Achieving Optimal Cache Utility in Constrained Wireless Networks through Federated Learning

Shanti Chilukuri
School of Computer Science and IT,
University College Cork,
T12 K8AF Cork, Ireland
Email: s.chilukuri@cs.ucc.ie

Dirk Pesch
School of Computer Science and IT,
University College Cork,
T12 K8AF Cork, Ireland
Email:d.pesch@cs.ucc.ie

***Abstract*—Edge computing allows constrained end devices in wireless networks to offload heavy computing tasks or data storage when local resources are insufficient. Edge nodes can provide resources such as the bandwidth, storage and in-network compute power. For example, edge nodes can provide data caches to which constrained end devices can off-load their data and from where user can access data more effectively. However, fair allocation of these resources to competing end devices and data classes while providing good Quality of Service is a challenging task, due to frequently changing network topology and/or traffic conditions. In this paper, we present Federated learning-based dynamic Cache allocation (FedCache) for edge caches in dynamic, constrained networks. FedCache uses federated learning to learn the benefit of a particular cache allocation with low communication overhead. Edge nodes learn locally to adapt to different network conditions and collaboratively share this knowledge so as to avoid having to transmit all data to a single location. Through this federated learning approach, nodes can find resource allocations that result in maximum fairness or efficiency in terms of the cache hit ratio for a given network state. Simulation results show that cache resource allocation using FedCache results in optimal fairness or efficiency of utility for different classes of data when compared to proportional allocation, while incurring low communication overhead.**

***Index Terms*—edge computing, resource allocation, fairness, federated learning**

I. Introduction

We have been experiencing a rapid growth in the number of heterogeneous low power mobile and wireless end devices. These devices differ in how they generate data, their battery capacities and communication and processing capabilities. This makes it difficult to achieve good network performance and application Quality of Service (QoS) as network management functions have to find a suitable solution that meets everyone's need. Software Defined Networking, network slicing, edge computing are some of the approaches that have been studied and successfully applied to networks to deal with this heterogeneity and manage performance and QoS.

Edge computing, as one such paradigm, has been applied with encouraging results in the context of resource-constrained Internet of Things (IoT) environments. In the edge computing model, data generated by an end device (the *producer*, e.g., a sensor) is sent to a nearby edge node. The edge node stores the data in a cache and serves it in response to incoming requests from a *consumer* with or without aggregation. Optimal edge cache management can yield desired or improved application QoS and reduced traffic in the core network [1].

A single edge node is typically connected to several end devices that generate different classes/flows of data with different QoS requirements. Furthermore, an edge node has (limited) resources such as up link bandwidth and cache space that are shared by these data classes/flows. Changing edge network topology and traffic patterns make effective allocation of edge resources to competing classes/flows of data a challenge. In such cases static resource allocation typically does not achieve a (minimum) guaranteed QoS, requiring a suitable dynamic resource allocation scheme.

Resource allocation to competing classes/flows of data with the goal of maximizing the fairness or efficiency of allocation is a well-researched topic. However, here we aim at achieving fairness or efficiency in terms of the *utility*[1] of the resource rather than the typical allocation fairness or efficiency. As the benefit of an allocation is more closely related to the QoS than the allocation itself, our interest is in *utility fairness* rather than *allocation fairness*.

The utility of an allocated resource varies with several factors other than the allocated resource share, e.g. the traffic characteristics, topology and channel condition. The relationship between the allocated resource share and its utility is often difficult to model. In such situations, Machine Learning (ML) has shown considerable promise in that the behaviour of the system is "learned" rather than modelled. If the network conditions are considered as a set of input features to an ML algorithm, the optimal resource allocation for that network condition can be learned from the behaviour of the network in the past as proposed in [3], [4]. Among others, deep and reinforcement techniques have been proposed and

This work has received funding in part, from the European Unions Horizon 2020 Research and Innovation Programme under the EDGE COFUND Marie Sklodowska Curie grant agreement No. 713567 and from the Science Foundation Ireland under CONNECT Centre grant no. 13/RC/2077.

[1]In this paper, we use the term utility to denote the benefit to a class of data from a resource allocation, rather than the overall efficiency of the allocation [2]

evaluated to do this.

For any ML task, the accuracy of the *model* learned depends on several factors, the most important of which is the amount of data available for training and its statistical distribution. When the data for training is available at different network locations, the model can be built either using local data (data from one network location) or global data (data from several network locations). In case local data is used to learn a model, the model may not be as accurate or comprehensive compared to centralised learning, where all data sets are sent to a central server to build a global model. However, centralised learning can be very communication intensive.

Federated Learning(FL) [5] is a relatively new ML paradigm in which the nodes possessing different sub sets of training data collaborate and build a model based on all the data available across the federation. The data does not need to be transmitted to a central location, but rather the local model information is shared, resulting in better accuracy with less communication overhead. When the participating nodes have limited bandwidth (e.g., edge nodes in IoT scenarios), this is very beneficial.

In this paper, we propose Federated learning-based dynamic Cache allocation (FedCache) for optimally fair or optimally efficient utilization of cache space to different classes of data in edge nodes for resource-constrained environments. The main contributions of this paper are

- an expression for the utility of the allocated cache space taking into consideration not only the allocated portion, but also the network state as seen by a class of data.
- the proposal and evaluation of Federated Learning-based cache allocation (FedCache) for dynamically allocating the available cache to ensure optimal fairness or efficiency in terms of the cache hit ratio.

