

A Load Balancing Scheme for Sensing and Analytics on a Mobile Edge Computing Network

Chen-Khong Tham and Rajarshi Chattopadhyay
Department of Electrical and Computer Engineering
National University of Singapore
E-mail: eletck@nus.edu.sg, e0009104@u.nus.edu

Abstract—The processing time of real-time sensing, data analytics and other applications is an important performance metric. When the application is being executed on a distributed system, the load balancing scheme among processing nodes significantly affects the total processing time of the application. We consider a load balancing scheme for distributed computing at the edge of the network. In the edge model considered, a group of nodes, either mobile or static, with processing and sensing capabilities, are connected to each other over a wireless ad-hoc network. Load balancing among edge nodes is formulated as a min-max optimization problem, with the objective of minimizing the overall processing time of the application while still satisfying the wireless channel capacity and link contention constraints. We use an aggregate utility method to convert the min-max problem into a convex optimization problem. The obtained constrained convex optimization problem is then relaxed with Lagrangian dual decomposition and solved with gradient descent. This form of the formulation can be implemented in a fully distributed manner among the edge nodes, which is consistent with the decentralized nature of edge networks. However, the convergence of this scheme may be slow. We further propose a heuristic algorithm which achieves fast convergence. Our simulation results show that it can give near-optimal performance most of the time.

Index Terms—Mobile Edge Computing; Ad-hoc Networks; Distributed Systems; Mobile Computing; Lagrangian Dual Decomposition

I. INTRODUCTION

In this age of ubiquitous computing and the Internet of Things (IoT), processing, sensing and wireless communication capabilities can be found in most handheld devices, connected vehicles, road-side processing units and wireless sensor networks. Edge devices include all such devices. The sensing capabilities of these devices generates a large amount of data. Smartphones are equipped with cameras, audio-recorders, accelerometers, gyroscopes and other sensors. The data being generated from these sensors can be analyzed and used for gaining insights in a variety of situations. This opens the door for a large number of new analytical applications which have the potential to improve our lives. An example application is traffic monitoring and prediction in nearby regions. In [1], the traffic situation in a road network is monitored by a group of mobile sensors specially deployed for this purpose. Another approach could be to use the sensors in the nearby connected vehicles to collect data and analyze it to gain insights about the local traffic.

In recent years, cloud computing has become a popular computing and storage solution. Each node sends its data to a

server in the cloud and the server analyses the data collected from many nodes and sends the results back to the users. However, this often incurs latency or delay and increases the traffic on back-end networks. To avoid this, we can turn to the computing capabilities of edge devices to locally process data, with possible cooperation among these devices when there is unbalanced data. This requires distributed data processing algorithms and good load balancing schemes implemented in a distributed manner with no centralized authority to coordinate the process.

Cisco has coined the term “fog computing”, which involves pushing data processing to edge devices [2]. It is also mentioned in [2] that about two exabytes of data is being generated daily from the Internet of Things (IoT). Processing this data at the edge devices will reduce processing latency, network delay and reduce the need for bandwidth additions to the core network. Moreover, the number and compute capacities of edge devices are increasing rapidly, and making use of this enormous distributed computing resource will make it possible to develop new applications and improve user experience.

Edge computing can also be considered in wireless sensor networks (WSN). Typically, WSN nodes send their sensed data to a sink node. The sink node is a gateway which further sends the data to cloud servers or other designated computing resources on the Internet for analysis. [3] proposes a distributed framework for minimizing the total energy consumed by the network for correlated data gathering in WSN's. [4] on the other hand, proposes a cross-layer optimization framework for network lifetime maximization while gathering correlated data. They consider varying the physical layer transmission power in their cross-layered framework. In-network data processing can also play a major role in improving the network lifetime and energy consumption performance of the network. One main reason for this is that, for typical applications, communication consumes a lot more energy per unit data as compared to processing. In-network processing of data can greatly reduce the amount of data required to be communicated. [5] describes the trade-off between communication and memory usage for distributed training of neural networks in WSNs. [6] discusses in-network data processing and sensing in energy harvesting sensor networks.

In many scenarios, several edge devices can be considered to have formed a wireless ad-hoc network, with computation, storage, sensing and data transmission capabilities in each node. We refer to such networks as “Edge Networks” (EN).

This is a common scenario in any crowded area with many people carrying smart devices like smartphones, or a vehicular ad-hoc network of connected cars on urban roads. There are many challenges to be overcome before this distributed computing resource can be fully utilized for executing useful applications. In [7], the state-of-the-art modeling techniques and research issues in ad-hoc networks has been summarized. Specifically, node mobility, wireless interference, node spatial distribution and information delivery aspects have been considered. Node mobility is an important characteristic of wireless ad-hoc networks, and it causes some randomness to it. Mobile nodes can move out of each other's communication range anytime and thus, lead to communication failures. Hence, the network is never stationary and the topology keeps changing. However, they can be assumed to have a static topology over short periods of time, depending on the relative velocity of the mobile nodes. Wireless interference is another major issue. Unlike wired networks, the wireless channel is a shared resource, and communication between two nodes can interfere with the communication of neighboring nodes. [8] describes how this interference limits the data rates that such a network can support. The work in [9] describes a cross-layer congestion control, routing and scheduling scheme for ad-hoc wireless networks. Each source node attains some utility when it is able to transmit data over the wireless ad-hoc network. The objective is to be able to maximize the overall utility given the wireless channel constraints in a distributed manner.

