Online Optimization for Low-Latency Computational Caching in Fog Networks

Gilsoo Lee†, Walid Saad†, and Mehdi Bennis‡
† Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA, USA, Emails: {gilsoolee,walids}@vt.edu.
‡ Centre for Wireless Communications, University of Oulu, Finland, Email: bennis@ee.oulu.fi.

***Abstract*—Enabling effective computation for emerging applications such as augmented reality or virtual reality via fog computing requires processing data with low latency. In this paper, a novel computational caching framework is proposed to minimize fog latency by storing and reusing intermediate computation results (IRs). Using this proposed paradigm, a fog node can store IRs from previous computations and can also download IRs from neighboring nodes at the expense of additional transmission latency. However, due to the unpredictable arrival of the future computational operations and the limited memory size of the fog node, it is challenging to properly maintain the set of stored IRs. Thus, under uncertainty of future computation, the goal of the proposed framework is to minimize the sum of the transmission and computational latency by selecting the IRs to be downloaded and stored. To solve the problem, an online computational caching algorithm is developed to enable the fog node to schedule, download, and manage IRs compute arriving operations. Competitive analysis is used to derive the upper bound of the competitive ratio for the online algorithm. Simulation results show that the total latency can be reduced up to 26.8% by leveraging the computational caching method when compared to the case without computational caching.**

I. Introduction

Emerging 5G applications such as augmented reality (AR) and virtual reality (VR) require ultra low latency transmission and computation [1]. The latency requirements of these applications cannot be accomplished by using traditional cloud computing due to the round-trip delay needed to reach the cloud [2], [3]. Thus, fog computing is proposed as an extension of cloud computing in which end-user devices, called fog nodes, perform key functions such as storing and computing data at the network edge [4]. Leveraging the physical proximity of fog nodes and pooling their resources allows for low-latency computation.

In particular, exploiting the caching capabilities of fog nodes is deemed as an essential step to improve system throughput and reduce the latency [5]–[10]. For instance, a case study that uses real measurement data in [5] shows that a fog caching architecture in vehicular networks can reduce the distance traveled by the data in the network. To show the impact of both caching and fronthaul on latency, the authors in [6] provide an information-theoretic latency analysis in fog radio access network with edge caching and cloud processing. Meanwhile, the work in [7] introduces a coding technique to reduce the latency of computation and bandwidth consumption, when data is redundantly stored in the network. The authors in [8] study the problem of caching for optimizing the users' quality of experience using unmanned aerial vehicles [11]. In [9], the authors study a distributed cluster formation of fog nodes with caching capability while maximizing a system throughput. Moreover, the authors in [10] investigate the problem of maximizing the minimum delivery rate of requested content in a cache-enabled fog network while considering the fronthaul capacity and power constraint.

In all of these existing works on caching for fog computing [6]–[10], it is generally assumed that the operation of the application running on the fog node has one corresponding input data given by either a single file or a set of files. Therefore, the goal is to gather all parts of the required input data. Such a caching technique can be viewed as *data caching*. However, for a given computational operation, multiple files can possibly be used for processing. Among the many possible input files, the operation can select a specific file as input. When each input file represents the intermediate computational result (IR) of an operation, storing a partial set of the possible input files can be seen as caching IRs, or more formally *computational caching*[1]. We therefore propose the paradigm of computational caching as a technique to reduce the computational latency at a fog node. Moreover, existing works such as [6] and [8] assume that information on the requested operation is completely known. However, in practice, a user can randomly use any application and arbitrarily request an operation in the application. Thus, the requested operation changes instantaneously and, hence, the sequential arrival of operations can be uncertain. Consequently, unlike the existing literature [6]–[10] that considers data caching, under prior knowledge on user and application behavior, our goal is to design an *online approach* to enable an online *computational caching* framework, under uncertainty, while minimizing the transmission and computational latency.

The main contribution of this paper is a novel framework for online computational caching in a fog network. This framework allows a given fog node to download the necessary IR from a neighboring fog node and use the downloaded or stored IR for computations by selecting the most suitable input in the presence of uncertainty on the arrival order of the user's operation. We formulate an online computational caching problem whose objective is to minimize the sum of the transmission and computational latency. To solve this problem, we propose an online computational caching algorithm. Then, we derive a competitive ratio of the formulated

This research was supported by the U.S. National Science Foundation under Grants CNS-1460333 and IIS-1633363, by the Office of Naval Research (ONR) under Grant N00014-15-1-2709, and by NOKIA donation on fog (FOGGY project).

