IEEE Open Journal of the Communications Society

Received 5 September 2022; revised 4 October 2022; accepted 23 October 2022. Date of publication 28 October 2022; date of current version 7 November 2022.

Digital Object Identifier 10.1109/OJCOMS.2022.3217447

An Accurate Model for Computation Offloading in 6G Networks and a HAPS-Based Case Study

TOLGA OVATMAN [1] (Senior Member, IEEE), GUNES KARABULUT KURT [2] (Senior Member, IEEE), AND HALIM YANIKOMEROGLU [3] (Fellow, IEEE)

[1]Department of Computer Engineering, Istanbul Technical University, 34469 Istanbul, Turkey

[2]Poly-Grames Research Center, Department of Electrical Engineering, Polytechnique Montréal, Montréal, QC H3T 1J4, Canada

[3]Department of Systems and Computer Engineering, Carleton University, Ottawa, ON K1S 5B6, Canada

CORRESPONDING AUTHOR: T. OVATMAN (e-mail: ovatman@itu.edu.tr)

ABSTRACT The undeniable potential of computation offloading has been attracting attention from researchers for more than a decade. With advances in multi-access edge computing (MEC), computation offloading has become a more critical issue because of the heterogeneity in the computational power of edge devices and the elevated importance of extending their lifespan. Due to the apparent advantages, the use of MEC in 6G networks, where a vertical heterogeneous network composed of space, air, and ground networks is only natural. The non-terrestrial networking elements constitute effective computational resources. However, recent research investigating the potential of computational offloading in 6G networks has involved models that do not adequately reflect the complexity of the underlying processes. In this study, we propose a realistic computation model for 6G networks that considers crucial properties of the offloaded job, including the inter-dependency of the job tasks and the decomposability of the job. Our model is based on the mature application domain of MEC, where proven solutions are already studied. We also investigate the potential of a high altitude platform station (HAPS)-aided MEC platform using this model. The proposed model allows us to design offloading strategies to enable adaptive computational offloading. Through numerical analyses, we show that the proposed model provides sufficient insight to reduce the total processing time significantly.

INDEX TERMS Computation offloading, high altitude platform station, multi-access edge computing.

I. INTRODUCTION

COMPUTATION offloading is the process of transferring a part of the computational workload of an over-utilized host to another (group of) underutilized host(s) [1], [2], [3]. The main challenge in computational offloading is the decision process, whether the additional cost of this transfer and processing compensates for the cost of delay caused by utilizing the local computation resources [3], [4], [5]. As the number of devices and resources continues to increase exponentially as a result of technological advancements, the opportunities to handle computational workloads of devices collaboratively increases as well.

It is only natural to benefit from the computational offloading aspects in the evolving wireless networks towards the network vision of 2030's, simply referred to as 6G [6]. The envisioned architecture will not only integrate the terrestrial, aerial (drone swarms), and space (satellite mega-constellations) networks from connectivity perspective, but it also will serve as a computation platform. 3GPP is working towards the consolidation of the potential impact of low Earth orbit (LEO) mega-constellations and definition of related solutions in Release (Rel.) 16, TR 38.821 covered by 5G Evolution, which also includes connectivity support from LEO satellites, unmanned autonomous vehicles, and high altitude platform station (HAPS) systems. Yet, the full benefit of LEO mega-constellations with advanced routing features and mobile-edge computing (MEC) are expected to be included in the near future [7] as the 3GPP is shaping the network evolution towards 6G in Rel. 21 (expected to be finalized in 2028). The inclusion of MEC and advanced routing features of mega-constellations is expected to be incorporated in Rel. 21. MEC enables the network

components to transfer a part of the computational workload of an over-utilized host to another (group of) underutilized host(s). Furthermore, as computational tasks can be completed at closer proximity to the user, the task completion times and reliability can be significantly improved in the presence of non-terrestrial network elements [8]. Interestingly, the MEC literature for 6G networks rarely handles service and job-specific properties.

Specifically, uncertainties about the communication channel, job decompositions, task dependencies, and task decomposition sizes are not taken into account. This application-agnostic performance restricts the straightforward applicability of a MEC framework for customized services over the vertical domains towards 6G networks. As it will be presented with a detailed literature survey in Section II, the current literature on MEC for 6G networks is based on unrealistic models that limit the applicability of the proposed techniques. However, the benefits of MEC are well-studied in a number of industries and applications such as vehicular networks.

In this study, our goal is to propose a realistic MEC model for 6G networks. We analyze different aspects of computation offloading for scenarios where traditional terrestrial networks are extended by vertical network services offered by non-terrestrial network elements. We exemplify our proposed approach through a simplified network model that is composed of a high-altitude platform station (HAPS) and a base station, as depicted in Figure 1. A HAPS is an aerial vehicle, such as a fixed-wing aircraft, airship, or balloon, that typically operates in the stratosphere at an altitude of around 20 km.

HAPSs are increasingly being used to provide communication infrastructure. An example of this is their use in vertical heterogeneous networks [6]. Due to their high-altitude deployment, HAPS systems have favorable characteristics, including cooling advantages, mobility, and a wide coverage range. These characteristics can benefit computer offloading scenarios as well. The use of a HAPS for computation, caching, and sensing has been studied by the authors in [9]. However, we also need to clarify that the proposed model is also applicable to satellite offloading, or integrated networks composed of aerial, space and ground components of 6G networks.

One primary contribution of our study is that we use communication models designed specifically for the networking properties of vertical networks. Additionally, we take into consideration some crucial properties of the offloaded job, such as inter-dependency between job tasks and the decomposability of the offloaded job. In the literature on computation offloading 6G networks, job properties are rarely considered; when they are considered, they involve oversimplified assumptions. In our study, in addition to analyzing the involvement of HAPS systems in computation offloading, we also discuss important properties of jobs that should not be ignored in such analyses.

In our numerical analyses, we have used our own discrete event simulator developed in Python and provide the source code of this simulator for the benefit of computation offloading researchers.[1] In our simulation implementation, in-line with our contributions, we have included software modules specific to vertical network communication properties of a HAPS and job-to-task decomposition based aspects. The insights offered by the realistic model that we propose provides us an opportunity to design an adaptive offloading strategy, which can significantly reduce the total processing time of computation jobs.

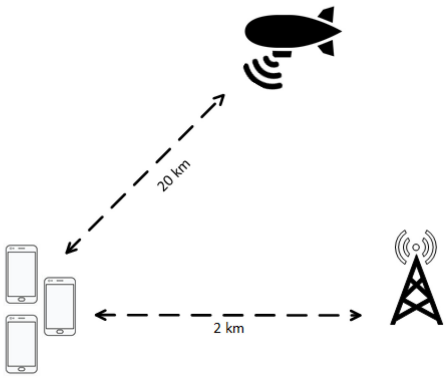


FIGURE 1. Our exemplary system model. The base station and the HAPS can be used to offload the computational tasks. The mobile user can also compute tasks locally.

Our contributions in this paper can be summarized as follows:

- Section II, reviews related work from the computation offloading literature in the vertical networking domain towards 6G and focuses on job decomposition related assumptions that may harm the validity of the studies.
- In Section III, we propose a system model for 6G networks that considers important aspects related to communication and job decomposition. We discuss several assumptions that have been frequently made in the computation offloading literature, and we explain how we handle each of them in detail. Our proposed approach is based on the techniques of the domains where mature MEC solutions are already available. Based on our findings and observations, we also present a realistic communication and computation model.
- We present a discrete event simulation environment based on a HAPS supported case study. We explain the simulation environment's design details in Section IV.
- We propose an offloading strategy that adaptively determines the offloading host in Section III-C and we present the numerical analyses in Section V. Through comparative scenarios we show that the proposed approach reduces the total processing time of computation jobs.

II. BACKGROUND AND RELATED WORK

Computation offloading is a widely studied area in computer networks and cloud computing [3], [4], [5], [10]. In this

1. https://github.com/ovatman/ComputationOffloadingSimulation

TABLE 1. Recent computation offloading studies in the terrestrial context (not containing aerial vehicles).

Study	Job Decomp.	Task Dep.	Task Comp. Sizes	Data Rate
[18]	No	No	Constant	Channel capacity
[16]	No	No	Uniform distr.	Fixed rate
[19]	No	No	Uniform distr.	Channel capacity (uplink and downlink)
[20]	No	No	Normal and Pareto distr.	Channel capacity
[21]	No	Partly	Uniform	Channel capacity
[22]	No	No	Constant (4 classes)	Channel capacity
[23]	Yes (as missions)	Yes	Constant (3 classes)	Channel independent
[24]	No	No	Constant	Channel capacity
[25]	No	No	Constant	Channel capacity
[26]	No	No	Uniform distr.	Channel capacity
[27]	No	No	Exponential distr.	Channel capacity
[28]	No	No	Exponential distr.	Channel capacity (with outage prob.)
[29]	No	Yes	Uniform distr.	Channel independent
[30]	No	Yes	Not available	Channel capacity
[31]	No	Yes	Not available	Channel independent
[13]	Yes	Yes (only comm.)	N/A	Channel capacity
[32]	Yes (as programs)	Yes	Uniform distr.	Channel capacity
[33]	No	Yes	Constant (classes)	Channel independent
[34]	Yes (as app.s)	Yes	Random vectors	Channel capacity
[35]	No	Yes (only sequent.)	Uniform distr.	Channel independent
[36]	No	Yes	Constant (3 classes)	Channel independent

TABLE 2. Recent computation offloading studies containing aerial vehicles in their contexts.

