Optimal Routing for Stream Learning Systems

Xinzhe Fu and Eytan Modiano
Lab for Information and Decision Systems, MIT

IEEE INFOCOM 2022 - IEEE Conference on Computer Communications | 978-1-6654-5822-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/INFOCOM48880.2022.9796959

***Abstract*—Consider a stream learning system with a source and a set of computation nodes that solves a machine learning task modeled as stochastic convex optimization problem over an unknown distribution $\mathcal{D}$. The source generates i.i.d. data points from $\mathcal{D}$ and routes the data points to the computation nodes for processing. The data points are processed in a streaming fashion, i.e., each data point can be accessed only once and is discarded after processing. The system employs local stochastic gradient descent (local SGD), where each computation node performs stochastic gradient descent locally using the data it receives from the source and periodically synchronizes with other computation nodes. Since the routing policy of the source determines the availability of data points at each computation node, the performance of the system, i.e., the optimization error obtained by local SGD, depends on the routing policy.**

In this paper, we study the influence of the routing policy on the performance of stream learning systems. We first derive an upper bound on the optimization error as a function of the routing policy. The upper bound reveals that the routing policy influences the performance through tuning the bias-variance trade-off of the optimization process, and gives rise to a framework for optimizing the routing policy for stream learning systems. By minimizing the upper bound, we propose an optimal static routing policy that achieves the best trade-off for stream learning systems with deterministic data generation process. We then propose a routing policy that can approximate the optimal static routing policy arbitrarily closely for systems where the data points are generated according to a stochastic process with unknown rate. Finally, we conduct simulations using Support Vector Machine as the machine learning task on a real data set, and show that the optimal static routing policy has excellent empirical performance in terms of minimizing the optimization error and the proposed stochastic routing policy closely matches the optimal static routing policy.

I. Introduction

Stream learning is a machine learning paradigm where the training data arrives as a continuous stream [1], [2] and the learning algorithm can only perform one pass over the training data. It models machine learning tasks in stream analytics applications such as video streaming analysis [3], network traffic classification [4], [5], and social network user profiling [6], where the data is naturally generated in a streaming fashion and has tight latency constraints. As machine learning-based analysis can discover valuable insights and patterns from streaming data [1], [7], systems for stream learning have become a key component in modern stream analytics platforms such as Twitter Heron [8], S4 [9], and Apache Flink [10].

A typical stream learning system has two logical components: a network component that consists of a source that generates the training data and a set of parallel computation nodes that receive the training data from the source, and an underlying learning algorithm that the computational nodes follow to process the training data and solve the machine learning task. There has been a flurry works that study the underlying learning algorithms for stream learning systems [7], [11], [12], [13], [14], [26]. Some of them work for general machine learning tasks while others are designed for specific machine learning tasks or models, such as Support Vector Machine [11], Neural Networks [7], and least-square regression [13]. On the other hand, the network component of stream learning systems has been largely overlooked. Different from other distributed machine learning systems, in stream learning systems, the training data is routed to the computation nodes by the source and each data point can be accessed only once. Due to this distinction in the availability of training data, the network component in stream learning system plays a fundamentally different role compared to previous studies on network operations in other learning systems, e.g., traditional distributed learning systems (where the training data is stored in a central memory and can be accessed by all computation nodes [15], [16], [17], and federated learning systems, where the training data is stored in the local memory of of each node and can be accessed on demand [18], [19], [20], [21].

The core operation of the network component of stream learning systems is the routing policy employed by the source. The routing policy determines the availability of the training data at each computation node, and thus controls the execution flow of the underlying machine learning algorithms. Therefore, the routing policy can have significant influence on the performance of stream learning systems.

In this paper, we consider a stream learning system with a source s and M computation nodes. The machine learning task is modeled as a convex optimization problem whose goal is to find the optimization variable w (often corresponding to model parameters) that minimizes the objective function $F(w) := \mathbb{E}_{x\sim\mathcal{D}}[f(w,x)]$, with $\mathcal{D}$ being an unknown distribution, x representing the data and f being the loss function. Such formulation is common in the machine learning literature [23]. We will discuss in Section II on how this formulation corresponds to real-life machine learning tasks. We first study a discrete time deterministic system where at each time slot t, the source s generates λ i.i.d. data points x_i from the distribution $\mathcal{D}$. Next, s sends a_m data points to the buffer of node m with $\sum_{m=1}^{M} a_m = \lambda$. Each node can process at most μ data points per time slot. To distill the effect of routing policies, we fix the underlying algorithm employed by the system as local stochastic gradient descent (local SGD). The main reasons behind choosing local SGD for our model are that in theory local SGD achieves near-optimal performance with minimal assumptions [25], and in practice local SGD

This work was supported by NSF CNS-1907905 and by Office of Naval Research (ONR) grant award N00014-20-1-2119.

is easy to implement and is widely adopted in real machine learning systems [26], [27]. In local SGD, each node m maintains a local iterate of the optimization variable $w^{(m)}$ and performs stochastic gradient descent based on the local iterate using its local data. Local SGD proceeds in rounds, where at the beginning of each round, each node with more than C data points in the buffer participates in the round by taking C steps of SGD using data points in the buffer. At the end of each round, each node updates its local iterate as the average of the local iterates of the nodes that have participated in the round (see Section II. B. for a detailed introduction to loca SGD). Given a time horizon of T time slots, the performance of the stream learning system is measured by the optimization error, i.e., the gap between the average of all the local iterates and the optimal solution after T time slots.

To quantitatively characterize the influence of the routing policy, we first derive an upper bound on the optimization error as a function of the routing policy $\{a_m\}$. The upper bound reveals that the routing policy influences the optimization error through tuning the trade-off between the number of rounds that local SGD completes by T time slots, which can be interpreted as the bias of the optimization process of the system, and the number of participating nodes of each round, which relates to the variance of the optimization process. The upper bound gives rise to an optimization framework for the routing policy of stream learning systems. Through minimizing the upper bound, we identify the optimal static routing policy that achieves the best bias-variance tradeoff. The optimal static routing policy has a special form that every time it sends a^* data points to approximately $\lfloor \lambda/a^* \rfloor$ nodes, where the value of a^* depends on the properties of F and system parameters λ and μ. As a^* decreases with the noise of the data generating process (which will be formally defined in Section II), our results yield interesting insights, namely, the optimal policy should concentrate more data points to fewer nodes when the noise is low (a^* is large), while balance the data points over more nodes when the noise is high (a^* is small).

Next, we extend our results to stochastic systems, where the source generates $a(t)$ data points at time t with $a(t)$ being a random variable such that $\mathbb{E}[a(t)] = \lambda$ and the parameter λ is unknown. Since computing a^* requires knowledge of λ, the optimal static routing policy cannot be applied in stochastic systems. We thus seek a stochastic routing policy that has close performance to the optimal static routing policy. We propose the Priority-K policy, that constructs virtual queues for each node and makes routing decision based on virtual queues. Priority-K does not rely on knowledge of λ, yet achieves a steady state distribution of routing decisions that can be arbitrarily close to the optimal static routing policy, with the difference decreasing exponentially with K. It can also be viewed as a method that solves a new kind of routing problems in stochastic queueing network that aim at maintaining the data rate of each node at a pre-defined level (rather than load-balancing or optimizing throughput). To summarize, our main results are as follows:

- We derive an upper bound on the optimization error of local SGD in stream learning systems as a function of the routing policy. The upper bound gives rise to an optimization framework for the routing policy in stream learning systems.
- Based on minimization of the upper bound, we derive the optimal static routing policy for deterministic stream learning systems.
- We then propose a stochastic routing policy, Priority-K, that can approximate the optimal static routing policy arbitrarily closely in stochastic stream learning systems.
- We empirically validate our theoretical results through simulations using SVM as the machine learning tasks on a real data set from libSVM [34].

