FedSLO: Towards SLO Guarantee for Federated Computing

Hao Che, Todd Rosenkrantz, Xiaoyan Shen, Hong Jiang, Zhijun Wang
UT Arlington
{hche, todd.rosenkrantz, xiaoyan.shen, hong.jiang, zhijun.wang}@uta.edu

***Abstract*—Federated computing, including federated learning and federated analytics, needs to meet certain task Service Level Objective (SLO) in terms of various performance metrics, e.g., mean task response time and task tail latency. The lack of control and access to client activities requires a carefully crafted client selection process for each round of task processing to meet a designated task SLO. To achieve this, one must be able to predict task performance metrics for a given client selection per round of task execution. In this paper, we develop, FedSLO , a general framework that allows task performance in terms of a wide range of performance metrics of practical interest to be predicted for synchronous federated computing systems, in line with the Google federated learning system architecture. Specifically, with each task performance metric expressed as a cost function of the task response time, a relationship between the task performance measure – the mean cost and task/subtask response time distributions is established, allowing for unified task performance prediction algorithms to be developed. Practical issues concerning the computational complexity, measurement cost and implementation of FedSLO are also addressed. Finally, we propose preliminary ideas on how to apply FedSLO to the client selection process to enable task SLO guarantee.**

I. Introduction

Federated learning (FL), originally proposed by Google engineers in 2016 [10], enables machine learning (ML) model training to use massively distributed data available from Internet-of-Things (IoT), mobile and edge devices, also known as clients. A model training task involves multiple training rounds. In each round, a central control unit, usually resided in a cloud, dispatches a number of subtasks of the task to different selected clients for local model training and the subtask results are sent back to be aggregated for global model update. Federated learning is data privacy preserving, meaning that in each round, it does not require clients to transmit their raw data, but rather the trained local model parameters, to the aggregator for global model update. This data privacy-preserving, distributed computing paradigm was proposed, again by Google engineers in 2020 [1], to be used for data-privacy-preserving data analytics as well, known as *federated analytics* (FA). Different from FL, a task for FA only consists of a single round, with clients computing the statistics from the local data and the server drawing the conclusion from the collective statistics. FL and FA are collectively known as *federated computing* (FC) [13].

This work was supported by the US NSF under Grants CCF SHF-2008835 and CCF-SHF2226117

FC tasks generally have to meet certain Service Level Objective (SLO), be it implicitly or explicitly. For example, an FL model training task may need to finish training between, midnight and 5am when a sufficient number of clients in the client population are eligible for training, called the eligible client population in this paper, i.e., they are idle, being charged and connected to the Internet via an unmetered network, such as a WiFi network [4]. With a given estimated number of rounds needed to meet a target model loss or accuracy, an upper bound of the mean task response time per round can be estimated, serving as the task SLO per round to guard against the possible training incompletion for the day. As another example, FA-based mobile crowdsensing [17], such as temperature, air quality, or water level sensing and alerting in a city, may call for a stringent task tail-latency SLO, e.g., the 99th-percentile of task response time of 60 seconds.

The task scheduler in an FC system generally has neither control nor access to the client activities [4] (Under certain circumstances, cross-silo FC may be an exception, as will be discussed in Section III). As such, *FC has to rely on a well-crafted client selection process that can identify a "right" subset of clients from the eligible client population to meet a given task SLO*. This is challenging given (1) the lack of efficient algorithms that can predict whether a given selection can meet the task SLO or not; and (2) the high combinatorial complexity of the selection process, e.g., selecting 500 clients out of one million eligible clients. Consequently, current practice mainly relies on random selection without task SLO guarantee [4] and the existing research works on client selection almost exclusively focus on how to strike a balance between task response time, also known as wall-clock time [9], and training quality, without task SLO guarantee [5], [9]. Although a client selection protocol for FL [11] does enforce a task response time target per round, the target is a deterministic, rather than statistic or probabilistic one. Given the randomness and stochastic nature of the task processing and communication processes, it is clear that setting a deterministic response time target for a task may lead to either the worst-case resource allocation, i.e., resource overprovisioning, or target miss. This explains why typical SLOs for distributed computing are expressed in terms of statistic or probabilistic metrics, as evidenced by the wide adoption of job tail latency (a probabilistic metric) SLOs for user-facing workloads and mean job response time (a statistic metric) SLOs for batch workloads [6], [15] for datacenter applications.

