Staleness Control for Edge Data Analytics

ATAKAN ARAL, Vienna University of Technology, Austria
MELIKE EROL-KANTARCI, University of Ottawa, Canada
IVONA BRANDIĆ, Vienna University of Technology, Austria

A new generation of cyber-physical systems has emerged with a large number of devices that continuously generate and consume massive amounts of data in a distributed and mobile manner. Accurate and near real-time decisions based on such streaming data are in high demand in many areas of optimization for such systems. Edge data analytics bring processing power in the proximity of data sources, reduce the network delay for data transmission, allow large-scale distributed training, and consequently help meeting real-time requirements. Nevertheless, the multiplicity of data sources leads to multiple distributed machine learning models that may suffer from sub-optimal performance due to the inconsistency in their states. In this work, we tackle the insularity, concept drift, and connectivity issues in edge data analytics to minimize its accuracy handicap without losing its timeliness benefits. To this end, we propose an efficient model synchronization mechanism for distributed and stateful data analytics. Staleness Control for Edge Data Analytics (SCEDA) ensures the high adaptability of synchronization frequency in the face of an unpredictable environment by addressing the trade-off between the generality and timeliness of the model. Making use of online reinforcement learning, SCEDA has low computational overhead, automatically adapts to changes, and does not require additional data monitoring.

CCS Concepts: • **Information systems** → **Online analytical processing engines**; • **Computer systems organization** → **Distributed architectures**; • **Networks** → *Mobile ad hoc networks*; • **Computing methodologies** → *Planning and scheduling*.

Additional Key Words and Phrases: Edge computing, data stream processing, concept drift, staleness control, non-stationarity, reinforcement learning.

ACM Reference Format:
Atakan Aral, Melike Erol-Kantarci, and Ivona Brandić. 2020. Staleness Control for Edge Data Analytics. *Proc. ACM Meas. Anal. Comput. Syst.* 4, 2, Article 38 (June 2020), 24 pages. https://doi.org/10.1145/3392156

1 INTRODUCTION

The past decade has seen the rapid development of the Internet of Things (IoT) and the introduction of an entirely new generation of Internet services that radically changed many traditional industries. In the IoT paradigm, all things – regardless physical or digital – are connected, and therefore, able to interact with each other remotely. The sectors that are immediately affected and revolutionized by the IoT are healthcare (smart medical devices), manufacturing (smart factories), energy (smart power grids) as well as urban development and transportation (smart buildings, cities, and vehicles). As a repercussion of this so-called smart revolution, there exists an ongoing paradigm shift from core

Authors' addresses: Atakan Aral, Vienna University of Technology, Favoritenstrasse 9-11, 1040, Vienna, Austria, atakan.aral@tuwien.ac.at; Melike Erol-Kantarci, University of Ottawa, 800 King Edward Avenue, ON K1N 6N5, Ottawa, Ontario, Canada, melike.erolkantarci@uottawa.ca; Ivona Brandić, Vienna University of Technology, Favoritenstrasse 9-11, 1040, Vienna, Austria, ivona.brandic@tuwien.ac.at.

2476-1249/2020/6 or June-ART38 $15.00
https://doi.org/10.1145/3392156

data analytics to edge data analytics (EDA) [14, 47]. Strict latency requirements and unprecedented velocity of data are the main driving factors for the disruption of core analytics [37, 62]. Many services in the aforementioned areas depend on near real-time decisions based on big streaming data, which render data aggregation and analytics at a central data center infeasible, due to high network delay [17, 49]. Moreover, recent efforts optimize even the most intensive machine learning (ML) algorithms for resource-constrained edge devices [33].

Distributed ML models, when deployed at the edge nodes, can be independently trained and periodically synchronized through a central parameter server. Using EDA techniques, trained model parameters, instead of training data, can be transmitted and aggregated. This brings not only lower network latency and less bandwidth usage but also better privacy. Edge computing [2, 53, 54] is a natural fit as it enables processing in close proximity (e.g. at network gateways) or even right at the data source [5]. EDA also enables higher scalability due to the concurrent use of a high number of resources. However, current EDA architectures for distributed learning, such as Federated Learning [10, 39] or Large-Batch Training [21] are not intended for near real-time applications, and therefore, do not consider dynamic synchronization periods for ML models. This is especially critical when consistently accurate decisions are required despite non-stationary ML models. For instance, concept drift, which is defined as the transformation of the target system over time in unforeseen ways [61], is a great threat to online data analytics. Although there exist effective solutions in a centralized setting [20], the problem escalates in a distributed and networked system, particularly under intermittent connectivity. Existing quorum [22, 63] or bound [24, 62] based consistency management techniques leave these challenges unanswered due to their rigidness in the face of unpredictability [4]. Thus, novel EDA techniques are needed to address time-sensitivity along with the consistency challenges that intermittent connectivity brings.

In this work, we first characterize the network-aware scheduling of ML model updates as a Markov decision process and then propose an efficient reinforcement learning based algorithm called *Staleness Control for Edge Data Analytics* (SCEDA). SCEDA makes dynamic scheduling decisions by learning individual network connectivity trends of edge nodes (ENs) as well as the significance of their updates. In designing SCEDA, particular attention is paid that it satisfies the following properties to facilitate its use in practice.

- P_1: The mechanism does not require human intervention and automatically learns from experience.
- P_2: After limited initial bootstrapping, it continues improving and adapting to the changes based on new experiences.
- P_3: It does not require any monitoring at the ENs and operates with the data already available at the parameter server.
- P_4: Its computational overhead is low enough to allow real-time decision making.

In the next section, we provide background information on EDA. Then in Section 3, we define the staleness control problem. We introduce and describe the details of SCEDA in Section 4. In the following Section 5, we provide a complexity analysis and discuss the family of problems that fit the proposed framework. Then, we present the experimental setup and the discussion of the numerical results in Sections 6 and 7. Finally, we discuss the related literature in Section 8 and conclude the paper in Section 9.

2 EDGE DATA ANALYTICS

Leading architectures for distributed ML currently are Large-Batch Training [21] and Federated Learning [39]. In the former, each distributed node performs a single optimization step for the ML model based on its local data and immediately communicates the update to a so-called parameter

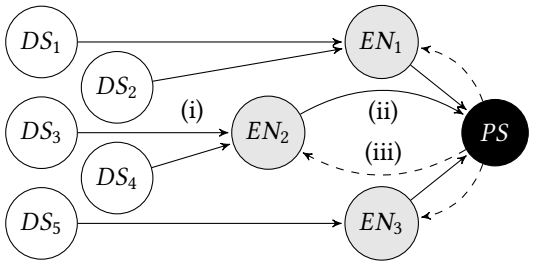

Fig. 1. Generic Edge Data Analytics Architecture.

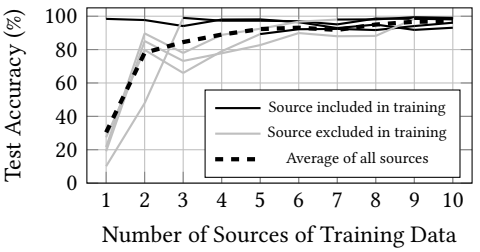

Fig. 2. Prediction Accuracy of Flight Delays.

server, where all updates are applied to the global model and broadcast to nodes. In the latter, on the other hand, nodes are able to run multiple iterations of optimization and update their local model. Local models are periodically communicated to the parameter server, aggregated, and broadcast. In this work, we consider the generic EDA architecture in Figure 1 that applies to both approaches. At step (i), data streams are sent to the ENs that are typically nearby their sources. Proximity and availability of local area bandwidth ensure that data are ready for processing without delay. Each EN maintains a local ML model and updates the model through online training in addition to inference. The parallel processing of data streams fosters the scalability of the system. Then at step (ii), the updated model is transmitted to the parameter server. Communication of the model parameters instead of training data has two-fold benefits. First, data privacy is preserved and second, the much smaller size of transmission expedites synchronization. Updates are aggregated at the parameter server and broadcast to all ENs at step (iii) for more accurate inference. We identify the following research challenges in EDA for distributed ML.

Intermittent connectivity Lack of connectivity or high packet loss affects not only the inflicted ENs, which are unable to obtain the most current global model, but also other well-connected ENs because they got deprived of the updates from the disconnected ones. Thus, intermittent network connectivity impedes timely model synchronization. This issue is not addressed by either of the aforementioned architectures.

Insularity Each EN collects data from the sources in its proximity, hence is unaware of global information [4]. This could lead to a series of critical problems including over-fitting, data sparsity, and sub-optimal accuracy of local trained models.

Non-stationarity Concept drift, also known as data set shift, is defined as the discrepancy between the training and test data of an ML model and results in non-stationarity [57]. Online learning algorithms are usually effective in coping with concept drift thorough frequent model updates.

We emulate an EDA application to better understand the impact of these three issues. To that end, we utilize the Reporting Carrier On-Time Performance data set by the U.S. Department of Transportation. We select the busiest ten airports from the data set and implement ten incremental on-line bagging classifiers (OzaBag) [44] to predict delays based on statistics such as airline, departure airport, flight duration, etc. Each classifier is assumed to be deployed in one of the airports; thus, has access to statistics of the flights destined for that airport. Additionally, we implement a synchronization mechanism between these classifiers, which we can control the airports contributing to the global model. In Figure 2, the x-axis is the gradually increasing involvement from only one airport to all ten airports, whereas the y-axis is the accuracy of each classifier. Light gray segments of the lines show the accuracy before that data source is in the global model and black segments after. Below are our findings from this proof of concept experiment.

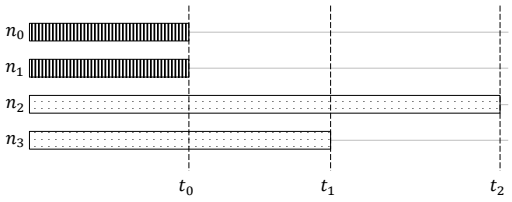

Fig. 3. Example Scenario Where Updates from Two Edge Nodes (n_2 and n_3) are Delayed.