It is worth mentioning that previous works on optimal routing for stream processing systems have considered the abstract problem where the objects to be routed are tasks that require a sequence of network resources from the network nodes [28], [29], [30], [31]. The problems therein are typically translated to a general network flow problem where the goal is to maximize the completion rate of the tasks subject to the resource constraints of the nodes [30], [31]. On the other hand, our work focuses specifically on the interaction between the routing of training data points and the underlying machine learning algorithm in stream learning systems.

The rest of the paper is organized as follows. In section II, we present our model for stream learning systems. The upper bound on optimization error as a function of the routing policy is derived in Section III. Based on the upper bound, we derive the optimal routing policy for deterministic systems in Section IV. We then extend the results to stochastic systems in Section V. We conduct simulations to evaluate the empirical performance of our routing policy in Section VI and we conclude the paper in Section VII.

II. Model and Problem Formulation

In this section, we present our model for the stream learning system that involves definitions of the machine learning task, network component, the underlying algorithm and the optimization error as the performance metric.

A. Machine Learning Task

The machine learning task is modeled as a stochastic optimization problem

$$\begin{aligned} \textbf{minimize } & F(w) := \mathbb{E}_{x\sim\mathcal{D}}[f(w,x)] \\ \textbf{s.t. } & w \in \mathcal{W}, \end{aligned} \tag{1}$$

where f is the loss function, w is the optimization variable with domain $\mathcal{W}$, and x is the data that follows distribution $\mathcal{D}$. The distribution $\mathcal{D}$ is unknown, but we will have access to i.i.d. data points that are generated from $\mathcal{D}$. We aim to (approximately) find the minimizer $w^* := \arg\min_{w\in\mathcal{W}} F(w)$ with first-order (gradient-based) algorithms. This formulation is common in machine learning literature [23], [24] for supervised learning tasks. As a concrete example, consider linear regression with regressor $\theta \in \mathbb{R}^n$ and dependent variable

$y \in \mathbb{R}$. The goal is to compute a linear model parameterized by $w \in \mathbb{R}^n$ such that the value of $w^{\mathrm{T}}\theta$ is close to the corresponding value of y, for (θ, y) from population distribution $\mathcal{D}$. The closeness is measured by the square-loss function $(y - w^{\mathrm{T}}\theta)^2$. We do not know the distribution $\mathcal{D}$ but we have access to i.i.d. training data coming from $\mathcal{D}$ and seek to learn a model with small expected loss with respect to $\mathcal{D}$ using the training data. In this example, a generic training data point x consists of the regressor θ and the dependent variable y and the loss function $f(w, x) = f(w, \theta, y)$ is the square function $f(w, \theta, y) = (y - w^{\mathrm{T}}\theta)^2$.

Throughout the paper, we will make the following assumptions on the optimization problem:

Assumption 1. *$\mathcal{W}$ is a convex and bounded set with $\sup_{w,w' \in \mathcal{W}} ||w - w'|| \leq D$. The gradient of F is bounded with $\sup_{w \in \mathcal{W}} ||\nabla F(w)|| \leq G_0$ and $\sup_{w \in \mathcal{W}} \sup_x ||\nabla f(w, x)|| \leq G_1$. $\mathbb{E}_{x \sim \mathcal{D}}[||\nabla f(w, x) - \nabla F(w)||^2] \leq \sigma^2$.*

Note that the gradient of F being bounded is implied by the gradient of f being bounded with probability one, and is thus a weaker assumption. Moreover, the quantity σ^2 essentially characterizes the variance of the gradient $\nabla f(w, x)$ over the distribution $\mathcal{D}$. If we use $\nabla f(w, x)$ as a stochastic gradient for $\nabla F(w)$, then σ^2 can be considered as the noise of the stochastic gradient, as the stochastic gradient $\nabla f(w, x)$ is computed over one data point x sampled from the distribution $\mathcal{D}$ rather than over the whole distribution.

Assumption 2 (Convexity)**.** *For each x, $f(w, x)$ is convex with respect to w, i.e., $\forall w_1, w_2 \in \mathcal{W}$, $f(w_1, x) \geq f(w_2, x) + \nabla f(w_2, x)^{\mathrm{T}}(w_1 - w_2)$.*

Note that we do not assume the function to be smooth (differentiable), so the notation $\nabla f(w, x)$ should be interpreted as the sub-gradient when the gradient does not exist.

B. Stream Learning System

1) Network Component: The network component of the stream learning system consists of a source s and M computation nodes. The source s generates i.i.d. data points from the distribution $\mathcal{D}$, and has a buffer Q_s that stores the generated data points. Each data point will be sent to one of the computation nodes for processing. Each computation node m has a buffer Q_m that stores the incoming data points. The nodes process the data points by computing gradients in a streaming fashion, i.e., for a generic data point x_i in the buffer of a node, the node can only compute the gradient of the loss function on the data point with respect to one value of the optimization variable, i.e., $\nabla f(w, x_i)$ for some w. After such computation, the data point leaves the buffer and cannot be reused.

The network component operates in discrete time with $t = 1, 2, \ldots$. The source s generates $a(t)$ data points at time slot t. We first consider deterministic systems where the generation of data points follows a deterministic process with $a(t) = \lambda$ for all t and the parameter λ is known. In section V, we will study stochastic systems where $a(t)$ is a sequence of i.i.d. random variables with $\mathbb{E}[a(t)] = \lambda$ and the parameter λ is unknown. In the deterministic system, the generated data points are first stored in the source buffer Q_s and then routed to the buffer of the nodes following some routing policy $\{a_m\}$ with each a_m being a non-negative real number with $\sum_{m=1}^{M} a_m = \lambda$. Thus, a_m can be interpreted as the time-average number of data points that node m receives. The routing process is conducted as follows: the source maintains a routing counter A_m for each node m. Every time slot, the source increments A_m by a_m, sends $\lfloor A_m \rfloor$ data points from the source buffer to the buffer of node m, and set $A_m := A_m - \lfloor A_m \rfloor$. The above procedure ensures an integer number of data points in sent in every time slot. The maximum processing rate of each node is μ, i.e., each node can compute the gradient on at most μ data points every time slot. Therefore, for system stability, we need to have $a_m \leq \mu$ for all m. Without loss of generality, we assume that $\lambda \leq M\mu$, i.e., the generating rate of data points is no larger than the maximum total processing rate of all the nodes. As can be seen from the model, the routing policy $\{a_m\}$ determines how the incoming data points are routed to the computation nodes. *The main goal of this paper is to explore how the routing policy influence the performance of the stream learning system.*

2) Underlying Algorithm: We adopt the local stochastic gradient descent (local SGD) as the underlying algorithm that the system uses to solve the optimization problem (1). In local SGD, each node maintains its own iterate of the optimization variable. It proceeds in rounds, with each round consisting of C iterations of stochastic gradient descent at each participating node and an averaging step on which the local iterates are averaged among the participating nodes and synchronized to all the nodes. The parameter C is prefixed and will be referred to as the averaging interval. The nodes participating in a round are the ones that have at least C data points in the buffer at the beginning of each round. A new round starts after the end of the previous round as soon as some node(s) has at least C data points in the buffer. We will use i to represent the index of iterations and use $w_i^{(m)}$ to denote the local iterate of node m at iteration i. Consider a generic iteration i in a round where the set of participating nodes is $\mathcal{M}$. Then, for each node $m \in \mathcal{M}$, it uses a data point $x_i^{(m)}$ in its buffer and updates $w_i^{(m)} := w_{i-1}^{(m)} - \eta_i \nabla f(w_i^{(m)}, x_i^{(m)})$, where η_i is the step size at iteration i. If i is the last iteration of the round, then all the nodes (including non-participating ones) set their local iterate to the average of the local iterate of the participating nodes projecting onto the domain $\mathcal{W}$, i.e., $\Pi_{\mathcal{W}}\left[\hat{w}_i := \frac{1}{|\mathcal{M}|}\sum_{m \in \mathcal{M}} w_{i-1}^{(m)}\right]$ for all m, with $\Pi_{\mathcal{W}}[\cdot]$ representing the projection onto $\mathcal{W}$. The pseudo-code of local SGD is presented in **Algorithm 1**. We present a schematic of our system in **Figure 1**.