As a first step towards providing task SLO guarantee for FC applications, in this paper, we develop, FedSLO, a general framework for task performance prediction for synchronous FC systems, in line with the Google federated learning system architecture. In FedSLO, with each performance metric expressed as a cost function of the task response time, a relationship between the task performance measure – the mean cost, and task/subtask response time distributions is established, allowing for unified task performance prediction algorithms to be developed for a wide range of task performance metrics of practical interest. In FedSLO, practical issues concerning the estimation of the mean cost, i.e., computational complexity, measurement cost and implementation, are also addressed. Finally, we discuss preliminary ideas on how to apply FedSLO to the client selection process to enable task SLO guarantee.

II. FEDSLO

A. System Model

While FA is synchronous by design, meaning that with a single round, a task is not complete until the results from all the subtasks are aggregated, FL may be synchronous or asynchronous[1] [2]. Since data privacy enhancement mechanisms for FL, such as Security Aggregation and differentiated privacy, require some sort of synchronization per training round, the most popular implementation of FL is synchronous by design, including the Google FL production system [4]. Hence, for FedSLO, we only consider synchronous FC, where a round is incomplete until either all the subtask results are aggregated or a timeout occurs. We further assume that for an FL task, the task SLO is already partitioned into per-round task SLOs for individual rounds. These two assumptions allow us to focus on a single round of task processing that may involve the following processing phases:

1) Central control unit selects participating clients in the eligible client population;
2) Central control unit dispatches subtasks of the task to the selected clients;
3) Clients process the subtasks and send the results to central control unit;
4) Central control unit aggregates the results.

we further assume that (a) only the task delays in phases 2) and 3) contribute to the task response time, i.e., from the instant the subtasks are dispatched to the selected clients to the instant when either the results of the slowest subtask reach the control unit or a timeout occurs, and the task SLO only applies to these two phases; and (b) the task processing delays due to phases 1) and 4) can be estimated and their impact on task performance can be accounted for separately. There are two main reasons for making this assumption. First, the task processing delays in phases 1) and 4) are much easier to predict than those in phases 2) and 3). This is because the selection and aggregation processes are typically carried out with dedicated computing resources, whereas phases 2) and 3) typically involve subtask processing in a large number of heterogeneous clients in parallel and incur highly variable communication delays for subtask dispatching and returning of subtask results. Second, the task delays in phases 1) and 4) may need to be estimated on a case-by-case basis. For example, for FL, depending on how one implements it, phase 1) may or may not overlap with the previous round [4] and thus may not contribute to the task response time except for the first round, and for FA, phase 1) may not always exist, as the task may involve the entire client population. As another example, for aggregation in terms of statistic measures, such as mean, it may be done incrementally, overlapping with the subtask results arrival process. In this case, the task delay in phase 4) may be negligible.

[1]Note that in this paper, we don't consider FL based on a fully distributed, peer-to-peer paradigm [7].

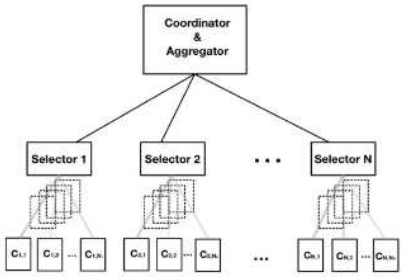

Fig. 1: FedSLO system model

In the FedSLO system model, the central control unit is composed of a coordinator and aggregator (CA) in the cloud, and N client selectors with each covering a subset of the client population, which may be collocated with CA or placed close to the clients they cover, as shown in Fig. 1. It is a two-tier star network with a CA-to-selector star network at the upper tier and N selector-to-client star networks at the lower tier. CA communicates with N selectors and the jth selector ($j = 1, ..., N$), in turn, communicates with a subset of the client population and selects N_j clients from the subset (note that N_j may differ from one round to another) on behalf of CA in phase 1). Hence, the total number of clients selected, or equivalently, the total number of subtasks of the task for the round, $N_s = \sum_{j=1}^{N} N_j$, also known as *task fanout*. A client is a selected one if it is connected with the selector by a solid line, otherwise, it is not selected (a dotted box). Without loss of generality, we label the selected ones as, $c_{j,k}$, for $k = 1, ..., N_j$ and $j = 1, ..., N$. In phases 2)-3), the selectors serve as the proxies to relay the communication and data exchanges between CA and selected clients. A round is not complete until phase 3) finishes.

The above FedSLO system model closely follows the Google FL production system architecture [4]. Specifically, a selector in this model plays the same role as a selector in the Google FL system. The CA in this model corresponds to the coordinator, together with the master aggregator and aggregators it spawns for a given FL training model in the Google FL system, which collectively enable rounds in lockstep and scalable aggregation.

