Scheduling in Visual Fog Computing: NP-Completeness and Practical Efficient Solutions

Abstract

The *visual fog* paradigm envisions tens of thousands of heterogeneous, camera-enabled edge devices distributed across the Internet, providing live sensing for a myriad of different visual processing applications. The scale, computational demands, and bandwidth needed for visual computing pipelines necessitates offloading intelligently to distributed computing infrastructure, including the cloud, Internet gateway devices, and the edge devices themselves. This paper focuses on the visual fog scheduling problem of assigning the visual computing tasks to various devices to optimize network utilization. We first prove this problem is NP-complete, and then formulate a practical, efficient solution. We demonstrate subminute computation time to optimally schedule 20,000 tasks across over 7,000 devices, and just 7-minute execution time to place 60,000 tasks across 20,000 devices, showing our approach is ready to meet the scale challenges introduced by visual fog.

1 Introduction

Visual data are collected everywhere nowadays. Surveillance cameras have been around every street corner across the world. There are more than 4 million cameras in UK and over 20 million in China. This can serve as the foundation of smart cities for physical security, smart transportation, etc. There are also increased number of cameras in retail stores for customer analytics and directed advertising. Cameras also penetrated into more and more households for security, senior care, baby care, etc. IHS forecasted that the amount of data generated by video surveillance cameras installed globally will be more than 859 petabytes per day by 2017 (IHS 2015). The visual data are rich in content and can open vast analytics opportunities.

However, the visual data are significantly underutilized due to the computation requirements and the outstanding data volume that can incur. The cloud computing paradigm has been well evolved in the past decade for easily scaling out with big data frameworks like Hadoop MapReduce, Spark, Spark Streaming, etc. However, the bounded bandwidth remains a bottleneck for scaling out video analytics applications.

Fog (Bonomi et al. 2012; Botta et al. 2016) is a paradigm for collaboratively using edge devices, intermediate gateways, and servers on premise or in the cloud as the computing platform. The problem of *visual fog computing* is to use the fog paradigm to process continuous video streams in real time. *Offloading* is a technique for a device to outsource tasks to another device (Satyanarayanan et al. 2009; Kumar et al. 2013) for better efficiency, lower latency or lower power consumption. Offloading can be particularly valuable in the context of visual fog, due to the large computational demands of visual processing tasks, and the relatively limited capabilities of edge devices. However, greedy offloading can be catastrophic, due to the high bandwidth data streams and shared upstream resource among multiple pipelines. Thus, global coordinated scheduling is needed.

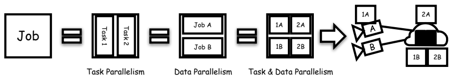


Figure 1: Scheduling in Visual Fog Computing

Scheduling in visual fog computing is much harder than the counterpart in cloud computing because (1) devices are way more heterogeneous in terms of resource types and resource capacities, (2) there are usually a few orders of magnitude more devices, and (3) there are more constraints in terms of device connectivity and task dependency. Unfortunately, the state of the art schedulers in cloud computing do not take into account device connectivity and task dependency as they don't exist in cloud computing. The current schedulers also addressed the scheduling problem at a much smaller scale. The differences between scheduling in cloud computing and scheudling in visual fog computing will be described in detail in Sec. 2.

The goal of this paper is to address the global scheduling problem for *visual fog computing*. Our key contributions include proving the NP-completeness of the problem and providing practical efficient exact solutions. The novelty of this paper is as follows.

- Our problem formulation considers all tasks as a whole and schedule onto heterogeneous devices with the best trade-offs leveraging task runtime resource estimation (Tumanov et al. 2016).
- Our problem formulation takes into account connectivity of devices and dependency of tasks. Essentially, devices are sparsely connected and may not be reachable from one another in the fog paradigm.
- Our solution to the scheduling problem in visual fog computing are optimal, scalable and efficient under the theoretic foundation of integer linear programming (Dantzig, Orden, and Wolfe 1955; Schrijver 1998) with novel tricks

to turn various non-linear constraints in linear forms.

The experiments confirmed the efficiency of our global scheduler in realistic settings that can schedule 20,000 dependent tasks for 7,000 devices with a sub-minute efficiency, and can schedule over 60,000 dependent tasks for 20,000 devices in 7 minutes. This demonstrates that our solution is scalable in the fog paradigm with more than tens of thousands of connected devices.