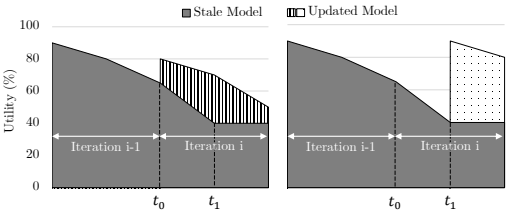

Fig. 4. An Illustration of Alternative Solutions (i) and (ii) for the Example Scenario.

- F_1: When only a few sources contribute to the global model (e.g. one in Figure 2), accuracy is only high for these sources.
- F_2: Including more sources to the global model (e.g. three in Figure 2) increases the accuracy for all classifiers.
- F_3: Not all sources are necessary to reach maximum accuracy. Partial information (e.g. five in Figure 2) is sufficient.

Further use cases that suffer from intermittent connectivity, insularity, and non-stationarity such as electric vehicle and virtual reality data analytics are discussed in Section 5.2.

3 PROBLEM FORMULATION

A trivial example is given in Figure 3 to illustrate the staleness control problem. There exist four ENs in this scenario and the chart shows the arrival times of their model updates to the parameter server. At time t_0, two ENs, n_0 and n_1, are connected to the parameter server and deliver their updates with a short network delay. The other two ENs, n_2 and n_3, on the other hand, are not accessible at t_0 and can only deliver their updates at t_1 and t_2, respectively. Building a global model and broadcasting it at each update would disseminate the timeliest information to ENs; however, this would also result in high network overhead due to the abundance and wide-area distribution of ENs [3, 35]. Therefore, we consider the case that local models are updated online, whereas, the global model is periodically synchronized (i.e. once in a predefined iteration length) [8, 37]. Ultimately, the solution options of a staleness controller in our example are (i) to build and broadcast the model immediately; (ii) to wait until one more EN responds before broadcasting; and (iii) to wait until both ENs respond before broadcasting.

Figure 4 illustrates the performance of solutions (i) and (ii) with a utility metric such as the mean accuracy of the local models. At iteration $i-1$, each EN hosts a model, however, the utility of this model (solid grey) decreases over time due to insularity and non-stationarity. Thus, ENs should be updated with a model that incorporates global information at iteration i. Solution (i) on the left has the advantage of timeliness (update at t_0) so the ENs avoid the stale model from iteration $i-1$. They would receive the updated model as soon as they are accessible by the parameter server. However, this would mean that the updates from lagging ENs (n_2 and n_3) are ignored for the iteration i. Solution (ii) on the right, on the other hand, updates the global model with additional information from n_3 at the cost of delaying the update. The updated model could eventually have higher utility, but the stale one has to be tolerated between t_0 and t_1. Omitted solution (iii) is the extreme case with even longer delay and more informed update than (ii).

Intuitively, the option with the largest area under the utility function (i.e. the union of the current and updated models) is the optimal solution that maximizes the mean utility over time. More formally, at each iteration i, we are looking for the future response j among m stragglers such that the mean utility given in objective function in Equation (1) is maximized. Here $\mathcal{U}_{i-1}(x)$ is the

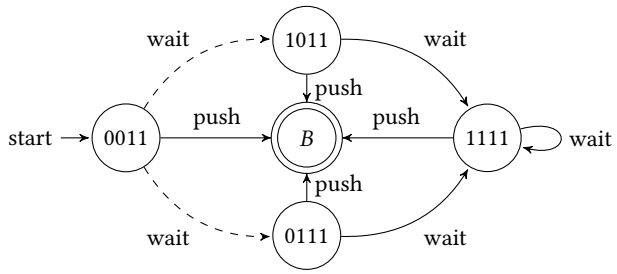

Fig. 5. Markov Decision Process for the Example Scenario.

utility function of the stale model from the previous iteration, $\mathcal{U}_i^j(x)$ is the utility function of the model that is broadcast at t_j within the current iteration, τ_i is the end of the iteration, and finally, m_i is the number of stragglers at the beginning of iteration i.

$$\begin{aligned} \underset{j}{\text{maximize}} \quad & \int_{t_0}^{t_j} \mathcal{U}_{i-1}(x)\,dx + \int_{t_j}^{\tau_i} \mathcal{U}_i^j(x)\,dx \\ \text{subject to} \quad & i, j \in \mathbb{Z},\ 0 \le i,\ 0 \le j \le m_i. \end{aligned} \tag{1}$$

A typical utility function could be the mean inference accuracy at all ENs. In an online setting, however, it is impossible to obtain $\mathcal{U}_i^j(x)$ functions a priori. The variables to be estimated also include the arrival times and order of the EN responses, significance of each response and its impact on the utility, and decaying behavior of each utility function. However, an accurate prediction is impractical due to the highly stochastic environment and many complex variables. Moreover, the number of possible scheduling decisions grows exponentially with m (see Lemma 5.3), hindering the real-world feasibility of an exact algorithm.

We model the staleness control problem as a Markov decision process (MDP) as shown in Figure 5. MDP, an extension of Markov chains, is a discrete-time stochastic control process that allows partly controlled and partly random outcomes. In staleness control, although it is possible to decide whether to wait for the next update, the arrival time and the source of the update (i.e. the next state) are indefinite. We define MDP as quadruple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_a(s,s'), \mathcal{R}_a(s,s') \rangle$. $\mathcal{S}$ is the set of states, where each state $s \in \mathcal{S}$ is characterized by a binary string given in Equation (2). Here, V_k is a binary random variable indicating whether the response of the kth EN is delivered to the parameter server up to the present time in the current iteration. Following the example scenario from Figure 3, the initial state is 0011 since only n_0 and n_1 respond immediately. We also define B as the end state, which is active between the broadcast time t_j, and the end of current iteration τ_i.

$$s = V_{N-1}V_{N-2}\ldots V_k\ldots V_0, \quad V_k = \begin{cases} 0, & n_k \text{ is delayed} \\ 1, & \text{otherwise} \end{cases} \tag{2}$$

$\mathcal{A}$, on the other hand, is the set of actions. Exactly two actions, wait and push, are defined for each state, i.e. $\mathcal{A} = \{wait, push\}$. Furthermore, $\mathcal{P}_a(s,s')$, defined in Equation (3), is the set of probabilities that action a in state s at time t will lead to state s' at time $t+1$. Push action is deterministic and always transitions to the end state B representing the broadcast of the updated global model as in Equations (4) and (5).

$$\mathcal{P}_a(s,s') = \Pr\left(s_{t+1} = s' \mid s_t = s, a_t = a\right) \tag{3}$$

$$\forall s \in \mathcal{S}\left(\mathcal{P}_{\text{push}}(s,B) = 1\right) \tag{4}$$

$$\forall s \in \mathcal{S}\ \forall s' \in \mathcal{S} \setminus B\left(\mathcal{P}_{\text{push}}(s,s') = 0\right) \tag{5}$$

Wait action delays the broadcast so as to gather more information. Since the arrival order of the responses is unknown, it transitions to a nondeterministic state. The next state must include exactly one more response upon the current state. Formally, the Hamming distance between s_t and s_{t+1} must be one and the Hamming weight (i.e. the number of set bits) of s_{t+1} must be greater than that of s_t since the responses cannot be withdrawn. Hence, the number of probable next states from s_t is equal to the number of its unset bits. Equation (6) defines the set of probable next states from state s_t after a wait action, where $d()$ is the Hamming distance metric. Here, we do not rule out the case that no further updates arrive after a wait decision due to poor network connectivity and the state remains the same ($s_{t+1} = s_t$) despite the wait action. We discuss such cases as well as the reward function $\mathcal{R}$ in the next section.

$$\mathcal{S}^+(s_t) = \{s_{t+1} : (d(s_t, s_{t+1}) = 1 \wedge s_{t+1} > s_t) \vee (s_{t+1} = s_t)\} \tag{6}$$

We define the below hypotheses for the staleness control problem.

- H_1: As the variety of the data involved in the training of the global model increases, its generality and accuracy improve.
- H_2: As the delay for broadcasting the global model increases, its staleness and accuracy worsen.
- H_3: There exists an inconstant point to broadcast during each iteration, where the trade-off between generality and staleness yields the optimum accuracy.

Initial findings F_1 and F_2 already support H_1, whereas H_2 is experimentally demonstrated in previous work [4, 58]. H_3, however, is the focus of this work and the main objective of SCEDA.

4 SCEDA

In this section, we introduce SCEDA based on the properties P_1–P_4 identified in Section 1, the findings F_1–F_4 made in Section 2, and the hypotheses H_1–H_3 constructed in Section 3. Our main design goal for the staleness control mechanism in this work is the practical applicability without overlooking the particular behavior of each EN, including connectivity, non-stationarity, and staleness factors. Traditional control theoretical or fuzzy logic based approaches would require a specific design for each additional factor (i.e. transfer functions or inference rules). Moreover, the entire control mechanism would need to be reconfigured, when applied to a new set of ENs. Instead, we resort to machine learning in order to avoid manual configuration for each application and to automatically take account of a large number of factors (P_1). More specifically, we train a decision making model via Q-learning [60], which is a widely used reinforcement learning algorithm. It allows learning a policy, that is a sequence of actions, rather than independent actions at each state. This is critical for the staleness control problem because each decision depends also on previous states and decisions.