B. Problem Statement

In FedSLO, the problem is formulated as finding the average cost, $\bar{C}$, for a cost function, $c(t)$, of task response time, t, for a given round. Mathematically, we have,

$$\bar{C} = \int_0^\infty c(t) \frac{dF(t)}{dt} dt \tag{1}$$

where $F(t)$ is the cumulative density function (CDF) of the task response time t for the round.

The above problem statement is a general one, covering a wide range of task performance metrics of practical interest. With a given performance metric or cost function $c(t)$, be it continuous, piecewise continuous, or single-valued, the corresponding task performance measure or mean cost, $\bar{C}$, can be estimated by Eq. (1). The following are three examples.

1. For the task performance metric in terms of task response time, a continuous cost function,

$$c(t) = t, \tag{2}$$

we have $\bar{C}$ = mean task response time.

2. For the task performance metric in terms of customer disengagement rate or churn rate [8], e.g., expressed as a piecewise continuous step cost function,

$$c(t) = \begin{cases} 0 & \text{if} \quad t \leq 100ms \\ 0.01 & \text{if} \quad 100ms < t \leq 500ms \\ 0.05 & \text{if} \quad 500ms < t \leq 1,000ms \\ 0.1 & \text{if} \quad 1,000ms < t, \end{cases} \tag{3}$$

i.e., the probability a customer may quit the service when the task response time falls in a given range, $\bar{C}$ = mean customer churn rate or the percentage of customers who may quit the service.

3. For the task performance metric in terms of task tail latency, a single-valued step cost function,

$$c(t) = \begin{cases} 0 & \text{if} \quad t \leq t_t \\ 1 & \text{if} \quad t > t_t, \end{cases} \tag{4}$$

we have $\bar{C} = 1 - F(t_t)$, i.e., the probability that the task response time exceeds the task tail latency, t_t,

Once a given task performance measure, $\bar{C}$, is estimated for a given selection of N_s clients, it can then be compared against the corresponding task SLO in terms of a cost target C^{SLO} to determine whether the selection meets the task SLO or not. For a task tail-latency SLO expressed in terms of the pth-percentile task latency of t_p, or equivalently, a target probability, $C^{SLO} = 1 - p/100$, for task response time to exceed t_p, $\bar{C} = 1 - F(t_t)$ at $t_t = t_p$ must be no greater than C^{SLO} in order to meet the task tail-latency SLO, or

$$F(t_p) \geq p/100. \tag{5}$$

For the other two types of task performance metrics, i.e., task response time and customer churn rate, the corresponding C^{SLO}'s are expressed in terms of a target mean task response time and a target mean churn rate, respectively. Again, the task SLO in terms of C^{SLO} is met, if $\bar{C} \leq C^{SLO}$.

The above problem statement and examples indicate that for a wide range of task performance metrics of practical interest, the problem boils down to how to estimate $F(t)$ for a given client selection in a round. A naive solution is to estimate $F(t)$ by constructing its histogram using historical training samples from different rounds of a training model in FL or samples from different tasks (also known as queries) in FA. This approach only works when the client selection algorithm for different rounds in FL or different tasks in FA selects exactly the same number of clients randomly from the eligible client population, so that $F(t)$ thus estimated can be applied to any round in FL or any task in FA. Unfortunately, this assumption does not hold true in general, as different rounds in FL and different tasks in FA may have different task fanouts, thus having different $F(t)$'s (as we shall see shortly, $F(t)$, is a function of task fanout). Moreover, non-random client selection algorithms (e.g., [5], [9]) in FL may be used to enhance the training performance, which generally leads to very different statistic behaviors for different rounds, even when the task fanout stays the same and hence, cannot be adequately characterized using the same $F(t)$. Instead, we consider the following solution.

Since the slowest subtask determines the task response time and subtasks are dispatched to and processed at the selected clients in parallel, the well-known order statistic [3] applies [4] and we have,

$$F(t) = \prod_{j=1}^{N} \prod_{k=1}^{N_j} F_{j,k}(t), \tag{6}$$

where $F_{j,k}(t)$ is the CDF of the subtask response time for subtasks corresponding to client $c_{j,k}$. Instead of attempting to estimate $F(t)$, one may estimate $F_{j,k}(t)$ for $k = 1, ..., N_j$ and $j = 1, ..., N$. Note that subtask response times for different subtasks of different rounds for a given FL training model or different tasks for a given FA application at the same client, $c_{j,k}$, are expected to share similar statistic behaviors and hence can be characterized by the same CDF, $F_{j,k}(t)$. Therefore, $F_{j,k}(t)$'s may be estimated based on the samples collected from different rounds for FL or different queries for FA. This expression also shows that $F(t)$ indeed is a function of task fanout in general.