Simulation results for sample IoT (Internet of Things) scenarios confirm that a model with better accuracy can be obtained with low communication overhead using FL. Results show FedCache can use the federated model to decide optimally fair or optimally efficient allocation of edge cache space for a dynamic wireless network.

II. Cache Space Management at the Edge

Consider an edge node that caches m classes of data, where data transmitted by end devices enter into the cache at an average rate γ_i for class i. The important factor effecting the cache hit ratio for a given request pattern is the average time a data item spends in the cache, which in turn depends on the rate at which new data enters the cache.

With a single (*unified*) cache for all classes of data, the average time a data item of any class spends in the cache is

$$\frac{1}{\sum_{i=1}^{m} \gamma_i}.$$

The data inflow rate of a class affects the average time spent in the cache by a data item of **all** classes.

This is shown in Figure 1 for a sample IoT case where a server requests data at an average request rate of δ_i for class i. Two classes of data share the cache and $\gamma_1 = 2\gamma_2$ with data entering the cache at a constant bit rate (CBR). For this sample case, $\gamma_1 > \delta_1$ and $\gamma_2 < \delta_2$. Data items are requested in the order of their generation as is typical for IoT. The hit ratio depicted is based on a First In First Out (FIFO) replacement policy. During the cold cache phase, requests can be satisfied as incoming data items can be stored in the empty cache. Once the cache fills-up with data, it is evident that the data of class 1 gets overwritten before it is requested and would result in a low hit ratio. However, since data of Class 2 gets replaced as well, its cache hit ratio will fall too, although $\gamma_2 < \delta_2$.

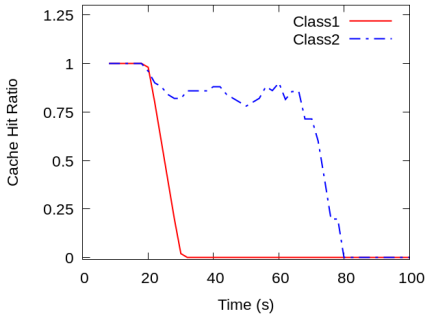

Fig. 1. Cache Hit Ratio over Time (CBR data, Unified Cache)

On the other hand, when the cache is structured such that data belonging to different classes occupy different sections of the cache memory, this does not happen as items in a particular cache section are not affected by data flowing into other cache sections. This can be seen in Figure 2, where Class 2 has a hit ratio unaffected by the data inflow of Class 1 (FIFO replacement, with each item requested once in the order of generation). Each class is allotted 50% of the available space in this example with the same data and request rates as above.

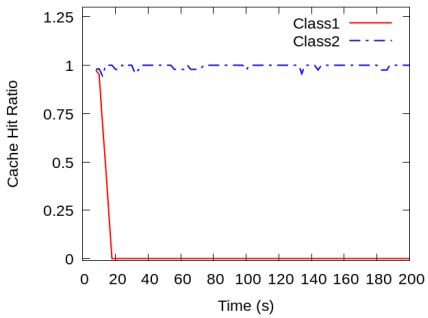

Fig. 2. Cache Hit Ratio over Time (CBR data, Split Cache)

Figure 3 shows the effect of cache space allocated in a split cache with bursty inflow (data pattern of a smoke sensor from Table II), with requests at a rate equal to the average rate of

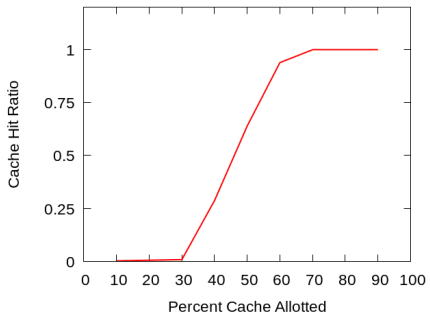

Fig. 3. Cache Hit Ratio with Allocated Space (Bursty data)

inflow. The cache size is 500 data items. Varying request patterns and/or burstiness may result in an average hit ratio that is not zero or one as in Figures1 and 2. Nevertheless, the hit ratio increases with allocated space.

In addition to isolating the hit ratio of a class from the data inflow of other classes, splitting the cache also allows the use of different replacement policies for different sections [6]. Typical edge computing scenarios have devices that generate bursty data and a split cache helps in isolating the effect of bursts on the hit ratio to that class alone. When the data inflow of different classes/flows has different burstiness patterns, the challenge is to partition the cache in a way to result in optimally efficient hit ratio for the entire cache or optimally fair hit ratio across different classes .