C. Performance Metric

The notion of optimization error, which essentially measures the gap between the produced solution and the optimal after a certain number of iterations, is commonly used to measure the

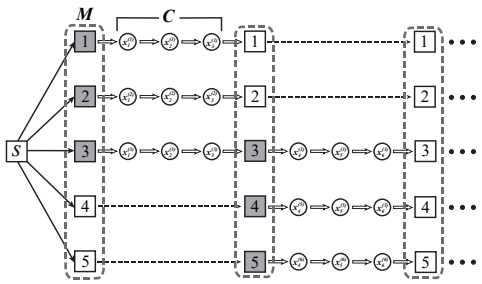

Fig. 1. Example of the stream learning system with $M = 5$ and $C = 3$. For iteration $i = 1, 2, 3$ (the first round), $\mathcal{M}_i = \{1, 2, 3\}$; for iteration $i = 4, 5, 6$ (the second round), $\mathcal{M}_i = \{3, 4, 5\}$.

Algorithm 1 Local SGD

Input: Averaging Interval C

Initialize: $w_0^{(1)} = \ldots = w_0^{(M)} = w_0 \in \mathcal{W}$.
$\mathcal{M} :=$ the set of nodes with at least C data samples in the buffer.
for $i = 1, 2, \ldots$ **do**
  for $m \in \mathcal{M}$ **do**
    Use one data point $x_i^{(m)}$ in the buffer.
    $w_{i+1}^{(m)} := w_i^{(m)} - \eta_i \nabla f(w_i^{(m)}, x_i^{(m)})$.
  for $m \notin \mathcal{M}$ **do**
    $w_{i+1}^{(m)} := w_i^{(m)}$.
  if $i \mod C \equiv C - 1$ **then**
    $\hat{w}_{i+1} := \Pi_{\mathcal{W}} \left[\frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} w_{i+1}^{(m)} \right]$.
    $w_{i+1}^{(m)} := \hat{w}_{i+1}$ for all m.
    Update $\mathcal{M}$ as the set of nodes currently with at least C data samples in the buffer.

performance of optimization algorithms [23], [24]. We adopt the optimization error as the performance metric of the stream learning system. However, a key difference in our setting is that the number of iterations completed in a certain number of time slots depends on the routing policy, and it is more appropriate to measure the performance of the system as the quality of the solution obtained after certain time. Therefore, we define the performance metric as the optimization error with respect to wall-clock time, as follows.

Definition 1 (Optimization Error)**.** *Suppose under some routing policy, local SGD completes I_T iterations after T time slots. Let $\mathcal{M}_i$ be the participating set at iteration i and $\bar{w}_T := \Pi_{\mathcal{W}} \left[\frac{1}{I_T} \cdot \sum_{i=1}^{I_T} \frac{1}{|M_i|} \sum_{m \in \mathcal{M}_i} w_i^{(m)} \right]$. Then, the optimization error under the routing policy after T time slots is $\mathbb{E}[F(\bar{w}_T) - F(w^*)]$.*

Define $\hat{w}_i = \frac{1}{|M_i|} \sum_{m \in \mathcal{M}_i} w_i^{(m)}$ to be the averaged iterate of iteration i (whether i is the last iteration of a round or not). In Definition 1, we have implicitly assumed that the solution produced by local SGD is the mean of the averaged iterates of all the iterations completed, which follows the convention of previous analysis of local SGD [25]. Returning the last averaged iterate $\hat{w}(T)$ as the solution might sound like a more natural choice, but as shown in [24], $\bar{w}_T$ used in our definition usually has a smaller error than $\hat{w}(T)$. Furthermore, using techniques from [23], we can extend our results to the case where the optimization error is defined with respect to $\hat{w}(T)$.

In this paper, we will seek routing policies that minimizes the optimization error after T time slots[1]. Our results can also be extended to other alternatives, e.g., using the averaged iterate of the last iteration as the solution in the definition of optimization error.

III. Influence of Routing Policies

In this section, we characterize the influence of routing policies on the stream learning system by deriving an upper bound on the optimization error as a function of the employed routing policy.

A. Intuition – Bias-Variance Tradeoff

Before delving into the details about the upper bound, it is useful to consider some special cases to develop intuition behind how routing policies influence the optimization error, and thus the performance of stream learning systems. Suppose $C = 1$, i.e., the local SGD performs averaging every iteration. It follows that in this case, for each m, $w_i^{(m)} = \hat{w}_i$ before every local gradient step, i.e., the local iterates never diverge from the averaged iterate $\hat{w}_i$. If we ignore the projections $\Pi_{\mathcal{W}}$ for now, then the evolution of $\hat{w}_i$ can be written as $\hat{w}_{i+1} := \hat{w}_i - \eta_i \cdot \frac{1}{|\mathcal{M}_i|} \sum_{m \in \mathcal{M}_i} \nabla f(\hat{w}_i, x_i^{(m)})$, where $\mathcal{M}_i$ is the set of participating nodes at iteration i. In this sense, the averaged iterate $\hat{w}_i$ in local SGD evolves as in mini-batch SGD with a iteration-varying batch size of $|\mathcal{M}_i|$, i.e., at every iteration $\hat{w}_i$ moves along the direction that is the average of $|\mathcal{M}_i|$ stochastic gradients.

Suppose $\lambda = M \leq \mu$. We consider two routing policies. The first one routes all the data samples into the first node, i.e., $a_1 = \lambda = M, a_2 = \ldots = a_M = 0$. The second policy, at each time slot, route the λ data samples generated from the source to the first λ nodes with each node receiving one data sample, i.e., $a_1 = \ldots = a_M = 1$. Under the first policy, for every iteration, only the first node participates in the updates and $|\mathcal{M}_i| = 1$. After T time slots, local SGD runs for λT iterations. Under the second policy, all the M nodes participate in every update. After T time slots, local SGD runs for T iterations, which is only $1/M$ of the number of iterations under the first policy. This suggests that under the first policy, the solution produced by the stream learning systems will have a lower *bias*, i.e., if the gradients were accurate, the solution under the first policy will have a smaller error. On the other hand, under the first policy, the averaged iterate is updated with gradients that are computed on only 1 data point while under the second policy the averaged iterate is updated with the averaged gradients on M data points, which have lower variances. It follows that the

[1]Here, we essentially ignore the time consumed by averaging steps, which may depend on the communication delay between nodes. It is easy to incorporate the communication delay in our analysis. However, doing so will render the results dependent on the relative scale of communication delay and computation time, which varies in different systems and can hide the insights we have on the impact routing policy on the optimization error.