To avoid excessively long or unbounded task response time due to subtask stragglers or unresponsive subtasks, two common practices are: (a) redundant subtask issues, i.e., issue N_s subtasks and aggregate the results from the first r subtasks with the rest $N_s - r$ discarded; and (b) Subtask timeout, i.e., subtask results that arrive at the aggregator after a given timeout, T_D, will not be included in the aggregation [4]. To account for the two common practices, we use a single index i to identify a selected client, i.e., use $G_i(t)$ ($i = 1, 2, ..., N_s$) to replace $F_{j,k}(t)$, e.g., $G_1(t) = F_{1,1}(t)$, $G_2(t) = F_{1,2}(t)$ ..., $G_{N_1+1}(t) = F_{2,1}(t)$, $G_{N_1+2}(t) = F_{2,2}(t)$ ..., and so on. Then to account for common practice (a), i.e., $F(t)$ represents the CDF when at least r subtasks finish within t, we have the

TABLE I: Prediction error versus number of samples M

M	2	5	10	20	31	75
$Error$	51%	20%	10%	5%	3%	1%

following [14],

$$F(t) = \sum_{i=r}^{N_s} \sum_{\{l_1,l_2\} \in P_i} \prod_{k=1}^{i} G_{l_{1k}}(t) \prod_{k=1}^{N_s - r} [1 - G_{l_{2k}}(t)], \quad (7)$$

where P_i is the set of all two-set partitions D, E of $\{1, 2, ..., N_s\}$ with $|D| = i$ and $|E| = N_s - i$, and l_{hk} is the kth selected client of the client vector l_h for $h = 1, 2$ corresponding to D and E, respectively. Clearly, Eq. (6) is a special case of Eq. (7) at $r = N_s$.

To also account for common practice (b), what we need to know is $F_D(t) = F(t|t \leq T_D)$, i.e., the CDF of the task response time, given that the task finishes before T_D, or

$$F_D(t) = \begin{cases} \frac{F(t)}{F(T_D)} & \text{if} \quad t \leq T_D \\ 0 & \text{if} \quad t > T_D. \end{cases} \quad (8)$$

In summary, in general, $F(t)$ in Eq. (1) should be replaced by $F_D(t)$ and $F(t)$ in Eq. (8) is given by Eq. (7). Assuming $G_i(t)$'s, or equivalently, $F_{j,k}(t)$'s, are known, the next question is how to estimate $\bar{C}$ at low sampling and computational costs for different task performance metrics.

C. Cost Analysis

We first convert Eq. (1) approximately into the following discretized form,

$$\bar{C} \approx \sum_{i=1}^{M} [F_D(t_i) - F_D(t_{i=1})] c(\frac{t_i + t_{i-1}}{2}), \quad (9)$$

where M is the number of sampled $F_D(t)$'s taken in between t_0 and T_D, not including $t_M = T_D$ as $F_D(T_D) = 1$, and t_0 is the largest t below which $F(t) = 0$. In general, as M grows larger, the approximation becomes more accurate.

Sampling Cost: For the task tail latency SLO with a single-valued cost function, only one sample of $F_D(t)$ at $t = t_p$ is needed to predict $\bar{C}$, since $\bar{C} = 1 - F_D(t_p)$, or one sample of $G_i(t)$ at $t = t_p$ for $i = 1, ..., N_s$, according to Eq. (7). For the churn rate with a piecewise continuous cost function given by Eq. (3), only three samples of $G_i(t)$ are needed to predict $\bar{C}$, i.e., at $t_1 = 100ms$, $t_2 = 500ms$ and $t_3 = 1,000ms$.

In contrast, for the mean task SLO corresponding to a continuous cost function in Eq. (2), the prediction error for $\bar{C}$ reduces as the number of samples M increases. To get a rough idea how many samples should be taken, Table I lists the prediction errors for the mean task response time at different M (evenly sampled) for an FL task CDF given in Fig. 2 (see Section III for detailed description). As one can see, the prediction error reduces quickly initially and then levels off as M increases. Since the mean task performance metric is normally applied to non-realtime applications, a 10% prediction error should be tolerable, which translates into about $M = 10$ samples.