TABLE I
NOTATION

Symbol	Meaning
N	Number of nodes participating in FL
M	Number of classes of data at any node
$\mathbf{X}$	m-dimensional vector of allocation to each class at a node
$\mathbf{S}_i$	network state of the i^{th} class at a node
A	set of possible allocation vectors
$u(x_i, \mathbf{S}_i)$	utility of an allocation x_i when the node is in network state $\mathbf{S}_i$
$F(\mathbf{X})$	fairness of an allocation $\mathbf{X}$
$\eta(\mathbf{X})$	efficiency of an allocation $\mathbf{X}$
$\mathbf{w}_G^{j+1}$	global model after the j^{th} round
$\mathbf{w}_i^{j+1}$	local model of the i^{th} node after the j^{th} round
α_i	rate of inflow of the i^{th} class at a node
β_i	avg. number of interests for the same data item of the i^{th} class at a node
t	number of iterations per round of federated learning
r	number of rounds in federated learning
T1	Time for which data is gathered during the test phase
T2	Time after which cache space is reallocated
T3	Time after which the training data set is changed

III. FAIRNESS

Fairness can be defined qualitatively or quantitatively. Quantitative fairness measures are based on the allocated portion of the resource alone. Jain's index [7] is a simple and well-known way of measuring quantitative fairness, but its minimum value depends on the bounds of the value being judged [8]. Qualitative fairness measures such as max-min, proportional fairness and Tian Lan's model [9] judge whether a given allocation is fair, instead of providing a measure of fairness. A detailed discussion of fairness in networks can be found in [2].

In general, the utilitarian philosophy of fairness aims to maximize the overall benefit of the allocation in the system. In contrast, we distinguish between the following as suggested in [10]:

- fairness to individuals, which increases with decreasing disparity in utility among competing individuals and
- efficiency, which is a measure of the overall utility across the system.

The goal can be to choose an allocation that optimizes fairness or efficiency of utility.

[8] introduces the concept of fairness in terms of the user Quality of Experience (QoE). The authors quantify fairness in a way that normalizes both the QoE and the standard deviation of the QoE experienced by different users as [8]

$$F(\mathbf{X}) = 1 - \frac{2\sigma}{H - L} \tag{1}$$

where H and L are the upper and lower bounds of QoE and σ is the standard deviation of the QoE values. As discussed in Section I, our goal is to optimize the utility fairness rather the fairness of the allocation itself. This is because the utility has a more direct impact to the QoS than the portion allocated as discussed in Section I. Hence, we base our fairness model on the definition in Equation[1].

Considering cache space allocation among M individuals (in our case, classes of data), let $\mathbf{X} = (x_1, x_2, .., x_m)$ be the vector of allocations, where x_i is the percentage allocation to the i^{th} individual (class). The fairness of this allocation is measured by a function $F(\mathbf{X}) : \mathbb{R}_n^+ \longrightarrow \mathbb{R}^+$ [10], which defines the utility of an allocation x_i to an individual as $0 \leq u_i(x_i) \leq 1$ where u_i is the utility function of the i^{th} individual. The notation used in this section is given in Table I.

In a network where some resource has to be allocated, the utility may not only depend on the quantity of the allocated resource, but on several other factors, e.g. the network *state* **S**). For example, the goodput of a flow depends not only on the bandwidth (the resource) allocated to a flow but also on other factors like the channel conditions at different receivers of the flow. Similarly, when allocating cache space (the resource), the cache hit ratio (the utility) depends on factors such as the rate at which new data enters the cache and the similarity of requests.

To factor this into fair allocation, we express the utility u of an allocation x_i to an individual as $0 \leq u(x_i, \mathbf{S}_i) \leq 1$, where $\mathbf{S}_i$ is the current condition (state) as seen by the competing individual (class of data). $\mathbf{S}_i$ is a vector that includes all the factors (other than allocation) influencing the utility.

To quantify fairness, we leverage on the formula for fairness introduced in [8]. When the cache hit ratio is measured, its upper and lower bounds H and L are 1 and 0 respectively. Let $\sigma_{\mathbf{X}}$ be the standard deviation of the cache hit ratios of different classes resulting from an allocation $\mathbf{X}$.

With Equation[1] we get

$$F(\mathbf{X}) = 1 - 2\sigma_{\mathbf{X}} \tag{2}$$

Further, the efficiency η of an allocation is given by the average of all the cache hit ratios resulting from the allocation. Thus, the efficiency is given by [10]-

$$\eta(\mathbf{X}) = \frac{1}{m}\sum_{i=1}^{m} u(x_i, \mathbf{S}_i) \tag{3}$$

With finite possible allocation vectors $\mathbf{X}$, the allocation that can result in the best possible fairness or efficiency can be chosen. However, to do this, one needs to know the utility function. Since the utility (hit ratio) depends on both x and $\mathbf{S}$, we propose to use machine learning to learn the hit ratio of an allocation for different states. The data for training is gathered by different edge nodes online. Since we consider allocation at edge nodes in constrained networks, we use federated learning to build a global model with low communication overhead.

IV. Federated Learning

Machine learning has been successfully applied for regression to learn the relation between the output (the *target*) and a set of inputs (the *features*). In supervised machine learning, this is done by training the ML algorithm to create a model based on a sample set of input-output pairs called the training data. The model thus created is tested on yet another set of input - output pairs called the test set. Once the model is refined enough to obtain the desired accuracy with the test set, it can be used to predict the output for any given input data with reasonable accuracy.