Edge Networks (EN) can be viewed as flexible distributed computing resource, with time varying network topology and constrained communication between nodes due to the shared wireless channel as challenges that need to be overcome. As mentioned above, some of the edge devices have sensing capabilities and can sense data to be utilized by analytical applications, e.g. to learn classification and regression models. Conventionally, cloud services are used to process this data. [2] and [10] state that processing data closer to the network edge, or closer to the source, results in lower latency and reduced network congestion. Hence, it makes sense to execute these analytical applications on edge networks. In this work, we consider a scheme to minimize the processing time of such applications, while considering the edge network characteristics. The fact that many machine learning and data analytics algorithms can be written in a "summation form" as shown in [11], enables us to speed them up using distributed computing resources. Here, we try to optimize this speed up, keeping in mind the edge network constraints. More specifically, we formulate a load balancing optimization problem for executing a learning algorithm on edge devices in an edge network.

The rest of the paper is organized as follows. In Section II, we position our work with respect to several related works. Section III contains the optimization problem formulation and Section IV talks about the distributed implementation. Section V describes the simulation setting and presents the results. Section VI concludes the paper.

II. RELATED WORKS

The work described in this paper lies at the intersection of wireless ad-hoc networks, distributed computing and edge computing. Wireless ad-hoc networks have been researched widely for some time. [7] summarizes some of the major research issues in the area including mobility models and wireless interference models. [9] gives an example of designing a distributed cross-layer protocol for wireless ad-hoc networks. They considered congestion control, and there is a large body of work utilizing similar approaches. [12] considers the network lifetime maximization problem for wireless networks.

Closely related to general ad-hoc wireless networks are WSNs. Sensor nodes are generally not mobile, but they do form a wireless ad-hoc network, and some of the protocol design methodology can be applied in the general WSN case. The work in [3] and [4] involve the design of cross-layer distributed protocols for correlated data gathering with minimum energy consumption and maximum network lifetime in WSNs respectively. [5] shows how a neural network can be distributed over a WSN whereas [6] considers in-network data processing in an energy harvesting WSN. All the design methodology of WSN protocols can be considered in designing edge network protocols, with certain quantities recomputed at regular intervals as the network topology changes slowly.

The learning algorithms for a large number of data analytics schemes can be executed in parallel on distributed systems. [11] describes parallel implementations of learning algorithms using the MapReduce framework [13]. MapReduce is a general framework for implementing a large number of data analytics algorithms on computer clusters. [14] presents an SDN-assisted deployment of MapReduce on WSNs. Although the learning algorithms can be implemented using the MapReduce framework, and this reduces the processing time significantly, it is not necessarily processing time optimal. In this work, we consider a scheme which is both processing time optimal and does not require a centralized SDN controller for its implementation, and can be implemented in a distributed manner. We will refer to the edge devices in the edge network as nodes.

III. SYSTEM AND PROBLEM FORMULATION

A. Considered Scenario and Network Setup

An edge device user wants to execute the learning algorithm for a particular data analytics model. The user's own smart device may or may not have data locally for the learning task. In any case, if the amount of data is not enough on one device, the learning algorithm will overfit the data. Furthermore, the user's device may not have sufficient processing capacity and the algorithm may take a long time to converge. One option available to the user's device is to get help from nearby devices, leveraging both their processing capacity and data. This is depicted in the edge network (EN) shown in Figure 1.

The user broadcasts a message asking for help from nearby devices. In the message, the user's device also sends its MAC address to distinguish it from other request messages. On

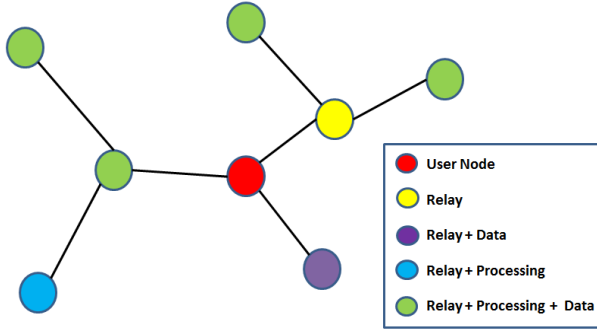


Fig. 1. Topology of a typical edge network with a user node and neighbouring nodes with different roles. Nodes which can contribute data have sensors attached to them.

receiving the request from the user's device, the nearby devices decide whether or not to help and further relays or broadcasts the message so that it reaches their neighbors. A node may receive the request message more than once in which case it will only consider the first time and discard the rest. In this way, the request for help propagates to the devices in the neighborhood. A particular device may decide not to provide processing capacity and data, but may still be useful as a relay. If a device does not want to help as a relay either, then it does not broadcast the request for help, since this may result in a disjoint network. There must be a termination criterion for this process. The initial user can put a time-stamp when the request is generated, along with a termination time. When the termination time is reached, the nodes stop propagating the request further, or the initial user may set a number which is decreased by one each time the message is broadcast. The process terminates once the number reduces to zero. In this way, an edge network will be set up.

B. Node Mobility and Processing Time

Some of the nodes in the EN just set up may be mobile and hence the network topology is not static. However, the network can be assumed to be static over short periods of time depending on the mobility, e.g. if the nodes are smart devices belonging to people walking nearby, then the network topology can be assumed to be constant for 1 to 2 seconds since the average walking speed is 1.4 m/s and the communication range of devices, denoted by R , is about 70 to 100 m. Hence, we can assume there will be no major change in network topology within this period.

The processing time of the learning algorithm being executed by an EN is defined as the time after which each node in the EN has no data remaining to be processed. Typically, data in the nodes will be unbalanced, i.e. some nodes will have more data than others, and hence, the processing time can be reduced if the nodes with more data transfer some of them to nodes with less data. Due to the mobility of nodes, some of the nodes may move out of the communication range of the other nodes while the learning algorithm is still being processed. The definition of processing time only makes sense when all the nodes which agreed to provide data and processing

are still connected in some way. Here, if some of the nodes get completely separated from others, we will ignore their contributions to processing time.