Computational Cost: Besides the sampling cost, the other major potential cost is the combinatorial computing cost for the evaluation of Eq. (7) per sampling point $t = t_i$ $(i = 1, ..., M)$. This is because the sum in Eq. (7) involves $\sum_{i=r}^{i=N_s} \frac{N_s!}{(N_s-i)!i!}$ terms, which grows much faster than exponential as N_s and $N_s - r$ increase. For example, consider 1% redundant subtask issues for an FL training round. At $N_s = 100$ and hence, $N_s - r = 1$, we only have 101 terms to be added per sampling point. In contrast, at $N_s = 200$ and $N_s - r = 2$, it grows to 20,001 terms, and at $N_s = 500$ and $N_s - r = 5$, it grows to 2.6x10^{11} terms! To mitigate combinatorial cost, we propose two possible approximate solutions.

Divide-and-conquer solution: Divide N_s selected clients with a given percentage of redundant subtask issues into a number of subsets of clients of equal size. Then evaluate the CDF for each subset by Eq. (7). Finally, take the products of the CDFs of all subsets as $F(t)$. It can be shown that the estimated $F(t)$ is smaller than that estimated directly, and serves as a conservative estimation of $F(t)$ and a conservative estimation of the task performance measure, $\bar{C}$.

Using the above example to illustrate the idea, instead of directly calculating the sum of 20,001 terms for the case where $N_s = 200$ and $N_s - r = 2$, one may split $N_s = 200$ clients into two subsets with $N_s = 100$ and $N_s - r = 1$ each and calculate the sum of 101 terms for each and then take the product of the two to be $F(t)$ for $N_s = 200$ and $N_s - r = 2$, reducing the total number of terms from 20,001 to only 202. By the same token, by splitting $N_s = 500$ clients into 5 subsets and following the same procedure, the total number of terms for $N_s = 500$ and $N_s - r = 5$ is reduced from 2.6x10^{11} to only 505.

Least-cost solution: Given that timeout, T_D, sets an upper limit as to how much subtask stragglers may negatively impact the task performance, the primary role of redundant subtask issues is to ensure with a high probability, at least r subtasks will successfully finish before timeout. This is important because a round of FL task or an FA task may fail if the number of successfully completed subtasks before timeout drops below a certain threshold [4]. Understanding this, for a case where N_s is large or the cost is high, one may consider using Eq. (6), instead of Eq. (7), with N_s replaced by r to estimate $F(t)$, which involves only one term, incurring the lowest possible cost while still allowing $N_s - r$ redundant subtask issues, which may be adjusted based on measurement, to ensure that with a high probability, r subtasks will indeed finish before timeout. $\bar{C}$ thus estimated is again conservative, i.e., higher than the actual one, as the possible deduction of task response time due to redundant subtask issues are not accounted for in the estimation of $F(t)$.

Both solutions are conservative ones and hence, can lead to task SLO guarantee. Nevertheless, their potential impact on the possible client overprovisioning (i.e., using a larger $N_s - r$ than actually needed) will be carefully studied as part of our future work.

D. Subtask CDF Estimation

The last point to address for FedSLO is how to estimate subtask response time CDFs, $G_i(t)$'s or $F_{j,k}(t)$'s for the entire client population. We propose a combined initial offline estimation and continuous online updating approach.

Offline Estimation: Before running an FC application in a production system, it normally has to go through an initial testing phase involving modeling, simulation and small scale system testing, using sample test data or other proxy data as inputs [4]. We propose to leverage this phase for the offline estimation of CDFs. The offline estimation may be performed on emulated or real devices that emulate the clients typically seen in the production system. Then different CDFs thus estimated are assigned to the devices in the production system as their initial CDFs based on the device types and models. In this phase, parameterized CDF models may also be developed to fit the measured CDFs for the FC application to further reduce the memory footprints of the CDFs. In what follows, we present some preliminary testing results for model fitting.

We experiment with two applications, i.e., FL model training for FedML's FedIoT: Anomaly Detection for Cybersecurity FL model training [16] and FA for temperature sensing. The experiments are performed in a testbed with Raspberry Pi 2's, Pi 3's and Pi 4's with temperature sensing capability serving as clients and a remote laptop running Ubuntu 22.04 as colocated CA and selectors. For both applications, we set $N_s = 5$. In particular, to test the impact of both system and statistic heterogeneities on the model fitting accuracy, for the FL application, we use a mixture of Pi models with one or two cores as clients and for the FA application, we use Pi 4 for all 5 clients with varied data sets. The subtask response time samples are collected for each FL round and different queries for temperature means of the temperature readings taken over different randomly selected periods of times. Then the CDF histograms are constructed using the collected samples as input.