4.1 Q-learning Preliminaries

Q-learning involves an agent, performing an action at each time step t on the MDP, which provides the agent with a reward $r_t \in \mathbb{R}$. The agent aims to maximize the expected value of the total reward over the successive steps. To this end, the function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ computes the quality of each combination of a state s_t and an action a_t that is possible at that state. At each step, the algorithm updates the corresponding quality value as given in Equation (7). Here, α is the learning rate and γ is the discount factor for future rewards.

$$Q(s_t, a_t) \leftarrow (1 - \alpha)\, Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q(s_{t+1}, a) \right) \tag{7}$$

4.2 Reward Function

The reward at each time step, r_t, is calculated by a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ defined in Equation (8). When the global model is built and broadcast with the push action, the reward is chosen as the utility value of the model, $\mathcal{U}(s_t)$, which is characterized by the responses received by the time t. No reward is given if the wait action is chosen so that double rewarding is avoided. The only exception to this is the case that no future response is received after a wait action until the end of the iteration. This case is undesired and is penalized with $p \in \mathbb{R}_{<0}$ because waiting until the end brings no additional value to the update but only delays it. An example of this is the self-loop to the state 1111 in Figure 5. A penalized wait may also occur at any other state because some ENs may be inaccessible throughout the iteration.

$$r_t = \mathcal{R}_a(s_t, s_{t+1}) = \begin{cases} \mathcal{U}(s_t), & a = \text{push} \\ 0, & a = \text{wait} \wedge s_{t+1} \neq s_t \\ p, & a = \text{wait} \wedge s_{t+1} = s_t \end{cases} \tag{8}$$

Any utility function, $\mathcal{U}(s_t)$, that is specific to the EDA use case can be used in Equation (8). Here, we propose a general approach as a concrete example. A typical utility function should be proportional to the generality of the model (H_1) and inversely proportional to the elapsed time since the beginning of the iteration, i.e. staleness (H_2), as given in Equation (9).

$$\mathcal{U}(s_t) = \frac{\mathcal{I}(s_t)}{t - t_0 + 1} \tag{9}$$

Generality can be measured via various metrics depending on the availability of relevant data at the parameter server, where SCEDA runs. We assume that the parameter server is unaware of the mean accuracy of the local data analytics due to the high overhead that its collection would bring (P_3). Alternatively and as supported by F_1 and F_2, we utilize training data variety ($\mathcal{I}$) as an indicator of model generality. In our proof of concept experiment (Figure 2), the number of data sources contributing to the training set is chosen to measure variety because the updates from the sources are of approximately equal significance. In real-world scenarios, as shown in Section 5.2, updates could be highly heterogeneous in terms of local training duration they correspond to. An update from a node that has been unavailable for several iterations should possess more variety and hence significance than one from a node that has contributed to the global model recently. Therefore, we sum up the duration of time that each update covers to estimate generality.

$$\mathcal{I}(s_t) = \sum_{k=0}^{N} \left(\left\lfloor \frac{s_t}{2^k} \right\rfloor \bmod 2 \right) (t - \phi_k) \tag{10}$$

The modulo operation in Equation (10) returns the kth bit of s_t, namely one if the response from n_k is received and zero otherwise. ϕ_k is the last time that n_k contributed to the global model. Hence $t - \phi_k$ is the time elapsed since the last broadcast that included an update from n_k. This metric of generality considers the variety both across and within data sources. It is also highly practical in the sense that it does not rely on the arrival of data streams to the parameter server, which would consume substantial bandwidth not to mention causing a delay in decisions. Local only consumption of the data also fosters privacy preservation [43].

4.3 Training

4.3.1 Offline. We propose an initial offline training based on recent traces in order to avoid the cold start problem. Each training instance in the traces is a set of EN–response time pairs that

belong to a single iteration of synchronization. For example, the instance for our example scenario would be as shown in Equation (11).

$$\mathcal{T}_i = \left\{\langle n_0, t_0\rangle, \langle n_1, t_0\rangle, \langle n_2, t_2\rangle, \langle n_3, t_1\rangle\right\} \tag{11}$$

At each training step, we let the agent choose a random instance i, take a random sequence of wait and push actions and update the Q values based on the reward outcomes. This allows us to evaluate the hypothetical actions that are not taken by the staleness controller in the runtime and significantly reduces the number of instances that need to be collected for meaningful training. Training ends after a predefined number of steps. Since state transitions in training are non-deterministic, we use a decreasing α value in Equation (12) to guarantee convergence [60]. Here, visits (s_t, a_t) is the total number of times the state–action pair has been visited.

$$\alpha = \frac{1}{1 + \text{visits}\,(s_t, a_t)} \tag{12}$$

4.3.2 Online. After the initial training, SCEDA continues to improve and adapt to the runtime changes through online learning (P_2). The quality of each decided policy is measured via the same reward function in order to improve subsequent decisions. Contrary to batch learning, future states are not known in the online case. Therefore, the maximum future quality (i.e. $\max_a Q\,(s_{t+1}, a)$) in Equation (7) must be calculated probabilistically. We use its expected value as in Equation (13).

$$Q\,(s_t, a_t) \leftarrow (1-\alpha)\,Q\,(s_t, a_t) + \alpha\left(r_t + \gamma \sum_s \Pr\,(s \mid s_t, a_t) \max_a Q\,(s, a)\right) \tag{13}$$

In online training, we opt for a constant but small α value (e.g. 0.1) so that the Q values evolve continuously and slowly.

5 DISCUSSION

5.1 Complexity Analysis

Here, we analyze the space, computation, and sample complexity of SCEDA. Computation complexity, in our context, refers to the computation effort required for a single training step to process a single instance, whereas sample complexity refers to the number of training steps required to approximate an optimum policy with a high probability. For quick reference, space, computation, and sample complexity are given in Equations (14), (15), and (16).

Lemma 5.1. *The MDP has $O(2^m)$ states.*

Proof. MDP has one state for every combination of m binary values (2^m states), in addition to the state B. Then, the total number of states, $|\mathcal{S}|$, is $2^m + 1$ and hence $O(2^m)$. □

Lemma 5.2. *The MDP has $O(2^m m)$ transitions.*

Proof. Each state in MDP except for B has the same number of outgoing transitions as its zeros that are triggered by the wait action and one outgoing transition that is triggered by the push action. State B has no outgoing transitions. Therefore, the number of ongoing transitions, m_s, from a single state, s, is $O(m)$.

The total number of transitions can be calculated as $\sum_{s\in\mathcal{S}} m_s$. Above we showed that m_s is $O(m)$ and in Lemma 5.1, we showed that $|\mathcal{S}|$ is $O(2^m)$. Thus, there are $O(2^m m)$ transitions in total. □

From Lemmas 5.1 and 5.2, we can calculate the **space complexity** of SCEDA as shown in Equation (14). This relatively high space complexity does not hinder the practical applicability of

SCEDA because the algorithm is solely stored in the parameter server, which typically resides in a cloud data center with abundant memory.

$$O(|\mathcal{S}||\mathcal{A}|) = O(|\mathcal{S}|)O(|\mathcal{A}|) = O(2^m)O(2^m m) = O(2^{2m} m) \tag{14}$$

SCEDA has the same per step **computation complexity** as general Q-learning, which is shown to be $O(\log |\mathcal{A}|)$ in [36]. This can be stated in terms of m with $O(m)$ as in Equation (15).

$$O(\log |\mathcal{A}|) = O(\log 2^m) = O(m) \tag{15}$$

The linear computational complexity of SCEDA, allows us to implement online learning (P_4). As a reference, general model-based approaches are $\Omega(|\mathcal{S}||\mathcal{A}|)$ or $\Omega(2^{2m} m)$.

Lemma 5.3. *There exist 2^m different policies in the MDP.*

Proof. Each policy is a path in MDP from the start to the state B. State B can be accessed from each state with the action push and the MDP is acyclic. Thus, the total number of policies is equal to the number of states except for B. In Lemma 5.1, we showed that there are $2^m + 1$ states. Subtracting one for B, we obtain 2^m. □

As a consequence of Lemma 5.3, at least 2^m training instances are required to evaluate all policies. However, the reward value for each policy is stochastic in our case. Since the arrival time of the updates and the information accumulated are real numbers, there exist infinitely many possible reward values for each policy. Therefore, it is not possible to define a precise lower bound for the number of training instances required to determine the optimal policy. Accordingly, we resort to a probabilistic bound on sample complexity for reaching a near-optimal policy and utilize the Probably Approximately Correct (PAC) learning framework [59].

In the PAC framework, we analyze whether and under what conditions a learner will probably output an approximately correct model. By approximately correct, we mean a model has an error over the distribution of inputs that is bounded by some ϵ, and by probably we mean that the learner will output such a model with probability $1 - \delta$. Knowing that a target concept is PAC-learnable allows us to bound the sample size necessary to probably learn an approximately correct model. Currently, it is not known whether Q-learning is PAC-learnable or what its sample complexity is. In our implementation of SCEDA, we opt for a variant of Q-learning called Delayed Q-Learning (DQL) [55]. DQL is the first model-free algorithm proved to be PAC-MDP (Probably Approximately Correct in Markov Decision Processes). The algorithm is called delayed because it waits until a state–action has been experienced k times before updating its Q-value, where k is an input parameter. When it updates the Q-value of a state–action, the update can be viewed as an average of the target values for the k most recently missed update opportunities. DQL has the same space and computation complexity as general Q-learning but it has bounded **sample complexity**, which is shown in Equation (16) [55].

$$\begin{aligned} &O\left(\frac{|\mathcal{S}||\mathcal{A}|}{\epsilon^4(1-\gamma)^8} \ln\frac{1}{\delta} \ln\frac{1}{\epsilon(1-\gamma)} \ln\frac{|\mathcal{S}||\mathcal{A}|}{\delta\epsilon(1-\gamma)}\right) \\ &= O\left(\frac{2^{2m}}{\epsilon^4(1-\gamma)^8} \ln\frac{1}{\delta} \ln\frac{1}{\epsilon(1-\gamma)} \ln\frac{2^{2m}}{\delta\epsilon(1-\gamma)}\right) \end{aligned} \tag{16}$$

The use of a PAC-learnable algorithm in SCEDA provides sufficient confidence that the broadcast decisions approach to the optimum point as more information gradually becomes available. This also ensures that SCEDA adapts well to the changes in the connectivity and concept drift behavior of ENs within a bounded number of training iterations.

5.2 Practical Use Cases

The family of problems that fit our framework and two examples of concrete applications are discussed in this section. First of all, applications should benefit from EDA. The following four criteria of the need for edge computing are presented in [52]. We argue that these criteria also apply to EDA.

Responsiveness The application requires timely data analytics; thus, wide bandwidth and low latency are imperative.

Scalability The size of data to be transmitted from many distributed sources to the cloud data center is not scalable in terms of bandwidth use.

Privacy Training data can potentially include private information that should not be exposed or aggregated.

Fault-tolerance Training and inference process must retain even if some of the ENs are unavailable.

Additionally, all three issues that are discussed in Section 2, namely, intermittent connectivity, insularity, and non-stationarity must be present. The four criteria above call for EDA, whereas these three issues call for SCEDA.