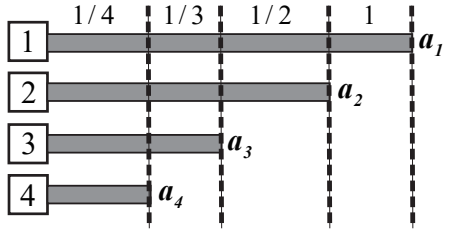

Fig. 2. Illustration of the multiplicity factor $\gamma(\{a_m\})$ with $M = 4$. Under this routing policy, after sufficient number of iterations, in a_4/a_1 of the iterations, the number of participating nodes $\mathcal{M}_i = 4$, in $(a_3 - a_4)/a_1$ of the iterations, $\mathcal{M}_i = 3$, in $(a_2 - a_3)/a_1$ of the iterations, $\mathcal{M}_i = 2$ and in $(a_1 - a_2)/a_1$ of the iterations, $\mathcal{M}_i = 1$. Correspondingly, $\gamma(\{a_m\}) := \sum_{m=1}^{4} \frac{a_m - a_{m+1}}{a_1 m}$, with a_5 defined as 0.

stochastic gradients used under the first policy can be more noisy and less accurate. Such noise naturally propagates to the final solution and the solution under the first policy will thus have a higher *variance*. The optimization error takes into account both the bias and the variance, or in other words, total number of updates and the quality of each update. Therefore, minimizing the optimization error requires the routing policy to balance between the number of updates and the variance of each update, which has a similar flavor as the classical bias-variance tradeoff in machine learning literature. Note that such tradeoff is in a sense unique to stream learning systems, since the training data is generated as a stream and cannot be reused. Other distributed machine learning systems do not possess this kind of tradeoff as the training data therein is available beforehand and typically the algorithm can perform multiple passes over the data.

B. Upper Bound on Optimization Error

We now derive the upper bound on the optimization error of local SGD as a function of the routing policy $\{a_m\}$. Without loss of generality, we assume $a_1 \geq a_2 \geq \ldots \geq a_M$. Also, as the processing rate of a server is μ, we only consider the case where $a_1 \leq \mu$. For a routing policy $\{a_m\}$, we define its multiplicity as $\gamma(\{a_m\}) := \sum_{m=1}^{M} \frac{a_m - a_{m+1}}{a_1 m}$ (with $a_{M+1} = 0$), which is essentially the average inverse of the number of participating nodes in each round (see Figure 2 for a graphic illustration.). The upper bound is formally presented in Theorem 1.

Theorem 1. *The optimization error of local SGD with step size $\eta_i = \frac{\alpha}{\sqrt{i}}$ is upper-bounded by*

$$\mathbb{E}[F(\bar{w}_T) - F(w^*)] \leq \frac{C_1}{\sqrt{a_1 T}} + \frac{C_2 \gamma(\{a_m\})}{\sqrt{a_1 T}} + O\left(\frac{1}{T}\right),$$

where $C_1 = 2\alpha C G_0 G_1 + \frac{2D^2}{\alpha} + \frac{\alpha G_0^2}{2}$ and $C_2 = \frac{\alpha \sigma^2}{2}$ are independent of the routing policy.

We prove the theorem by analyzing the averaged iterate $\hat{w}_i = \frac{1}{|\mathcal{M}_i|} \sum_{m \in \mathcal{M}_i} w_i^{(m)}$, with $\mathcal{M}_i$ being the set of participating nodes at iteration i. The multiplicity $\gamma(\{a_m\})$ shows up in the upper bound because it is related to the variance of the stochastic gradients applied to the averaged iterate. The proof is omitted due to spce limitations.

We define the upper bound of the optimization error in Theorem 1 (ignoring the lower order term in T) as a function of the routing policy $\{a_m\}$ as

$$U(\{a_m\}) = \frac{C_1}{\sqrt{a_1 T}} + \frac{C_2 \gamma(\{a_m\})}{\sqrt{a_1 T}}, \tag{2}$$

We will derive the optimal routing policy based on $U(\{a_m\})$.

Remark: (i). The upper bound on the optimization error in Theorem 1 depends on the step size. The choice of step size in the theorem, i.e., $\eta_i = \frac{\alpha}{\sqrt{i}}$ has been shown [23], [24] to be in the optimal regime in terms of the scaling with respect to the iteration i. (ii). Although $U(\{a_m\})$ is an upper bound on the optimization error, which might not exactly match the true value of the optimization error for every objective function, the upper bound is tight in terms of scaling with respect to T (see e.g. [32]). Furthermore, it can be seen from Section VI that routing policies that minimizes the upper bound $U(\{a_m\})$ also enjoys superior performance in terms of minimizing the true value of the optimization error. As the exact dependence of the optimization error on the routing policies is mathematically intractable, the upper bound $U(\{a_m\})$ provides a good proxy to measure the affect of the routing policies on the performance of stream learning systems. (iii). To demonstrate the significance of the routing policy, consider the value of $U(\{a_m\})$ of the two routing policies discussed in Section IV.A. The first one has $a_1 = \lambda = M$ and $\gamma(\{a_m\}) = 1$ while the second one has $a_1 = 1$ and $\gamma(\{a_m\}) = 1/M$. Therefore, the values of upper bound $U(\{a_m\})$ of the two policies are $\frac{C_1 + C_2}{\sqrt{MT}}$, and $\frac{C_1}{\sqrt{T}} + \frac{C_2}{M\sqrt{T}}$, respectively. The ratio of the two upper bounds can grow as $O(\sqrt{M})$, which scales with the size of the system and can grow much larger than the constants C_1, C_2 which depend solely on the properties of the optimization problem. Therefore, although the routing policy does not affect the scaling of optimization error with respect to the time horizon T, it can determine the scaling of optimization error with respect to the size of the system.

IV. Optimal Routing in Deterministic Systems

The upper bound sets up a framework for optimizing the routing policy in stream learning systems. In this section, we use the framework to derive the optimal routing policy in deterministic stream learning systems through minimizing the upper bound $U(\{a_m\})$ on the optimization error. We start by analyzing the multiplicity factor $\gamma(\{a_m\})$ which is a key term that characterizes the influence of routing policy on the optimization error. We define Γ_a as $\{\{a_m\} \mid a_1 = a, \mu \geq a_1 \geq a_2 \geq \ldots \geq a_M, \sum_{m=1}^{M} a_m = \lambda\}$, i.e., the set of all routing policies with the maximum data rate to the nodes being $a_1 = a$. Since for a given value of $a_1 = a$, $U(\{a_m\})$ is optimized by minimizing $\gamma(\{a_m\})$, in Lemma 1, we derive the optimal routing policy and the optimal value of $\gamma(\{a_m\})$ in the set Γ_a.

Lemma 1. *For all* $a \leq \mu$, $\min_{\{a_m\}\in\Gamma_a} \gamma(\{a_m\}) = \left(\lambda/a - \lfloor \lambda/a \rfloor\right) \cdot \frac{1}{\lceil \lambda/a \rceil} + \left(1 - \lambda/a + \lfloor \lambda/a \rfloor\right) \cdot \frac{1}{\lfloor \lambda/a \rfloor}$. *Furthermore,* $a/\lambda \leq \min_{\{a_m\}\in\Gamma_a} \gamma(\{a_m\}) \leq \frac{9}{8} \cdot a/\lambda$.

The proof of Lemma 1 consists of simple algebraic manipulations, and is omitted due to space limitations. From Lemma 1 and its proof, we have the following corollary that specifies the form of the routing policy that minimizes $U(\{a_m\})$ in Γ_a.

Corollary 1. *The routing policy in* Γ_a *that minimizes* $U(\{a_m\})$ *is of the form:* $a_1 = a_2 = \ldots = a_{\lfloor \lambda/a \rfloor} = a$, $a_{\lfloor \lambda/a \rfloor + 1} = \lambda - a \cdot \lfloor \lambda/a \rfloor$, *and* $a_{\lfloor \lambda/a \rfloor + 2} = \ldots = a_M = 0$.