We find that the following two different normalized (i.e., scale to the range of $[0, 1]$), two-parameter functions, both are variations of the logistic function [12], actually match the subtask CDFs for the two applications pretty accurately. They are: For FL application,

$$G_i(t) = \begin{cases} \frac{1+e^{\alpha_i t_{0,i}}}{1+e^{-\alpha_i t}} - \frac{1+e^{\alpha_i t_{0,i}}}{1+e^{-\alpha_i t_{0,i}}} & \text{if} \quad t \geq t_{0,i} \\ 0 & \text{if} \quad t < t_{0,i}, \end{cases} \tag{10}$$

where α_i and $t_{0,i}$ are the two parameters to be fixed, capturing the steepness of the CDF curve and the smallest possible subtask response time at client i, respectively. For the temperature sensing application,

$$G_i(t) = \frac{1 + e^{-\beta_i t_{c,i}}}{1 + e^{-\beta_i (t - t_{c,i})}} - e^{-\beta_i t_{c,i}} \tag{11}$$

where β_i and $t_{c,i}$ are also two parameters to be fixed, capturing the steepness of the CDF curve and the average of the subtask response time, respectively. Figs. 2 and 3 give the matched subtask CDFs using a least-mean-square algorithm, along with task response time CDF (i.e., $F(t)$ model), calculated by Eq. (6), using the matched subtask CDFs as input, together with the measured task CDF, $F(t)$. It turns out that the one calculated from the model matches the measured one very well, within 7% error for the FL application and 3% for the FA application. One also notes that both system and statistic heterogeneities result in the dominance of the slowest client in determining the task response time. This implies that out of a potentially large number of selected clients, N_s, it is likely that only a small number of clients will effectively determine the task response time distribution in practice. This observation, if further confirmed, may open up opportunities for the design of low complexity algorithms for the evaluation of Eqs. (6) and (7).

The above results also suggest that the CDF corresponding to a given client i may be represented by only a few parameters, such as $\{\alpha_i, t_{0,i}\}$ or $\{\beta_i, t_{c,i}\}$, called the fingerprint of client i in this paper. To avoid having to maintain a large array of histogram data for each client CDF, it is possible to only maintain a fingerprint per client for the client population. With the client population potentially reaching billions [4], this means potentially huge savings of memory/storage spaces to accommodate the CDFs for the client population. Moreover, by grouping the clients based on the proximity of their fingerprints, effective client selection algorithms may be developed, as discussed briefly in Section 3.

Online Updating: At the beginning of the production phase, based on the type and model of the device for each client in the client population, an initial CDF with a given fingerprint or histogram is assigned to aid the early client selection processes. Then the CDF for a client is updated over time as the corresponding subtask response time samples are accumulated, making the subsequent selection processes more and more accurate.

The online updating process can be easily incorporated in a production system with low cost. Using the Google FL production system [4] as an example, the subtask response time samples can be easily extracted from the logs in the analytics layer in a CA, which "logs an event in every state of a training round and uses these logs to generate ASCII visualizations of the sequence of state transitions happening across all devices" [4]. With the samples extracted over time, the CDFs for all the clients in the client population can then be updated on a continuous basis with low measurement cost.

III. On Client Selection Process

Here are some preliminary ideas on how to apply FedSLO to enable task-SLO-guaranteed client selection. The problem can be stated as follows: Maximizing the mean cost $\bar{C}$, among all possible client selections of N_s clients from a given Eligible client Population set (EP) of size $|EP|$, provided that $\bar{C}$ does not exceed C^{SLO}, or Mathematically,

$$Max_{N_s \in EP} \quad \bar{C} \tag{12}$$

Subject to:

$$\bar{C} \leq C^{SLO}. \tag{13}$$

Succinctly, the objective is to select the most costly, or equivalently, the slowest possible N_s clients including $N_s - r$