In some applications of machine learning, the data used to train the machine learning algorithm (the *training set*) and the creation of a model are located in the same place, e.g. computing node. For example, training machine learning models for recognizing characters or image classification have been done successfully with a centralized training set. However, in some scenarios, all the training data is not located at one computing node but is available as subsets spread across several networked computing nodes (e.g., training data gathered by crowd-sourcing). In this case, there are two options,

- train the model with the local data available; or
- communicate all data to a central node which trains the model.

Training individual local models may result in sub-optimal solutions. On the other hand, communicating all data to a central node to create a single global model may result in a more accurate model but may be at high communication cost - especially for training deep learning models with a lot of data. Such communication overhead may be prohibitive, particularly in wireless networks with limited bandwidth and low-energy devices.

Federated learning is a relatively new machine learning paradigm suited to such problems. Using federated learning, an optimal, global model can be trained without communicating the training data itself. This is done by building local models and sending them to the FL server which aggregates them.Since the communication overhead of transmitting model data is much lower than transmitting all the training data, federated learning can work well in networks with unreliable or slow network connections [5]. In addition, as the training data does not need to be communicated, privacy is enhanced. As such, federated learning is an attractive option for learning in wireless and ad hoc networks [11].

A typical federated learning setting consists of two types of actors - the participant nodes, each of which has a subset of the training data and the FL server, which performs the model aggregation. Each participating node i with $i = 1, ..., N$ has a subset D_i of the training data, runs the training algorithm locally and transfers the resulting model $\mathbf{w}_i$ to the FL server. The server then aggregates all the models to create an enhanced (optimal), global model $\mathbf{w}_G$, which is sent back to each participating node i for its own use.

However, aggregating the local models just once may not result in a solution that is accurate enough. Most modern algorithms used for optimization are iterative in nature and need a large number of iterations to converge to a solution. Consequently, a global model from distributed data can be built by running some iterations at each participating node and sharing the model with the FL server, which performs model aggregation and sends the aggregated model back to the nodes. The process is repeated until the algorithm converges to a solution.

A. Steps in Federated Learning

Algorithm 1 Federated_Learning(D_i)

Input: Local Training Data set D_i
Output: Global model $\mathbf{w}_G$

1: $j = 0$
2: **while** $j <= r$ **do**
3: Receive $\mathbf{w}_G^j$ from the FL server
4: Using $\mathbf{w}_G^j$ and D_i, build local model as $\mathbf{w}_i^{j+1}$
5: Send $\mathbf{w}_i^{j+1}$ to the FL server
6: Increment j
7: **end while**

Assuming that the each participating node i ($i \in [1, N]$) has a subset D_i of the training data, the FL server first initializes the task of federated learning. To this effect, it selects the initial model parameters $\mathbf{w}_G^0$ and hyper parameters such as the learning rate and sends them to the participating nodes. To balance the trade-off between model accuracy and communication overhead, the local models are communicated

by the participating nodes after locally refining the model for t iterations (one *round*). A large value of t results in lower communication overhead but may result in an inaccurate model, while a smaller value of t increases communication overhead but may yield a better model.

The following steps are repeated iteratively in each j^{th} round.

1) Each participating node i uses $\mathbf{w}_G^j$ to run a learning algorithm and refines $\mathbf{w}_G^j$ in t iterations to build a local model $\mathbf{w}_i^{j+1}$. The goal of the participating nodes is to minimize the loss function based on the local data subset.
2) The FL server aggregates the received local models $\mathbf{w}_i^{j+1}$ ($i \in [1..N]$) to create the global model $\mathbf{w}_G^{j+1}$, which it sends back to the participating nodes. This finishes one round.
3) Steps 1 and 2 are repeated for r rounds or until a sufficiently accurate model is built.

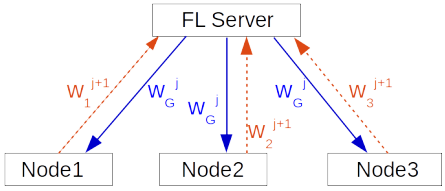

Fig. 4. Federated Learning with three participating nodes

Algorithm 1 describes the operation of the participating nodes in learning local models. Figure 4 illustrates federated learning for three participating nodes (N = 3).

A simple and classic way to aggregate models at the FL server is using the FedAvg algorithm [12], which is based on Stochastic Gradient Descent (SGD). Several improvements have been proposed for the basic FL algorithm and the FedAvg algorithm [13]. While a detailed discussion of all these is beyond the scope of this paper, we discuss below one such important variation that is beneficial to ad hoc networks.

In the simple version of FL, the FL server knows the number of participating nodes beforehand and waits for local updates from each of them in each round. However, not all of the participating nodes may respond in each round. This can be due to poor network connection between the FL server and the nodes, node mobility or low battery power. In such cases, the server may have to wait for a very long time (in the extreme case, forever) for a round to finish. To avoid this, several variations of FL have been proposed. Some of these are to wait for the first N' local model update messages ($N' \leq N$) instead of all N or to wait for a fixed time to receive local model updates. This ensures that the system converges to a global model even in the case of a varying number of participating nodes.

V. Related Work

An overview of fairness in wireless networks is given in [2]. The authors discuss popular qualitative and quantitative measures of fairness and put-forth several issues and challenges in achieving fairness in wireless networks. They identify that the relationship between a network resource and its utility is an important open issue that needs to be studied.