[1]Note that computational caching here is different from the original notion of computational caching used in [12] in which computer networks cache the act of computation, i.e., they store the trajectories that are encountered during the execution of a software's instructions, and apply it in new contexts.

online problem. In addition, we propose a bandwidth allocation scheme to achieve a desired target competitive ratio. Simulation results show that the proposed online algorithm minimizes the latency while achieving a performance that is near-optimal compared to an offline solution that has full information on all input arrivals.

II. System Model

Consider a fog network that consists of a fog node i and a set $\mathcal{J}$ of J neighboring fog nodes as shown in Fig. 1. Fog node i is running a set $\mathcal{K}$ of K applications that require high computational resources. For instance, when fog node i is running an AR or VR application supporting six degree of freedom videos, it must process a huge volume of input data that includes video clips at different angles. While using application $k \in \mathcal{K}$, the user of fog node i may want to change the angle of view or watch video scenes related to specific persons. We assume that each application k has $L = |\mathcal{L}|$ different commands. The user's specific input command is indexed by $l \in \mathcal{L}$. When command l is executed by using application k, the computational *operation* is denoted by $\alpha_{k,l}$.

To compute operation $\alpha_{k,l}$ at time t, the required input of the operation will be $\beta^{(t)}_{k(t),l(t)}$. When $\alpha_{k,l}$ is computed, this yields an IR $r_{k,l}$. $\alpha_{k,l}$ can correspond to the operation that is used to find a relevant image or video data about a certain location by using application k. Also, there will be another operation $\alpha_{k,l'}$, $l' > l$, that is used to find the image or video about a more specific location. These two operations $\alpha_{k,l}$ and $\alpha_{k,l'}$ yield outputs $r_{k,l}$ and $r_{k,l'}$, respectively. Then, $r_{k,l}$ can be seen as the result including the broader information, and $r_{k,l'}$ can be the result including the information about the more specific requirement. Therefore, $\alpha_{k,l'}$ can be computed by having IR $r_{k,l}$ as an input. In other words, if $r_{k,l}$ is stored in fog node i, then this IR can be reused for other operation $\alpha_{k,l'}$, $l' > l$. We also define $r_{k,0}$ as the raw data that can be used to compute operation $\alpha_{k,1}$, for notational simplicity, even though the raw data is not an IR. In the VR example, $r_{k,0}$ can correspond to the raw video data that includes all degree angles of videos.

Fog node i can store (cache) a different set of IRs in its memory (e.g., RAM, flash memory, or others). The set of cached IRs of fog node i at time t is denoted by $\mathcal{R}^{(t)}_i$. We assume that $r_{k,0} > r_{k,1} > \ldots > r_{k,L}$ since the operation indexed by a larger l includes information about more specific requirements. If fog node i can store up to $M_{\max}$ bits in its memory, its memory constraint will then be $\sum_{r_{k,l} \in \mathcal{R}^{(t)}_i} r_{k,l} \leq M_{\max}$.

Fog nodes compute their operations and cache the associated IRs. If fog node i must use, at time t, a certain IR that is not cached in $\mathcal{R}^{(t-1)}_i$, it can download it over a wireless link from a neighboring fog node $j \in \mathcal{J}$ that has the IR. The data rate needed to download this information using a wireless link is $R^{(t)}_{ji} = B\log_2\left(1 + \frac{g_{ji}P_{\text{tx},j}}{BN_0}\right)$ where g_{ji} is the wireless channel gain between fog nodes i and j, $P_{\text{tx},j}$ is the transmission power of fog node j, N_0 is the noise power spectral density, and B is the bandwidth. The downloaded IR at time t is denoted by $u^{(t)}_{k(t),l(t)}$. Thus, if $u^{(t)}_{k(t),l(t)}$ is

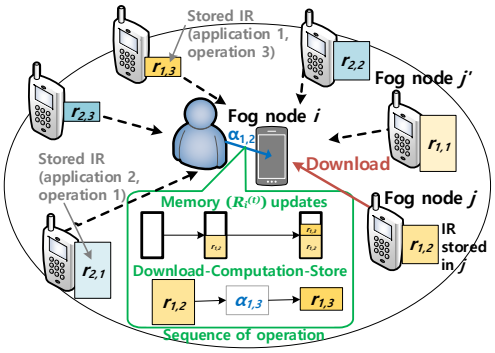

Fig. 1: Example of the computing system of a fog node i that consists of six neighboring fog nodes. In this example, the user request is operation $\alpha_{1,3}$, so IR $r_{1,1}$ or $r_{1,2}$ can be used as input. Since the memory of fog node i is empty at the beginning, $r_{1,2}$ can be downloaded from neighboring fog node j. After finishing the computation, the output IR $r_{1,3}$ is stored in the memory.