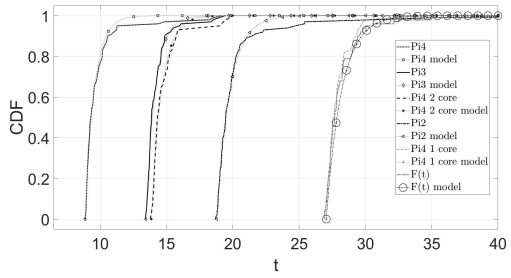

Fig. 2: Model fitting for FL application

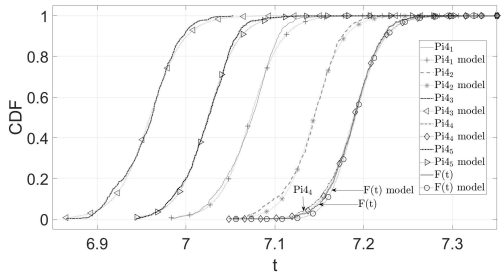

Fig. 3: Model fitting for FA application

redundant ones[2] from EP that meets the target, C^{SLO}. Doing so, leaves as many fast clients in EP for other FL round or FA tasks to meet their targets.

A key challenge for solving the above client selection problem is how to cope with the combinatorial computing complexity as the problem size in terms of $|EP|$ and N_s increases, as the total number of possible selections is $\frac{|EP|!}{(|EP|-N_s)!N_s!}$. In what follows, we first sketch a low-time-complexity algorithm that solves the above problem approximately in the case of C^{SLO} being a task tail-latency SLO, and then propose the ideas of a possible low-time-complexity heuristic solution to the above problem in general.

Client selection with task tail-latency SLO Guarantee: As explained earlier, for task tail-latency SLO, $\bar{C} = 1 - F_D(t_p)$ for any given client selection, where $F_D(t_p)$ is in turn determined by $G_i(t_p)$, for $i = 1, 2, ..., N_s$. The algorithm works as follows.

First, sample, $Q_k(t_p)$, the CDF at, t_p, for all eligible clients, k, for $k = 1, 2, ..., |EP|$, and then sort them in an ordered list. Without loss of generality, assume $Q_k(t_p)$'s are already ordered, meaning $Q_k(t_p) \leq Q_{k+1}(t_p)$, for $k = 1, 2, ..., |EP| - 1$. Then do the binary search of the client selections, with each selection being N_s consecutive $Q_k(t_p)$'s assigned as $G_i(t_p)$'s. It starts with assigning the middle N_s consecutive $Q_k(t_p)$'s and calculate $\bar{C}$. If $\bar{C} < C^{SLO}$, do the binary search in the first half of the ordered list, otherwise the second half, until finding the selection corresponding to the largest $\bar{C}$ smaller than or equal to C^{SLO}. The time complexity of this algorithm is $O(log_2(|EP| - N_s)$. For example, for $|EP| = 1,000,000$ and $N_s = 500$, the number of search steps is about 20, which is well manageable, e.g., 20x500 $= 10,000$ multiplications, plus 20 comparisons, when $\bar{C}$ is estimated using the least-cost solution proposed in Section II-C.

A General Solution: Finding a low-time-complexity solution to the problem given in Eq. (12) in general (i.e., for arbitrary cost functions, be it continuous or piecewise continuous) can be difficult, if not impossible. In this paper, we present some initial ideas about how a heuristic algorithm may solve the problem approximately at low time complexity.

The idea is to sample the subtask response time, $t_{k,q}$, at a given qth-percentile for eligible client, k, for $k = 1, 2, ..., |EP|$. For example, one may set, $q = 50$, capturing the center point of subtask response time distribution, $t_{k,50}$, for subtask, k, i.e., with $50 : 50$ chance the subtask response time will be below or above, $t_{k,50}$. Then sort $t_{k,q}$'s in a ordered list. Again, without loss of generality, assume that $t_{k,q}$'s are already in order with $t_{k,q} \geq t_{k+1,q}$ for $k = 1, 2, ..., |EP| - 1$. The proposed heuristic algorithm then works exactly the same way as the one for the task tail latency SLO above, by simply replacing $Q_k(t_p)$'s in the list with $t_{k,q}$'s. Of course, for each selection in the algorithm, $\bar{C}$ must be calculated using M samples in general. The time complexity of this algorithm is M times the time complexity of the previous algorithm, which again should be manageable, given that we only need $M = 10 - 30$ to attain reasonably high prediction accuracy of $\bar{C}$ (within $3 - 10\%$ prediction errors according to Table I).

Note that the above heuristic algorithm cannot guarantee that the client selection found maximizes the average cost in Eq. (12), as $\bar{C}$ for the selected clients corresponding to the largest N_s consecutive $t_{k,q}$'s, subject to $\bar{C} \leq C^{SLO}$, is not necessarily the largest among all possible selections. Nevertheless, the client selection obtained guarantees that $\bar{C} \leq C^{SLO}$, i.e., the selection will indeed lead to task SLO guarantee. A possibly better solution is to order the eligible clients based on their fingerprints, as they capture the entire CDFs, which will be investigated as part of our future work.