Study	Vehicle	Job Decomp.	Task Dep.	Task Comp. Sizes	Data Rate
[17]	UAV	Data decomp.	No	Constant	Fixed rate
[37]	UAV	Data decomp.	N/A	Constant	Channel capacity
[38]	UAV	N/A	No	Uniform Distr.	Channel capacity
[39]	UAV	Data decomp.	No	Uniform Distr.	Channel capacity
[40]	UAV	Data decomp.	No	N/A	Channel capacity
[41]	UAV	Data decomp.	No	Constant	Channel capacity
[42]	UAV	Data decomp.	N/A	Random	Channel capacity
[43]	HAPS	Data decomp.	No	Constant	Channel capacity
[14]	UAV	N/A	Yes	Uniform Distr.	Channel capacity
This study	HAPS	Task decomp.	Yes	Multi-modal	Variable rates less than channel capacity

section we give an overview of the latest five years' studies on computation offloading in terms of multi-access edge computing and aerial vehicles towards 6G networks. We provide a brief comparison in Tables 1 and 2 enlisting the properties related to the contributions of our studies.

To begin, however, we would like to clarify a major distinction in terms of computation terminology. Even though they are sometimes used interchangeably in the literature we will use the terms *job* and *task* as two separate entities with a one-to-many relation. More specifically, in this paper we refer to a job as a collection of tasks, which may have dependencies between them in terms of computation or data [11], [12], [13]. An important ramification of this definition is that we can break a job down into tasks, but there may be constraints about inter-dependencies of tasks that make up a job.

In reviewing related works in the literature, we focused on the following properties:

- *Domain of study:* We classified the studies that we reviewed into three categories. The first category involved studies that do not specifically refer to aerial or space vehicles; rather they approached the problem from the perspective of terrestrial base stations or by using more abstract or generic system models. The second category involved system models that contained Unmanned Air Vehicles (UAVs) as first order citizens. In the last category, we included studies that focused on systems involving high-altitude and/or non-UAV elements. Hence, non-terrestrial network elements towards the evolution of 6G networks are covered.
- *Job decomposition:* We reviewed the studies to find out if they use any kind of job decomposition during offloading process. The majority of the vertical network studies did not perform any kind of decomposition, although they involved multiple agents offloading the job. They basically either offloaded a job as a whole

or not. Studies with aerial elements typically involved a single agent offloading the jobs, so their approach was to offload the input data over a replica of the job instead of offloading the job itself. This approach was applied even when there were multiple agents that performed the same job, which may be limited according to the computations' domain because of possible data dependencies.

- *Task dependencies:* Task dependencies in the computation offloading process are cumbersome to handle. However, it is also naive to assume jobs can be broken down into pieces arbitrarily, disregarding any kind of possible process or data dependencies between them. Even if we assume that a job is irreducible, it is still realistic to assume that there may be data dependencies among the jobs. Most of the time, arbitrarily distributing input data is costly compared with caching the data and dispatching the jobs at the cached agent. Yet hardly any recent studies from vertical networks literature handle task dependency as a part of their offloading scheme. An exception to this situation is [14], where the authors considered flow dependency between the tasks.
- *Task computation sizes:* We also compared the studies according to the workload distribution they used. We found a variety of approaches used, from constant work-loaded jobs to normal distributed workload. However, it is well known especially in cloud literature that the workload profile changes greatly according to the domain and application area [15]. We may expect to experience a distributed computation scheme fitting a mixture model in case a job decomposition is applied.
- *Data rate:* The majority of the studies in the literature assumed that the data transmission can be achieved by a data rate equal to the channel capacity. Noting that the channel capacity is merely an upper bound for the data rate that can be used to transmit information, the necessity of a more realistic communication model becomes apparent. As noted in Tables 1 and 2, only two of the studies used fixed data rates [16], [17]; however, the communication standards also define multiple transmission rates that change according to the status of the channel. In this work, we consider variable rate transmission as defined by the communication standards.

Tables 1 and 2 presents recent studies from computation offloading literature comparing studies that do not contain aerial vehicles and studies that contain aerial vehicles, respectively.

Especially in recent years, researchers began to focus on building more realistic models that handle inter-task dependencies, input and output sizes and task characteristics especially in the horizontal domain as listed in Table 2. However, lesser number of studies consider such concepts in offloading for vertical networks as Table 2 suggests. Our model is influenced by the studies in the horizontal domain and propose a model for the vertical networks that is able to take those concepts into account.

For instance, task inter-dependency is mostly handled by using task graphs and Directed Acyclic Graphs (DAG) in the literature. We also use a similar approach and used connected components in the task graphs to perform task dispatching.

Another novel concept (for vertical networks) that we use is task classification. In horizontal networks task classification is done in some of the studies in the augmented reality domain [21], [33], in the big data domain [31] and in other domains [32]. There also exist earlier studies that considers differences in job requests and response sizes [19], [22]. In [22] the authors classified computation jobs as either small or big.

In our study not only we use different profiles for computation workload, computation data sizes (these two are frequently separately handled in other studies as well [23], [36], [14]). We also use different profiles for response sizes, since it may not be possible to assume that the input data and/or computation sizes are correlated with response sizes.

In a preliminary study by [13], authors considered communication dependency in their analysis. It may also be possible to consider other types of dependencies, such as process dependencies and data dependencies, among tasks within a job and among jobs that have common domains or users. Specific types of dependencies may be considered to further elaborate the analysis of computation offloading schemes.Here, however, we do not specifically focus on the type, instead we accept any type of dependency to enforce dispatching the related tasks to common agents. Recent horizontal network literature also uses the term "workflow dependency" to broadly refer to mentioned types of dependencies (see last few rows of Table 1).

Some studies, especially in the aerial domain, make dependency tracking during offloading obsolete since the jobs are offloaded to a single UAV [42] and [37]. But these two studies handle all the computation at the UAV level and do not perform any local computation. If the local computation is partially offloaded even to a *single* UAV, arbitrary decomposition becomes susceptible to data dependencies of the job. An example of such a job may be a time series analysis, which may require *integrity* of historical data during computation.

As a summary, it can be seen that recently the horizontal network literature has evolved to cover most of the aspects that we cover in our model. However in the vertical networks and especially for HAPS involved domains such evolution has not yet taken place. In our study we provide a simple yet holistic a model and validate it with realistic workloads based on the cloud workload literature.

III. PROPOSED COMMUNICATION AND COMPUTATION MODEL FOR INCLUSIVE NON-TERRESTRIAL NETWORK ELEMENTS

In this section, we present our model for MEC offloading scenarios for 6G networks based on our observations about

TABLE 3. Symbols used in the definitions, in their order of appearance.

Symbol	Definition
J	An instance of a computation job.
T	Set of tasks that is required to complete a certain job.
D	A lower triangular matrix with Boolean entries that represent inter-dependency between two tasks of a job.
t_i	A task with IDi. Each task is defined with its computational size class, request size class and response size class for task i.
is_i	Input data size for for task i.
os_i	Output data size for for task i.
ps_i	Processing requirement for task i.
$I^{s,l}$	Input data size of the job. It can be either small or large input data.
$O^{s,l}$	Output data size of the job. It can be either small or large output data.
$P^{s,l}$	Processing requirement of the job. It can require either small or large amount of processing power.
$dp_{i,j}$	A Boolean variable that represents presence of a dependency between tasks i and j.
E	Environment of the offloading host.
S	Stations, known to host in its environment.
A	Task assignment matrix that represents the assigned station IDfor each task of a job.
s_j	An task offloadable station with IDj. Index $j = 0$ denotes the host's index (e.g. s_0 is the host itself).
$proc_j$	Processing power of the station.
ul_j	Uplink communication capacity between the host and the station j.
dl_j	Downlink communication capacity between the host and the station j.
$C^{s,m,l}$	Communication link capacity class. It can be either small, medium or large capacity.
d_j	Distance from the host to the station j.
op_j	Operational overhead of the station.
y_j	Type of the station. It can be either `host`, `terrestrial` or `HAPS`.
$a_{i,j}$	A Boolean variable that represents if task i has been assigned to station j or not.
χ_i	The computation characteristic of the offloaded job. This variable can take one of four values explained in Table IV.

the inadequacies of current models proposed in the literature. As previously discussed, previous models either do not contain any job decomposition or they contain oversimplified assumptions about how a job can be decomposed into tasks. Additionally, most of the studies contain debatable assumptions about how job/task characteristics like computation size, data size, and response size should be determined on the basis of the numerical analysis of the proposed offloading schemes.

Before introducing our system model, we present the diagram in Figure 1 that depicts a typical system architecture that will be the subject of the computation offloading scenarios that we will examine. In a typical scenario, we handle the jobs of mobile users in a given area with a combination of terrestrial base stations and base stations mounted on one or more HAPS, UAV, or satellite.

We also present, as a reference, the list of symbols that we use in our system model in Table 3. Below, we use these symbols to refer to specific elements in our system model definition.