C. Overview of Proposed Scheme

The main objective of our scheme is to perform load balancing among the nodes of the EN with processing capabilities. Load balancing requires the movement of data from one node to another. Since the nodes communicate with each other via the shared wireless channel, feasible data rates between nodes are constrained. Hence, the data transfer between nodes may take a significant amount of time. Our proposed scheme determines the data transfer rates between nodes to balance the load so that the overall processing time is minimized. In ENs, the feasible data rates are determined by the channel capacity and network topology. As discussed in the previous subsection, although the network topology changes due to mobility, it can be considered nearly static for short periods of time. We assume that the network topology remains nearly static for τ seconds. Hence, we recompute the appropriate data rates between nodes every τ seconds. Our scheme can be computed in a distributed manner and converges quickly after making several simplifications.

D. Practical Implementation

So far, we have described the scenario being considered and the main purpose of our proposed scheme. Before proceeding further, we discuss practical implementation considerations. The load balancing scheme in this paper is designed keeping four things in mind. First is improvement in user experience which ultimately drives the development of new protocols and algorithms, and reducing the processing time of applications is one of the major ways to improve user experience. Second is its compatibility with the underlying system it is designed for, in this case, edge networks. Compatibility with the underlying system here implies that it satisfies all the constraints imposed on it by the system. If a scheme does not satisfy the system constraints, then the solution or policy obtained will not be feasible, resulting in poor user experience. The proposed scheme tries to provide the best possible user experience within the given constraints. Third is its applicability for a variety of data analytics algorithms which can be written in the "summation form" as in [11]. A load balancing scheme which is applicable only for a small number of algorithms is not useful because then, there will be a need to design many new schemes. Fourth is the reasonably low computation time of the scheme. A load balancing scheme should not take up too much time for its own computation as this slows down the entire algorithm. These features make our scheme practical and suitable for implementation.

This scheme can be used by data analytics based edge device applications as a platform to build upon. The data analytics algorithms can focus on their own processing, while leaving the details of data transfer over edge networks to the proposed scheme. This kind of interaction with the algorithms will enable the quick development of data analytics based edge

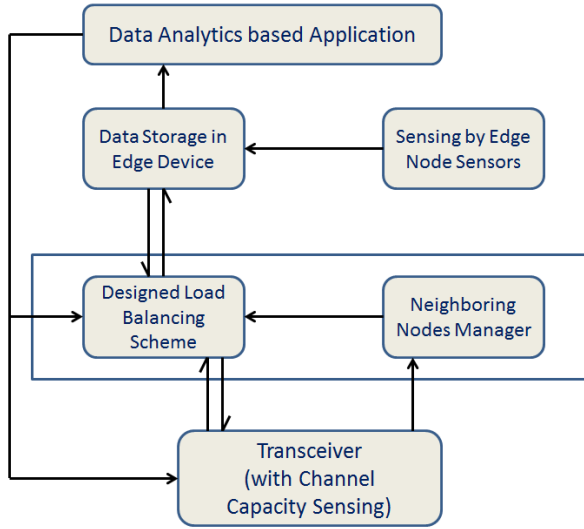


Fig. 2. Various functional modules at each node.

device applications as the developer will only need to focus on the application's goals and objectives, being assured of best possible workload distribution, without getting into the underlying details of edge networks.

Figure 2 shows the overall working of the Load Balancing Scheme on an edge device. There are several modules involved. Each node must have all these modules for the successful execution of the proposed scheme. As mentioned earlier, the proposed scheme is designed specifically for learning algorithms of data analytics models which can be written in a specific "summation form". The arrows show the transfer of data and information between the various modules. The Data Analytics based Application module consists of the distributed version of the learning algorithm being executed on the edge network. It only receives the data it needs to process from the data storage module, which handles the storage of data in the memory of the edge device. The rest of the stored or sensed data is either processed gradually or transferred to neighbouring nodes to process. The role of the Neighbouring Nodes Manager is to keep track of the MAC address, processing capacity and workload of the neighbouring edge nodes within communication range and it updates these information periodically every τ seconds with the help of the transceiver module. It exchanges such information with the load balancing scheme so that it can decide on the optimal data transfer rates for load balancing. Note that the load balancing scheme also requires some application specifications from the Data Analytics based Application module, such as how many CPU cycles are required to process one training sequence. The transceiver module communicates the data and information to and from all the other nearby edge devices within communication range. From the point of view of other modules, it simply accepts data and a destination MAC address and transfers the data as desired. We assume that the transceiver can sense the channel capacity. The role of

the sensing module is to store the data sensed by edge node sensors in the local memory of the edge node.

The Transceiver module, Data Storage module and Sensing modules are fairly standard, and will typically be present in an edge device. The Neighbouring Nodes Manager module may not be typically present on an edge device, and there may be a need to install it. In the following sections, we describe the algorithms running in the Designed Load Balancing Scheme module in detail. In Figure 2, these two modules have been boxed together because the Neighbouring Nodes Manager module may not be present on all edge devices and it is better to install both together. In terms of various networking layers, the Transceiver module contains all the layers from physical to transport, and is responsible for providing reliable communication between the edge network nodes. All other modules are in the application layer and ensure that the data analytics application interacts optimally with the underlying edge network. Note that the application developer only needs to focus on the Application module, thus making the development of new applications straightforward.

E. Wireless Channel and Network Topology Constraints

The nature of the wireless channel brings about the concept of *channel capacity*, which is the maximum rate at which data can be reliably communicated using the channel. The channel capacity depends on the signal to noise ratio (SNR) of the received signal at the receiver. Typically, SNR is determined by the transmit power, channel gain and environmental noise. We assume that a fixed channel capacity C is maintained between nodes. Note that this means we are assuming that there are transmit power control schemes present which maintain a fixed channel capacity in the face of varying environmental noise and channel gain. Furthermore, the shared wireless channel causes interference between the EN links. There are various ways of modeling this interference as summarized in [7]. Here, we consider the one hop interference model.