In [14], the authors compare the communist (maximizes fairness), utilitarian (maximizes benefit to the whole group) and capitalist (unregulated allocation) cache policies for shared system caches. They identify that near-optimal performance cannot be obtained by static uniform partitioning of the cache, no matter which policy is used for evaluation. Our simulations corroborate this fact for allocation in the case of caching at network nodes. Hence, the proposed allocation mechanism in this paper performs dynamic non-uniform partitioning of the cache space and can be used either to maximize fairness or the efficiency (communist or utilitarian respectively, as per [14]).

There has been a lot of interest in applying machine learning to improve efficiency in communication networks. Specifically, machine learning has been explored for automatic resource management by the authors of [15]. Authors of [16] establish that the device-to-device sharing model is an NP-hard optimization problem. Subsequently, the edge caching replacement problem is derived as a Markov Decision Process(MDP) and Double Deep Q-Network algorithm is used for the edge caching strategy. In [17], computing resources on the network edge are allocated using the ϵ-greedy Q-learning algorithm. In [4], a resource allocation method for content-centric IoT networks is proposed. The authors use Deep Q-learning and present results that show increased Quality of Experience (QoE) when caching is done at all the nodes as in Content Centric Networking. In [3], the authors propose and evaluate cooperative Q-learning for fair power allocation in Hetnets. [18] presents a new caching policy using deep reinforcement learning for networks where the content popularity distribution is not known.

All these papers consider deep and/or reinforcement learning for resource allocation. This requires all the data to be at a single location where the model is trained. This may not be feasible in environments where different data subsets are gathered by nodes at different network locations and with bandwidth constraints. Our approach proposes federated learning to deal with such scenarios. In addition, our main goal is to optimize the fairness or efficiency of the utility of cache space among competing classes, something which the above-cited work does not consider.

[11] presents a comprehensive survey of various federated learning techniques and their applicability to learning in mobile edge computing. The authors identify areas in which FL can be used for networks - cyber attack detection, edge caching and computation offloading, base station association and vehicular networks. They also identify challenges faced by FL from the network perspective i.e., open issues in making FL better in the face of unreliable communication, such as dropped participants, communication security and so on. Our work is different from those works as our focus is

on the usage of FL to improve communication performance and not on better communication for FL.

A study of caching and computation offloading trade-off using federated learning in mobile edge computing (MEC) systems has been studied in [19]. The authors use deep reinforcement learning with federated learning to make optimal caching and computation offload decisions in a MEC system. Similar to this, [20] studies federated learning for making computation off-loading decisions. In [21], the authors apply FL for proactive content caching in the network, where content popularity over different geographical regions can be leveraged to cache data proactively. Although the goal of these papers is to improve the network performance and QoS using FL, their focus is not on cache allocation or fairness/efficiency optimization.

VI. FedCache - Federated Learning-based Optimal Cache Allocation

Let the total available cache space at an edge node be ***R*** units and the number of classes of data ("individuals" as mentioned in Section III) among which the cache space is shared be *M*. If each class i of data gets x_i units of the cache space, the following constraint must hold:

$$\sum_{i=1}^{M} x_i \quad \leq \boldsymbol{R} \tag{4}$$

Since the goal is to optimize the utility fairness, we need to

$$\begin{aligned} &\max F \\ &\text{s.t.} \quad \textstyle\sum_{i=1}^{M} x_i \quad \leq \boldsymbol{R} \end{aligned} \tag{5}$$

where F is given by Equation (2). Alternately, the goal can be to maximize the efficiency of allocation i.e.,

$$\begin{aligned} &\max \eta \\ &\text{s.t.} \quad \textstyle\sum_{i=1}^{M} x_i \quad \leq \boldsymbol{R} \end{aligned} \tag{6}$$

where η is given by Equation (3).

Algorithm 2 FedCache($\mathbf{w}_G$)

```
Initialize timer2 and timer3
while 1 do
  if timer2 = T2 then
    Sense S_m for each m^th class
    for X in A do
      for m = 1 to M do
        Use w_G to predict u_m(x_m, S_m)
      end for
    end for
    Find X* = {X | F(X) = max_{∀X∈A} F(X)}
    for m = 1 to M do
      Apply X* and note the resulting actual cache hit ratio u'_m
      Augment (x_m, S_m, u'_m) to the new local data set D'_i
    end for
    Reset timer2
    if timer3 = T3 then
      Federated_Learning(D'_i)
      Reset timer3
    end if
  end if
end while
```

Our proposed cache allocation mechanism FedCache is specified in Algorithm 2. The notation used here is given in Table I. In a typical edge node, data from end devices is gathered and held in the cache from where it can be requested by application users or a server. As the amount of cache space allocated increases, the cache hit ratio improves as can be seen from Figure 3. For a given cache space, if the rate at which fresh data flows into the cache increases, items already present in the cache are replaced faster and cannot be retrieved from the cache anymore, resulting in lower hit ratio. In typical IoT scenarios, data is requested in the order of its generation. If the same data item is requested multiple times, it may result in multiple hits or misses depending on whether it is present in the cache or not.