Table I: Summary of key notations

$\alpha_{k,l}$	Computational operation of command l in application k.
$r_{k,l}$	IR obtained by computing $\alpha_{k,l-1}$.
$\beta^{(t)}_{k(t),l(t)}$	Input is IR $r_{k(t),l(t)}$ when computing operation at time t.
$u^{(t)}_{k(t),l(t)}$	Downloaded IR at time t is IR $r_{k(t),l(t)}$.
$\mathcal{R}^{(t)}_i$	Set of stored IRs in the memory.

transmitted from fog node j to fog node i, the transmission latency will be:

$$T_{ij}(u^{(t)}_{k(t),l(t)}) = \frac{u^{(t)}_{k(t),l(t)}}{R^{(t)}_{ji}}. \quad (1)$$

We assume that the computational latency is quadratically increasing with the size of input $\beta^{(t)}_{k(t),l(t)}$. This assumption can capture the fact that the time-complexity of an application is $\mathcal{O}(n^2)$ where n is the data size[2]. Also, the latency can be scaled by the computational speed [13]. Thus, when input $\beta^{(t)}_{k(t),l(t)}$ is used, the computational latency at fog node i is given by

$$C_i(\beta^{(t)}_{k(t),l(t)}) = \frac{\xi}{c}\left(\beta^{(t)}_{k(t),l(t)}\right)^2, \quad (2)$$

where c is the computation speed of the fog node, e.g., value proportional to CPU frequency, and $\xi = 1$ is the unit computational price per bit. The computational latency can decrease as the data size of $\beta^{(t)}_{k(t),l(t)}$ decreases and c increases. Therefore, to compute operation $\alpha_{k,l}$, instead of using other inputs such that $\beta^{(t)}_{k(t),l(t)}$=$r_{k,l'}$, l'<$l-1$, reusing IR $r_{k,l-1}$ can reduce the computational latency. Clearly, when $u^{(t)}_{k(t),l(t)}$ is downloaded from neighboring fog node j and $\beta^{(t)}_{k(t),l(t)}$ is used to compute the given operation $\alpha_{k,l}$ by fog node i, the total latency can be defined by

$$f_i(u^{(t)}_{k(t),l(t)}, \beta^{(t)}_{k(t),l(t)}) = T_{ji}(u^{(t)}_{k(t),l(t)}) + C_i(\beta^{(t)}_{k(t),l(t)}). \quad (3)$$

Our goal is to analyze how this system can exploit the IRs stored in different fog nodes so as to minimize its latency.

III. Problem Formulation

Our goal is to minimize the end-to-end latency by enabling a fog node to properly choose the IR that must be used for the computation of its operations. Since IRs stored in fog

[2]For instance, in signal processing or image processing, the discrete Fourier transform can have quadratic time complexity $\mathcal{O}(n^2)$.

node i can be reused to compute future operations, storing IRs that will be used in the near future is beneficial to reduce the latency. However, in practice, the inputs of the operations can be dynamically determined by the user, and, thus, they cannot be anticipated. As a result, fog node i is generally unable to know a priori what the future inputs will be and, hence, optimizing the total latency becomes a challenging problem. Since the memory size of fog node i is limited, not all IRs can be stored. This limited memory has to be properly managed by erasing and storing IRs depending on the inputs. While considering unknown information on future input, downloading a new IR and managing the stored IRs at each time is essential to reduce the latency. Under such uncertainty, determining the IR needed to compute the current operation must also account for a prospective arrival of future operations. In consequence, we introduce a novel *online optimization scheme* that makes a sequential decision for the online arrival of operations to minimize the latency for computational caching.