The EN can be thought of as a directed graph $G = (V, E)$, where the vertices set V is the set of nodes and edge set E is the set of all links between the nodes. A link exists between two nodes only if they are within each other's communication range R . Note that we assume that all nodes have the same communication radio range R . If node i and node j are within each other's range, then there are two directed links ij and ji . A contention set of a link ij is defined as the set of links with which link ij cannot transmit simultaneously. It is denoted by ϕ_{ij} . The contention sets of all the links in the EN will depend on the choice of interference model. In the case of the one hop interference model, the contention set of a link will contain all its one hop neighbor links. For convenience, we include a link in its own contention set. The solid lines in Figure 3 shows the links in the contention set of link ij , which is itself shown in red. The dashed lines indicate links which are not included in the contention set of link ij .

Let r_{ij} denote the flow rate on link ij . The channel capacity and link contention sets impose certain restrictions on the feasible values of flow rates of all the links. [8] proposes

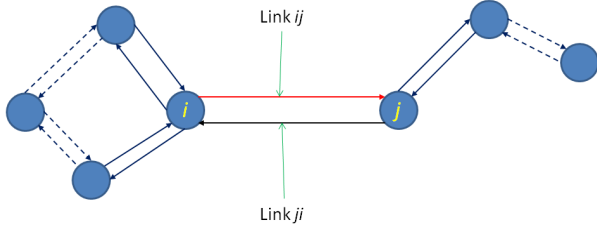


Fig. 3. Links with solid lines are in the contention set ϕ_{ij} of link ij .

a sufficient condition for the feasibility of the network flow rate values. It is shown that if the following condition holds for a particular choice of link rate values, then that choice is feasible:

$$\sum_{(p,q) \in \phi_{ij}} r_{pq} \leq C; \forall (i,j) \in E \quad (1)$$

F. System Formulation

We formulate processing time minimization as an optimization problem. The objective is to find link data rates r_{ij} so as to minimize the processing time while satisfying constraint 1. Table I lists all the notations used. The processing time of a node is the time it takes to process all the training data samples it has. We assume that the computational effort required is proportional to the number and size of the training data samples. If there are no data transfers between nodes, then the time each node takes is the amount of data it has, divided by the amount of data it processes in one second, i.e. its data processing capacity, and the processing time of the group of nodes would be determined by the node that finishes last.

However, transferring data between nodes at the assigned rates r_{ij} may help to reduce the processing time. The overall processing time would be minimized if, after data transfer, the ratio of data to processing capacity is the same for each node. We will refer to this optimal redistribution of data as Equal Ratio Distribution (ERD). However, given constraint (1), data transfer is not instantaneous, and it may take a significant amount of time to achieve ERD. On top of that, changes in the EN topology affect constraint (1), since the contention sets change with the topology and this would in turn affect the time required to achieve ERD. As mentioned earlier, our scheme recomputes the link data rates every τ seconds to deal with the mobility of nodes; hence, a set of calculated link data rates is used for τ seconds. Although it may not be possible to achieve ERD within τ seconds, the r_{ij} values can be selected such that we get as close to ERD as possible. The ratio of data to processing capacity of a node is the same as its processing time. Hence, the objective is to get the processing time of each node as close as possible to the processing time at ERD, after τ seconds. More specifically, the intention is to minimize the maximum processing time at the end of τ seconds, since the ultimate goal is to minimize the overall processing time. Note also that the overall processing time cannot be lower than the processing time at ERD. After τ seconds, the remaining

TABLE I
NOTATIONS USED

Parameter	Notation
D_i	Initial number of training samples at node i (in Mb)
d_i	Sensing rate at node (in Mbps)
τ	Time duration for which the EN topology is assumed static (in seconds)
r_{ij}	Data transfer rate from node i to node j (in Mbps)
N	Total number of nodes in the EN
$Ne(i)$	Set of neighbouring nodes of node i
l_s	Size of one training sequence (in Mb)
P_i	Processing power of node i (in cycles per second)
C_s	CPU cycles required by the learning algorithm to process one training sample
ϕ_{ij}	Contention set of link ij
C	Channel capacity (in Mbps)
R	Wireless radio range of nodes (in meters)

processing time (or the ratio of data to processing capacity) for node i , denoted by T_i , would be:

$$T_i = \frac{\frac{D_i}{l_s} + \tau \cdot \left[\frac{d_i + \sum_{k \in Ne(i)} r_{ki} - \sum_{k \in Ne(i)} r_{ik}}{l_s} - \frac{P_i}{C_s} \right]}{\frac{P_i}{C_s}} \quad (2)$$

See Table I for the definition of the terms in this expression.

Here, the overall processing time remaining after τ seconds is $\max_i T_i$. We aim to minimize overall processing time subject to constraint (1). In summary, we need to solve the following:

$$\begin{aligned} P1 : \quad & \min_{r_{ij}} \max_i T_i \\ \text{s.t.} \quad & \sum_{(p,q) \in \phi_{ij}} r_{pq} \leq C; \quad \forall (i,j) \in E \\ & r_{ij} \geq 0; \quad \forall (i,j) \in E \end{aligned} \quad (3)$$

IV. DISTRIBUTED IMPLEMENTATION

Note that to solve $P1$, global network information is required. Specifically, it requires the knowledge of D_i , d_i and P_i of each node in the EN. This can be viewed as a centralized implementation and may not be practical for ENs with a large number of nodes. Another problem with a centralized implementation is that, in general, there is no central coordinator in ENs. The network is formed out of nodes which just happen to be in each other's range with the exact topology changing over time due to mobility. Hence, a distributed solution approach is not only desired, but also natural for a distributed computing resource like an EN. To this end, we perform Lagrangian Dual Decomposition (LDD) of the constrained optimization problem $P1$, an approach that is commonly used for vertical

and horizontal decomposition of Network Utility Maximization (NUM) problems, as demonstrated in [3], [4] and [15]. However, before performing LDD, we need to convert the min-max optimization $P1$ into a convex optimization problem.