A. JOB DECOMPOSITION AND TASK DEPENDENCIES

We begin with a system model presented in Definition 1 that supports decomposing a job into smaller tasks that may contain inter-dependencies between them. In so doing, we define a job as a set of tasks and a triangular matrix with Boolean entries that represent the inter-dependencies between the tasks.

In our task definition, o_i is a Boolean variable that represents the offloading decision for task i. According to this variable's value, the task is either offloaded to the external processing environment of the host or processed by the task's host itself.

Definition 1: Computation model with task characteristics and dependencies

$$\begin{aligned} J &\triangleq \{T, D\} \\ T &\triangleq \{t_0, t_1, \ldots, t_n\} \\ t_i &\triangleq \{is_i, os_i, ps_i\} \\ is_i &\in \left\{I^s, I^l\right\} \\ os_i &\in \left\{O^s, O^l\right\} \\ ps_i &\in \left\{P^s, P^l\right\} \\ D &\triangleq dp_{ij} \in \{0, 1\}^{|T|\times|T|} \end{aligned} \tag{1}$$

Our system model separates request size from response size while considering task properties. Only a handful of studies in recent literature covered in former sections distinguish between the request and response characteristics of computation jobs. There may be some cases, such as image processing, where the request and response have similar sizes (and characteristics), but the case is different if the computation involves transforming the input into a totally different output. While it may not be entirely possible to determine the response size for a task when it is created, previously analyzed distributions and estimations can be used for this purpose. Recent literature performs such distinctions according to domain of the study; for instance for augmented reality domain a distinction is performed according to the stages of the augmented processing [21], [33]. Our study generalizes this approach and presents a more holistic

TABLE 4. Overview of the computation classes that will be assigned to offloading tasks.

t_i class (χ_i)	t_i char.	Definition	Example case
χ_C	$\{I^s, O^s, P^l\}$	Comp. intensive task	prime decomp.
χ_R	$\{I^s, O^l, P^l\}$	Reproductive task	decompression
χ_I	$\{I^l, O^s, P^l\}$	Input processing task	hashing
χ_T	$\{I^l, O^l, P^l\}$	Input transform. task	image filtering
χ_O	$\{-, -, P^s\}$	Comp. small task	simple calc.

$$\begin{array}{c} \\ t_0 \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{array} \begin{array}{c} \begin{array}{cccccc} t_0 & t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{pmatrix} & 1 & 0 & 0 & 0 & 0 \\ & & 0 & 0 & 0 & 0 \\ & & & 0 & 0 & 0 \\ & & & & 1 & 1 \\ & & & & & 1 \\ & & & & & \end{pmatrix} \end{array}$$

FIGURE 2. An example task dependency matrix.

approach since the number of classes in Definition 1 can be expanded to produce a more fine grained classification in Table 4.

Additionally, χ_i represents task characteristics for task i, where each task is classified as one of five different characteristic instances. A characteristics instance contains a tuple of random variables, each having different distribution parameters according to their classification. These variables are defined as task input data size (request size) and task output data size (response size), denoted with I and O random variables. To each of these variables, a superscript s or l is added (s for small; l for large) to indicate the class of the characteristic. Similar approaches can be seen also in previous studies, as covered in related work section.

A very important observation in computationally expensive jobs is it may not be possible to estimate the workload amount or workload characteristics of the job at all by only considering its size. Most of the of current MEC literature affiliates the workload amount for a task with its input data size, which may be possible if the data of the job is packed together with the job. However, especially in cloud environments, data being processed and the computational definition of the job are often separate, which results in incorrect estimations about the computational workload of the job when the size is used.

A possible remedy to address this situation is to model the computational workload of the tasks separately. This approach is not uncommon; there are measures like millions-of-instructions to represent the processing workload. We add a new element to our task characteristic property to model small and large sized computation workloads with three separate P random variables. This parameter is different from both of the size parameters, since the required computational power may not always be correlated with the job's request and response sizes. For instance, some tasks may share common data where there is no need to resend data with each task every time. Take a video processing task as an example: a video file affiliated with a set of processing tasks are uploaded once to an offloading station; the rest of the processing tasks are uploaded separately. These tasks might be small in size, but their computational needs are considerably higher since they all operate on frames of video data.

In Table 4 we present task characteristic classes regarding the amount of computation they require. Due to the task dependencies, however, tasks that require smaller amounts of computation may sometimes be offloaded as well; usually it may be preferred to handle them locally. In the literature there are some studies that employ such a classification approach, such as [44].

We use the term *inter-dependency* to indicate any kind of reason that requires two tasks to be executed in the same host because of a temporal, input-output or data dependency between the tasks. We would like to emphasize that we currently do not consider dependencies that induce an execution order. These types of inter-dependencies transform the problem into a scheduling problem, which we will focus on a future study. For an example of the type of inter-dependency we consider in this study, consider the dependency matrix of a job with six tasks, shown in Figure 2. This matrix is upper triangular, since our definition of dependency is undirected. Examining this matrix it can be seen that t_0 and t_1 constitute a dependent group that should be assigned to the same processing station just like t_3, t_4 and t_5. On the other hand, t_2 does not have any dependencies with any other tasks, which means it can be assigned freely to any stations or the host itself.

Defining computational offloading jobs like this allows us, in a fine grained way, to define possible inter-dependency constraints that may be present for an offloading job. On the one hand, it is possible to utilize this model to mimic fully independent decomposition of tasks, as current literature suggests. On the other hand, it would be possible to define in great detail the specific decomposition constraints for a job, as well as specific properties of a particular task.

Even though there might be differences in minor details, handling task inter-dependencies by using task graphs is a commonly used approach in the recent literature (Tables 1, 2). In contrast, our approach, initially considers undirected dependency graphs to form task clusters [33] from connected components of the graph and perform a cluster-wide dispatching to the resources.

It is important to note that properties of task dependencies may also be diversified according to the context of the system that is being modeled. However, we leave this discussion about the relative value of considering diverse properties of these dependencies to a future study. We would also like to note that the proposed model is generic enough to be used in other contexts as well, by adapting the model parameters appropriately.

B. ENVIRONMENTAL MODEL

Our last set of observations is about the environment used to process the offloaded tasks. For a mobile computing environment, transmissions over the wireless channels are not performed at channel capacity. Only discrete valued rates are allowed. Also, the processing stations that the tasks will be offloaded to need to provide processing power that can be used in the calculation of task completion times.

Definition 2: Processing environment model

$$
\begin{aligned}
E &\triangleq \{J, S, A\} \\
S &\triangleq \{s_0, s_1, \ldots, s_k\} \\
s_j &\triangleq \left\{proc_j, ul_j, dl_j, d_j, op_j, y_j\right\} \\
&\qquad 0 \leq j < |S| \\
&\qquad ul_j, dl_j \in \left\{C^s, C^m, C^l\right\} \\
&\qquad y_j \in \{\texttt{host}, \texttt{terrestrial}, \texttt{haps}\} \\
A &= a_{i,j} \in \{0, 1\}^{|T| \times |S|}
\end{aligned} \tag{2}
$$

In this model, we have four specific high-level components, namely the jobs to be offloaded denoted by J, which has been defined in detail in the previous sections, offloadable processing stations denoted by set S, and the assignment matrix denoted with A.

Set S contains a list of processing stations to which the host may offload the task. Each station denoted by s_j is represented by the processing capacity $proc_j$ of station j. Variables ul_j and dl_j represent the uplink and downlink capacity of the communication channel between the offloading host and the station that the task is being offloaded to. The channel capacity can be small, medium or large depending on the different portions of capacity that can be used at a time by the offloading host. Variable op_j represents the operational delay experienced in a processing station. Operational delay contains all the additional delay to communication and processing times, which may be due to queuing and other operational issues. Finally, y_j variable holds the type of the processing station since the communication time calculation may change due to the type of station. For instance, if the processing station is the host itself, there will be no communication delay. Additionally, the distance will have difference effects on the communication time calculations depending on the type of station (i.e., terrestrial or aerial).

The environment also contains an assignment matrix (A) where each task is assigned to at most one station in the environment. If the task is not assigned to any station, it means that it has not been offloaded. More specifically, $a_{i,j}$ contains a Boolean variable to indicate if task t_i has been assigned to station s_j or not.

This model can be subject to expansion by adding new properties to C, s_j and H variables, such as energy consumption for each processing station, or additional properties for the offloading host. We wanted to build a simple yet extensible model to be able to add new properties that might be the focus of future research.

Algorithm 1 Proposed Computational Offloading Strategy

Require: connected component c from dependency graph
  if $\Sigma_P >> \Sigma_I + \Sigma_O$ **then**
    assign to aerial
  else if $\Sigma_P << \Sigma_I + \Sigma_O$ **then**
    assign to local
  else
    count the number of tasks belonging to each χ in c
    $max\_class \leftarrow max(|\chi_C|, |\chi_R|, |\chi_I|, |\chi_T|, |\chi_O|)$
    if $max\_class = \chi_O$ **then**
      assign to local
    else if $max\_class = \chi_C$ **then**
      assign to aerial
    else
      assign to terrestrial
    end if
  end if

TABLE 5. Channel parameters.