We observe the system during a finite period of time $t \in \mathcal{T} = \{1,\dots,T\}$. At time $t \in \mathcal{T}$, one operation $\alpha_{k(t),l(t)}$ where $k(t) \in \mathcal{K}$ and $l(t) \in \mathcal{L}$, $\forall t \in \mathcal{T}$, arrives to fog node i. Then, the sequence of operations that have arrived during the observation period will be $\boldsymbol{\sigma} \triangleq (\sigma_t)_{t=1}^{T} = (\alpha_{k(1),l(1)}, \dots, \alpha_{k(t),l(t)}, \dots, \alpha_{k(T),l(T)})$. For a sequence $\boldsymbol{\sigma}$, the latency minimization problem becomes:

$$\min_{\boldsymbol{u},\boldsymbol{\beta}} \quad \sum_{t=1}^{T} f_i\left(u_{k(t),l(t)}^{(t)}, \beta_{k(t),l(t)}^{(t)}\right), \tag{4}$$

$$\text{s.t.} \quad \sum_{r_{k,l} \in \mathcal{R}_i^{(t)}} r_{k,l} \leq M_{\max}, \forall t \in \mathcal{T}, \tag{5}$$

$$\mathcal{R}_i^{(t)} \subset \mathcal{R}_i^{(t-1)} \cup \{u_{k(t),l(t)}^{(t)}\}, \forall t \in \mathcal{T}, \tag{6}$$

$$\beta_{k(t),l(t)}^{(t)} = r_{k(t),l'} \in \mathcal{R}_i^{(t)}, l' < l(t), \forall t \in \mathcal{T}, \tag{7}$$

where the time-varying vector of downloaded IRs is $\boldsymbol{u} = [u_{k(1),l(1)}^{(1)},\dots,u_{k(t),l(t)}^{(t)},\dots,u_{k(T),l(T)}^{(T)}]$ and the time-varying vector of the IRs used for computation is $\boldsymbol{\beta} = [\beta_{k(1),l(1)}^{(1)},\dots,\beta_{k(t),l(t)}^{(t)},\dots,\beta_{k(T),l(T)}^{(T)}]$. The goal of (4) is to minimize the sum of transmission and computational latency during a finite time period. (5) constrains the sum of the stored IRs at time t within the limited memory size. (6) shows that $u_{k(t),l(t)}^{(t)}$ is the downloaded IR from the network. (7) states that any IR $r_{k(t),l'}$, $l' < l(t)$, can become input $\beta_{k(t),l(t)}^{(t)}$ that is used to compute operation $\alpha_{k(t),l(t)}$ at time t. IR $r_{k(t),l'}$ must also be stored in the memory $\mathcal{R}_i^{(t)}$.

In this problem, $\sigma_t = \alpha_{k(t),l(t)}$ is revealed to fog node i at time t, and, then, fog node i must determine $\beta_{k(t),l(t)}^{(t)} = r_{k(t),l'}$. To make a decision at time t, the previously stored IRs in $\mathcal{R}_i^{(t-1)}$ can be used. Also, fog node i can download $u_{k(t),l(t)}^{(t)}$ from neighboring node $j \in \mathcal{J}$. For effective computational caching, node i can form a network in which all IRs can be downloaded from neighbors. To form a network, for example, fog nodes can exchange a beacon signal that contains the information about the stored IR. Then, fog node i can select the set of neighboring nodes such that each neighbor has a different IR. By doing so, the network of fog node i can be formed. Hereinafter, we assume that network formation is given using online approaches such as [14].

Algorithm 1 Computational Caching Algorithm

```
while time t ≤ T
  Operation α_{k(t),l(t)} arrives.
  if new application arrives,
    mark all stored IRs as removable end if
  if fog node i has any IR that can be used,
    Compute by using the best available stored IR.
  elseif fog node i does not have an available IR,
    Download IR r_{k(t),l*} from fog node j* where
      min_{j∈J,l∈L} T_ij(r_{k(t),l}) s.t. r_{k(t),l} ∈ R_j, l < l(t).
    Store the downloaded IR r_{k(t),l*} ∈ R_i^(t).
    Compute by using r_{k(t),l*}.
  end if
  Store output r_{k(t),l(t)} in R_i^(t).
end while
```

In this online computational caching problem, fog node i can reuse one of the stored IRs or download a new IR. While reusing a stored IR can reduce the transmission latency, the computational latency can increase unless IR $r_{k(t),l(t)-1}$ was already stored. On the other hand, if fog node i decides to download IR $r_{k(t),l(t)-1}$, transmission latency will be incurred, however, the computational latency will be minimized since $r_{k(t),l(t)-1}$ is the IR having the smallest size among those can be used to compute input operation $\alpha_{k(t),l(t)}$. Therefore, there is a tradeoff between computational latency and transmission latency, and, thus, choosing the appropriate IRs to compute the sequence under uncertainty, is not trivial. Under such incomplete information, finding the optimal solution of (4) in a conventional offline manner is clearly not feasible and, therefore, an online solution is needed.