5.2.1 Electric Vehicles (EV) and Smart Grid. In the smart grid, efficient integration of EVs to the power infrastructure calls for communication technologies that will allow EVs to exchange data with the grid while on-the-go [42]. EVs communicate their state of charge and schedules when they are charging with a plug, using power line communications or in some cases Zigbee (IEEE 802.15.4) or WiFi (IEEE 802.11) [19]. However, data collected from the road allows better prediction and planning capabilities for the charging station operators and the utilities. Such data that are potentially available while the EV is on-the-move have been under-utilized. Thus, the use of EV data is limited due to the lack of efficient data analytics techniques at the edge. Near real-time nature of many EV applications requires low-latency communications, which cannot be attained by accessing the cloud. Recently, mobile edge computing emerged to address the latency issue which also brings data analytics to the edge. However, data analytics at the edge is not straightforward since EV mobility brings in new challenges to the management of distributed data over the ENs [6].

We consider the scenario that the streaming data generated at an EV are processed directly on that vehicle in order to avoid communication delay and tolerate disconnection from the rest of the network. To that end, each EV is equipped with an in-vehicle edge computing platform as proposed in [48]. Consequently, the inference is made in near real-time based on the current ML model at the EV. The local model is updated with the same data in an online learning setting. EVs occasionally connect to road-side units (RSUs) through wireless protocols such as WAVE or 5G in order to synchronize their ML models. A centralized parameter server, typically in a cloud data center, is responsible for the aggregation of the model updates and periodically broadcasts a global model. EV is chosen as a challenging use case due to its properties of wide distribution and constant mobility, which satisfies all seven criteria.

Responsiveness EVs continuously produce a large amount of data from many sensors around the vehicle that must be processed promptly. Charge planning and range estimation for EVs, for instance, is critical for the timely capacity planning of smart power plants or distributed generators (e.g. solar panels).

Scalability The popularity of EVs and the amount of data that is collected at each EV are rising rapidly.

Privacy The data includes information about the geographical location and driving style of individuals.

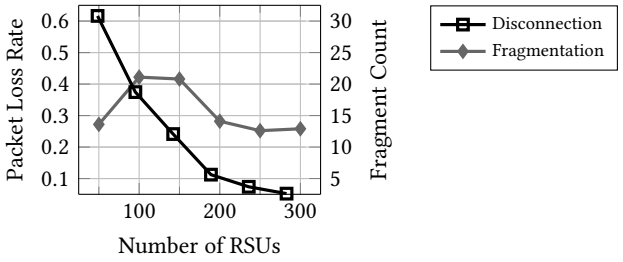

Fig. 6. EV Network Connectivity Measurements.

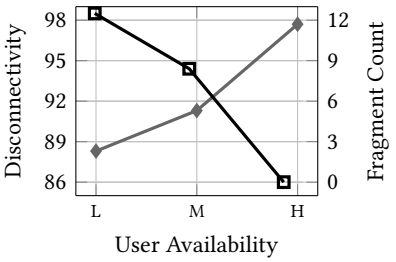

Fig. 7. VR Network Connectivity Measurements.

Fault-tolerance Training data should not be lost in case of a failure at the parameter server.

Intermittent connectivity Mobility of EVs results in loss of connectivity particularly in rural areas.

Insularity Previous training data collected by a single EV is unlikely to generalize to all future environmental conditions.

Non-stationarity Drive and environment change over time.

We conduct an analysis of EV mobility to better understand the extent of the network challenges that lie at the bottom of the above criteria. Figure 6 demonstrates two network connectivity measurements with varying number of RSUs located on a 100km highway. Evaluation of multiple RSU configurations obstructs the use of real EV connectivity data for this analysis. Therefore, we resort to simulation via the widely used and validated ns-3 environment [50] with real-world crowd-sourced EV mobility traces [56] (see Section 6.1 for details). Since mobility directly determines the connectivity patterns, we believe the following findings give a realistic picture of EV connectivity. First, the mean loss rate of the packets sent from EVs to the RSUs is intensive with a reasonable number of RSUs. Although it is possible to reach almost zero (0.05) packet loss with 300 RSUs (i.e. three RSUs in every km), this could be an unrealistic expectation. The second measurement is the mean fragment count that is the number of uninterrupted connectivity periods as EVs remain in the RSU coverage area. The count initially increases as more coverage areas become available but later decreases due to the intersection of these areas. However, the number of interruptions during the three-hour period is always over 12. The results demonstrate that intermittent connectivity and consequently insularity are prevalent in the EV use case.

5.2.2 Virtual Reality (VR) Headset Movement. Virtual and augmented reality are shown among the applications that will benefit the most from edge computing due to their real-time and data-intensive characteristics [52]. A spherical video, which is a specific type of VR experience, is recorded in every direction at the same time with an omnidirectional camera. They are usually intended to be viewed via a head-mounted display (VR headset). Gyroscope sensors on the device are used to pan the video based on the orientation of the device that is the viewing direction of the user.

Omnidirectional recording increases the bit rate significantly. As a result, not all directions can be downloaded to the user device particularly in low bandwidth conditions. When only the relevant parts of the video are downloaded, buffering is minimized. However, if the parts that the user is viewing are not downloaded, further delays may invalidate the benefits of partial streaming. Therefore, accurate and timely prediction of the user viewing direction is imperative. In this use case, we consider an ML model that explores the user behavior in spherical video streaming to predict the next viewing direction based on data from multiple headsets. This would allow pre-loading only relevant parts of the spherical videos and bring substantial bandwidth savings. Due to the

real-time use of VR headsets, the decision should be made in real-time, as well. This use case also satisfies all seven criteria as elaborated below.

Responsiveness Spherical videos are particularly large in terms of data size and it is critical to load relevant parts of them ahead of time for smooth video experience.

Scalability VR headsets are quickly getting popularity. Additionally, headset movements need to be monitored in high granularity resulting in big streaming data.

Privacy Video watching history and gaze locations of an individual are private information.

Fault-tolerance VR headset can continue streaming even if the cloud data center is inaccessible by means of caches in content delivery networks.

Intermittent connectivity VR headsets are used in arbitrary periods and go offline otherwise.

Insularity A head movement prediction model trained on a video may not apply to other videos.

Non-stationarity Behavior of VR headset users, as well as the videos streamed, change over time.

In Figure 7, we present disconnectivity rates and fragment counts for video streaming traces [45] (see Section 6.2 for details). Real wide-area connectivity data for video services exemplify the network challenges between edge nodes and parameter server (ii and iii in Figure 1) in contrast to the ones between data sources and edge nodes (i) as in the EV use case. From the perspective of SCEDA, any loss of connectivity in the continuum between the local models and the global model has the same impact, that is delayed or lost model updates. We group the users into three categories based on how long they are connected to the streaming service, namely low (L), medium (M), and high (H) availability. Results show that connectivity is sparse even with the most available users, who also exhibit high fragmentation (i.e. frequent interruptions up to 11 times in an hour on average).

6 PERFORMANCE ANALYSIS

6.1 Experimental Setup for the EV Use Case

We evaluate the runtime performance of the SCEDA algorithm through extensive network simulation with real-world EV mobility traces as well as emulation of EDA on a computer cluster with energy data streams from the same EVs. The experimental scenario consists of multiple EVs driving on a highway while running EDA (inference and training) and synchronization tasks.

An extensive corpus of real-world commute data from EVs [56] is used for realistic vehicle mobility. The data was collected from hundreds of volunteers internationally and made public in the scope of the ChargeCar project of the Carnegie Mellon University, Robotics Institute. It contains mobility information (speed, acceleration, distance, etc.) in the granularity of seconds for 423 EV trips over the course of nine years. Since the data set does not contain exact location information, we randomly generate the departure point of each trip on the highway. Similarly, RSUs are placed uniformly at random. To generalize the outcomes, experiments are repeated with several subsets of EVs that are not necessarily disjoint.

Connectivity of each EV through an RSU during its mobility is simulated via the state-of-the-art discrete-event network simulator, ns-3 [50]. The simulator is fed with the above-described mobility traces converted to the ns mobility format [27]. We implement a point-to-point star topology for the connectivity of RSUs to the parameter server, whereas EVs and RSUs communicate through the IEEE 802.11p wireless access in vehicular environments (WAVE) protocol. The output of the network simulation is the connectivity traces, which is the network latency between each EV and the parameter server (positive infinity if disconnected) at each second.

Many EDA services implement unsupervised learning due to the unavailability of labeled data instances. We implement a clustering algorithm, a typical unsupervised learning technique, to emulate EDA on a computer cluster. Clustering is critical for the online detection of driving, traffic,

Table 1. Experimental Parameters

Parameter		Description	Value
SCEDA	α	Learning rate	Eq. (12)
	γ	Discount factor	0.5
	p	Penalty for undesired wait	-10
	r	Reward for an action	Eq. (8)
Baselines		DLINE deadline	20 s / 6 s
		QRUM quorum size	6 / 17
		THOLD threshold	500 s / 400 s
		PRIOR iterations	2 / 3
EV use case	N	Number of EVs	10
		Number of RSUs	100
		Highway length	100 km
		Simulation duration	3 hrs
	τ	Iteration length	60 s
		StreamKM++ cluster count	5
VR use case	N	Number of VR Headsets	63
		Simulation duration	1 hr
	τ	Iteration length	30 s
		AMRules target count	4
		AMRules confidence	0
		AMRules tie constant	0.05

or energy consumption patterns and can be used in charge planning or range estimation. In addition to the mobility traces, the ChargeCar data set also contains energy measurements from each EV, namely power (kW), current (amp), and voltage (V) in a data stream format. In our emulation, each EV uses these data to train a `StreamKM++` clusterer, which is a modern and streaming version of the k-means algorithm and one of the most widely used techniques for stream clustering [1]. We used the `StreamKM++` implementation available in the Massive Online Analysis data stream mining framework [9].