Parameter	Value
Carrier frequency	10 GHz
Bandwidth	10 MHz
UE-to-HAPS Channel	Ricean K, $(K = 10)$
HAPS Transmit Power	10 dB
HAPS Antenna Pattern	ITU-R F.1336-5, *recommends* 3.2.1
HAPS Height	20 km
HAPS Beam	Pointing directly towards UE
Supported Data Rates	{7, 14.1, 21.1, 31.7, 42.3, 64.3, 85.7, 161.9} Mbps

C. OFFLOADING BY INTER-DEPENDENCIES AND TASK CHARACTERISTICS

Our proposed offloading strategy is shown in Algorithm 1. The presented strategy uses the computation classes defined in Table 4. This strategy works for each cluster (connected component) in the dependency graph, so we decide to offload the tasks in the cluster altogether according to the majority of the tasks. Exploring connected components in a graph takes $O(|V| + |E|)$ by known sequential algorithms [45].

In the first steps of the strategy, we compare the total computation requirement of the cluster to the total input and output size of the tasks in the cluster. If either one is substantially larger than the other, we offload to aerial or compute locally, respectively. This step is deemed necessary since for some corner cases the cluster may contain a balanced number of tasks from each computational group but the overall processing requirement of the cluster is substantially larger than the input and output size (or vice versa). This step requires computing the accumulated properties of tasks in the cluster so its computation complexity is $O(V)$.

Another point that needs clarification for the first steps is the definition of "substantially larger." For our application, we use thresholds for the ratio between processing size and input and output size but this comparison can be done in

TABLE 6. Random variable distributions used in our study.

Variable	Distribution	Unit	Distribution parameters
P^s	lognormal	MI	$\mu = 4.32$ and $\sigma = 1.31$
P^l	power-law	MI	$\alpha = 1.5$
$\{I^s, I^l\}$	lognormal	Mbytes	$\mu = 8.08$, $\sigma = 2.58$
$\{O^s, O^l\}$	zipf	Mbytes	$\alpha = 1.7748$
task per job	Pareto	number	$\alpha = 1.517$
job inter-arrival	lognormal	sec.	$\mu = 5.15$ and $\sigma = 0.5$
task graph	Erdös-Renyi $G(n, p)$	undir. graph	$n =$ task per job, $p = 0.05$

many different ways. We leave a discussion of this for a future study.

Contents of the else statement in the strategy counts the majority of the classes that belong to a certain computational class in the cluster and decides where to offload the cluster accordingly. Using majority voting might not result in the best results but as we will discuss in the next section even majority voting provides better results than offloading jobs as a whole.

This step requires iterating through all the tasks in a given cluster for each characteristic, so its computation complexity is $O(V)$. In our study, we use exactly five characteristics, so it does not increase the overall complexity of the computation, but if we assume there might be a varying number of characteristic classes, the computational complexity will be bound to the number of these classes as well. However, it is very unlikely that the number of characteristic classes will surpass the number of tasks for the worst case, so we may safely assume that the computation complexity for this step is $O(V)$.

In terms of computational complexity, the proposed strategy as a whole has $O(|V|+|E|)+O(V)+O(V) = O(|V|+|E|)$ computation complexity. For our case $|V|$ is the number of tasks and $|E|$ changes according to the amount (and characteristics) of inter-dependency among the tasks. To measure an estimate for the actual running time of the proposed strategy, we used Python's NetworkX package and performed connected component extraction. For a graph with 10^4 nodes, using parameters in Table 6 we obtained a running time of less than 100 milliseconds. When we increase the inter-dependency density to 0.9 the running time was still less than one second.

IV. SIMULATION ENVIRONMENT AND EXPERIMENTAL SETUP

The foregoing has allowed us to come up with an architectural model for a job to be offloaded and the tasks contained by the job definition in Definition 1. We also present a preliminary model for the environment that the MEC is going to operate in Definition 2.

For our experiments we use a very simple setup with a single mobile device (or a group of mobile devices with close proximity) that is the source of the generated jobs and tasks where the single source communicates with either a base station within approximately 2 kilometers of distance or with a HAPS station that is approximately within 20 kilometers of distance. A diagrammatic representation of the physical environment used in the experiments is presented in Figure 1. On this setup, we present some numerical values and formulation for the calculation of variables presented in Definitions 1 and 2.

A. STATISTICAL DISTRIBUTIONS

First, we begin by introducing the parameters of the distributions for the random variables belonging to various classes presented in our former definitions. Table 6 presents a list of all the random variable distributions, ranges and parameters that we have used in our experiments. Definitions in this table should be interpreted by the help of the symbol in Table 3 and Definitions 1 and 2.

Despite the obstacles noted above, we have focused on workload studies that might be meaningful from computational offloading perspective, where the workload that has been characterized fits into the definition of computationally intensive cases. Most of the studies that we examined were based on Google cluster data set [46] but there are also studies that focus on different organizational or public clusters.

We start with the computation workload distributions that we use in this study. To be more precise, we only focus on the size of the jobs and tasks in terms of the computational work inherently present in them. Firstly, for almost all the studies that we examined, there is a great emphasis on the distinction between small and considerably larger jobs. This supports our choice of using a bimodal distribution with classes such as P^s and P^l. For instance, in Google cluster data set, the top 20% of the jobs follow power law distributions [47]. On the other hand, a user-based analysis of the same data set shows that, from the users' perspective, task lengths follow lognormal distributions [48]. An additional study from Hadoop also supports the lognormal distribution finding [49]. We have chosen to use a mixture model where small jobs (80% of overall) follow a lognormal distribution and larger jobs (20% of overall) follow a power law distribution for task computation requirement generation. We should also note that most of the studies in this area, including the ones cited above, states that an obvious distribution is not present in their findings and it may also be suitable to perform sampling from the data sets provided above.

Another important parameter in our simulation is the input and output data size of the tasks. In terms of computationally intensive Web-based tasks, workload characterization literature is not very rich on examining directly the request and response sizes of the workloads. Instead, most of the time, they naturally chose to examine the I/O behavior of the tasks. An interesting exception might be the papers that analyze the workload of Dropbox [50], [51]. However, these studies have focused more on I/O heavy workloads which does not fit with our case. Nevertheless, those studies report that

most of the requests are fairly small and follow Pareto-like distributions. On the other hand, studies that fit better into our case, such as in Hadoop and map/reduce jobs, similar findings have been reported [49], [52], [53]. Furthermore, a study of smartphone applications [54] also reported that most of the requests and responses are small in terms of data size. We chose to use distributions mentioned in a very recent study [53]. It was reported here that the *read data* sizes usually follow a lognormal distribution while *written data* sizes follow a Zipf distribution.

Examining the distributions for processing requirement, input and output sizes of the computational jobs, we continue with computational job and task generation. Studies in the literature indicate that, most of the time, workloads consist of small jobs with single tasks, even for the Google cluster data [55]. However, in an Hadoop environment, the rate of smaller jobs, containing less than 10 tasks drops down to 40% of the workload [49]. These results are in-line with the majority of tasks having small I/O values, so we use a Pareto distribution in generating the number of tasks in a job.

For the task dependency graph generation, a number of techniques are often used in the literature for similar simulation studies to ours [56]. However, in our study we do not need a directed graph since we will not be focusing on the scheduling of the tasks and hence the sequential relations between the tasks may be ignored. We use the undirected task graphs generated over Erdös-Renyi $G(n, p)$ generation and use connected components of this graph as offloadable clusters. Note that a cluster represents a subset of tasks which are interdependent. Examining more realistic task inter-dependencies will improve the quality of our simulation, but we leave this issue for a future study and pseudo-randomly assign dependencies between the tasks by following the randomly generated dependency graph.

In terms of selection of χ_i class of the tasks that are going to be involved in job generation, there are a few studies that use this perspective, such as [44]. However it has been commonly mentioned that the majority of tasks are smaller in terms of computation and data size. We apply our first distinction in terms of χ_O and χ_C,χ_R,χ_I,χ_T classes, or in other words between the tasks that require a small amount of computation and those that require a large amount of computation. Distribution of these classes are explained above, but briefly we use a mixture model where 80% of the tasks are small in terms of computation requirements. For the input and output sizes, we just sample from the distributions explained earlier because there are no studies that clearly state if there is a relation between those classes, to the best of the authors' knowledge. In our classification we classify the tasks that belong to the 20% tail of the distribution as *large* and the rest as *small*.

Finally, we also cover job interarrival times. A relatively more recent study reports that the jobs arrive in lognormal distribution to Alibaba's cloud servers [53], while for the Google case it seems that they follow not a particular distribution at all [57]. Another interesting result from the literature focuses on task interarrivals and suggests that interdependent tasks arrive at closer times, with a loglogistic distribution in differences of arrival times [58]. Since we will be offloading task clusters inside a job altogether we believe following lognormal job generation should be appropriate for our case.

B. RUNTIME CALCULATIONS

In our simulations, we have calculated a task's end-to-end processing time(τ_i) as in Definition 3. In this definition, each task's end-to-end processing time is comprised of three components τ^{com}, τ^{proc} and τ^{oper} which refer to communication, processing and operation time respectively. Each of these variables are affiliated with tasks and station IDs since the times may vary according to the task and station that the task was processed in.