A. Conversion to Convex Optimization Problem

We achieve this by applying the method introduced in [12], which was also applied in [4]. Formulation $P1$ is converted to a convex, constrained minimization problem. The min-max objective function is converted into one that minimizes the aggregate utility, where utility is given by $T_i^{\alpha-1}/(\alpha-1)$. The formulation is as follows:

$$P2: \quad \min_{r_{ij}} \quad \frac{\sum_{i=1}^N T_i^{\alpha-1}}{\alpha-1} \\ \text{s.t.} \quad \sum_{(p,q) \in \phi_{ij}} r_{pq} \leq C; \quad \forall (i,j) \in E \\ r_{ij} \geq 0; \quad \forall (i,j) \in E \quad (4)$$

Theoretically, formulations $P1$ and $P2$ are the same when $\alpha \rightarrow \infty$. However, practically, a value of $\alpha = 4$ works well, as demonstrated in [12]. The convexity of formulation $P2$ can be easily shown. All T_i 's are linear in r_{ij} 's and the objective function of $P2$ is a sum of higher powers of T_i , making it convex. The constraints are simply linear inequalities.

B. Lagrangian Dual Decomposition

The convexity of formulation $P2$ implies that there is no duality gap between the optimal primal and dual solutions. The Lagrangian function of $P2$ can be written as:

$$L(r, \mu) = \sum_{i=1}^N \frac{T_i^{\alpha-1}}{\alpha-1} + \sum_{(i,j) \in E} \mu_{ij} \left(\sum_{(p,q) \in \phi_{ij}} r_{pq} - C \right) \quad (5)$$

Here, r is used to denote the vector of all r_{ij} 's and μ the vector of the Lagrangian multipliers μ_{ij} 's. The dual function of the Lagrangian function above is:

$$W(\mu) = \min_{r \geq 0} L(r, \mu) \quad (6)$$

On differentiating the Lagrangian function (5) with respect to r_{ij} , we get:

$$\frac{\partial L}{\partial r_{ij}} = -T_i^{\alpha-2} \frac{\tau}{l_s P_i} + T_j^{\alpha-2} \frac{\tau}{l_s P_j} + \sum_{\forall (x,y) \in E; s.t. (i,j) \in \phi_{xy}} \mu_{xy} \quad (7)$$

We can update the values of r_{ij} in an iterative manner using the following update rule:

$$r_{ij}(t+1) = \left[r_{ij}(t) - \lambda \frac{\partial L}{\partial r_{ij}}(t) \right]^+ \quad (8)$$

where λ is the step size and $[a]^+ = \max(a, 0)$. The dual problem of formulation $P2$ is:

$$\max_{\mu \geq 0} W(\mu) \quad (9)$$

The gradient of the dual function with respect to μ_{ij} is:

$$\frac{\partial W}{\partial \mu_{ij}} = \sum_{(x,y) \in \phi_{ij}} r_{xy} - C \quad (10)$$

Hence, the following update rule can be used for the μ_{ij} 's:

$$\mu_{ij}(t+1) = \left[\mu_{ij}(t) + \lambda \frac{\partial W}{\partial \mu_{ij}}(t) \right]^+ \quad (11)$$

C. Distributed Solution

The LDD of $P2$ in the previous subsection provides a way of obtaining a distributed solution. We need to compute the values of r and μ to solve the problem. The values are computed using the iterative update rules (8) and (11). The update rules only require the gradients of the Lagrangian function with respect to (w.r.t.) r_{ij} 's and the dual function w.r.t. μ_{ij} 's. The computation of these gradients do not require global network information, i.e. the information from all the nodes in the EN.

Let node i compute the values of T_i , r_{ij} and μ_{ij} ; $\forall j \in Ne(i)$. Here, computing the value of r_{ij} and μ_{ij} means performing the required updates at each iteration. It can be seen from expression (2) that computation of T_i requires information from only neighboring nodes. Specifically it requires the r_{ki} values from all nodes within the communication range of node i . As per expression (7), computing the gradient of the Lagrangian w.r.t. r_{ij} requires the knowledge of T_i , T_j , P_i , P_j , τ , l_s and μ_{pq} of all the links whose contention set contains the link ij . Out of these only T_j , P_j and μ_{pq} have to be obtained from other nodes. T_j and P_j can be obtained from direct neighbors. Note that if a link wz belongs to the contention set of link xy , then link wz also belongs to the contention set of link xy , since we are using the one hop interference model.

This means that all the required μ_{pq} values in Equation (7) and r_{pq} values in Equation (10) correspond to the links in the contention set of link ij . Hence, node i can obtain these values from its direct neighbors and one hop neighbors. The information from the one hop neighbors can be routed to node i through its direct neighbors. Thus, nodes can first gather all the required information from its direct neighbors, and then relay this information to its own neighbors. Assume that a node has β neighbors on an average. Table II summarizes the information received by node i from node j , $j \in Ne(i)$. Node i has to receive a total of 4β parameters and send 4β parameters to each direct neighbor. The sending part can be made more efficient through the use of broadcasting.

Hence, all the computations required for the updates require local network information and the scheme can be implemented in a distributed manner. Algorithm 1 summarizes the distributed algorithm. It is implemented in the Designed Load Balancing Scheme module at each node.