IV. Online Computational Caching

We propose an online computational caching algorithm that schedules the IRs to minimize the total latency given by (3). To reduce the computational latency, the stored IRs are reused. However, since not all possible IRs can be stored within a limited memory size, the fog node may download IRs from neighboring nodes. In such a case, if the wireless data rate is low, the transmission latency can become a bottleneck in minimizing the total latency. Thus, to prevent a large latency over the wireless links, we introduce an online algorithm focusing on minimizing the transmission latency.

The online algorithm must also manage the limited size of memory by evicting outdated IRs. For storing IRs, if the memory does not have sufficient free space, then existing IRs must be removed, to include the new data. Therefore, the current decision to remove a certain IR can also affect the set of the stored IRs in the future. Then, to manage the stored IRs, we define two events in which all stored IRs are marked so that the marked items can be evicted from the memory.

The proposed online computational caching algorithm shown in Algorithm 1 is a transmission-centric algorithm that minimizes the use of wireless resources. Thus, if any

IR, i.e., $r_{k(t),l}, l < l(t)$, is stored in $\mathcal{R}_i^{(t)}$, it is used to compute $\alpha_{k(t),l(t)}$ without downloading other IRs. However, if fog node i does not have an IR to compute $\alpha_{k(t),l(t)}$, it downloads IR $r_{k(t),l^*}$ from neighboring node j^* so as to minimize the transmission latency $T_{ij}(r_{k(t),l})$ such that $l < l(t)$. The downloaded IR is stored in $\mathcal{R}_i^{(t)}$ and used to compute the operation. Then, the output of this operation $r_{k(t),l(t)}$ is stored in the memory.

In Algorithm 1, after computing the incoming operation at each time or after downloading the IR, the output IR is stored for possible future computation. To store the output IR, fog node i must have free memory space. If $\mathcal{R}_i^{(t)}$ does not have enough space, some of the previously stored IRs must be erased. To this end, we propose a modified marking algorithm. In the so-called paging problem, a marking algorithm is typically developed to evict a marked page from the memory when eviction is necessary [15]. To exploit the marking-and-eviction structure from marking algorithms, we define two events to mark all stored IRs. The motivation behind designing two events is to determine the stored IRs that are unlikely to be used in the near future. The first event is triggered when the application type is changed. For instance, if the application at time $t-1$ is different from the current application at t, all IRs in the memory are marked as erasable. In the second event, all IRs are marked if the sum of all unmarked IRs in the memory reaches or exceeds the maximum memory capacity.

Once IRs are marked by the two events, the eviction scheme will follow a least-recently-used (LRU) algorithm [15]. LRU replaces an old page that is least recently used if one empty slot for a new page is required. We propose an IR eviction scheme (IRES) that gives priority to recently used IRs. Similar to LRU, IRES removes the least-recently-used IR from $\mathcal{R}_i^{(t)}$, but it only selects the IR that is marked by the two proposed events. IRES is different from LRU in that LRU replaces one page to store a page, but IRES will repeatedly remove the marked IRs until enough free space for a new IR is available.

Next, we analyze the online computational caching problem by using *competitive analysis* [15]. Competitive analysis measures the performance of an online algorithm denoted by $ALG(\boldsymbol{\sigma})$ by comparing it to the performance of an ideal, offline optimal algorithm represented by $OPT(\boldsymbol{\sigma})$. Then, a competitive ratio can be defined by $\Gamma(\boldsymbol{\sigma}) = ALG(\boldsymbol{\sigma})/OPT(\boldsymbol{\sigma})$. While an online algorithm has information only on the current and past input sequence of operations, the offline optimal algorithm is ideal and knows the entire input sequence $\boldsymbol{\sigma}$. For analysis, we divide the whole input sequence $\boldsymbol{\sigma}$ into multiple partitioned sequences using two marking events. While the online algorithm processes input sequence $\boldsymbol{\sigma}$, if one of two marking events is triggered at time t' and time t'', respectively, the partitioned sequence $\hat{\boldsymbol{\sigma}}$ will be defined as $(\sigma_{t'},\ldots,\sigma_{t''-1})$. By doing so, for an online algorithm, any input sequence can be divided into partitioned sequences. For notational simplicity, we omit the index for the partitioned sequences. Then, we respectively measure the minimum latency of the offline optimal algorithm and the maximum latency of an online algorithm during processing $\hat{\boldsymbol{\sigma}}$. If the input operations are given by $\hat{\boldsymbol{\sigma}}$, the minimum latency achieved by the optimal offline algorithm is denoted by $OPT(\hat{\boldsymbol{\sigma}})$. To derive a lower bound for $OPT(\hat{\boldsymbol{\sigma}})$, we find the lower bound of the transmission and computational latencies, respectively. The lower bound of the transmission latency can be zero if fog node i does not download any IRs. The minimum computational latency is achieved by using the smallest size of IR for each input operation in $\hat{\boldsymbol{\sigma}}$. Therefore, $OPT(\hat{\boldsymbol{\sigma}})$ is at least larger than the minimal computational latency to compute operations $\hat{\boldsymbol{\sigma}}$, so $OPT(\hat{\boldsymbol{\sigma}}) \geq \sum_{t=t'}^{t''-1} C_i(r_{k(t),l(t)-1})$. We denote the latency of the online algorithm by $ALG(\hat{\boldsymbol{\sigma}})$. To find the upper bound of $ALG(\hat{\boldsymbol{\sigma}})$, we consider the worst-case scenario in which node i does not have any IR. As an example of this worst case, if the requests of applications arrive in a round-robin fashion, any cached IR will be removed and can no longer be used. Here, $ALG(\hat{\boldsymbol{\sigma}})$ becomes at most the sum of the transmission and computational latency for all operations in $\hat{\boldsymbol{\sigma}}$. Thus, $ALG(\hat{\boldsymbol{\sigma}}) \leq \sum_{t=t'}^{t''-1} T_{ij}(r_{k(t),l(t)-1}) + C_i(r_{k(t),l(t)-1})$.