Potentially limited processing power of each EV is represented with a virtual machine (VM) with a single-core CPU and 2 GB of memory. At each iteration, VMs are allowed to send their updates to the parameter server if and when ns-3 generated network traces indicate connectivity. Similarly, when the parameter server broadcasts a new model, it is only received by the connected VMs at that time. The parameter server is emulated with a powerful Intel Xeon E5 Cloud server. The model updates submitted by the VMs are the current coordinates of their cluster centroids and sizes, whereas the global model is built with weighted means. The weight of a centroid is the number of data points at the corresponding cluster.

6.2 Experimental Setup for the VR Use Case

The head movement data set of users that watch spherical videos using a headset is provided by IMT Atlantique [16]. The data set contains the changes in headset orientation, as the users watch spherical videos. Head movement data are collected from 63 users during five 70 seconds-long spherical videos and consists of four angles per instance. A wide range of video content is used for data collection so that various head movement patterns are captured. The granularity varies from 33.5 to 199.9 instances per second.

We utilize the four-dimensional directions in the data set to train a regressor. Since the four angles are not independent, we utilize a multi-target regression model, which predicts the future values for multiple variables simultaneously. MOA implementation of a state-of-the-art streaming rule learning algorithm, AMRules [18] is chosen due to its comparable performance to batch learners. Because of the higher number of VR headsets, we opt for a shorter iteration length and simulation duration than the EV use case.

The head movement data set has been collected in a laboratory environment, hence does not exhibit timing information such as when the user started watching the video and when they stopped. Consequently, we use another data set for the connectivity of headsets. It consists of streaming videos watched by 158 participants tracked by a browser extension that is used to examine whether and how indicators of collective preferences and reactions are associated with the viewing duration of videos [45]. This data set contains the video properties and sentiment of comments for each of the 1,125 videos collected through the YouTube API. We select the most active 63 participants and map them randomly to the users in the head movement data set. We assume that the VR headset is connected to the parameter server as long as the user is watching a video. Thus, connectivity is directly obtained from real-world data.

6.3 Baseline Algorithms

The performance of SCEDA is compared to the following baseline algorithms, parameters of which are provided in Table 1.

DLINE broadcasts the model at a constant time at each iteration.

QRUM broadcasts when a constant number of updates are received (i.e. quorum is met). This approach is proposed in [22].

THOLD broadcasts when the accumulated information at the parameter server reaches a threshold. To estimate the accumulated information, we use the function in Equation (10).

PRIOR assumes that the connectivity of the ENs remains the same and broadcasts when the updates from the ENs that were connected in the previous iterations arrive.

All baseline algorithms and SCEDA broadcast the model at the end of the iteration if the condition is not met earlier. Baseline algorithms can be configured to wait until a greater (or fewer) number of updates arrive at the expense of timeliness. For a fair comparison, we choose the parameters in Table 1 such that each algorithm, including SCEDA, receives and incorporates roughly the same number of updates during the whole simulation duration. Since the baselines are not adaptive like SCEDA, the values vary by use case separated with a slash. The values on the left correspond to the EV use case and the ones on the right to the VR use case. Additionally, we implement the bounded staleness algorithm described in [24], which always waits for an update if its source node is not included in the model in the last k iterations (i.e. staleness bound). However, this approach assumes that all nodes eventually respond in every iteration. This is not the case for either of our two use cases and ENs can stay disconnected for several iterations, therefore the algorithm waits until the end of almost all iterations. Since this is analogous to the DLINE baseline with a deadline equal to the iteration length, we exclude this algorithm from the results.

Following two baselines, on the other hand, represent the best- and worst-case connectivity in EDA without staleness control.

SYNC is the hypothetical case that the ENs are always connected with infinite bandwidth and no latency; thus, all local models are perfectly synchronized.

DISC is the case that there is no global model synchronization and each EN trains its model with only local data.

6.4 Performance Metrics

Due to the novelty of the problem, there does not exist standard metrics to compare different approaches. We overcome this issue by adopting several performance metrics from network science and artificial intelligence and devising two new area-specific ones.

6.4.1 Age of Information (AoI). AoI measures the freshness of information that is continuously updated through a network queue. It is originally proposed for vehicular networks [30] and recently drew significant attention [28, 29, 32]. AoI is defined as the difference between the time the information is observed and generated.

$$\Delta(i,j) = t_o(i,j) - t_g(i) \tag{17}$$

In Equation (17), generation time ($t_g(i)$) is when a data instance, $\mathcal{T}_i$ is processed at its source EN; whereas observation time ($t_o(i,j)$) is when a global model that incorporates this data instance is first received by another node, n_j. A global model is assumed to incorporate $\mathcal{T}_i$, if its source node sends an update after $t_g(i)$, and the update is received by the parameter server before the model is built.

6.4.2 Value of Information of Update (VoIU). Authors in [31] consider the case that the level of *dissatisfaction* for having aged status updates increases non-linearly with time. Furthermore, they introduce VoIU metric, which captures the degree of importance of the information received at the destination as given in Equation (18).

$$\Upsilon(i,j) = \frac{f(t_o(i,j) - t_g(i-1)) - f(t_o(i,j) - t_g(i))}{f(t_o(i,j) - t_g(i-1))} \tag{18}$$

We use a logarithmic function, $f(x) = log(x+1)$, to implement non-linear aging. VoIU is bounded in the real interval $[0,1]$.

6.4.3 Number of Penalized Waits. As explained in Section 4.2, waiting until the end of iteration without a push decision is undesirable since the expected update(s) does not arrive and the received updates are delayed in vain. This metric indicates the number of iterations in which an undesired wait decision occurs.

6.4.4 Silhouette Score. Although VoIU measures the significance of an update, we are also interested to what extent its significance corresponds to the performance of the data analytics. The silhouette score [51] captures the discriminative capacity of clustering by measuring how similar instances are to their assigned clusters.

$$\Sigma(i) = \frac{d_n(i) - d_c(i)}{\max(d_c(i), d_n(i))} \tag{19}$$

In Equation (19), $d_c(i)$ is the mean Euclidean distance of instance $\mathcal{T}_i$ to all other instances within the same cluster; whereas $d_n(i)$ its mean distance to all instances in the neighboring cluster (i.e. the cluster that is next best fit for $\mathcal{T}_i$). Silhouette score is bounded in the real interval $[-1,1]$ and corresponds to the utility in the EV case.

6.4.5 Root Mean Squared Error (RMSE). RMSE is a frequently used measure for evaluating the accuracy of regression models. It indicates the difference between values predicted by the model and the values observed in the VR use case. In Equation (20), $\hat{y}_t$ is the prediction, y_t is the actual value, and T is the number of predictions.

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}} \tag{20}$$

6.4.6 The Slope of Trend Line. This metric evaluates whether the model consistency deteriorates over time. First, a trend line is fit to RMSE values via least squares regression. Then, the slope of the regression line is used as a measure of deterioration.

7 NUMERICAL RESULTS

7.1 EV Use Case

Figure 8 presents the cumulative AoI values at each iteration averaged over all EVs. It is apparent from these figures that SCEDA provides the local models in a significantly timelier fashion than the baselines. PRIOR performs the tardiest broadcasts, which would jeopardize the relevance of accumulated information due to aging. This is because PRIOR is dependent on the responses from particular EVs. With DLINE, AoI is almost constant as it broadcasts always at the same time. That is substantially longer than SCEDA on average, although they take the same number of updates into account. QRUM and THOLD perform comparably well in the earlier phase but are outperformed in the long run as SCEDA adapts. Interestingly, while the baselines are quite consistent with the broadcast time, SCEDA pushes the model early in certain periods (e.g. around 100 to 150 minutes) but waits longer in others (e.g. around 80 to 100 minutes and from 160 minutes on). The main reason for this is that it is the only algorithm that learns the network connectivity and adapts itself as the connectivity degrades.

Additionally, mean AoI per EV can be seen in Figure 11. SCEDA has the earliest AoI in 8 of the 10 EVs. Remaining two EVs represent the extreme cases of connectivity; n_4 stays connected only 37% of the time (mean connectivity of all EVs is 53%), hence receives less frequent updates; whereas n_9 is connected 92% of the time. VoIU results in Figure 9 are generally consistent with Figure 8; early AoI results in higher VoIU. This is particularly apparent in the periods that SCEDA performed later and less valuable broadcasts than average. The only difference is that PRIOR achieves higher VoIU than DLINE, although its AoI is older. This is understandable because PRIOR broadcasts the model immediately after the expected response arrives, whereas DLINE waits regardless. This adds value to the model as the information from the expected response is still fresh but is not enough to outperform other baselines as they accumulate comparable information in a shorter time.

Figure 12, on the other hand, illustrates that broadcasts by SCEDA have the greatest value for 8 out of 10 EVs. For n_3, results are very similar for three algorithms in both AoI and VoIU; whereas for n_4, SCEDA manages to provide more value than DLINE, despite later broadcast. This indicates that SCEDA makes the right decision to wait and significant updates arrive in the meantime. Finally, n_9 is the only EV, where SCEDA is noticeably outperformed. Here, QRUM and DLINE broadcast very early on average, whereas SCEDA's decision to wait does not serve its purpose.

To better understand the inner workings of the algorithms, we demonstrate the number of penalized waits and the distribution of broadcast times in Figure 14. As shown on the left side, SCEDA sustains fewest undesired waits in 180 iterations, excluding DLINE, which never waits after the deadline by definition. SCEDA's performance in penalty avoidance is 37% better than the next best algorithm, THOLD. Color-coded distribution of waiting times is illustrated on the right side of the figure. Here, white indicates an immediate broadcast and black a penalized wait. It is clear that PRIOR takes completely different actions than the other algorithms. QRUM and THOLD behave quite similar over time, which indicates that the amount of accumulated information is roughly proportional to the number of received updates. SCEDA, on the other hand, performs fewer late broadcasts and more mid-range ones than the baselines until around 160 minutes. We notice that, after this point, many EVs are not in the range of an RSU and hence inaccessible. PRIOR performs earlier broadcasts during this period.