D. Issues with Convergence

The distributed implementation in Algorithm 1 is iterative. Although it requires the gathering of only local information at each node, it requires a large number of iterations to converge. This has been observed in simulations as shown in the next

TABLE II
PARAMETERS RECEIVED BY NODE i FROM NODE j DURING ONE
ITERATION OF ALGORITHM 2 ($i \in Ne(j)$)

Parameter	Average number of parameters
T_j	1 parameter
P_j	1 parameter
μ_j	1 parameter
r_j	1 parameter
$\mu_{kj}; \forall k \in Ne(j), k \neq i$	$\beta - 1$ parameters
$r_{kj}; \forall k \in Ne(j), k \neq i$	$\beta - 1$ parameters
$\mu_{jk}; \forall k \in Ne(j), k \neq i$	$\beta - 1$ parameters
$r_{jk}; \forall k \in Ne(j), k \neq i$	$\beta - 1$ parameters

Algorithm 1

```

1:  $Iterations \leftarrow Set\ Number\ of\ Iterations$ 
2: for  $t = 1 : Iterations$  do
3:   for  $k \in Ne(i)$  do
4:     Get  $T_k$  and  $P_k$  from node  $k$ 
5:      $\forall (p, q) \in \phi_{ik}$ , get  $\mu_{pq}$  from node  $p$ 
6:     Compute  $\frac{\partial L}{\partial r_{ik}}(t)$  using (7)
7:      $\forall (p, q) \in \phi_{ik}$ , get  $r_{pq}$  from node  $p$ 
8:     Compute  $\frac{\partial W}{\partial \mu_{ik}}(t)$  using (10)
9:      $r_{ik}(t+1) = [r_{ik}(t) - \lambda \frac{\partial L}{\partial r_{ik}}(t)]^+$ 
10:     $\mu_{ik}(t+1) = [\mu_{ik}(t) + \lambda \frac{\partial W}{\partial \mu_{ik}}(t)]^+$ 
11:   end for
12: end for

```

section. Here, we propose an alternative heuristic algorithm, presented in Algorithm 2, which is based on the gradient expressions (7) and (10) and update expressions (8) and (11). It converges much faster as compared to the scheme proposed earlier. Its fast convergence in simulations is shown in the next section. The faster convergence of Algorithm 2 as compared to Algorithm 1 can be explained in terms of how the method of Lagrangian multipliers works for constrained optimization problems.

Usually, when decision variables and Lagrangian multipliers are updated in an iterative manner, as in Algorithm 1 and Algorithm 2, the initial values of Lagrangian multipliers are set to zero. Therefore, the Lagrangian function, which is now the objective function, is initially the same as the original objective function, and hence, the iterative optimization proceeds as an unconstrained optimization problem and the decision variables are updated to satisfy the original optimization objective. The Lagrangian multipliers become positive only when the corresponding constraints are violated. If the step size parameter is small, it takes many iterations for the Lagrangian multipliers to be large enough to affect the Lagrangian function significantly and the solution cannot converge unless all constraints are satisfied. This makes the convergence very slow. Applying this reasoning in the context of the algorithms above, the main reason for the quicker convergence of Algorithm 2 is

that whenever there is contention (constraint violation), we allow the link rates to decrease quickly as fractions of its previous value as opposed to the slow and gradual gradient based decrease when Algorithm 1 is used, as explained above.

Algorithm 2

```

1:  $Iterations \leftarrow Set\ Number\ of\ Iterations$ 
2:  $dR = 0.005$ 
3: for  $t = 1 : Iterations$  do
4:   for  $k \in Ne(i)$  do
5:     Get  $T_k$  and  $P_k$  from node  $k$ 
6:      $\forall (p, q) \in \phi_{ik}$ , get  $\mu_{pq}$  from node  $p$ 
7:     Compute  $\frac{\partial L}{\partial r_{ik}}(t)$  using (7)
8:      $\forall (p, q) \in \phi_{ik}$ , get  $r_{pq}$  from node  $p$ 
9:     Compute  $\frac{\partial W}{\partial \mu_{ik}}(t)$  using (10)
10:     $Count = 0$ 
11:    for  $(p, q) \in \phi_{ik}$  do
12:      if  $\mu_{pq} > 0$  then
13:         $Count = Count + 1$ 
14:      end if
15:    end for
16:    if  $Count = 0$  then
17:       $r_{ik}(t+1) = [r_{ik}(t) - \lambda (\frac{iterations-t}{iterations})^{20} \frac{\partial L}{\partial r_{ik}}(t)]^+$ 
18:    else
19:       $r_{ik}(t+1) = [(1 - dR) + dR(\frac{|\phi_{ik}| - Count}{|\phi_{ik}|})] r_{ik}(t)$ 
20:    end if
21:    if  $\frac{\partial W}{\partial \mu_{ik}} \leq 0$  then
22:       $\mu_{ik}(t+1) = 0$ 
23:    else
24:       $\mu_{ik}(t+1) = 1$ 
25:    end if
26:  end for
27: end for

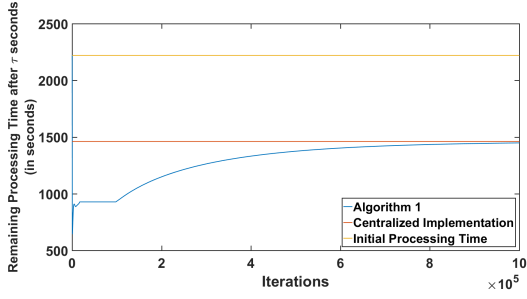
```

Quick convergence is essential for this load balancing scheme, especially since the algorithm recomputes the link data rates after every τ seconds. If each run of the algorithm requires a large number of iterations to converge, then there will be a very large overhead, making it unsuitable for practical implementation.

