\epsilon$ is the update threshold, then we invoke SGD locally at node $i$. Similar step happens at node $j$ as well. Also, parallel updates can happen between any other pairs in the network independently. In case other nodes try to communicate with a pair say $i$ and $j$ while they are in between their update phase, then the received packet is either buffered or dropped.

**Algorithm 3 Edge Stochastic Gradient Descent**

1. **Initialize**
   - Node ID $i$, $\epsilon$, $\alpha$, $\beta$, iteration $T$, Total Nodes $N$.
   - Node $i \in V$ has $m_i$ samples
   - $\theta^0 \gets$ previously recorded estimates.

2. **Upon the detection of an event**
   - Node receive samples $x$, $y$.
   - $k \gets 0$, $\theta^k \gets \theta^0$

3. **while not converged do**
   - Node $i \in V$ wakes up uniformly.
   - Node $i$ now selects node $j \in N_i$
   - At Node $i$:
     - $\tilde{\theta}^k_i \leftarrow (\beta \theta^k_i + (1 - \beta) \theta^k_j)$
     - if $||\theta^k_i - \theta^k_j|| > \epsilon$ then
       - $\theta^{k+1} = SGD(x, y, T, \alpha, \tilde{\theta}^k_i)$
   - end if
   - At Node $j$:
     - $\tilde{\theta}^k_j \leftarrow (\beta \theta^k_i + (1 - \beta) \theta^k_j)$
     - if $||\theta^k_i - \theta^k_j|| > \epsilon$ then
       - $\theta^{k+1} = SGD(x, y, T, \alpha, \tilde{\theta}^k_j)$
   - end if
   - $k \gets k + 1$

4. **end while**

5. **TERMINATE**

Algorithm 3 outlines the EdgeSGD algorithm. In Line 10 and 15, the SGD randomly picks few samples from the local training set to obtain the local partial estimates. These partial updates are merged using gossip reduce given in Line 8 and 13. This process is repeated until convergence. This algorithm is adaptable to new arrival of the updates and the random sample size can always be adjusted based on the internal memory of the edge device. In algorithm 3 we show the update process only in terms of node $i$ and $j$, and similar updates happen in parallel with other nodes in the network. The proposed EdgeSGD is adaptable, robust and fault tolerant. We will evaluate these characteristic in the next section.

**V. Evaluation**

In this section, we evaluate the EdgeSGD algorithm and provide the empirical results using both synthetic and real seismic data set. We carry our evaluation on a edge computing testbed consisting of 16 BeagleBone nodes Fig. 2(a). We compare our results with MapReduce algorithm [4], DGD [16] and EXTRA [11]. Our results indicate that the proposed method is faster in terms of execution time, robustness and has balanced communication cost. We also observe that MapReduce takes fewer iteration than EdgeSGD, however, takes longer time per iteration due to synchronization overhead.
X and Y direction. Since EdgeSGD outperforms DGD and EXTRA, we now only compare it with MapReduce. From this we can see that the proposed Edge algorithm’s result is closer to the optimal and MapReduce method.

**TABLE I**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Distributed Computation</th>
<th>Decentralized Communication</th>
<th>Epochs</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGD</td>
<td>No</td>
<td>No</td>
<td>22</td>
<td>205</td>
</tr>
<tr>
<td>MapReduce</td>
<td>Yes</td>
<td>No</td>
<td>35</td>
<td>175</td>
</tr>
<tr>
<td>EdgeSGD</td>
<td>Yes</td>
<td>Yes</td>
<td>60</td>
<td>110</td>
</tr>
<tr>
<td>EXTRA</td>
<td>Yes</td>
<td>Yes</td>
<td>175</td>
<td>232</td>
</tr>
<tr>
<td>DGD</td>
<td>Yes</td>
<td>Yes</td>
<td>200</td>
<td>251</td>
</tr>
</tbody>
</table>

**Fig. 3.** Visual comparison of the result obtain from a) Layer 7 of Ground truth or the optimal solution b) MapReduce algorithm with distance from the optimal \( \phi = 0.0019b \) Edge SGD algorithm with distance from the optimal \( \phi = 0.0054 \). These results are obtained after 30 epochs and from this we can see that the proposed Edge algorithm’s result is closer to the optimal and MapReduce method.

**Fig. 4.** (a) Convergence characteristic of different algorithms against number of Epochs. We see that EdgeSGD converges to the same solution as MapReduce. (b) Time (sec) taken by EdgeSGD and MapReduce to decrease its relative error. We see that around 85th sec MapReduce stalls due to a delay in the node 10. The speed of MapReduce is governed by the slowest node in the network and is not suitable for Edge analytics.