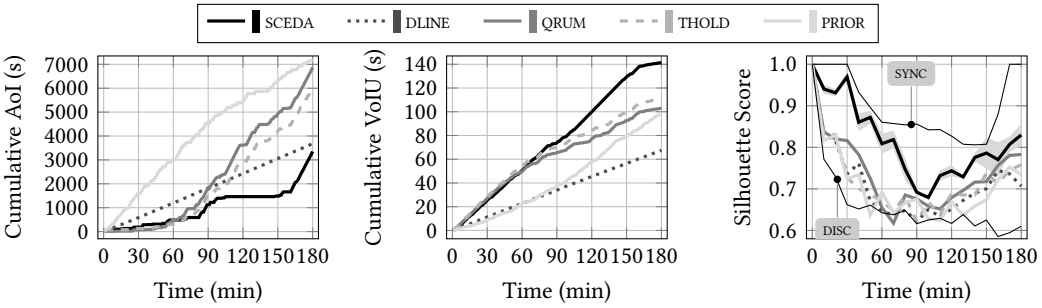

Fig. 8. AoI per Minute (EV).

Fig. 9. VoIU per Minute (EV).

Fig. 10. S. Score per Minute (EV).

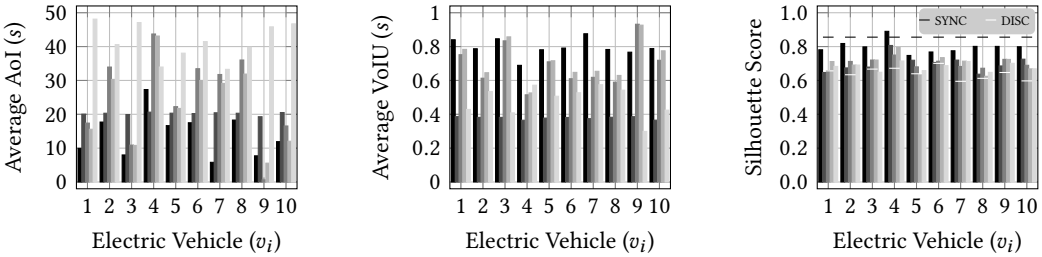

Fig. 11. AoI per EV.

Fig. 12. VoIU per EV.

Fig. 13. Silhouette Score per EV.

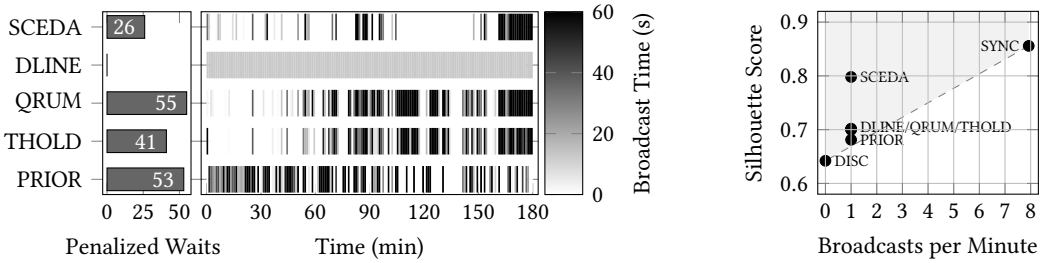

Fig. 14. Penalized Waits and Distribution of Broadcast Times (EV).

Fig. 15. Trade-off Analysis (EV).

For the second part of the evaluation, we emulate EDA on a computer cluster. We use the traces from the first part in order to simulate the network connectivity of the nodes. Figure 10 presents the mean silhouette score of all EVs in 10-minute intervals. Scores for the hypothetical scenarios of SYNC and DISC are also provided. All results start with the maximum score (i.e. 1.0) because we train an omniscient model and initialize every EV with it before the start of the emulation. The shaded area around SCEDA scores indicates the 95% confidence interval. It can be seen from the figure that SCEDA and the baselines outperform DISC significantly, which indicates that model synchronization and staleness control are imperative. Global information from other EVs informs the local model about the cases before it encounters them and consequently increases its accuracy. Moreover, SCEDA's performance is even comparable to SYNC except for the period between around 80 to 130 minutes. We notice that a concept drift begins around 40 minutes, which is weathered by SYNC with low loss of performance. SCEDA, however, takes a longer time and higher loss to

recover due to the realistic connectivity. We demonstrate the mean silhouette scores for each local model in Figure 13. Consistent with the previous results, SCEDA achieves the highest scores among the algorithms in every case and outperforms even the SYNC scenario for n_4.

Finally, we present a scatter plot in Figure 15 to visualize the trade-off between accuracy and bandwidth consumption. DISC does not consume bandwidth, whereas SYNC consumes the most by broadcasting each update immediately. All others broadcast exactly once in an iteration. Assuming a linear relationship between the parameters, shown with the dashed line, they all stand in the shaded area, which represents more efficient solutions than both extremes.

7.2 VR Use Case

We obtain similar promising results for SCEDA with our second use case, VR headset movement prediction. In this case, we provide only aggregate results due to the high number of simulated ENs. Figures 16 and 17 demonstrate that SCEDA achieves the lowest age and highest value of information outperforming the other baselines. The reasoning behind this is already discussed for the EV use case. Regarding penalized waits shown in Figure 20, SCEDA performs even better without a single penalty in 120 iterations. DLINE and SCEDA achieve the most uniformly distributed broadcast times. The ranking of other baselines is the same as the EV use case except for the tie between SCEDA and DLINE.

Figure 18 presents the deviation of the prediction from the actual values. Here, SYNC and DISC define the best and worst possible cases, respectively. SCEDA achieves nearly as small error as SYNC and outperforms other baselines in the whole duration with 90% confidence. The only period that one of the baselines, namely DLINE, is within the 95% confidence interval (shaded area) is between 50th and 60th minutes. Our investigation of the reason for such a large interval reveals

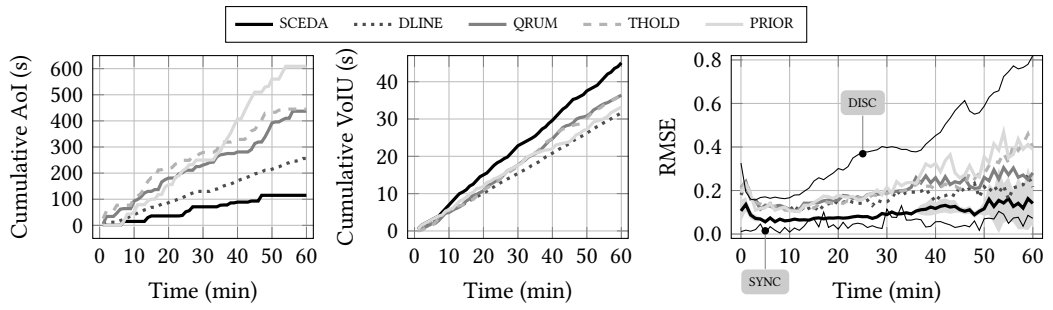

Fig. 16. AoI per Minute (VR). Fig. 17. VoIU per Minute (VR). Fig. 18. Prediction Error per Minute (VR).

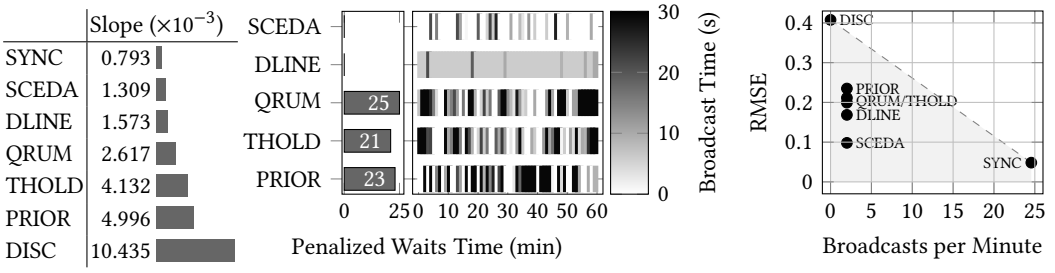

Fig. 19. RMSE Slopes. Fig. 20. Penalized Waits and Broadcast Dist. (VR). Fig. 21. Trade-off Analysis (VR).

that only a few users are online in this period. Thus, the results after the 50-minute mark are not as conclusive.

An interesting trend in Figure 18 is the rise in the RMSE value over time, which is particularly visible with DISC. This is due to the concept drift as it is hard to cope with when the synchronization does not function well, and therefore, any change in the environment is unprecedented for ENs. In this specific case, when VR users watch a new video and their prediction model is not enhanced by the experience of others who already watched it, prediction error increases. In Figure 19, this trend is further analyzed by comparing the slopes of the least square regression lines for RMSE trends in Figure 18. Higher values of slope indicate that RMSE increases faster over time. Here, SCEDA performs closest to SYNC. The bandwidth–accuracy trade-off in Figure 21 shows that all solutions again stand in the shaded area that outperforms the SYNC–DISC boundary and the baselines are dominated by SCEDA. Finally, we measure the computation latency of the baselines and SCEDA to determine whether the time it takes to make a scheduling decision is a significant factor within AoI results. We observe that all baselines reach a decision in sub-μs time, whereas SCEDA takes 2 μs on average. Thus, decision time is negligible compared to AoI, which is in the order of seconds. These results apply to both use cases.

7.3 Sensitivity Analysis

In this section, we analyze the sensitivity of the previous results to the parameters of SCEDA, namely discount factor, training steps, penalty value, and reward function. To avoid redundancy, we present only the results corresponding to the EV use case as the results with the VR data are in a similar vein. In Figures 22–25, we present the relative accuracy and VoIU performance of SCEDA normalized by the parameter value that is used in the previous experiments (indicated with a vertical line). Additionally, relative accuracy values (i.e. Silhouette score) of the two best-performing baselines (i.e. THOLD and QRUM) are shown as a reference. In almost all cases in four experiments, evaluating a wide range of parameter values, SCEDA outperforms the baselines in accuracy. This demonstrates that the results are not limited to a certain configuration and can be achieved even under suboptimal parameters.