Definition 3: End-to-end processing time of a task

$$\tau_i = \tau_{i,j}^{com} + \tau_{i,j}^{proc} + \tau_{i,j}^{oper} \tag{3}$$

Next, we expand on these components by drawing on our job-task model and environment model. Before the expansion, we should note that we abstract away operational overhead ($\tau_{i,j}^{oper}$) by using a random distribution to estimate the overhead presented in Table 6. This estimation, of course, may have important ramifications, but we leave the details of this to a future study.

Definition 4: Communication time components of a task

$$\begin{aligned} \tau_{i,j}^{com} &= (a_{i,0})\left(\tau_{i,j}^{req} + \tau_{i,j}^{resp}\right) \\ \tau_{i,j}^{req} &= \frac{is_i}{ul_j} + \frac{d_j}{c} \\ \tau_{i,j}^{resp} &= \frac{os_i}{dl_j} + \frac{d_j}{c} \end{aligned} \tag{4}$$

In this set of equations, we define communication time of a task by adding its request's communication time ($\tau_{i,j}^{req}$) to its response's communication time ($\tau_{i,j}^{req}$). However, this calculation is done only when it is decided that the task should be offloaded to the external processing stations. This decision is modeled by multiplying $a_{i,0}$ value from the assignment matrix A. This value contains both the propagation and transmission time of the task to be offloaded.

In our processing time calculation, we also show how the request and response communication times are calculated as in definition 5. In these equations, ul_j and dl_j represents the uplink and downlink data rates. The propagation time is calculated by the fraction of the distance between the host and the j-th MEC server (d_j) to the speed of light (c).

Definition 5: Processing time components of a task

$$\tau_i^{proc} = \left(\frac{ps_i}{proc_j}\right) \tag{5}$$

We calculate the processing time by dividing the processing requirement of the task (ps_i) by the processing capability of the offloaded station ($proc_j$).

With the help of the previous models and time calculations defined above, we may define constraints for our computational offloading models below. In this definition we use Ω to represent the set of constraints.

Definition 6: MEC offloading constraints

$$\Omega = \left\{ \begin{array}{l} o_i = \sum_{j=0}^{|S|} a_{i,j}, \\ dp_{t_1,t_2} \implies \left(o_{t_1} = o_{t_2}\right) \wedge \left(a_{t_1} = a_{t_2}\right) \end{array} \right\} \quad (6)$$

- $o_i = \sum_{j=1}^{|S|} a_{i,j}$ is used to represent each offloaded task that should be assigned to exactly one station. Here the value of o_i will be *true* or 1 whenever it is decided to be offloaded. For this case, the sum of the row in the assignment matrix should be exactly one.
- $dp_{t_1,t_2} \implies (a_{t_1} = a_{t_2})$ ensures that dependent tasks are assigned to the same processing element: either processing stations or the host. It checks this by comparing the offloading decision variables and assignment matrix rows of dependent tasks.

Constraints presented in Definition 6 can be seen as a minimal set of constraints that can be used to demonstrate the validity of the proposed model. In different domains, under different assumptions this set of constraints can be expanded to exhaustively define the environmental assumptions.

For instance many of the studies presented in the related work section consider energy related constraint to account for the energy consumption in participating devices. Another example might be given from vehicular domain where offloading time is very limited. Studies in vehicular domain mostly use computation time limits to ensure that offloading is completed within a determined amount of time. Similar constraints arise also in studies that involve drones as well. Resource and/or cost based constraints can also be introduced to account for the limited presence of resources in offloading studies.

Lastly, we should also note that during our simulations we assume the operational delay and storage based operations of the tasks are included in the computational workload of the tasks. Even though for specific domains these parameters may more seriously effect the offloading operation, we chose to focus on the perspective of decisions related to communication delay versus computation workload trade-offs.

We would like to emphasize that in our study we don't perform any optimization; instead we demonstrate the validity of the proposed model via simulations with realistic workload distributions. We try to provide only a very basic set of constraints so that our model can be expanded for different domains by additional constraint.

V. SIMULATION RESULTS AND DISCUSSION

Before presenting the results of the simulation experiments, we are going to briefly explain the development environment of our simulation. In our simulations we have developed a simple discrete time simulation using Python as programming language. We have used `networkx`[2] and `numpy`[3] libraries to generate random graphs and probabilistic distributions respectively. A single run of an experimental simulation for realistic amount of tasks/jobs presented below takes about one minute in a powerful PC with 16 GB of memory and ninth generation core i7 Intel processor.

TABLE 7. Properties of the processing infrastructure.

	Local	Terrestrial	Aerial (HAPS)
Proc. power	10^5MIPS	10^6MIPS	10^7MIPS
Data rate	-	800 Mbit/s	calculated via Table V
Distance	-	2 km.	20 km.

TABLE 8. Relative properties of the job's tasks used in experiments.

Task clusters	I	O	P
Tasks 1-2-3	X	X	X
Tasks 4-5-6	100X	100X	1000X
Tasks 7-8-9	X	X	1000X

In our experiments, when resolving task interdependencies we follow a non-deterministic approach. During the execution of interdependent tasks, the execution of the dependent task is blocked until the blocking task finishes and thus the execution order is dynamically determined. In real-world there might even be longer chain of dependencies that affect the run-time of tasks but our experiments show that even such a simple case provides a significant difference. We repeat each of our experiment twenty times and use average values in order to smooth out the random errors; we provide error bars in our charts for the cases where significant.

A. SIMULATION WITH A BASIC SET OF JOBS

In our first set of experiments, we use a basic job-task model with a single job comprised of nine different tasks and a typical infrastructure configuration for more comprehensive results. Table 8 presents the relative sizes for input, output, and the processing requirements of each task of the job used in our experiments.

In Table 8, each row represents a task cluster that contains three interdependent tasks with identical properties. For instance, for our first experiment we set the input (I) and output (O) unit size as 1 MB, results in the tasks in the second cluster having an input and output size of 100 MB. For this set of experiments, we aim to offload (or process locally) all the tasks in the single job at once in the beginning of the simulation.

For this set of experiments, we use the parameters in Table 7 for the properties of the processing infrastructure. We use ten times more processing power in the terrestrial station compared to local computation power and again ten

2. https://networkx.org/
3. https://numpy.org/

times more processing power in the HAPS compared to the terrestrial station. The terrestrial station is assumed to have a communication link with a data rate of 800 Mbit/s and to be located two kilometers away from the local processor, whereas the HAPS is located at a distance of 20 kilometers and the data rate is calculated with parameters in Table 5, the values of which are adapted from [59]. The supported data rates are determined according to the provided rates, according to 3GPP TS 36.211.

In our first experiment, we fix the unit I and O size to 1 MB and change the unit P size from 10^2 to 10^6 to observe the effect of using larger sized tasks in terms of processing requirements. The smallest unit size of 10^2 results in relatively smaller tasks to be completed in approximately 10^{-3} seconds locally and larger tasks are completed in approximately one second locally. By contrast, for the largest unit size of 10^6, a small task is completed in approximately 10 seconds locally and a large task completes approximately in 10^4 seconds ($\sim$ 2.7 hours). With a HAPS, however, for the largest unit size of 10^6 a small task is completed within 10^{-1} seconds and large tasks are completed in approximately in 100 seconds.

Figure 3 presents the results from our first set of experiments. The label "PROPOSED" represents the proposed strategy to consider the task characteristics in the cluster during offloading. As expected, the local computation keeps growing as the tasks become more computationally intensive. Yet for terrestrial and aerial stations, the increase in completion time keeps stable for a few steps since the processing time of the job is surpassed by the communication time of the job by a great extent. This results in the communication time of the job dominating the end-to-end processing time. But after a few steps, as the tasks in the job get larger, the terrestrial station and HAPS start to show a similar increase in end-to-end processing time.

Figure 3 shows the proposed strategy provides, at worst, as good performance as the best station in each experiment. This is due to the strategy of offloading according to the job characteristics and selecting the most appropriate station for the job to be offloaded to.

In our second experiment, we fix the unit P size to 10^4 MI and change the unit I and O sizes from 0.1 MB to 10^3 MB to observe the effect of using larger sized tasks in terms of input and output size. For the fixed task processing size in this set, a small task can be processed in 0.1 seconds locally, whereas a big task can be processed in 100 seconds locally and one second in HAPS.

Figure 4 presents the results from our second set of experiments. As expected, the local computation is observed to be stable since it is not effected by the I and O size. On the other hand for terrestrial station the increase in completion time keeps stable for a few steps since the processing time of the job surpasses the communication time of the job by a great extent. Figure 4 also shows the proposed strategy provides, at worst, as good performance as the best station in each experiment.

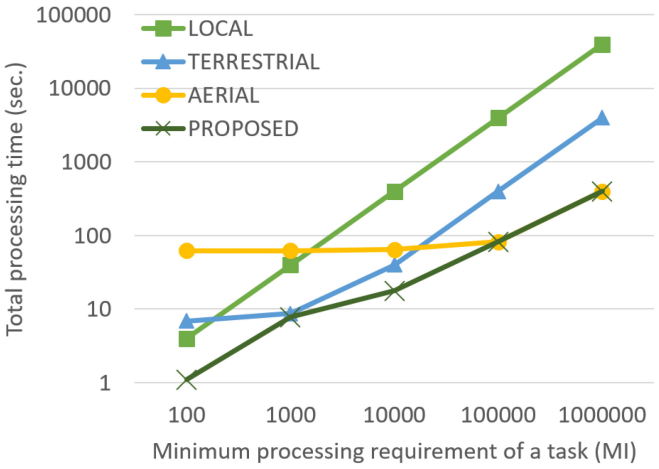


FIGURE 3. The total processing time vs. the processing requirements of a task.