MapReduce takes lesser epochs to converge to a true solution. However, from Fig. 4(b) we see that MapReduce takes longer time per epoch due to synchronization overhead. The speed of the MapReduce is governed by the slowest node in the network. In Fig. 4(b) we see this effect, where, node 10 temporarily stalls at 85th sec for about a minute preventing other nodes to continue. In EdgeSGD, due to the decentralized nature, failure of one or group of nodes does not have significant effect on the overall convergence rate and is shown in Fig. 5(c).

A detailed comparison of different algorithms are given in Table I. Except SGD, all the other algorithms are run on 16 nodes. Each node has 5K training set with 4096 features. Since SGD is centralized, it takes very few epochs to reach true solution, on the other hand, each epoch requires frequent disk access due to large training set. This increases the execution time. DGD and EXTRA both have their computation and communication distributed, however these methods are slower than EdgeSGD which also has similar property. Properties such as robustness, asynchrony, distributed computation and communication makes EdgeSGD algorithm a suitable choice for Edge analytics.

Next, we demonstrate the correctness of our algorithm through visualization. Fig. 3 shows the layer 7 of 16 along X and Y direction. Since EdgeSGD outperforms DGD and EXTRA, we now only compare it with MapReduce. From this experiment, we can see that EdgeSGD is able to generate sharper image similar to MapReduce. This evaluation suggests that EdgeSGD which has other advantages over MapReduce can be a good candidate for decentralized seismic imaging.

**D. Robustness**

One of the important characteristic of EdgeSGD is its fault tolerance and here we validate it by simulating node and link failure. We run each experiments for 200 sec and with three different cases. **Case 1)** No Failure **Case 2)** 25% of the node fail for 10% of the time **Case 3)** 50% of the nodes fail for 10% of the time. From Fig. 5(d) we see that there is no significant effect on the convergence due to node failure. However, if more than half of the nodes fail then the algorithm slows as on an average there would be no neighbors to communicate with. However, MapReduce requires all nodes to be running and its speed depends on the slowest node. This property is extremely important when choosing algorithms for edge computing.

**E. Communication Cost**

We now compare the communication pattern of MapReduce and EdgeSGD. We plot the number of messages exchanged by each node on a grid to finish 200 epochs. The x and y axis in Fig 5 shows the 4 × 4 grid layout and the z axis denotes the number of messages exchanged. We can see that the communication is balanced in EdgeSGD and each node exchanges roughly around 250 messages. Whereas in MapReduce since we use center node as a coordinator, we observe an imbalance communication pattern. There is an increased message overhead near the center node roughly accumulating to 300 messages. Although, messages exchanged by other nodes are lower than in EdgeSGD, reliance on central coordinator will limit the systems scalability and the performance. Building the infrastructure with special coordinator, routing protocols and synchronization adds extra overhead and is not suitable for decentralized systems such as edge network.

**F. Effect of Topology**

Topology plays a key role in decentralized algorithms like EdgeSGD. Since the method communicates only with its immediate neighbors, the topology decides how fast the information diffuses in the network. Therefore, EdgeSGD on a strongly connected topology such as complete graph (Kn) converges faster than Grid topology as shown in Fig. 5(d).

**G. Imaging Parkfield, CA**

In this section, we will apply our algorithm to a real-data set and compare the results with the existing centralized method [17]. The dataset we use here is obtained from Parkfield, CA which is located over the San-Andreas fault region. The goal is to obtain the P-wave velocity map of the region or
in other words image of the fault zone. We preprocess the raw seismic signal using the algorithms given in [5, [7].

Fig 6 shows the horizontal slices of the P-wave velocity at 1 Km depths. Image in Fig 6(a) is obtained using centralized algorithm as presented in [17]. This is currently the widely accepted model of the San-andreas fault region. The result from our proposed algorithm is shown in Fig 6(b). From the visual comparison our decentralized algorithm is able to obtain the main features of the fault similar to the centralized algorithm. These results are very promising and we hope to extend it to other seismic monitoring sites as well.

VI. CONCLUSION

In this paper we presented a novel decentralized stochastic gradient descent algorithm, EdgeSGD for solving machine learning problem such as linear regression on a edge network. We demonstrated the applicability of such method using a real-world problem of learning/predicting seismic anomaly via real-time imaging. EdgeSGD outperformed existing MapReduce algorithm and also decentralized P2P based machine learning methods such as DGD and EXTRA on an edge computing testbed. In future, we intend to explore the use of one-sided communication protocol for edge analytics.

ACKNOWLEDGMENT

Our research is partially supported by NSF-CNS-1066391, NSF-CNS-0914371, NSF-CPS-1135814 and NSF-CDI-1125165.

REFERENCES