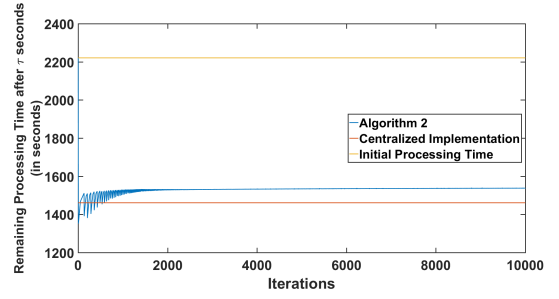
V. NUMERICAL SIMULATION AND RESULTS

To simulate an edge network, 15 edge devices were randomly distributed in a square $150m \times 150m$ area. All the nodes are assumed to have a communication radio range R of 100 m. The workload at each node D_i is uniformly distributed between 0 and 100 Mb¹. The size of each training example is taken to be 0.02 Mb. The data sensing rate at each node d_i is uniformly distributed between 0 and 3 Mbps. The processing capacity of each node P_i is distributed uniformly between 0 and 50 cycles/s and 4 cycles are required to process each training sample. τ is set to 1 s. α is set to 4 and the channel capacity C is set to 10 Mbps. The step size parameter λ is set to 10^{-7} throughout the

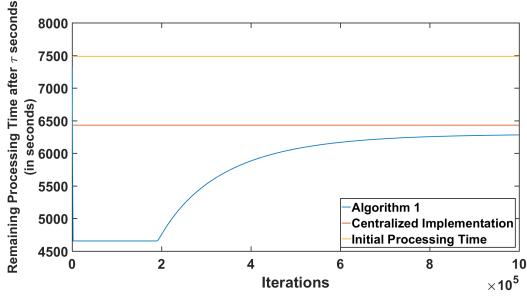
¹As stated earlier, we assume that the computational effort is proportional to the number and size of the training data samples.



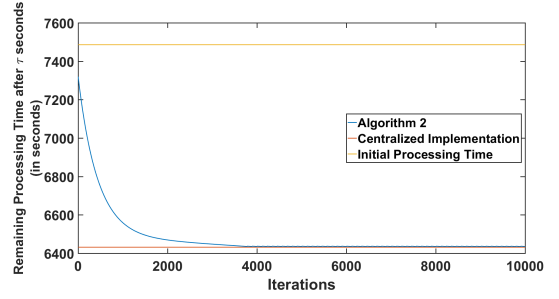
(a) Slow convergence of Algorithm 1 (Simulation 1)



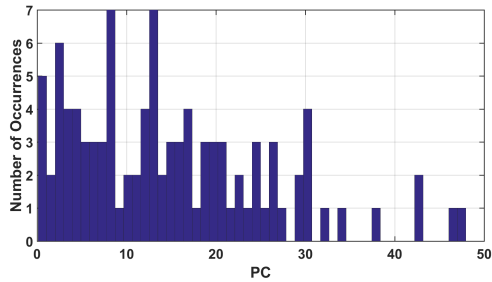
(b) Fast convergence of Algorithm 2 (Simulation 1)



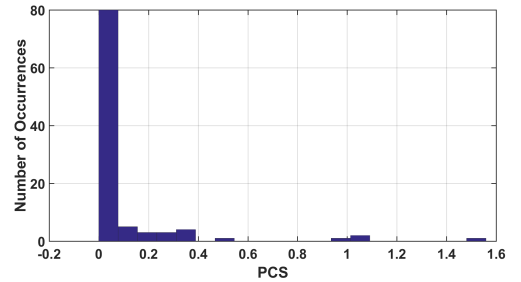
(c) Slow convergence of Algorithm 1 (Simulation 2)



(d) Fast convergence of Algorithm 2 (Simulation 2)



(e) Histogram of PC (from 100 simulations using Algorithm 2)



(f) Histogram of PCS (from 100 simulations using Algorithm 2)

Fig. 4. Convergence and accuracy of Algorithm 1 and Algorithm 2.

simulations. The initial values of decision variables r_{ij} is set to be between 0 and 0.01 and all the Lagrangian multipliers are initially set to 0.

First, through simulations, we show the fast convergence of Algorithm 2 compared to Algorithm 1. The previous paragraph described the choice of simulation parameters. Due to space limitations, we show convergence plots from only two simulation settings, referred to as Simulation 1 and Simulation 2, which differ in their D_i , d_i and P_i settings, as well as having different positions of nodes resulting in different network topologies and contention sets. Similar results were observed for other settings as well. Figures 4a to 4d show the convergence results. In all the plots, the horizontal axis represents the number of iterations and vertical axis shows the remaining processing time after τ seconds (i.e., equation (2)). Two lines and one curve are present in each plot. The yellow line is the remaining processing time if load balancing is not done. More specifically, it is the value of $\max_i T_i$, with T_i given by equation (2), with the values of link rate parameters r_{ij} set

to zero. We call this the Initial Processing Time. The red line shows remaining processing time after the first τ seconds when load balancing is done using the centralized implementation. More specifically, it is value of $\max_i T_i$ with the link rates r_{ij} set to the solution of optimization formulation $P2$ after its first execution. The yellow and the blue lines are plotted for reference and are not versus iteration. The blue curves show the convergence of either Algorithm 1 or Algorithm 2 (as mentioned in the respective legends) versus the number of iterations. More specifically, it plots the value of $\max_i T_i$ after each iteration when algorithms are being executed.

Note that the red line, or the solution of the centralized implementation, is like the baseline scheme against which the distributed iterative solutions are being compared. It is the lowest possible remaining processing time after τ seconds, satisfying all the channel capacity and contention constraints. As we can see in the plots of Figures 4a to 4d, the iterative algorithms converge close to the solution from the centralized scheme. However, they may not satisfy the constraints

exactly. We will discuss this later. Here, we can see the fast convergence of Algorithm 2 compared to Algorithm 1 for both Simulation 1 and Simulation 2. Algorithm 1 takes nearly 10^6 iterations to converge, whereas Algorithm 2 converges reasonably well within the first 2,000 to 3,000 iterations. Note also that Algorithm 2 converges differently in Figures 4b and 4d. In one case, it converges steadily and in the other it oscillates and then converges. This difference in convergence behavior was observed in other simulations, and it is caused by differences in the network topology. However, almost all the convergence plots become stable after 2,000 to 3,000 iterations.