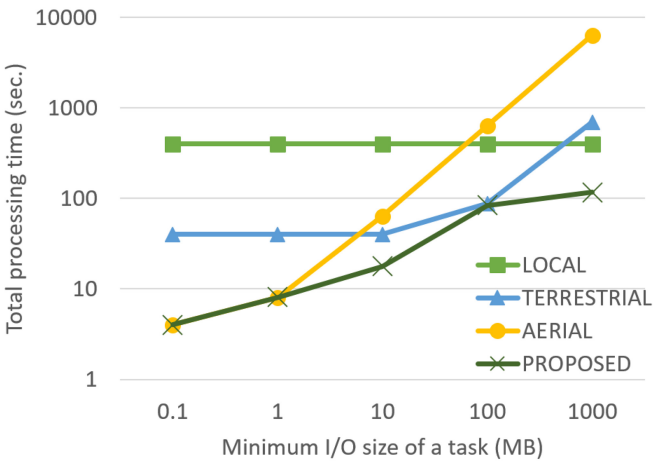


FIGURE 4. The total processing time vs. I/O size of a task.

B. SIMULATION WITH REALISTIC JOBS

By using the distributions presented in Table 6 and explained in Section IV we repeat our experiments by using station configurations differing only by varying relative processing powers.

For the results presented in Figure 5, each tick in the x-axis presents a different base and relative processing power between the processing stations. Each triplet in the tick label represents the processing power of each station in MIPS with respect to a unit processing power. The first element in the triplet is the processing power of the local station. The second element is terrestrial station's processing power, and the third is the HAPS' processing power. The unit processing power we use is 10^5 MIPS, which corresponds to the MIPS value of an early Intel Core i7 processor.

In our experiments we limit the I and O size to be at most one gigabyte when the distribution used (very rarely) produces a larger value. For the processing power of the tasks being used, we scale our distribution in two different ways, so that for Figure 5(a) the largest job is 10^8 MI in processing size, whereas for Figure 5(b) the largest job is 10^9 MI in processing size.

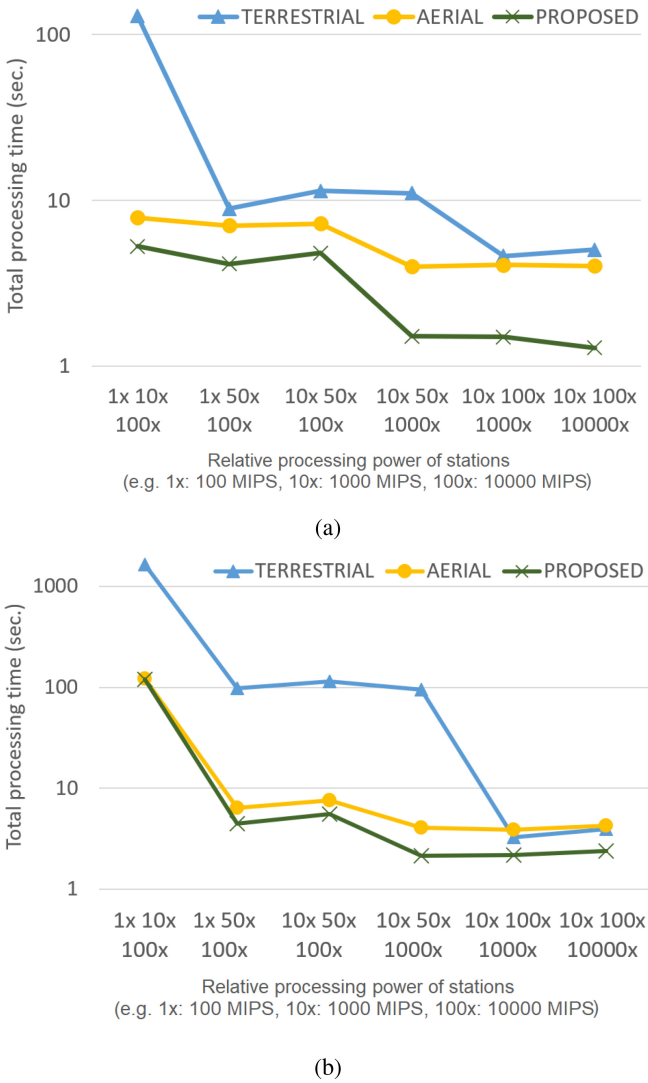


FIGURE 5. Offloading jobs by the proposed model (a) Results for largest P of 10^8 MI, (b) Results for largest P of 10^9 MI.

In Figure 5, we have excluded the results of the local computation, since it always produced the worst results for the large number of jobs we use in our experiments. Results in Figure 5 (a) and (b) show that the proposed strategy can reduce the total processing time than simply offloading all the jobs to any of the stations. The offloading decisions in the proposed strategy rely on a realistic model that performs clustering by inter-dependencies instead of arbitrarily dividing a job for offloading.

C. COMPARISON WITH PER-TASK AD-HOC ASSIGNMENT

In our experiments, as described earlier, we handle dependencies between tasks in a non-deterministic way, where the dependent task is suspended until the execution of a dependent task finishes. In this section we present a comparison between performing the task assignment in a dependency-cluster based way, as proposed in this paper, and performing the assignment considering the characteristics between them but disregarding the dependencies between the tasks (we relax the second term of Definition 6). As a base-line we also included the random assignment where we perform the assignment without even considering task characteristics.

TABLE 9. Threshold values used in experiments.

	Parameters in Fig. 7a		
	Local	Terrestrial	Aerial
BY IO	> 5 GB I/O	< 5 GB I/O	< 1 GB I/O
BY PROC	$< 50 \times 10^6$ MI	$< 100 \times 10^6$ MI	$> 100 \times 10^6$ MI
BY RATIO	< 1000	< 10000	> 10000
	Parameters in Fig. 7b		
	Local	Terrestrial	Aerial
BY IO	> 10 GB I/O	< 10 GB I/O	< 100 MB I/O
BY PROC	$< 10^5$ MI	$< 10^7$ MI	$> 10^7$ MI
BY RATIO	< 10	< 1000	> 1000

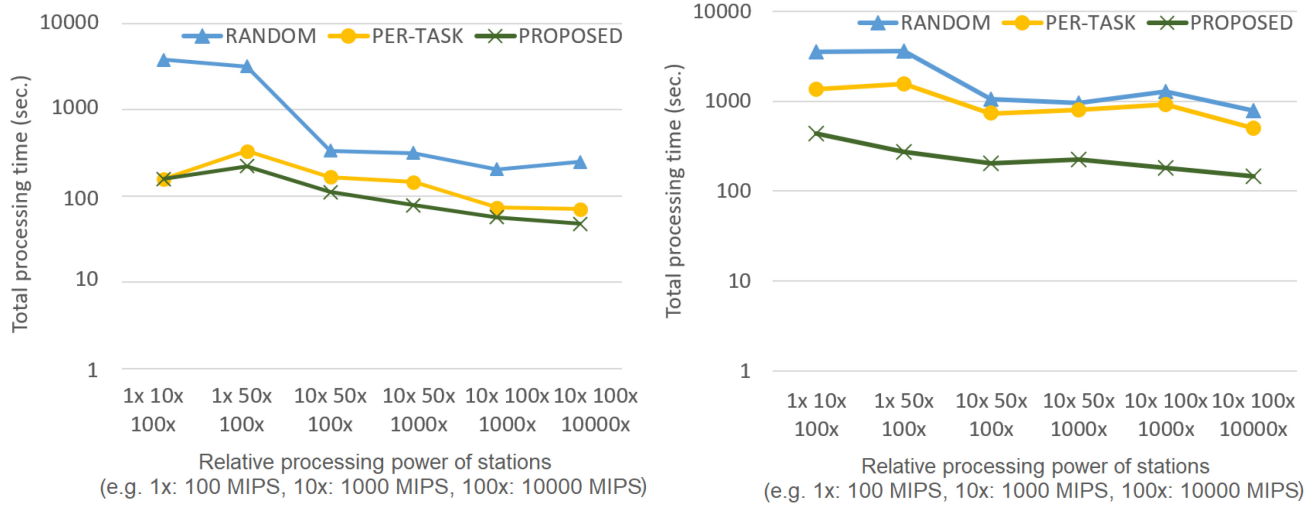
In Figure 6 we present the proposed model's performance over the baseline (random) and ad-hoc (assignment without dependency consideration) task assignment. Naturally, when the inter-dependencies are considered, a more performance task assignment is achieved. Moreover, as typical I/O size of the task increases, ignoring the inter-dependencies is task assignment result in a larger performance degradation. This situation is due to the increase in the communication time between the base station and the computing stations which also increases the suspended time of the inter-dependent task.