With slight abuse of notation, we define $\gamma(a)$ as $\left(\lambda/a - \lfloor \lambda/a \rfloor\right) \cdot \frac{1}{\lceil \lambda/a \rceil} + \left(1 - \lambda/a + \lfloor \lambda/a \rfloor\right) \cdot \frac{1}{\lfloor \lambda/a \rfloor}$. From Lemma 1 and Corollary 1, to find the routing policy that minimizes the upper bounds U, it suffice to find the $a \in [\lambda/M, \mu]$ that minimizes $\frac{C_1}{\sqrt{aT}} + \frac{C_2\gamma_a}{\sqrt{aT}}$ and the policy of the form in Corollary 1 in Γ_a. Here $a \geq \lambda/M$ since it corresponds to the largest data rate among all the nodes. As we have the closed-form expression of γ_a, we can directly minimize $\frac{C_1}{\sqrt{aT}} + \frac{C_2\gamma_a}{\sqrt{aT}}$. However, the minimizer would not have a closed-form expression due to the floor and ceiling functions in $\gamma(a)$. Instead, we will approximate $\gamma(a)$ by a/λ, which by Lemma 1, is at most a factor of $\frac{9}{8}$ of $\gamma(a)$. As will be seen later, such approximation will lead to a much cleaner and insightful expression on the optimal value of a. Hence, we replace $\gamma(a)$ with a/λ and seek to find $a \in [\lambda/M, \mu]$ that minimizes

$$U(a) = \frac{C_1}{\sqrt{aT}} + \frac{C_2\sqrt{a}}{\lambda\sqrt{T}}, \tag{3}$$

The function $U(a) = \frac{C_1}{\sqrt{aT}} + \frac{C_2\sqrt{a}}{\lambda\sqrt{T}}$ is monotonically decreasing for $a \in [0, \frac{C_1\lambda}{C_2}]$ and monotonically increasing for $a \in [\frac{C_1\lambda}{C_2}, \infty)$. Define $C_3 = C_1/C_2$ It follows that the $a^* = \arg\min_{a\in[\lambda/M,\mu]} U(a)$ is equal to

$$\arg\min_{a\in[\lambda/M,\mu]} U(a) = \begin{cases} \mu, & \text{if } \mu \leq \frac{C_1\lambda}{C_2} \\ \frac{C_1\lambda}{C_2}, & \text{if } \frac{\lambda}{M} \leq \frac{C_1\lambda}{C_2} \leq \mu \\ \frac{\lambda}{M}, & \text{if } \frac{C_1\lambda}{C_2} \leq \frac{\lambda}{M} \end{cases}$$

Based on the preceding discussion, we have the following optimal routing policy that (approximately) minimizes the upper bound on the optimization error in deterministic stream learning systems. We will refer to such policy as the optimal static routing policy, to distinguish it with the dynamic policy that we will propose for stochastic systems.

Definition 2 (Optimal Static Routing Policy)**.** *The optimal static routing policy* $\{a_m^*\}$ *is one such that* $a_m^* = a^*$ *for* $m = 1, \ldots, \lfloor \lambda/a^* \rfloor$, $a_{\lfloor \lambda/a^* \rfloor + 1} = \lambda - a^* \cdot \lfloor \lambda/a^* \rfloor$, *and* $a_m^* = 0$ *for* $m = \lfloor \lambda/a^* \rfloor + 2, \ldots, M$, *where*

$$a^* = \begin{cases} \mu, & \text{if } \mu \leq C_3\lambda \\ C_3\lambda, & \text{if } \frac{\lambda}{M} \leq C_3\lambda < \mu \\ \frac{\lambda}{M}, & \text{if } C_3\lambda < \frac{\lambda}{M} \end{cases}$$

Remark: In the definition of the optimal static routing policy, C_3 is defined based on the properties of the objective functions which can be known or estimated beforehand. Also, λ and μ are parameters that are typically known for deterministic systems. Thus, the optimal static routing policy can be implemented in deterministic systems. In the next section, we will discuss stochastic systems, where the parameter λ is unknown and the optimal static routing policy is thus inapplicable.

Discussion: Definition 2 suggests that the optimal routing policy should approximately route the same amount of data (a^*) to as many computation nodes as possible, and determine the optimal value of a^*. Although the value of a^* involves three cases based on the relative magnitude of $\mu, C_3\lambda, \frac{\lambda}{M}$, the first one and the third one exist due to the physical condition that $a^* \in [\frac{\lambda}{M}, \mu]$. The core lies in the second case, where $a^* = C_3\lambda$. This reflects the intuition of bias-variance tradeoff in Section III-A. As in our setup, σ^2 characterizes the level of noise in the gradients as the gradients are computed on data pointed generated from $\mathcal{D}$ rather than directly from the whole distribution $\mathcal{D}$. Since the constant C_1 is independent of σ^2 and C_2 increases with σ^2, the factor $C_3 = C_1/C_2$ decreases as the noise increases. Therefore, in the high-noise regime (C_3 is small), the optimal static routing sets $a^* = \frac{\lambda}{M}$, which essentially routes an equal number of data points to each server, so that each update of the local SGD averages over all the M nodes. This implies that decreasing the noise of each update outweighs increasing the total number of iterations in terms of minimizing the optimization error. On the other hand, in the low-noise regime (C_3 is large), the optimal static routing sets $a^* = \mu$. This maximizes the total number of iterations completed by T time slots, which implies that having more updates is more beneficial than decreasing the noise of each update (as the noise is already low).

V. Extension to Stochastic Systems

In this section, we extend the optimal routing problem to stochastic stream learning systems where the data generation follows a stochastic process. The optimal static routing policy is no longer applicable here since we do not have knowledge of the data generation rate λ. We instead propose a queue-based routing policy that can arbitrarily closely approximate the optimal static routing policy in stochastic systems.

A. Stochastic System Model

The setup of a stochastic stream learning system is similar as the deterministic system. The main difference is that the number of data points $a(t)$ generated from the source at each time t is no longer deterministic, but a sequence of i.i.d. random variable with $\mathbb{E}[a(t)] = \lambda$, where λ is unknown.[2] In the stochastic system, we allow for dynamic routing policies $\{a_m(t)\}$ with $\sum_m a_m(t) = a(t)$. As in the deterministic system, each node can compute the gradient on at most μ

[2]Our results can be easily generalized to the case where the processing rate is also a stochastic process with unknown rate μ. For simplicity of presentation, we only consider the case where the data generation is stochastic.

data points at each time slot. We assume that there exists B such that $\mu \leq B$ and $\max\{C_3, 1\} \cdot a(t) \leq B$ almost surely.

Since λ is unknown, we cannot compute the a^* in the optimal static routing policy, which renders the policy inapplicable in stochastic systems. Instead, our goal is to design a routing policy that works for stochastic stream learning systems that closely approximate the optimal static routing policy, albeit without knowing a^*. We will consider policies where $\lim_{T\to\infty} \frac{\sum_{t=1}^{T} a_m(t)}{T}$ exists and use

$$\sum_{m=1}^{M} |a_m^* - \lim_{T\to\infty} \frac{\sum_{t=1}^{T} a_m(t)}{T}|$$

as the measure of closeness, with $\{a_m^*\}$ being the optimal static routing policy.

B. Stochastic Routing Policy

Before presenting our routing policy for stochastic systems, we first make some observations regarding the optimal static routing policy. In Definition 2, the third case $C_3\lambda \leq \frac{\lambda}{M}$ is equivalent to $C_3 \leq \frac{1}{M}$, which can be checked without knowing the value of λ. Furthermore, since in the third case all the nodes get the same number of data points, round-robin routing or join-the-shortest-queue work even in stochastic systems [33]. In the following, we will thus focus on approximating the first and the second cases of the optimal static policy in stochastic systems since the third case can be easily dealt with.