Thus, the factors effecting the utility u of the i^{th} class in such a cache are

- the cache space allocated to class i, x_i,
- the average rate with which data of that class flows into the cache α_i and
- the average number of times the same data item is requested β_i

Hence, the state vector $\mathbf{S}_i$ of the i^{th} class at a given instant is (α_i, β_i). The FedCache algorithm is executed iteratively repeating the phases described below.

A. Learning the Utility Function using Federated Learning

During the set-up phase, nodes using FedCache gather the following data while varying the space allocation x_i to different classes of data:

- the state $\mathbf{S}_i$ of class i at that time and

- the resulting cache hit ratio (utility) for class i, $u(x_i, \mathbf{S}_i)$

After gathering this data (D_i is used to denote the data set gathered by the i^{th} node) for T_1 seconds, the nodes participate in a federated learning phase to learn a model that can predict the utility u of a particular allocation x_i in a state $\mathbf{S}_i$. This global model is used to predict the cache hit ratio resulting from an allocation for the given network conditions. By iterating over all possible allocations for a class in its current state, the node can choose an allocation that is optimal in terms of fairness and/or overall efficiency.

B. Dynamic Allocation of Cache space

Once the model is trained, after every time interval T_2, each node measures the inflow rate of data α_i and similarity of requests β_i. It then calculates the utility of different possible space allocations and adjusts the cache space allocated to the different classes of data to achieve maximum fairness or efficiency.

C. Refining the Model

The actual fairness or efficiency of the cache allocation strategy depends on the FL model built. In networks with fixed number of nodes or traffic patterns, the model can be built once during the setup phase. For networks with widely varying traffic conditions and network topologies, the model predictions may result in cache hit ratios that are not optimally fair or efficient. To deal with this, nodes continue to gather data during the operational phase. The actual cache hit ratio u'_m resulting from allocation according to the current model is recorded together with the allocation and state as $(x_m, \mathbf{S}_m, u'_m)$. This builds a new data set based on the node's experience. After T_3 seconds, the nodes run the federated learning phase again to refine the model, using the new local data sets $D'_i \quad i \in [1, N]$. The model built during the setup phase is refined by running federated learning after every T_3 seconds of the operational phase. This life-long learning helps build a better model, which can result in a better allocation.

VII. Simulation Results

In order to evaluate the proposed resource allocation scheme FedCache, we performed simulations using ndnSIM [22], an ns3-based network simulator for Information Centric Networks (ICN). ICN is a name-based network architecture where data is requested, routed and returned by its name, rather than the host (IP) address. Several ICN architectures have been proposed including Named Data Networking (NDN) [23], which is pull-based. In an NDN network, the node interested in a data item (the *consumer*) sends an interest packet with the unique name of the data item. This interest is routed based on the forwarding information in the Forwarding Information Base (FIB) at intermediate nodes. Once the interest reaches the node generating/serving the data item (the *producer*), the data item is returned along the reverse path and is stored in a content store (CS) (a cache) at all the network nodes along the path. Subsequent interests for this data item may be satisfied by any network node that has the item in its cache.

Our choice of the NDN architecture for simulation is due to its innate support for caching of data at any network node. The proposed algorithm, FedCache, can be applied to any (name or host-based) network that supports caching at network nodes.

The network considered for simulation consists of three edge nodes, each of which is connected to a variable number of (up to 25) end nodes randomly located in a 100x100m area. Edge nodes are connected to the devices with 802.11 links and the loss model considered is ThreeLogDistance-PropagationLossModel. The end nodes themselves are IoT devices with varying data generation patterns. The data generation patterns for these devices have been taken from [24], where the researchers study the traffic from real devices and characterize them. The data rates of the devices considered are given in Table II. Data coming from the devices is divided into three classes, depending on their criticality. For example, data from smoke and motion sensors is considered to be of Class 0, data from smart plugs and cameras is considered to be Class 1 and data from photo frames and weather stations is treated as of Class 2.

TABLE II
SAMPLE TRAFFIC PARAMETERS OF VARIOUS END DEVICES

Type of device	Avg. sleep time(s)	Mean data rate (bps)	Peak to mean ratio	Packet length(bits)
Smoke sensor	4	462	11	234
Motion sensor	4	11388	11	234
Smart Plug	241	462	11	144
Wireless Speaker	4	462	66	144
Smart bulb	4	462	11	94
Smart camera	4	2461	11	144
Smart photo frame	4	462	11	234
Hub for smart things	4	462	11	94
BP monitor	24832	462	11	144

Data gathered by end nodes is cached at the nearest edge node from where it is served to the consumers requesting it. We consider a FIFO replacement policy because for typical IoT applications, fresh data is more important than older data. Keeping such applications in view, we consider a simple request pattern where data is requested in the order of its generation. However, the choice of the replacement policy is orthogonal to the working of FedCache. The rate at which data of class m is sent by end devices to a given edge node is α_m and the average number of consumers requesting the same data item of class m from the edge node is β_m. Even for a fixed set of end devices, α_m varies over time because the data gathered by end devices is bursty, as observed by [24].