The competitive ratio with respect to $\boldsymbol{\sigma}$ is upper bounded by the largest competitive ratio of a partitioned sequence denoted by $\hat{\boldsymbol{\sigma}}$. Thus, the competitive ratio can be found as follows:

$$\begin{aligned}\Gamma(\boldsymbol{\sigma}) &\leq \max_{\hat{\boldsymbol{\sigma}}} \{\Gamma(\hat{\boldsymbol{\sigma}})\}, \\ &\leq \max_{\hat{\boldsymbol{\sigma}}} \left\{ \frac{\sum_{t=t'}^{t''-1} (\rho_t + 1) C_i(r_{k(t),l(t)-1})}{\sum_{t=t'}^{t''-1} C_i(r_{k(t),l(t)-1})} \right\}, \end{aligned} \quad (8)$$

where ρ_t is defined by $T_{ij}(r_{k(t),l(t)-1}) = \rho_t C_i(r_{k(t),l(t)-1})$.

A. Bandwidth allocation

From (8), we propose a wireless bandwidth allocation scheme that yields a constant competitive ratio for problem (4). Suppose that a target performance is predetermined by ρ. Then, when fog node j transmits an IR to fog node i, the bandwidth used by neighboring node j is the solution of:

$$B \log_2 \left(1 + \frac{g_{ji} P_{\text{tx},j}}{B N_0} \right) = \frac{c}{\rho\, \xi\, r_{k(t),l(t)-1}}. \quad (9)$$

If the maximum bandwidth that fog nodes can access is $B_{\max}$, then $0 \leq B \leq B_{\max}$. Since the left-hand side of (9) corresponds to the data rate $R_{ji}^{(t)}$, the data rate is scaled by ρ in (9). Now, we find the competitive ratio by using ρ.

Proposition 1. *For a given ρ, if there exists B satisfying* (9)*, $0 \leq B \leq B_{\max}$, then the competitive ratio becomes $\rho+1$.*

Proof. From (8), by replacing ρ_t, $\forall t \in \mathcal{T}$, with ρ, the competitive ratio of all partitions becomes $\rho + 1$, and it provides the upper bound of the competitive ratio with respect to $\boldsymbol{\sigma}$. □

This show how ρ can be a design parameter used to adjust the competitive ratio. For any given ρ satisfying Proposition 1, the competitive ratio is fixed as $\rho + 1$. Also, from (9), the bandwidth allocation scheme results in a higher data rate if computational speed c increases and the size of stored IR $r_{k(t),l(t)-1}$ decreases. The competitive ratio (8) is derived from (4) without any assumption on the online algorithms. Hence, this competitive ratio can be applied to any online algorithm for our model.

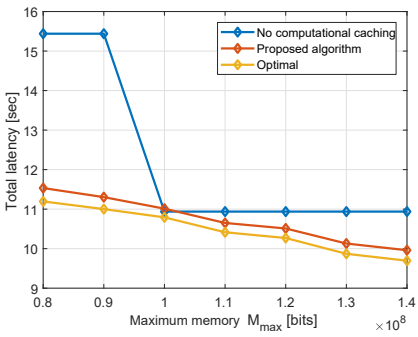

Fig. 2: Total latency for different memory sizes of fog node i.