D. COMPARISON WITH OTHER OFFLOADING APPROACHES

In Figure 7 we present performance comparison of our approach with some other possible task dispatching approaches. Very briefly, the proposed approach uses the most dominant task characteristic in a cluster to decide where to offload the cluster. In Figures 7(b) and 7(a) we use varying values for thresholds over (a) the total amount of input and output data size of the cluster (BY IO) (b) the total processing requirement of the cluster (BY PROC) and (c) ratio between total processing requirement and total I/O size of the cluster to decide where to offload the whole cluster (BY RATIO). The selected set of parameters are given in Table 9.

Figure 7(a) uses an initial setting of the mentioned thresholds which resulted in mostly worse performance than the proposed approach, which corresponds to the first set of parameters. An important point to note here is that the proposed algorithm *counts* the dominant task characteristic in a cluster and does not need to set any thresholds in this final decision procedure. Other approaches need threshold configuration that may need effort to determine the optimal values for each different setting.

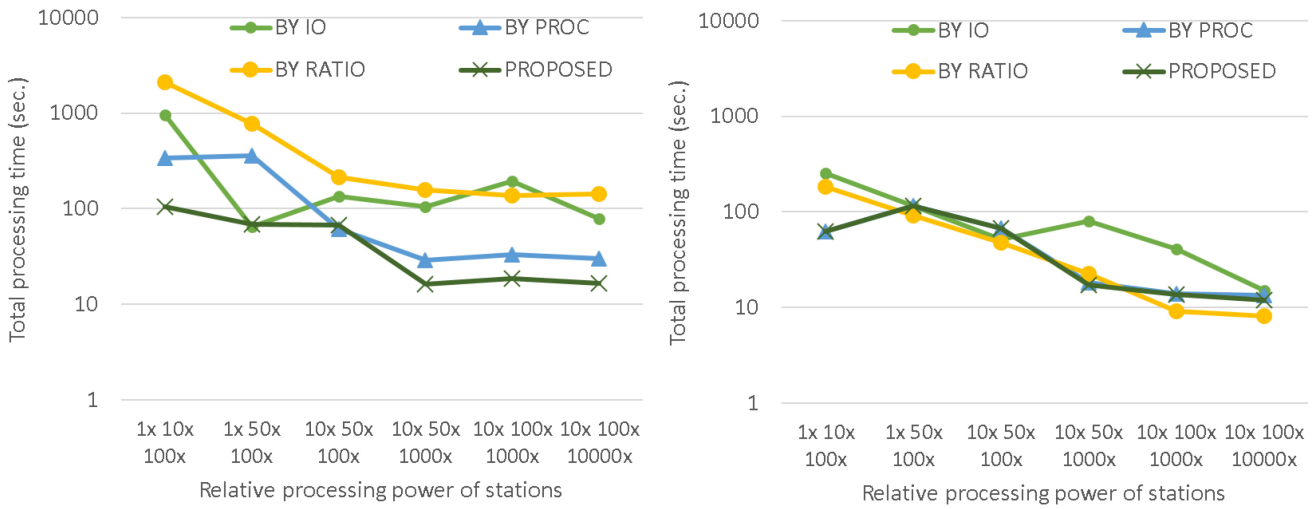
Figure 7(b) present the results after such an effort of trials and errors in setting thresholds for the competitive approaches, by using the second set of parameters. As seen, the proposed approach still performs as good as the competitive approaches. In a real world setting most of the



(a) A maximum I/O of 1 GB is allowed

(b) A maximum I/O of 10 GB is allowed

FIGURE 6. Comparison of per-task offloading with relatively growing maximum input/output sizes.



(a) Using the first set of parameters (see Table IX)

(b) Using the second set of parameters (see Table IX)

FIGURE 7. Comparison of the proposed approach with decision by I/O sizes, by processing requirements, and by a ratio between processing requirements and I/O size.

parameters relating to offloading domain (such as relative processing power of the offloading stations, communication quality, etc.) can change in time, it may be hard to obtain optimal parameter configurations for competitive approaches and rapidly adapt them to these changes. However, proposed approach is more robust in this sense, it does not need any special parameter tweaking for changing environment conditions.

E. COMPARISON WITH FULL CHANNEL CAPACITY ASSIGNMENT

Our experiments take into consideration the channel capacity assignment specific to HAPS-based case study that we apply our model on. We experiment with the channel capacity assignment to compare the case where maximum channel capacity is used with the HAPS communication case in our experiments. Figure 8 presents the case where the maximum channel capacity is used for a basic set of experiments that we perform in our study. As can be seen from thus figure, by using the maximum rate possible achieves in a slightly better performance than using the capacity rate assignment model in our model, as expected.

VI. CONCLUSION AND FUTURE WORK

This paper highlighted the realistic implications for modeling and provided performance metrics for task completion in the presence of wireless fading channels for 6G networks, exemplified via a HAPS-powered multi-access edge computing scenario. We proposed a realistic task offloading model through four key observations, based on measurements and

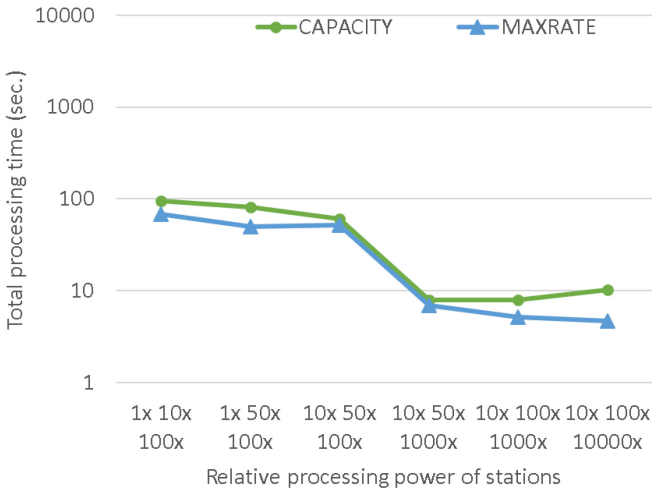


FIGURE 8. Comparison with full channel capacity usage.

insights provided in the literature. Our observations were that;

- Tasks cannot be divided into infinitesimal intervals;
- Request and the response size distributions of real-world tasks may independently vary;
- The computational load of each task is not necessarily linearly proportional to the request frame size;
- Data transmission rates need to be selected from a set of discrete values, in accordance with the capacity of the wireless channel.

Our novel communication and computation offloading model accurately validates these observations, specifically for the presence of non-terrestrial network elements. We also introduced a dynamic offloading algorithm that reflects the characteristics of the proposed model. Numerical results demonstrates that the total processing time could be significantly reduced with respect total processing at the local host, the base station of the HAPS system. As future, we plan to extend the network architecture to include multiple users and multiple networking elements. Additionally, we plan to investigate energy efficiency aspects of the proposed computational offloading approach.

Since the model proposed in this study targets an abstract level, additional open problems include expansion of our model to meet the needs of specific application domains. Different task characteristic classes and statistical distributions may be chosen to address specific domains such as more I/O intensive applications. Moreover, different domains may need focusing different aspects of task inter-dependency resolution and/or scheduling of tasks. For instance, applications in storage domain contain tasks with heavy I/O based dependencies where applications of more time-critical nature need adjustments based on how the tasks are scheduled. Our model can be used to draw architectural guidelines on building domain specific approaches in 6G computation offloading. We believe such models with greater accuracy will speed up the way to future MEC deployments, making them a part of our daily lives.

REFERENCES

[1] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive offloading for pervasive computing," *IEEE Pervasive Comput.*, vol. 3, no. 3, pp. 66–73, Jul.–Sep. 2004.

[2] K. Yang, S. Ou, and H.-H. Chen, "On effective offloading services for resource-constrained mobile devices running heavier mobile Internet applications," *IEEE Commun. Mag.*, vol. 46, no. 1, pp. 56–63, Jan. 2008.

[3] L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1584–1607, Aug. 2019.

[4] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.

[5] K. Akherfi, M. Gerndt, and H. Harroud, "Mobile cloud computing for computation offloading: Issues and challenges," *Appl. Comput. Inform.*, vol. 14, no. 1, pp. 1–16, 2018.

[6] G. K. Kurt et al., "A vision and framework for the high altitude platform station (HAPS) networks of the future," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 729–779, 2nd Quart., 2021.

[7] L. Bariah et al., "A prospective look: Key enabling technologies, applications and open research topics in 6G networks," *IEEE Access*, vol. 8, pp. 174792–174820, 2020.

[8] T. Darwish, G. K. Kurt, H. Yanikomeroglu, G. Senarath, and P. Zhu, "A vision of self-evolving network management for future intelligent vertical HetNet," *IEEE Wireless Commun.*, vol. 28, no. 4, pp. 96–105, Aug. 2021.

[9] Q. Ren, O. Abbasi, G. K. Kurt, H. Yanikomeroglu, and J. Chen, "Caching and computation offloading in high altitude platform station (HAPS) assisted intelligent transportation systems," *IEEE Trans. Wireless Commun.*, early access, May 10, 2022, doi: 10.1109/TWC.2022.3171824.

[10] T. Ovatman, M. E. Tırnakçı, and A. Yılmaz, "Utilizing HAPS deployed data centers in offloading cloud workloads," in *Proc. Int. Conf. Comput. Sci. Comput. Intell. (CSCI)*, 2021, pp. 386–391.