For gathering the initial training data (the setup phase), each edge node varies the available cache space (200 data items) for each class of data from 10% to 80%. For each cache allocation, the number of consumers requesting the

same data item β is varied from 1 to 6[2]. For each cache allocation vector $\mathbf{X}$ and β, the rate at which data arrives into the cache (α) and the corresponding cache hit ratio u is noted per class. The tuple $(u_m, x_m, \alpha_m, \beta_m)$ consisting of the readings for the m^{th} class is noted after every 100 simulation seconds and is inserted into the training data set.

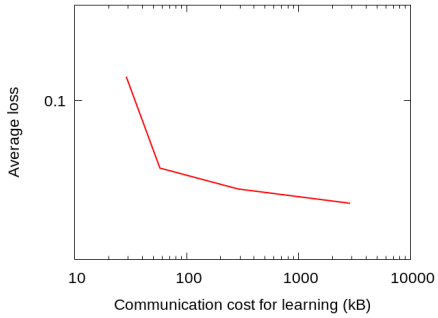

Fig. 5. Average loss with communication cost for federated learning

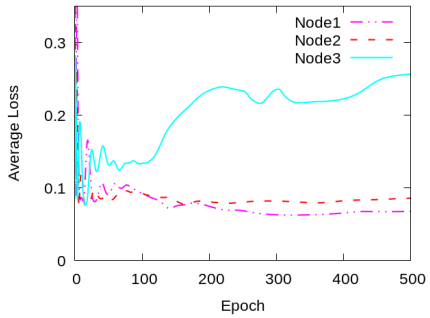

Fig. 6. Average loss without federated learning

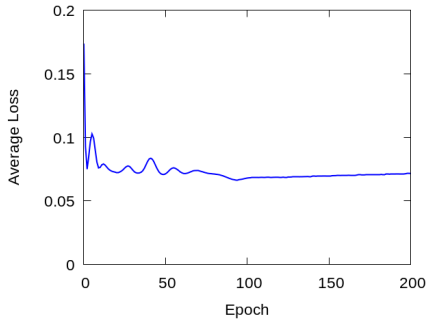

Fig. 7. Average loss with federated learning

Once the training data is gathered, the nodes run a federated learning algorithm with an Adam optimizer [25].

The sigmoid activation function and mean square error loss are considered for training, with the learning rate starting at 0.1 and decaying with a factor of 0.95 for every 5 steps. Since we consider just three features, the network consists of a 3x16 input layer, one 16x8 hidden layer and a 8x1 output layer. This requires 1472 bytes of data to be exchanged between the FL server and an edge node per round. Table III shows the variation in average loss as the number of rounds increases, for a total number of 1000 iterations. It can be seen that the accuracy increases with frequent model updates (more number of rounds), but the total communication cost also increases with the number of rounds as illustrated in Figure 5 (x-axis in logscale with base 10). For our data, we chose 20 rounds with 50 iterations per round, as this gives a reasonable balance of good accuracy and communication cost.

[2]For IoT data, "popularity" may not be as high as that for the World Wide Web data.

TABLE III
AVERAGE LOSS WITH NUMBER OF ROUNDS OF FL

Number of rounds	Iterations per round	Average Loss
10	100	0.107501
20	50	0.078738
100	10	0.072162
1000	1	0.067673

Figure 6 shows the variation of average loss over time when the nodes do not use FL but learn from their local data with the same model hyper parameters as discussed above. Figure 7 shows the average loss with federated learning. It can be seen that federated learning results in a model with less loss and better convergence.

Figures 8 and 9 illustrate the fairness and efficiency for all the possible allocation vectors (36 for 3 classes and allocation in steps of 10%) for a state $\mathbf{S} = (1.5, 4)$ (data inflow rate of 150 per 100 seconds and similarity of 4 same requests per second), as per the model. Of these, the allocation that results in maximum efficiency or fairness can be chosen.

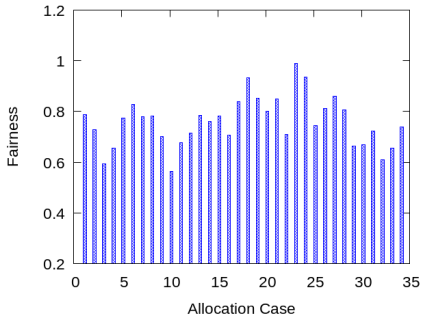

Fig. 8. Fairness for different allocations when $\mathbf{S} = (1.5, 4)$

Figures 10 and 11 depict the optimal fairness and efficiency of allocation for eight different scenarios with different network states. FedCache is compared with an allocation where the cache is split in proportion to the data rate α (denoted by Split (Proportional)) and a unified cache that is not split. For all sample network conditions, it can be seen that FedCache achieves its goal (either efficiency or fairness)

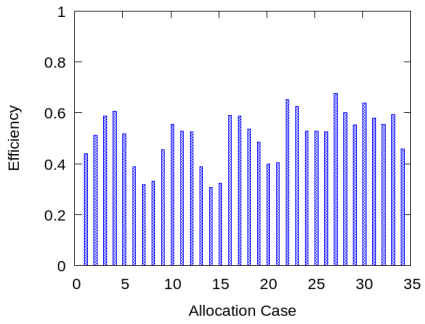

Fig. 9. Efficiency for different allocations when $\mathbf{S} = (1.5, 4)$

better than proportional resource allocation in a split cache or a unified cache.