The first and the second cases cannot be directly distinguished without the knowledge of λ. However, the optimal static routing in both cases still have the structure that $\lfloor \lambda/a^* \rfloor$ nodes receive the same amount of data and one node gets the remaining data points. Therefore, for a routing policy that works in stochastic systems and approximates the optimal static routing policy, it needs to leverage the structure and approximately distinguish the first and the second cases with auxiliary information. To do so, our proposed policy, Priority-K, maintains two virtual queues $Y_m(t)$ and $\tilde{Y}_m(t)$ for each node m that evolve as follows:

$$Y_m(t+1) := [Y_m(t) - \mu]^+ + a_m(t)$$
$$\tilde{Y}_m(t+1) := [\tilde{Y}_m(t) - C_3 a(t)]^+ + a_m(t),$$

where $[z]^+ = \max\{z, 0\}$. Note that maintaining the virtual queues does not rely on knowledge of λ. At time t, the service to virtual queue Y_m is equal to μ and the service to virtue queue $\tilde{Y}$ is equal to $C_3 a(t)$. Intuitively (as will be formally shown later), if Y_m is bounded and maintained at a non-zero level, then the time-average data rate to m is close to μ while if $\tilde{Y}_m$ is bounded and maintained at a non-zero level, then the time-average data rate to m is close to $C_3\lambda$. Therefore, if $\mu \leq (C_3\lambda)$, to approximate the optimal static policy, it is sufficient to keep Y_m for $m = 1, \ldots, \lfloor \lambda/\mu \rfloor$ positive for the majority of the time; On the other hand, if $\mu > C_3\lambda$, it is sufficient to keep $\tilde{Y}_m$ for $m = 1, \ldots, \lfloor \lambda/C_3\lambda \rfloor$ positive for the majority of the time. Priority-K achieves this based on a priority-based routing rule, with the priority simply being an arbitrary ordering of the computation nodes. The details of the Priority-K policy are presented in **Algorithm 2**.

Algorithm 2 The Priority-K Policy

Input: Parameter K

Initialize: $Y_m(1) = 0, \tilde{Y}_m(1) = 0$ for each m.
for $t = 1, 2, \ldots,$ **do**
  $a_m(t) := 0$ for each m.
  for $m = 1, 2, \ldots, M$ **do**
    if $\max\{Y_m(t), \tilde{Y}_m(t)\} \leq K$ **then**
      $a_m(t) := a(t)$.
      End for loop.
  Update $Y_m(t), \tilde{Y}_m(t)$ for each m.

The Priority-K policy takes a parameter K as input. It first checks node 1. If the maximum value of the two virtual queues of node 1 at the current time is no larger than K, then it sets $a_1(t)$ to be $a(t)$. Otherwise, it proceeds to check if the maximum value of the two virtual queues of node 2 at the current time is no larger than K, and sets $a_2(t)$ as $a(t)$ if the condition holds. The process proceeds until node M, i.e., the routing policy will set $a_m(t)$ to be $a(t)$ as soon as it finds a node m with $\max\{Y_m(t), \tilde{Y}_m(t)\} > K$. In **Algorithm 2**, we have omitted the case where $\max\{Y_m(t), \tilde{Y}_m(t)\} > K$ for all m. We will show that the probability of such events decreases exponentially with K. Furthermore, when such events happen, we can either set $a_m(t)$ to be zero for all m (essentially discard the data points generated at t) or set $a_m(t)$ to be $a(t)/M$ for all t. We will study the variant of Priority-K policy that discards the generated data points when $\max\{Y_m(t), \tilde{Y}_m(t)\} > K$ for all m. The analysis holds similarly for the other variant.

Before formal analysis, we first give an intuitive explanation of the design rationale behind Priority-K. Without loss of generality, we consider the case where $\mu > C_3\lambda$ and $\lfloor \lambda/(C_3\lambda) \rfloor = L > 1$. In this case, at every time t, the virtual queues $Y_m(t)$ and $\tilde{Y}_m(t)$ have the same arrival, while the service to $Y_m(t)$ is μ and the service to $\tilde{Y}_m(t)$ is $C - 3a(t)$. As Y_m and $\tilde{Y}_m$ have the same arrivals but the service to Y_m is in expectation larger than that of $\tilde{Y}_m$, $\tilde{Y}_m(t)$ is usually larger than $Y_m(t)$. Therefore, the condition $\max\{Y_m(t), \tilde{Y}_m(t)\} \leq K$ is usually equivalent to $\tilde{Y}_m(t) \leq K$. Recall that the maximum arrival is bounded by B. Consider node 1, as it has the top-priority under Priority-K and $\lambda > C_3\lambda$, $\tilde{Y}_1(t)$ is maintained at a level that is greater than 0 but smaller than $K + B$. It follows that the data rate to 1 is approximately $C_3\lambda$ (it cannot be greater than $C_3\lambda$ since $\tilde{Y}_1$ is bounded). Next, consider node 2, since it has the priority next to node 1, applying the same argument, we have $\tilde{Y}_2(t)$ is also maintained at a level that is greater than 0 but smaller than $K + B$. It follows that the data rate to 2 is approximately $C_3\lambda$. Repeat the argument for $m = 1, \ldots, L$ and noting that the residual data points when $\max\{Y_m(t), \tilde{Y}_m(t)\} > K$ for $m = 1, \ldots, L$ will mostly go to node $L + 1$, we have that the time-average data rates to the nodes under Priority-K are close to those under the optimal static policy.

To begin with analyzing Priority-K, note that $\boldsymbol{Y}(t) = (Y_1(t), \tilde{Y}_1(t), \ldots, Y_M(t), \tilde{Y}_M(t))$ forms a discrete-time Markov chain, and $\{a_m(t)\}$ is a function of $\boldsymbol{Y}(t)$. Furthermore, from the construction of the policy, $0 \leq Y_m(t), \tilde{Y}_m(t) \leq K + B$ for all m, t, where recall that B is an upper bound on $a(t)$. Therefore, the virtual queues are stable and there exists a steady state distribution for the Markov chain. Denote the steady state distribution of $\boldsymbol{Y}(t)$ under the Priority-K policy as π, and the probability and expectation under π as $\mathbb{P}_\pi$ and $\mathbb{E}_\pi$. It follows that $\lim_{T\to\infty} \frac{\sum_{t=1}^{T} a_m(t)}{T} = \mathbb{E}_\pi[a_m(t)]$. Thus, we can equivalently use $\sum_{m=1}^{M} |a_m^* - \mathbb{E}_\pi(a_m(t))|$ to measure the closeness of Priority-K to the optimal static routing. We begin by proving the following two lemma that link $|a_m^* - \mathbb{E}_\pi(a_m(t))|$ for each m to the steady-state distribution of virtual queues. The proofs of the lemmas are omitted due to space limitations.

Lemma 2. *For each m, $\mathbb{E}_\pi[a_m(t)] \leq \mu$ and $\mathbb{E}_\pi[a_m(t)] \leq C_3\lambda$.*

Lemma 3. *For each m, $\mathbb{E}_\pi[a_m(t)] \geq \mu \cdot \mathbb{P}_\pi[Y_m(t) \geq B]$ and $\mathbb{E}_\pi[a_m(t)] \geq C_3\lambda \cdot \mathbb{P}_\pi[\tilde{Y}_m(t) \geq B]$.*

Based on Lemmas 2 and 3, we have that if $a_m^* = \mu$, then $|a_m^* - \mathbb{E}_\pi a_m(t)| \leq \mu(1 - \mathbb{P}_\pi[Y_m(t) \geq B])$ and if $a_m^* = C_3\lambda$, then $|a_m^* - \mathbb{E}_\pi a_m(t)| \leq C_3\lambda(1 - \mathbb{P}_\pi[Y_m(t) \geq B])$. This corresponds to the informal claim we made in the preceding discussion that to approximate the optimal static policy, it is sufficient to maintain Y_m or $\tilde{Y}_m$ at a non-zero level for the majority of the time. Next, we present the central theorem of this section that establishes that the closeness between Priority-K and the optimal static routing policy decreases exponentially with K. The proof is done by showing that $\mathbb{P}_\pi[Y_m(t) \geq B]$ is close to 1 when $\mu > C_3\lambda$ and $\mathbb{P}_\pi[\tilde{Y}_m(t) \geq B]$ is close to 1 when $\mu < C_3\lambda$, through constructing suitable potential functions of the virtual queues and establishing concentration bounds of the potential functions. The proof of the theorem is omitted due to space limitations.