[11] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, "GRAPHENE: Packing and dependency-aware scheduling for data-parallel clusters," in *Proc. USENIX Symp. Oper. Syst. Design Implement.*, 2016, pp. 81–97.

[12] R. Han et al., "Workload-adaptive configuration tuning for hierarchical cloud schedulers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2879–2895, Dec. 2019.

[13] X. Zhang and S. Debroy, "Adaptive task offloading over wireless in mobile edge computing," in *Proc. ACM/IEEE Symp. Edge Comput.*, 2019, pp. 323–325.

[14] S. Zhu, L. Gui, D. Zhao, N. Cheng, Q. Zhang, and X. Lang, "Learning-based computation offloading approaches in UAVs-assisted edge computing," *IEEE Trans. Veh. Technol.*, vol. 70, no. 1, pp. 928–944, Jan. 2021.

[15] M. C. Calzarossa, L. Massari, and D. Tessera, "Workload characterization: A survey revisited," *ACM Comput. Surveys*, vol. 48, no. 3, pp. 1–43, 2016.

[16] E. Meskar, T. D. Todd, D. Zhao, and G. Karakostas, "Energy efficient offloading for competing users on a shared communication channel," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2015, pp. 3192–3197.

[17] S. Jeong, O. Simeone, and J. Kang, "Mobile edge computing via a UAV-mounted cloudlet: Optimization of bit allocation and path planning," *IEEE Trans. Veh. Technol.*, vol. 67, no. 3, pp. 2049–2063, Mar. 2018.

[18] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[19] J. Guo, Z. Song, Y. Cui, Z. Liu, and Y. Ji, "Energy-efficient resource allocation for multi-user mobile edge computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2017, pp. 1–7.

[20] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018.

[21] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint computation offloading and user association in multi-task mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 67, no. 12, pp. 12313–12325, Dec. 2018.

[22] S. Mu, Z. Zhong, D. Zhao, and M. Ni, "Joint job partitioning and collaborative computation offloading for Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 1046–1059, Feb. 2019.

[23] F. Sun et al., "Cooperative task scheduling for computation offloading in vehicular cloud," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 11049–11061, Nov. 2018.

[24] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[25] E. El Haber, T. M. Nguyen, and C. Assi, "Joint optimization of computational cost and devices energy for task offloading in multi-tier edge-clouds," *IEEE Trans. Commun.*, vol. 67, no. 5, pp. 3407–3421, May 2019.

[26] Z. Kuang, L. Li, J. Gao, L. Zhao, and A. Liu, "Partial offloading scheduling and power allocation for mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6774–6785, Aug. 2019.

[27] L. Li, T. Q. Quek, J. Ren, H. H. Yang, Z. Chen, and Y. Zhang, "An incentive-aware job offloading control framework for multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 20, no. 1, pp. 63–75, Jan. 2021.

[28] L. Li, M. Siew, and T. Q. Quek, "Learning-based pricing for privacy-preserving job offloading in mobile edge computing," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, 2019, pp. 4784–4788.

[29] F. Liu, Z. Huang, and L. Wang, "Energy-efficient collaborative task computation offloading in cloud-assisted edge computing for IoT sensors," *Sensors*, vol. 19, no. 5, p. 1105, 2019.

[30] S. Pan, Z. Zhang, Z. Zhang, and D. Zeng, "Dependency-aware computation offloading in mobile edge computing: A reinforcement learning approach," *IEEE Access*, vol. 7, pp. 134742–134753, 2019.

[31] X. Xu et al., "A computation offloading method over big data for IoT-enabled cloud-edge computing," *Future Gener. Comput. Syst.*, vol. 95, pp. 522–533, Jun. 2019.

[32] S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 19, no. 7, pp. 4947–4963, Jul. 2020.

[33] T. Braud, Z. Pengyuan, J. Kangasharju, and H. Pan, "Multipath computation offloading for mobile augmented reality," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. (PerCom)*, 2020, pp. 1–10.

[34] L. Chen, J. Wu, J. Zhang, H.-N. Dai, X. Long, and M. Yao, "Dependency-aware computation offloading for mobile edge computing with edge-cloud cooperation," *IEEE Trans. Cloud Comput.*, early access, Nov. 11, 2020, doi: 10.1109/TCC.2020.3037306.

[35] L. Fan, X. Liu, X. Li, D. Yuan, and J. Xu, "Graph4Edge: A graph-based computation offloading strategy for mobile-edge workflow applications," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, 2020, pp. 1–4.

[36] A. Naouri, H. Wu, N. A. Nouri, S. Dhelim, and H. Ning, "A novel framework for mobile-edge computing by optimizing task offloading," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 13065–13076, Aug. 2021.

[37] Y. Du, K. Wang, K. Yang, and G. Zhang, "Energy-efficient resource allocation in UAV based MEC system for IoT devices," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2018, pp. 1–6.

[38] G. Lee, W. Saad, and M. Bennis, "Online optimization for UAV-assisted distributed fog computing in smart factories of industry 4.0," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2018, pp. 1–6.

[39] Q. Hu, Y. Cai, G. Yu, Z. Qin, M. Zhao, and G. Y. Li, "Joint offloading and trajectory design for UAV-enabled mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1879–1892, Apr. 2019.

[40] J. Zhang et al., "Computation-efficient offloading and trajectory scheduling for multi-UAV assisted mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 2114–2125, Feb. 2020.

[41] N. N. Ei, M. Alsenwi, Y. K. Tun, Z. Han, and C. S. Hong, "Energy-efficient resource allocation in multi-UAV-assisted two-stage edge computing for beyond 5G networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 9, pp. 16421–16432, Sep. 2022.

[42] Y. K. Tun, Y. M. Park, N. H. Tran, W. Saad, S. R. Pandey, and C. S. Hong, "Energy-efficient resource management in UAV-assisted mobile edge computing," *IEEE Commun. Lett.*, vol. 25, no. 1, pp. 249–253, Jan. 2020.

[43] S. Wang et al., "Federated learning for task and resource allocation in wireless high-altitude balloon networks," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17460–17475, Dec. 2021.

[44] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, "Analysis, modeling and simulation of workload patterns in a large-scale utility cloud," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 208–221, Apr.–Jun. 2014.

[45] J. Iverson, C. Kamath, and G. Karypis, "Evaluation of connected-component labeling algorithms for distributed-memory systems," *Parallel Comput.*, vol. 44, pp. 53–68, May 2015.

[46] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: Format+schema," Google Inc., Menlo Park, CA, USA, White Paper, 2011.

[47] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. ACM Symp. Cloud Comput.*, 2012, pp. 1–13.

[48] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, "An approach for characterizing workloads in Google cloud to derive realistic resource utilization models," in *Proc. IEEE Int. Symp. Service-Oriented Syst. Eng.*, 2013, pp. 49–60.

[49] Z. Ren, J. Wan, W. Shi, X. Xu, and M. Zhou, "Workload analysis, implications, and optimization on a production hadoop cluster: A case study on Taobao," *IEEE Trans. Services Comput.*, vol. 7, no. 2, pp. 307–321, Apr.–Jun. 2013.

[50] I. Drago, M. Mellia, M. M. Munafo, A. Sperotto, R. Sadre, and A. Pras, "Inside dropbox: Understanding personal cloud storage services," in *Proc. Internet Meas. Conf.*, 2012, pp. 481–494.

[51] G. Gonçalves, I. Drago, A. P. C. Da Silva, A. B. Vieira, and J. M. Almeida, "Modeling the dropbox client behavior," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2014, pp. 1332–1337.

[52] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads," 2012, *arXiv:1208.4174*.

[53] Z. Ren, W. Shi, J. Wan, F. Cao, and J. Lin, "Realistic and scalable benchmarking cloud file systems: Practices and lessons from AliCloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3272–3285, Nov. 2017.

[54] D. Zhou, W. Pan, W. Wang, and T. Xie, "I/O characteristics of smartphone applications and their implications for eMMC design," in *Proc. IEEE Int. Symp. Workload Characterization*, 2015, pp. 12–21.

[55] P. Minet, E. Renault, I. Khoufi, and S. Boumerdassi, "Analyzing traces from a Google data center," in *Proc. 14th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, 2018, pp. 1167–1172.

[56] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J.-M. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *Proc. Int. ICST Conf. Simulat. Tools Techn. (SIMUTools)*, 2010, pp. 1–10.

[57] S. Di, D. Kondo, and F. Cappello, "Characterizing and modeling cloud applications/jobs on a Google data center," *J. Supercomputing*, vol. 69, no. 1, pp. 139–160, 2014.

[58] D.-C. Juan, L. Li, H.-K. Peng, D. Marculescu, and C. Faloutsos, "Beyond poisson: Modeling inter-arrival time of requests in a datacenter," in *Proc. Pacific-Asia Conf. Knowl. Disc. Data Min.*, 2014, pp. 198–209.

[59] G. K. Kurt and H. Yanikomeroglu, "Communication, computing, caching, and sensing for next-generation aerial delivery networks: Using a high-altitude platform station as an enabling technology," *IEEE Veh. Technol. Mag.*, vol. 16, no. 3, pp. 108–117, Sep. 2021.