Second, we study the convergence of Algorithm 2 by fixing the number of iterations to 2,000. The reason for reducing the number of iterations is to reduce the overhead introduced by the load balancing scheme. At the same time, we must not reduce the number of iterations too much because the solution would not have converged. To justify that 2,000 iterations is a good choice, we study how well Algorithm 2 converges and how well the constraints are satisfied after 2,000 iterations. To do this, we introduce two metrics PC and PCS . Let $RPT2000$ stand for the remaining processing time when Algorithm 2 is executed with 2,000 iterations, $RPTC$ stand for the remaining processing time when the centralized solution is used and $RPTI$ be the initial processing time. The metrics PC and PCS are defined in equations (12) and (13) below:

$$PC = \frac{|RPT2000 - RPTC|}{RPTI - RPTC} * 100 \quad (12)$$

$$PCS = \frac{\sum_{(i,j) \in E} \max(0, \sum_{(p,q) \in \Phi_{ij}} r_{pq} - C)}{C * |E|} * 100 \quad (13)$$

The metric PC is a measure of how well Algorithm 2 converges after 2,000 iterations. The lower its value the better the convergence as the remaining processing time is closer to the value of remaining processing time when the centralized implementation is used. The metric PCS is a measure of the extent to which the constraints are not satisfied. This should be as low as possible for good performance. Figures 4e and 4f show the histograms of PC and PCS for 100 simulations using Algorithm 2 in each case. We can see that most of the PC values are less than 30, with only 7 values out of 100 being greater than 30. Similarly, for PCS , all the values are below 1.6 with 80 values out of 100 being less than 0.1. Thus, Algorithm 2 converges reasonably well within 2,000 iterations, with most solutions being within a 30% band of the optimal centralized solution (in terms of PC) and with less than 1.6% constraint violation (in terms of PCS).

VI. CONCLUSION AND FUTURE WORK

There is a growing interest in utilizing the processing power of edge devices for training models for data analytics. In this paper, we proposed a distributed and fast converging load balancing scheme, as shown in Algorithm 2, to minimize the processing time of learning algorithms being executed on

edge networks. Our scheme is applicable to a large number of learning algorithms which can be written in the “summation form”. Algorithm 2 is iterative and simulation results show that it gives near optimal performance within 2,000 iterations most of the time. Due to the quick convergence and reliable performance, Algorithm 2 can handle the transient nature of the edge network topology due to node mobility by recomputing the link data rates at regular intervals, and hence, can harness the processing power of edge networks appropriately.

In the future, we plan to look into the energy consumption of distributed processing in edge networks. As most edge devices are energy-constrained, energy consumption is crucial and joint optimization schemes considering both energy and processing time must be looked into. Furthermore, appropriate incentive schemes will increase edge device participation.

VII. ACKNOWLEDGMENTS

This research is supported by a Ministry of Education (MOE) Singapore AcRF Tier 1 FRC grant.

REFERENCES

- [1] J. Chen, K. H. Low, C. K.-Y. Tan, A. Oran, P. Jaillet, J. M. Dolan, and G. S. Sukhatme, “Decentralized data fusion and active sensing with mobile sensors for modeling and predicting spatiotemporal traffic phenomena,” *arXiv preprint arXiv:1206.6230*, 2012.
- [2] Cisco, “Fog computing and the Internet of Things: Extend the cloud to where the things are,” 2015.
- [3] K. Yuen, B. Liang, and B. Li, “A distributed framework for correlated data gathering in sensor networks,” *IEEE Transactions on Vehicular Technology*, vol. 57, no. 1, pp. 578–593, 2008.
- [4] S. He, J. Chen, D. K. Yau, and Y. Sun, “Cross-layer optimization of correlated data gathering in wireless sensor networks,” *IEEE Transactions on Mobile Computing*, vol. 11, no. 11, pp. 1678–1691, 2012.
- [5] M. Holenderski, J. Lukkien, and C. K. Tham, “Trade-offs in the distribution of neural networks in a wireless sensor network,” in *International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP 2005)*, 2005.
- [6] S. Yang, Y. Tahir, P.-y. Chen, M. Alan, and J. McCann, “Distributed optimization in energy harvesting sensor networks with dynamic in-network data processing.”
- [7] J. Liu, H. Nishiyama, N. Kato, T. Kumagai, and A. Takahara, “Toward modeling ad hoc networks: current situation and future direction,” *IEEE Wireless Communications*, vol. 20, no. 6, pp. 51–58, 2013.
- [8] B. Hamdaoui and P. Ramanathan, “Rate feasibility under medium access contention constraints,” in *IEEE Global Telecommunications Conference (GLOBECOM’03)*, vol. 6. IEEE, 2003, pp. 3020–3024.
- [9] L. Chen, S. H. Low, M. Chiang, and J. C. Doyle, “Cross-layer congestion control, routing and scheduling design in ad hoc wireless networks,” 2006.
- [10] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The case for VM-based cloudlets in mobile computing,” *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.
- [11] C. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bratski, A. Y. Ng, and K. Olukotun, “Map-Reduce for machine learning on multicore,” *Advances in Neural Information Processing Systems*, vol. 19, p. 281, 2007.
- [12] Y. Xue, Y. Cui, and K. Nahrstedt, “A utility-based distributed maximum lifetime routing algorithm for wireless networks,” in *Second International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks (QSHINE’05)*. IEEE, 2005.
- [13] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- [14] A.-C. Anadiotis, G. Morabito, and S. Palazzo, “An SDN-assisted framework for optimal deployment of MapReduce functions in WSNs,” *IEEE Transactions on Mobile Computing*, vol. 15, no. 9, pp. 2165–2178, 2016.
- [15] J. Trdlicka and Z. Hanzalek, “Distributed multi-commodity network flow algorithm for energy optimal routing in wireless sensor networks,” *RadioEngineering*, 2010.