Theorem 2. *Assume $\mu \neq C_3\lambda$. For $K \geq 2MB$, there exist positive constants ϵ_0, M_0, N_0 that only depend on λ, B, μ, C_3, such that*

$$\forall m,\ |a_m^* - \mathbb{E}_\pi[a_m(t)]| \leq M_0 \cdot \left(\frac{B}{B+\epsilon_0}\right)^{K/N_0}$$

Theorem 2 establishes that the steady state distribution of the Priority-K policy can get arbitrarily close to the optimal static routing policy by picking a large enough K. Although the distance of the steady-state distribution to the optimal decreases with K, the convergence time to the steady-state distribution generally increases with K. In simulations, we observe that Priority-K with K being about 10 times λ (or B) can already achieve similar performance as the optimal static routing policy.

It is worth mentioning that the objective of approximating the optimal static routing policy in stochastic stream learning systems can be equivalently put into the context of stochastic queueing networks, where the goal is to maintain the data rate received by the network nodes at a certain pre-defined level (corresponding to a_m^* in the optimal static routing), with the level depending on unknown network statistics (the relationship between $C_3\lambda$ and μ). Such routing objective is of different nature than traditional objectives such as load-balancing or throughput maximization, and we are not aware of any traditional routing policies that can be used to achieve the objective. Therefore, to our best knowledge, our Priority-K policy is the first that solves this new kind of routing problems in stochastic networks, which may appear in other scenarios than performance optimization of stochastic stream learning systems.

VI. Simulations

In this section, we empirically validate our theoretical results. Specifically, we focus on evaluating the performance of the optimal static routing policy and Priority-K using the optimization error of the stream learning system, and measuring the closeness of Priority-K and the optimal static routing policy. We base our simulations on the classification task of Support Vector Machine (SVM) with a real data set from [34]. The data set contains $n = 690$ data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where each $x \in \mathbb{R}^N$ ($N = 14$) is a feature vector of the data and $y \in \{-1, 1\}$ is the label. The goal is to compute vector $w \in \mathbb{R}^N$ and $b \in \mathbb{R}$ that minimizes the loss

$$\frac{1}{n}\sum_{i=1}^{n} \max[0, 1 - y_i(w_i^{\mathrm{T}} x_i - b)].$$

The rationale behind the loss function $\max[0, 1-y_i(w_i^{\mathrm{T}} x_i - b)]$ is that it is small when $(w_i^{\mathrm{T}} x_i - b)$ has large absolute value and has the same sign as y_i, and it is large otherwise. Through minimizing the loss function, we seek for parameters w and b that predicts the correct label with high confidence (large absolute value of $(w_i^{\mathrm{T}} x_i - b)$).

Projecting into our framework, each data point is a feature-label (x, y) pair, and the optimization variable is (w, b). The objective function is equal to $\mathbb{E}_{(x,y)\sim\mathcal{D}}[\max[0, 1 - y_i(w_i^{\mathrm{T}} x_i - b)]]$, where $\mathcal{D}$ is the empirical distribution defined by the data set. Note that we choose the empirical distribution as $\mathcal{D}$ so that we know $\mathcal{D}$ and can thus generate data points and evaluate the optimization error based on it. The optimization algorithm (local SGD) of stream learning does not have access to $\mathcal{D}$. We consider a stream learning system with one source and $M = 15$ computation nodes. The source generates data points (which is simulated by sampling from the empirical distribution) at a rate of $\lambda = 150$ per time slot. For the deterministic system, the source generates 150 data points every time slot, while for stochastic routing, the number of data points generated by the source is a uniform random integer in $[100, 200]$. The processing rate of each node is $\mu = 30$.

A. Effect of Routing Policy

We first investigate how the routing policy influences the evolution of the optimization error with respect to the wall-

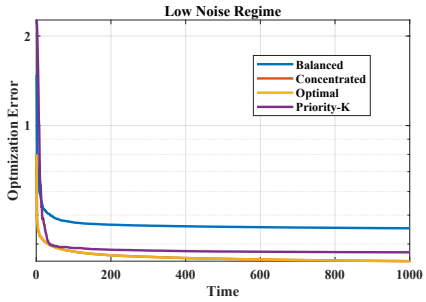

Fig. 3. Evolution of the optimization error with time in the low noise regime.

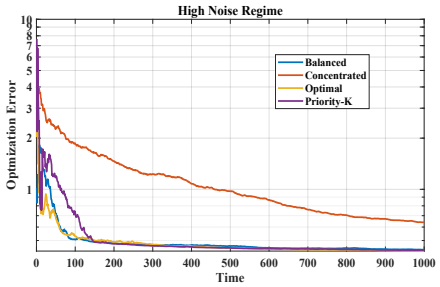

Fig. 4. Evolution of the optimization error with time in the high noise regime.

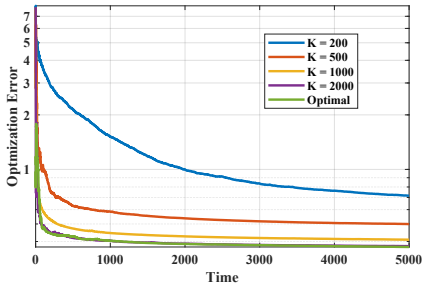

Fig. 5. Performance of Priority-K with different values of K.

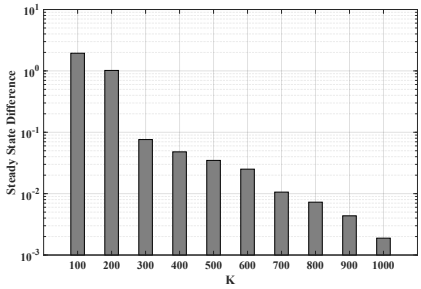

Fig. 6. Steady state rate difference of Priority-K with different values of K.

clock time. We consider four different routing policies. The first three are static policies in the deterministic system, and the last one works in the stochastic system:

- Balanced: each node receives 10 data points for each slot, i.e., $a_m = 10$ for all m.
- Concentrated: five nodes receives 30 data points for each slot, i.e., $a_m = 30$ for $m = 1, \ldots, 5$.
- Optimal: the optimal static routing policy defined in Definition 2, with the parameters C_3 estimated from the data set.
- Priority-K: the stochastic routing policy with the same C_3 as the optimal static routing. $K = 2000$

We run the routing policies together with local SGD in two regimes: a low noise regime where the noise of the gradients only comes from the data generation process, and a high noise regime where we add artificial noise (a uniform random variable in $[-10, 10]$) to the gradient evaluated at each data point. The results are plotted in Figures 3 and 4. Note that the y-axis (optimization error) is in log-scale.

From Figure 3, we can see that in the low noise regime, concentrated routing has better performance (in terms of optimization error) than balanced routing and the optimal static policy actually coincides with concentrated routing ($a^* = 30$). On the other hand, in the high noise regime (Figure 4), balanced routing outperforms concentrated routing, and is close to the optimal static policy. This validates the discussion on the bias-variance tradeoff in Section III, which indicates that when the noise level is low, increasing the number updates (reducing the bias) outweighs improving the accuracy of the updates (reducing the variance) for local SGD, and the contrary holds when the noise level is high. Finally, we can also see that the performance of Priority-K is close to the optimal static policy in both regimes.