A Remark: The above did not consider the case when a feasible client selection that satisfies $\bar{C} \leq C^{SLO}$ cannot be found. When this happens, in general, there are two options: (a) to relax the task SLO by setting a larger C^{SLO} and then rerun the algorithm iteratively until at least one feasible selection becomes available; and (b) to add more resources to clients to reduce $\bar{C}$. For cross-device FC [7] where the central control unit or CA has no control over the clients' resources, option (a) is the only way to go, however, at the cost of reduced task performance and hence reduced customer satisfaction.

IV. Conclusions

In this paper, we develop, FedSLO, a general framework that allows task performance in terms of a wide range of performance metrics of practical interest to be predicted for synchronous federated computing systems. Practical issues concerning the computational complexity, measurement cost and implementation of FedSLO are also addressed. Finally, we propose preliminary ideas on how to apply FedSLO to the client selection process to enable task SLO guarantee.

[2]This is true because the cost is an increasing function of task response time.

REFERENCES

[1] Federated Analytics: Collaborative Data Science without Data Collection. https://research.google/blog/federated-analytics-collaborative-data-science-without-data-collection/. Accessed on 9/5/2024.

[2] P. Bellavista, L. Foschini, and A. Mora. Decentralised learning in federated deployment environments: A system-level survey. *ACM Comput. Surv.*, 54(1), feb 2021.

[3] G. Bolch, S. Greiner, H. de Meer, and K. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley-Interscience publication. Wiley, 2006.

[4] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage, and J. Roselander. Towards federated learning at scale: System design. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems*, volume 1, pages 374–388, 2019.

[5] Z. Chai, A. Ali, S. Zawad, S. Truex, A. Anwar, N. Baracaldo, Y. Zhou, H. Ludwig, F. Yan, and Y. Cheng. Tifl: A tier-based federated learning system. In *Proceedings of the 29th international symposium on high-performance parallel and distributed computing*, pages 125–136, 2020.

[6] C. Iorgulescu, R. Azimi, Y. Kwon, S. Elnikety, M. Syamala, V. Narasayya, H. Herodotou, P. Tomita, A. Chen, J. Zhang, et al. {PerfIso}: Performance isolation for commercial {Latency-Sensitive} services. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 519–532, 2018.

[7] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, H. Eichner, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konecný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, H. Qi, D. Ramage, R. Raskar, M. Raykova, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[8] N. Li, H. Jiang, H. Che, Z. Wang, M. Q. Nguyen, and T. Rosenkrantz. User disengagement-oriented target enforcement for multi-tenant database systems. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*, pages 394–409, 2023.

[9] B. Luo, W. Xiao, S. Wang, J. Huang, and L. Tassiulas. Tackling system and statistical heterogeneity for federated learning with adaptive client sampling. In *IEEE INFOCOM 2022-IEEE conference on computer communications*, pages 1739–1748. IEEE, 2022.

[10] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In A. Singh and J. Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017.

[11] T. Nishio and R. Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019-2019 IEEE international conference on communications (ICC)*, pages 1–7. IEEE, 2019.

[12] L. J. Reed and J. Berkson. The application of the logistic function to experimental data. *The Journal of Physical Chemistry*, 33(5):760–779, 1929.

[13] R. Schwermer, R. Mayer, and H.-A. Jacobsen. Federated computing – survey on building blocks, extensions and systems, 2024.

[14] I. Tsimashenka, W. J. Knottenbelt, and P. G. Harrison. Controlling variability in split–merge systems and its impact on performance. *Annals of Operations Research*, 239(2):569–588, 2016.

[15] Z. Wang, H. Li, L. Sun, T. Rosenkrantz, H. Che, and H. Jiang. Tailguard: Tail latency slo guaranteed task scheduling for data-intensive user-facing applications. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, pages 898–909. IEEE, 2023.

[16] T. Zhang, C. He, T. Ma, L. Gao, M. Ma, and S. Avestimehr. Federated learning for internet of things. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, SenSys '21, page 413–419, New York, NY, USA, 2021. Association for Computing Machinery.

[17] B. Zhao, X. Li, X. Liu, Q. Pei, Y. Li, and R. H. Deng. Crowdfa: A privacy-preserving mobile crowdsensing paradigm via federated analytics. *IEEE Transactions on Information Forensics and Security*, 18:5416–5430, 2023.