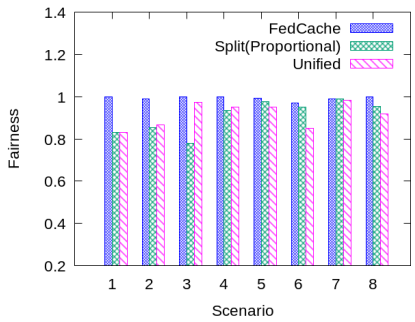

Fig. 10. Fairness for different scenarios

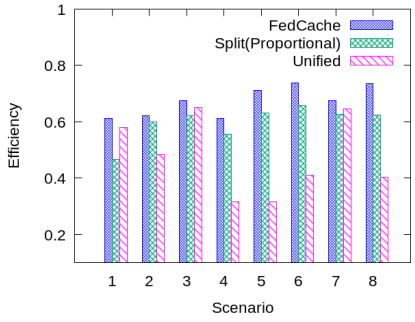

Fig. 11. Efficiency for different scenarios

VIII. Conclusions and Future Work

In this paper, we present a dynamic cache allocation scheme FedCache for optimal fairness or efficiency in terms of utility in edge caches of constrained networks. FedCache uses a continuous, federated learning mechanism to build a model that can predict the utility of a cache allocation under a particular network state. This model is used for choosing the optimal allocation under the network condition at that time. Simulation results show that FedCache results in better fairness or efficiency compared to proportional allocation, while incurring low communication overhead for training.

While the present work aims to optimize fairness/efficiency of cache hit ratio treating all classes to be equal, we intend to study it further with different priorities for each class. The effect of more complex request patterns on split cache allocation can be explored. Allocation of a single resource (the cache) has been studied and evaluated in this work. An interesting idea would be to extend the proposed scheme for joint resource allocation.

References

[1] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[2] S. Huaizhou, R. V. Prasad, E. Onur, and I. Niemegeers, "Fairness in wireless networks: Issues, measures and challenges," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 5–24, 2013.

[3] R. Amiri, H. Mehrpouyan, A. Fridman, R. K. Mallik, A. Nallanathan, and D. Matolak, "A machine learning approach for power allocation in hetnets considering qos," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–7.

[4] X. He, K. Wang, H. Huang, T. Miyazaki, Y. Wang, and S. Guo, "Green resource allocation based on deep reinforcement learning in content-centric iot," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2018.

[5] "Federated learning: Collaborative machine learning without centralized training data," https://ai.googleblog.com/2017/04/federated-learning-collaborative.html, accessed: 2020-01-21.

[6] V. S. Varanasi and S. Chilukuri, "Adaptive differentiated edge caching with machine learning for v2x communication," in *2019 11th International Conference on Communication Systems Networks (COMSNETS)*, Jan 2019, pp. 481–484.

[7] R. JAIN, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," *DEC Research Report*, 1984.

[8] T. Hofeld, L. Skorin-Kapov, P. E. Heegaard, and M. Varela, "Definition of qoe fairness in shared systems," *IEEE Communications Letters*, vol. 21, no. 1, pp. 184–187, Jan 2017.

[9] C. Y. Ma, D. K. Yau, J.-c. Chin, N. S. Rao, and M. Shankar, "Matching and fairness in threat-based mobile sensor coverage," *IEEE transactions on mobile computing*, vol. 8, no. 12, pp. 1649–1662, 2009.

[10] L. G. Boiney, "When efficient is insufficient: Fairness in decisions affecting a group," *Management science*, vol. 41, no. 9, pp. 1523–1537, 1995.

[11] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *arXiv preprint arXiv:1909.11875*, 2019.

[12] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.

[13] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *arXiv preprint arXiv:1908.07873*, 2019.

[14] L. R. Hsu, S. K. Reinhardt, R. Iyer, and S. Makineni, "Communist, utilitarian, and capitalist cache policies on cmps: Caches as a shared resource," in *2006 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2006, pp. 13–22.

[15] N. J. Yadwadkar, *machine learning for automatic resource management in the Datacenter and the cloud*. University of California, Berkeley, 2018.

[16] W. Li, C. Wang, D. Li, B. Hu, X. Wang, and J. Ren, "edge caching for d2d enabled hierarchical wireless networks with deep reinforcement learning," *Wireless Communications and Mobile Computing*, pp. 1–12, 2019.

[17] X. Liu, Z. Qin, and Y. Gao, "Resource allocation for edge computing in iot networks via reinforcement learning," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, May 2019, pp. 1–6.

[18] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, March 2018, pp. 1–6.

[19] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, no. 5, pp. 156–165, 2019.

[20] J. Ren, H. Wang, T. Hou, S. Zheng, and C. Tang, "Federated learning-based computation offloading optimization in edge computing-supported internet of things," *IEEE Access*, vol. 7, pp. 69 194–69 201, 2019.

[21] Z. Yu, J. Hu, G. Min, H. Lu, Z. Zhao, H. Wang, and N. Georgalas, "Federated learning based proactive content caching in edge computing," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–6.

[22] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnsim 2.0: A new version of the ndn simulator for ns-3."

[23] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[24] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Characterizing and classifying iot traffic in smart cities and campuses," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, May 2017, pp. 559–564.

[25] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.