Figure 22 shows that lower values of discount factor (< 0.2) result in a greedy behavior for immediate rewards, which translates to high VoIU but also low accuracy. On the contrary, with a high discount factor (> 0.5), the scheduler puts too much emphasis on expected future rewards, which increases the risk of penalized waits. In terms of the number of training steps used for the initial offline training (Figure 23), we observe erratic behavior due to overfitting when few steps are allowed. With only 2,500 steps, however, the results stabilize rapidly. This is in line with our

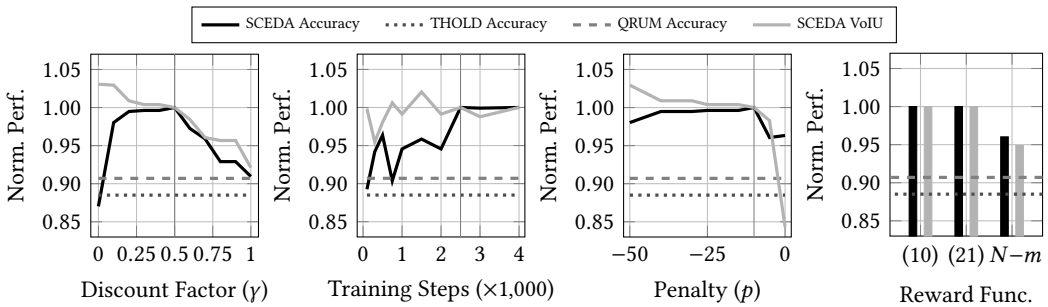

Fig. 22. Sensitivity to the Discount Factor.

Fig. 23. Sensitivity to the Offline Training Length.

Fig. 24. Sensitivity to the Undesired Wait Penalty.

Fig. 25. Sensitivity to the Reward Function.

theoretical analysis of sample complexity in Section 5.1. As shown in Figure 24, when the penalty for an undesired wait is weak (e.g. −5 or 0), SCEDA tends to delay the broadcasts inducing slightly lower accuracy. Penalties stronger than −10 do not seem to be more effective. Finally in Figure 25, we analyze the sensitivity to the reward function by comparing the proposed reward function in Equation (10) to two others. Equation (21) is the information entropy of the data set X, which can be used to measure the usefulness of updates, whereas the third one is a simple reward function that counts the number of received updates ($N - m$) as used for the analysis in Figure 2.

$$\mathcal{I}(s_t) = H(X) = -\sum_{x \in X} \Pr(x) \log \Pr(x) \tag{21}$$

The results do not show high variance with the choice of the reward function. We conclude that learning the connectivity behavior of ENs is the main reason why SCEDA outperforms the baselines.

8 RELATED WORK

There exist two different approaches to the synchronization of an ML model in the distributed setting. The quorum-based approach [22, 63] allows updates as long as a certain number of responses are received, which lacks the flexibility to adapt to the unpredictable evolution of local models and connectivity of ENs. Bounded staleness [15, 24, 34, 62] instead, allows asynchronous execution unless staleness is over a predefined bound and suffers from the same rigidness problem. Moreover, both these approaches rule out the network aspect of the problem. One novelty of our work lies in its exemption from static thresholds or bounds, and its capability to adapt to changes in communication and learning patterns.

Previous work [7] has experimentally demonstrated that a prospective online scheduler (called the potential-heuristic) would be more efficient than existing algorithms in limiting the updates from evolving ML models pushed to edge devices. SCEDA not only realizes this potential as a concrete algorithm but also extends the scope of the problem with online training at the edge (instead of only inference). Gaia [25] is a geographically distributed ML system that maintains multiple parameter servers at cloud data centers and only synchronizes major updates across them through a significance filter. ML applications can, however, update the parameter server at the local data center freely. Since Gaia is intended for cloud architecture with reliable network connectivity, it does not deal with undelivered updates between distributed ML models. A related problem in EDA is the trade-off between the timeliness and accuracy [23], where local models deliberately reduce accuracy (allow errors) to meet real-time deadlines or vice versa. Similarly, VideoEdge [26] addresses the trade-off between multiple hierarchical resources and the accuracy of video analytics by determining query plans and task placements that maximize accuracy. Further objectives in geographically distributed ML such as regulatory compliance, fault tolerance, and privacy preservation are identified in the literature [12], which also proposes query-based communication between models instead of a parameter server.

Many distributed services already employ data analytics to some extent, yet none of the existing data stream processing (DSP) engines (e.g. Apache Storm, Flink, Samza, and Spark Streaming) explicitly deal with insularity, non-stationarity, and intermittent connectivity. Apache SAMOA framework is proposed [41] to act as an abstraction for the aforementioned DSP engines and it provides rudimentary snapshot-based model consistency. However, this is not capable of providing the dynamicity in model synchronization and adaptability to concept drift, required by EDA. Continuum [58] is another platform to provide model consistency on top of the existing ML frameworks. It determines when to update the model and avoids frequent retraining due to its high computational cost. However, Continuum maintains a single ML model and does not support multiple geographically distributed ones. Thus, the interplay between local and global models

Table 2. Qualitative Assessment of the Relevant Literature

Publication	IC	IN	NS	P_1	P_2	P_3	P_4
Babu and Stewart [7]		●	●	N/A	●		N/A
Cano et al. [12]		●	●	●	●		●
Cipar et al. [15]		●	●			●	●
Hara and Madria [22]	●	N/A	N/A			●	●
Heintz et al. [23]	●		●		●	N/A	●
Ho et al. [24]		●	●			●	●
Hsieh et al. [25]		●	●			●	●
Hung et al. [26]	●	N/A	N/A	●		●	●
Lee et al. [34]		●	●			●	●
Morales and Bifet [41]			●				
Tian et al. [58]			●	●	●	N/A	●
Xing et al. [62]		●	●			●	●
Yu and Vahdat [63]	●	N/A	N/A	●	●	N/A	●
SCEDA	●	●	●	●	●	●	●

is not addressed. Although several DSP architectures exist for edge computing [11, 13, 46], the management of model consistency across multiple nodes is not yet studied to the best of our knowledge. Other works on DSP within the edge computing context can be found in recent surveys [17, 64].

In Table 2, a selection of studies from the literature is assessed qualitatively based on the addressed challenges and design constraints. The former include the challenges addressed by SCEDA, namely intermittent connectivity (IC), insularity (IN), and non-stationarity (NS), whereas, the latter are defined in Section 1 (P_1 to P_4). Here, (●) indicates the work addresses the criterion, a gap indicates otherwise, and (N/A) indicates the criterion does not apply to the use case. We conclude from the literature review that there does not exist a work prior to this paper, to the best of our knowledge, that addresses all challenges subject to the constraints.

9 CONCLUSION

In this work, we propose a dynamic staleness control algorithm, SCEDA, for edge data analytics. SCEDA addresses the insularity challenge, which is prevalent in the scenarios with non-stationary data streams and intermittent network connectivity, through effective dissemination of new information throughout the edge network. It employs reinforcement learning to jointly take into account individual behavior of each edge model in terms of its connectivity and stationarity. Experimental results show that SCEDA minimizes the age of information while maximizing its value. Moreover, edge data analytics with the proposed dynamic model synchronization mechanism can achieve a comparable level of accuracy as core data analytics, yet with near real-time decisions. It also outperforms the baseline algorithms, including the state-of-the-art quorum solution.

The impact of this work goes far beyond our initial use case scenarios of electric vehicles or virtual reality and it is possibly applicable to many stateful analytics tasks on distributed and streaming big data, in general. As future work, we plan to evaluate SCEDA with further machine learning tasks and use cases. Additionally, we will investigate the possible performance impacts of the use of more comprehensive deep reinforcement learning algorithms (e.g. [38, 40]) in scheduling model updates and look into approximation mechanisms to reduce the state-space for the initial training of the Q-learning model.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers and our shepherd Ganesh Ananthanarayanan, for many insightful comments and constructive suggestions that helped us improve the quality of the paper greatly. We also thank Dogan Altan for the valuable discussions regarding reinforcement learning techniques. This work has been partially funded through the Rucon project (Runtime Control in Multi Clouds), Austrian Science Fund (FWF): Y904-N31.

REFERENCES

[1] Marcel R Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. 2012. StreamKM++: A clustering algorithm for data streams. *Journal of Experimental Algorithmics* 17 (2012), 2–4.

[2] Sharad Agarwal, Matthai Philipose, and Paramvir Bahl. 2014. Vision: the case for cellular small cells for cloudlets. In *International Workshop on Mobile Cloud Computing & Services*. ACM, Bretton Woods, NH, USA, 1–5.

[3] Joon Ahn, Maheswaran Sathiamoorthy, Bhaskar Krishnamachari, Fan Bai, and Lin Zhang. 2014. Optimizing content dissemination in vehicular networks with radio heterogeneity. *IEEE Transactions on Mobile Computing* 13, 6 (2014), 1312–1325.

[4] Atakan Aral and Ivona Brandic. 2018. Consistency of the Fittest: Towards Dynamic Staleness Control for Edge Data Analytics. In *International European Conference on Parallel and Distributed Computing Workshops*. Springer, Turin, Italy, 40–52.

[5] Atakan Aral and Tolga Ovatman. 2018. A Decentralized Replica Placement Algorithm for Edge Computing. *IEEE Transactions on Network and Service Management* 15, 2 (2018), 516–529.

[6] Ashwin Ashok, Peter Steenkiste, and Fan Bai. 2018. Vehicular cloud computing through dynamic computation offloading. *Computer Communications* 120 (2018), 125–137.

[7] Naveen TR Babu and Christopher Stewart. 2019. Energy, latency and staleness tradeoffs in ai-driven iot. In *ACM/IEEE Symposium on Edge Computing*. ACM, Washington D.C., USA, 425–430.

[8] Yael Ben-Haim and Elad Tom-Tov. 2010. A Streaming Parallel Decision Tree Algorithm. *Journal of Machine Learning Research* 11, Feb (2010), 849–872.

[9] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. 2010. MOA: Massive Online Analysis. *Journal of Machine Learning Research* 11, May (2010), 1601–1604.

[10] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konecný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. *CoRR* abs/1902.01046 (2019), 15.

[11] Antonio Brogi, Gabriele Mencagli, Davide Neri, Jacopo Soldani, and Massimo Torquati. 2017. Container-based Support for Autonomic DSP through the Fog. In *International Workshop on Autonomic Solutions for Parallel and Distributed Data Stream Processing*. Springer, Santiago de Compostela, Spain, 17–28.