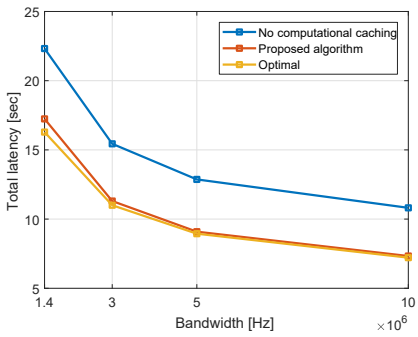

Fig. 3: Total latency for different bandwidths $B_{\max}$.

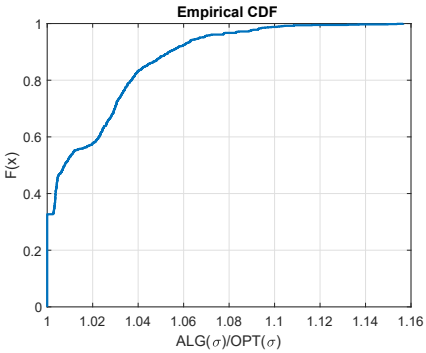

Fig. 4: The CDF of the empirical competitive ratio for the proposed online algorithm.

Table II: Simulation parameters.

Figure	c [bps]	$B_{\max}$	$M_{\max}$
Fig. 2	30×10^{14}	3 MHz	Variable
Fig. 3	30×10^{14}	Variable	90 Mbits
Fig. 4	5×10^{14}	10 MHz	90 Mbits

Table III: Computational latency in the case of Fig. 2.

No comp. caching	Proposed algorithm	Optimal
8.33 sec	5.38 sec	5.19 sec

V. Simulation Results

For our simulations, we consider that a fog node is connected to eight neighboring fog nodes uniformly distributed within 50 m. As in [14], we set $P_{\text{tx},i} = 20$ dBm and the power spectral density of the noise is -174 dBm/Hz. Also, the channel gain is $g_{ji}{=}\gamma_1 d_{ji}^{-\gamma_2}$ where d_{ji} is the distance between fog nodes i and j, $\gamma_1 = 10^{-3}$, and $\gamma_2 = 4$. Each neighboring fog node stores a different IR. Two applications are used by the user with equal probability while the arrival of four operations at each time follows the Zipf distribution [9]. The size of IRs is given by $[r_{k,0}, r_{k,1}, r_{k,2}, r_{k,3}]{=}[50, 30, 20, 10]$ Mbits, $\forall k \in \mathcal{K}$. We set $T{=}10$, and all statistical results are averaged over a large number of simulation runs. For comparison, an exhaustive search with complete knowledge about the inputs is used to find the optimal latency when IRES is used for managing the memory policy. Other parameters are shown in Table II.

In Fig. 2, we show the total latency defined in (4) for different sizes of the fog node i's memory with $B = B_{\max}$. Since the computational latency does not change significantly along with $M_{\max}$, the value averaged over different $M_{\max}$ for each case is presented in Table III. From this figure, we can first see that the proposed computational caching algorithm decreases the latency. Compared to the case without computational caching, the proposed algorithm can reduce the latency by up to 26.8% at $M_{\max} = 90$ Mbits. In the case of no computational caching, the latency is reduced at $M_{\max} = 100$ Mbits, since all raw data can be stored in the memory. However, in computational caching, the latency decreases as the memory size increases since all possible IRs cannot be stored in $M_{\max}$. Also, Fig. 2 shows that the gap between the proposed online and offline solutions, in terms of the total latency, is 3% at $M_{\max} = 80$ Mbits. From Fig. 2, we can also see that the total latency decreases as the memory size increases since the transmission latency can be reduced by caching more IRs in a larger size memory. For instance, the latency decreases by 13.6% if $M_{\max}$ increases from 80 Mbits to 130 Mbits.

Fig. 3 shows the total latency for different maximum accessible bandwidth when $B = B_{\max}$. We first see that the latency decreases with the increasing bandwidth used to transmit IRs. For example, the proposed algorithm reduces the latency by 57.4% by increasing $B_{\max}$ from 1.4 MHz to 10 MHz. Moreover, Fig. 3 shows that the total latency resulting from the proposed algorithm and the offline scenario are very close. This demonstrates the effectiveness of the proposed online algorithm. Also, in Fig. 3, the gap between the proposed online and offline solutions, in terms of total latency, can be reduced by up to 1.7% at $B_{\max} = 10$ MHz. Moreover, compared to the case without computational caching, the total latency can be reduced by up to 32.1% at $B_{\max} = 10$ MHz.