B. Robustness of Priority-K

We proceed to evaluate the sensitivity of the performance of Priority-K with respect to the value of the parameter K. We first run local-SGD in the high noise regime with Priority-K routing for $K = \{200, 500, 1000, 2000\}$ and plot the results in Figure 5. We can see that, in the range of $\{200, 500, 1000, 2000\}$, the performance of Priority-K improves as the value of K increases, and setting $K = 1000$ can already closely match the performance of the optimal static policy.

We then compute the steady state rate difference of Priority-K with the optimal static routing policy, i.e., the value $\sum_{m=1}^{M} |a_m^* - \mathbb{E}_\pi a_m(t)|$, for $K = \{100, 200, \ldots, 1000\}$ and plot the results in Figure 6. From the figure, we can see the rate difference roughly decreases exponentially with K, matching the theoretical results in Section V.

VII. Conclusion

In this paper, we studied the optimal routing problem in stream learning systems. We first proposed an framework for optimizing routing policy for stream learning through deriving an upper bound on the optimization error as a function of the routing policy. By minimizing the upper bound, we proposed an optimal static routing policy for stream learning systems with deterministic data generating process. We then designed a stochastic routing policy that can approximate the optimal static routing policy arbitrarily closely for systems where the data points are generated according to a stochastic process with unknown rate. Our stochastic routing policy can also been seen as a solution to a new class of routing problems in stochastic queueing networks where the goal is to maintain the data rate to nodes at a target level.

REFERENCES

[1] H. M. Gomes, J. Read, A. Bifet, J. P. Barddal, and J. Gama, "Machine learning for streaming data: state of the art, challenges, and opportunities." in *ACM SIGKDD Explorations Newsletter*, vol. 21, no. 2, pp. 6-22, 2019.

[2] T. Fu, J. Ding, R. Ma, M. Winslett, Y. Yang, and Z. Zhang, "DRS: Auto-scaling for real-time stream analytics." in *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3338-3352, 2017.

[3] C. Gutterman, K. Guo, S. Arora, T. Gilliland, X. Wang, L. Wu, E. Katz-Bassett, and G. Zussman, "Requet: Real-time QoE metric detection for encrypted YouTube traffic." in *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 16, no. 2, pp. 1-28, 2020.

[4] M. Soysal, E. G. Schmidt, "Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison." in *Performance Evaluation*, vol. 67, no. 6, pp. 451-467, 2010.

[5] T. Wagner, S. Guha, S. Kasiviswanathan, and N. Mishra, "Semi-supervised learning on data streams via temporal label propagation." in *International Conference on Machine Learning*, pp. 5095-5104, 2018.

[6] R. Gonzalez, C. Soriente, and N. Laoutaris, "User profiling in the time of https." in *Proceedings of the 2016 Internet Measurement Conference*, pp. 373-379, 2016.

[7] H He, S. Chen, K. Li, and X. Xu, "Incremental learning from stream data." in *IEEE Transactions on Neural Networks*, vol. 22, no. 12, pp. 1901-1914, 2011.

[8] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, "Twitter heron: Stream processing at scale." in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp: 239-250, 2015.

[9] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform." in *IEEE International Conference on Data Mining Workshops*, pp. 170-177, 2010.

[10] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine." in *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[11] P. Rai, H. Daumé III, and S. Venkatasubramanian. "Streamed learning: one-pass SVMs." arXiv preprint arXiv:0908.0572 (2009).

[12] R. Frostig, R. Ge, S. M. Kakade, and A. Sidford, "Competing with the empirical risk minimizer in a single pass." in *Conference on learning theory*, pp. 728-763, 2015.

[13] P. Jain, S. M. Kakade, R. Kidambi, P. Netrapalli, and A. Sidford, "Parallelizing stochastic gradient descent for least squares regression: mini-batching, averaging, and model misspecification." in *Journal of Machine Learning Research*, 2018.

[14] H. Daneshmand, A. Lucchi, and T. Hofmann, "Starting small-learning with adaptive sample sizes." in *International conference on machine learning*, pp. 1463-1471, 2016.

[15] A. S Berahas, R. Bollapragada, N. Keskar, and E. Wei, "Balancing communication and computation in distributed optimization." in *IEEE Transactions on Automatic Control*, vol. 64, no. 8, pp. 3141-3155, 2018.

[16] M. M. Amiri,and D. Gündüz, "Computation scheduling for distributed machine learning with straggling workers." in *IEEE Transactions on Signal Processing*, vol. 67, no. 24, pp. 6270-6284, 2019.

[17] B. Woodworth, B. Bullins, Ohad Shamir, and N. Srebro, "The Min-Max Complexity of Distributed Stochastic Convex Optimization with Intermittent Communication." arXiv preprint arXiv:2102.01583, 2021.

[18] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems." in *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205-1221, 2019.

[19] M. Chen, H. V. Poor, W. Saad, and S. Cui, "Convergence time optimization for federated learning over wireless networks." in *IEEE Transactions on Wireless Communications*, 2020.

[20] H. Yang, Z. Liu, T. Quek, and H. V. Poor, "Scheduling policies for federated learning in wireless networks." in *IEEE transactions on communications*, vol. 68, no. 1, pp. 317-333, 2019.

[21] N. H. Tran, W. Bao, A. Zomaya, M. Nguyen, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis." in *IEEE INFOCOM*, pp. 1387-1395, 2019.

[22] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li, "Parallelized Stochastic Gradient Descent." in *NeurIPS*, vol. 4, no. 1, pp. 4, 2010.

[23] A. Rakhlin, O. Shamir, and K. Sridharan, "Making gradient descent optimal for strongly convex stochastic optimization." in *International Conference on Machine Learning*, pp. 1571-1578, 2012.

[24] O. Shamir, and T. Zhang, 'Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes." in *International conference on machine learning*, pp. 71-79, 2013.

[25] S. Stich, "Local SGD Converges Fast and Communicates Little." in *International Conference on Learning Representations*, 2018.

[26] T. Lin, S. Stich, K. Patel, and M. Jaggi, "Don't Use Large Mini-batches, Use Local SGD." in *International Conference on Learning Representations*, 2019.

[27] B. Woodworth, K. Patel, S. Stich, Z. Dai, B. Bullins, B. Mcmahan, O. Shamir, and N. Srebro, "Is local SGD better than minibatch SGD?." in *International Conference on Machine Learning*, pp. 10334-10343, 2020.

[28] J. Zhang, A. Sinha, J. Llorca, A. Tulino, and E. Modiano. "Optimal control of distributed computing networks with mixed-cast traffic flows." in *IEEE INFOCOM*, pp. 1880-1888, 2018.

[29] H. Feng, J. Llorca, A. Tulino, and A. F. Molisch, "Optimal dynamic cloud network control." in *IEEE/ACM Transactions on Networking*, vol. 26, no. 5, pp. 2118-2131, 2018.

[30] V. Valls, G. Iosifidis, G. de Mel, and L. Tassiulas, "Online Network Flow Optimization for Multi-Grade Service Chains." in *IEEE INFOCOM*, pp. 1329-1338, 2020.

[31] C-S. Yang, R. Pedarsani, and A. Avestimehr. "Communication-aware scheduling of serial tasks for dispersed computing." in *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1330-1343, 2019.

[32] A. Nemirovsky and D. Yudin, "Problem complexity and method efficiency in optimization." 1983

[33] M. Harchol-Balter, "Performance modeling and design of computer systems: queueing theory in action." Cambridge University Press, 2013.

[34] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html