[12] Ignacio Cano, Markus Weimer, Dhruv Mahajan, Carlo Curino, and Giovanni Matteo Fumarola. 2016. Towards Geo-Distributed Machine Learning. *CoRR* abs/1603.09035 (2016), 10.

[13] Valeria Cardellini, Francesco Lo Presti, Matteo Nardelli, and Gabriele Russo Russo. 2018. Decentralized self-adaptation for elastic Data Stream Processing. *Future Generation Computer Systems* 87 (2018), 171 – 185.

[14] Aakanksha Chowdhery, Marco Levorato, Igor Burago, and Sabur Baidya. 2018. Urban IoT Edge Analytics. In *Fog computing in the internet of things*. Springer, Cham, Switzerland, 101–120.

[15] James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Gregory R Ganger, Garth Gibson, et al. 2013. Solving the Straggler Problem with Bounded Staleness.. In *Workshop on Hot Topics in Operating Systems*, Vol. 13. ACM, Santa Ana Pueblo, NM, USA, 22–22.

[16] Xavier Corbillon, Francesca De Simone, and Gwendal Simon. 2017. 360-degree video head movement dataset. In *ACM Conference on Multimedia Systems*. ACM, Taipei, Taiwan, 199–204.

[17] Marcos Dias de Assuncao, Alexandre da Silva Veith, and Rajkumar Buyya. 2018. Distributed Data Stream Processing and Edge Computing: A Survey on Resource Elasticity and Future Directions. *Journal of Network and Computer Applications* 103 (2018), 1–17.

[18] João Duarte, João Gama, and Albert Bifet. 2016. Adaptive model rules from high-speed data streams. *ACM Transactions on Knowledge Discovery from Data* 10, 3 (2016), 30.

[19] Melike Erol-Kantarci, Jahangir H Sarker, and Hussein T Mouftah. 2011. Communication-based plug-in hybrid electrical vehicle load management in the smart grid. In *IEEE Symposium on Computers and Communications*. IEEE, Corfu, Greece, 404–409.

[20] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys* 46, 4 (2014), 44.
[21] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *CoRR* abs/1706.02677 (2017), 12.
[22] Takahiro Hara and Sanjay Kumar Madria. 2005. Consistency Management Among Replicas in Peer-To-Peer Mobile Ad Hoc Networks. In *IEEE Symposium on Reliable Distributed Systems*. IEEE, Orlando, FL, USA, 3–12.
[23] Benjamin Heintz, Abhishek Chandra, and Ramesh K Sitaraman. 2016. Trading timeliness and accuracy in geo-distributed streaming analytics. In *ACM Symposium on Cloud Computing*. ACM, Santa Clara, CA, USA, 361–373.
[24] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. 2013. More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. In *Conference on Neural Information Processing Systems*. Curran Associates, Lake Tahoe, NV, USA, 1223–1231.
[25] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R Ganger, Phillip B Gibbons, and Onur Mutlu. 2017. Gaia: Geo-Distributed Machine Learning Approaching {LAN} Speeds. In *USENIX Symposium on Networked Systems Design and Implementation*. USENIX, Boston, MA, USA, 629–647.
[26] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. 2018. Videoedge: Processing camera streams using hierarchical clusters. In *ACM/IEEE Symposium on Edge Computing*. IEEE, Seattle, WA, USA, 115–131.
[27] Teerawat Issariyakul and Ekram Hossain. 2012. *Introduction to Network Simulator NS2*. Springer, New York, NY, USA.
[28] Zhiyuan Jiang, Bhaskar Krishnamachari, Xi Zheng, Sheng Zhou, and Zhisheng Niu. 2018. Decentralized Status Update for Age-of-Information Optimization in Wireless Multiaccess Channels. In *IEEE International Symposium on Information Theory*. IEEE, Vail, CO, USA, 2276–2280.
[29] Clement Kam, Sastry Kompella, Gam D Nguyen, Jeffrey E Wieselthier, and Anthony Ephremides. 2018. On the age of information with packet deadlines. *IEEE Transactions on Information Theory* 64, 9 (2018), 6419–6428.
[30] Sanjit Kaul, Marco Gruteser, Vinuth Rai, and John Kenney. 2011. Minimizing age of information in vehicular networks. In *IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*. IEEE, Salt Lake City, UT, USA, 350–358.
[31] Antzela Kosta, Nikolaos Pappas, Anthony Ephremides, and Vangelis Angelakis. 2018. The Cost of Delay in Status Updates and their Value: Non-linear Ageing. *CoRR* abs/1812.09320 (2018), 32.
[32] Qiaobin Kuang, Jie Gong, Xiang Chen, and Xiao Ma. 2019. Age-of-Information for Computation-Intensive Messages in Mobile Edge Computing. *CoRR* abs/1901.01854 (2019), 6.
[33] Nicholas D Lane, Sourav Bhattacharya, Akhil Mathur, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. 2017. Squeezing deep learning into mobile and embedded devices. *IEEE Pervasive Computing* 16, 3 (2017), 82–88.
[34] Joo Hwan Lee, Jaewoong Sim, and Hyesoon Kim. 2015. BSSync: Processing near memory for machine learning workloads with bounded staleness consistency models. In *International Conference on Parallel Architecture and Compilation*. IEEE, San Francisco, CA, USA, 241–252.
[35] Ilias Leontiadis, Paolo Costa, and Cecilia Mascolo. 2009. A hybrid approach for content-based publish/subscribe in vehicular networks. *Pervasive and Mobile Computing* 5, 6 (2009), 697–713.
[36] Lihong Li. 2012. Sample complexity bounds of exploration. In *Reinforcement Learning*. Springer, Berlin, Germany, 175–204.
[37] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, et al. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *USENIX Conference on Operating Systems Design and Implementation*. USENIX, Broomfield, CO, USA, 583–598.
[38] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971 (2015), 14.
[39] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *International Conference on Artificial Intelligence and Statistics*. PMLR, Lauderdale, FL, USA, 1273–1282.
[40] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.
[41] Gianmarco De Francisci Morales and Albert Bifet. 2015. SAMOA: Scalable Advanced Massive Online Analysis. *Journal of Machine Learning Research* 16, 1 (2015), 149–153.
[42] Hussein T. Mouftah and Melike Erol-Kantarci. 2013. Smart Grid Communications: Opportunities and Challenges. In *Handbook of Green Information and Communication Systems*. Elsevier, Amsterdam, The Netherlands, 631–663.
[43] Seyed Ali Osia, Ali Shahin Shamsabadi, Ali Taheri, Hamid R Rabiee, and Hamed Haddadi. 2018. Private and Scalable Personal Data Analytics Using Hybrid Edge-to-Cloud Deep Learning. *Computer* 51, 5 (2018), 42–49.

[44] Nikunj C Oza. 2005. Online bagging and boosting. In *IEEE international conference on systems, man and cybernetics*, Vol. 3. IEEE, Waikoloa, HI, USA, 2340–2345.
[45] Minsu Park, Mor Naaman, and Jonah Berger. 2016. A data-driven study of view duration on Youtube. In *International AAAI Conference on Web and Social Media*. AAAI Press, Cologne, Germany, 651–654.
[46] Pankesh Patel, Muhammad Intizar Ali, and Amit Sheth. 2017. On Using the Intelligent Edge for IoT Analytics. *IEEE Intelligent Systems* 32, 5 (2017), 64–69.
[47] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. 2015. Low latency geo-distributed data analytics. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 421–434.
[48] Bozhao Qi, Lei Kang, and Suman Banerjee. 2017. A vehicle-based edge computing platform for transit and human mobility analytics. In *ACM/IEEE Symposium on Edge Computing*. ACM, San Jose, CA, USA, 1–14.
[49] Rajiv Ranjan. 2014. Streaming Big Data Processing in Datacenter Clouds. *IEEE Cloud Computing* 1, 1 (2014), 78–83.
[50] George F Riley and Thomas R Henderson. 2010. The ns-3 network simulator. In *Modeling and Tools for Network Simulation*. Springer, Berlin, Germany, 15–34.
[51] Peter J Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20 (1987), 53–65.
[52] Mahadev Satyanarayanan. 2017. The emergence of edge computing. *Computer* 50, 1 (2017), 30–39.
[53] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. 2009. The Case for VM-based Cloudlets in Mobile Computing. *IEEE Pervasive Computing* 8, 4 (2009), 14–23.
[54] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.
[55] Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. 2006. PAC model-free reinforcement learning. In *International conference on Machine learning*. ACM, New York, NY, USA, 881–888.
[56] Alexander Styler, Gregg Podnar, Paul Dille, Matthew Duescher, Christopher Bartley, and Illah Nourbakhsh. 2011. Active management of a heterogeneous energy store for electric vehicles. In *IEEE Forum on Integrated and Sustainable Transportation Systems*. IEEE, Vienna, Austria, 20–25.
[57] Masashi Sugiyama, Neil D Lawrence, Anton Schwaighofer, et al. 2017. *Dataset Shift in Machine Learning*. The MIT Press, Cambridge, MA, USA.
[58] Huangshi Tian, Minchen Yu, and Wei Wang. 2018. Continuum: A Platform for Cost-Aware, Low-Latency Continual Learning.. In *ACM Symposium on Cloud Computing*. ACM, New York, NY, USA, 26–40.
[59] Leslie G Valiant. 1984. A theory of the learnable. In *ACM Symposium on Theory of Computing*. ACM, New York, NY,US, 436–445.
[60] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3-4 (1992), 279–292.
[61] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. 2016. Characterizing concept drift. *Data Mining and Knowledge Discovery* 30, 4 (2016), 964–994.
[62] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. 2015. Petuum: A New Platform for Distributed Machine Learning on Big Data. *IEEE Transactions on Big Data* 1, 2 (2015), 49–67.
[63] Haifeng Yu and Amin Vahdat. 2002. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Transactions on Computer Systems* 20, 3 (2002), 239–282.
[64] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. 2019. Deep Learning in Mobile and Wireless Networking: A Survey. *IEEE Communications Surveys & Tutorials* 21, 3 (2019), 2224–2287.

Received January 2020; revised February 2020; accepted March 2020