Fig. 4 shows the empirical competitive ratio for problem (4) with $\rho = 1$. The bandwidth B for each neighboring fog node is calculated from (9). We can first see that 32.7% of iterations achieve a competitive ratio of 1 which means that the result of the online algorithm coincides with the offline optimal solution of (4). Fig. 4 also shows that the empirical competitive ratio in the worst case is shown to be 1.157. From Fig. 4, we can also see that the empirical competitive ratio is less than the upper bound $\rho{+}1 = 2$ from Proposition 1. Thus, the results from Fig. 4 show that Algorithm 1 can effectively select the input IR, in an online manner, while minimizing latency.

VI. Conclusion

In this paper, we have proposed a novel framework for online computational caching in a fog network that allows the optimization of the selection of the input IR under uncertainty on the arrival order of the user's operation. We have formulated an online computational caching problem to minimize the transmission latency and computational latency. The proposed online algorithm schedules the IRs to compute each of the sequentially arriving operations while managing the stored IRs in the memory. We have also shown an upper bound of the competitive ratio for the formulated online problem. Simulation results have shown that the proposed online computational caching algorithm is effective in reducing latency.

REFERENCES

[1] M. S. ElBamby, M. Bennis, and W. Saad, "Proactive edge computing in latency-constrained fog networks," in *Proc. 26th European Conference on Networks and Communications (EuCnC)*, Oulu, Finland, May 2017, pp. 1–6.

[2] Cisco, "Fog computing and the Internet of Things: Extend the cloud to where the things are," *Cisco white paper*, 2015.

[3] M. Chen, W. Saad, and C. Yin, "Resource management for wireless virtual reality: Machine learning meets multi-attribute utility," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Singapore, Dec. 2017.

[4] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, Dec. 2016.

[5] F. Malandrino, C. Chiasserini, and S. Kirkpatrick, "The price of fog: A data-driven study on caching architectures in vehicular networks," in *Proc. 1st Int. Workshop on Internet of Veh. and Veh. of Internet*, Paderborn, Germany, July 2016, pp. 37–42.

[6] R. Tandon and O. Simeone, "Cloud-aided wireless networks with edge caching: Fundamental latency trade-offs in fog radio access networks," in *Proc. IEEE Int. Symp. on Inform. Theory*, Barcelona, Spain, July 2016, pp. 2029–2033.

[7] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coding for distributed fog computing," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 34–40, April 2017.

[8] M. Chen, M. Mozaffari, W. Saad, C. Yin, M. Debbah, and C. S. Hong, "Caching in the sky: Proactive deployment of cache-enabled unmanned aerial vehicles for optimized quality-of-experience," *IEEE J. Sel. Areas in Commun.*, vol. 35, no. 5, pp. 1046–1061, May 2017.

[9] Y. Sun, T. Dang, and J. Zhou, "User scheduling and cluster formation in fog computing based radio access networks," in *Proc. IEEE Int. Conference on Ubiquitous Wireless Broadband*, Nanjing, China, Oct. 2016, pp. 1–4.

[10] S.-H. Park, O. Simeone, and S. Shamai, "Joint cloud and edge processing for latency minimization in fog radio access networks," in *Proc. IEEE 17th Int. Workshop on Signal Process. Adv. in Wireless Commun.*, Edinburgh, UK, July 2016, pp. 1–5.

[11] M. Mozaffari, W. Saad, M. Bennis, and M. Debbah, "Unmanned aerial vehicle with underlaid device-to-device communications: Performance and tradeoffs," *IEEE Trans. Wireless Commun.*, vol. 15, no. 6, pp. 3949–3963, Jun. 2016.

[12] A. Waterland, E. Angelino, E. D. Cubuk, E. Kaxiras, R. P. Adams, J. Appavoo, and M. Seltzer, "Computational caches," in *Proc. 6th Int. Systems and Storage Conference*, Haifa, Israel, 2013, pp. 1–7.

[13] Y. Yu, J. Zhang, and K. B. Letaief, "Joint subcarrier and CPU time allocation for mobile edge computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Washington DC, USA, Dec. 2016.

[14] G. Lee, W. Saad, and M. Bennis, "An online secretary framework for fog network formation with minimal latency," in *Proc. IEEE Int. Conf. on Commun. (ICC)*, Paris, France, May 2017, pp. 1–6.

[15] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. Cambridge University Press, 2005.