2 Background

Our work extends prior research in the literature of cloud computing that schedules diverse workloads on heterogeneous devices with heterogeneous resource capacities by explicitly taking into account *device connectivity* and *task dependency*. Specifically, device connectivity represents the device hierarchy of a fog paradigm, and task dependency refers to the dependency between (primitive) tasks. There are two major bodies of related work – reservation systems and DAG scheduling. We will first describe how these two problems are formulated and show why they can't comprehend the need for scheduling in visual fog computing.

The state of the art reservation systems (Curino et al. 2014) and scheduling systems (Tumanov et al. 2016) do not have the concept of task dependency. Reservation systems are designed to ensure that resource guarantees are not overcommitted in a per task manner. Scheduling systems are used to determine task-to-device mapping for allocating tasks onto devices for potentially optimal performance. However, both of them are not capable of addressing the scheduling problem in visual fog computing without taking into account device connectivity and task dependency. Specifically, in visual fog computing, video analytics tasks are usually *divisible* and can be easily partitioned into multiple interdependent stages. Hence, task parallelism and data parallelism can be exploited for simultaneous execution of tasks.

In DAG scheduling, task dependency is represented explicitly as directed acyclic graphs (DAGs). The problem had attracted outstanding attentions for decades for its expressiveness in addressing many real world heterogeneous resource scheduling problem (Radulescu and Van Gemund 2000; Topcuoglu, Hariri, and Wu 2002; Sakellariou and Zhao 2004; Deelman et al. 2009; Zheng and Sakellariou 2013); however, the resource types are usually limited to no more than a few, e.g., CPU, GPU and FPGA on one server. For example, in the experiments of (Zheng and Sakellariou 2013), the number of resources and the number of nodes are considered up to 8 and 123, respectively. In visual fog computing there may are orders of magnitude more resources (hereinafter, we may refer to resource and device interchangeably) and nodes (hereinafter, we may refer to task and node interchangeably); on the other hand, the resource connectivity (i.e., device connectivity) can be rather sparse and directional, which shall be represented in another DAG.

Note that in either of the aforementioned body of work, there is no hard constraint on device capacity and network capacity. Yet another key difference is the objective. The objective in the prior art is to minimize the makespan (i.e., the total execution time). However, in visual fog computing, the (video) data come in continuously, and, therefore, there's no concept of makespan. The scheduling problem in visual fog computing, in effect, optimizes for certain resource utilization, e.g., device resource, link resource or a utility of a mixture of them.

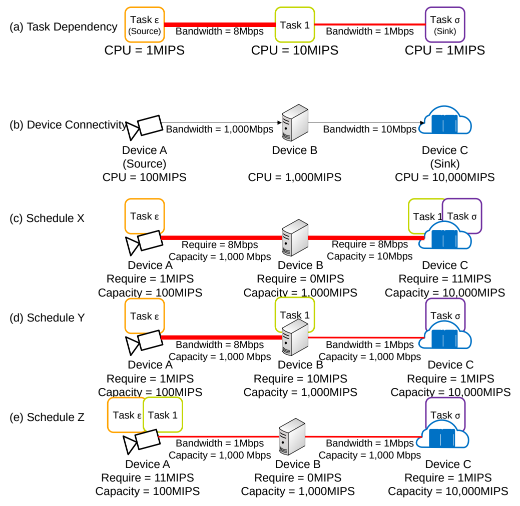

(a) Task Dependency
Task ε (Source)
Bandwidth = 8Mbps
Task 1
Bandwidth = 1Mbps
Task σ (Sink)
CPU = 1MIPS
CPU = 10MIPS
CPU = 1MIPS
(b) Device Connectivity
Bandwidth = 1,000Mbps
Bandwidth = 10Mbps
Device A (Source) CPU = 100MIPS
Device B CPU = 1,000MIPS
Device C (Sink) CPU = 10,000MIPS
(c) Schedule X
Task ε
Task 1
Task σ
Require = 8Mbps Capacity = 1,000 Mbps
Require = 8Mbps Capacity = 10Mbps
Device A Require = 1MIPS Capacity = 100MIPS
Device B Require = 0MIPS Capacity = 1,000MIPS
Device C Require = 11MIPS Capacity = 10,000MIPS
(d) Schedule Y
Task ε
Task 1
Task σ
Bandwidth = 8Mbps Capacity = 1,000 Mbps
Bandwidth = 1Mbps Capacity = 1,000 Mbps
Device A Require = 1MIPS Capacity = 100MIPS
Device B Require = 10MIPS Capacity = 1,000MIPS
Device C Require = 1MIPS Capacity = 10,000MIPS
(e) Schedule Z
Task ε
Task 1
Task σ
Bandwidth = 1Mbps Capacity = 1,000 Mbps
Bandwidth = 1Mbps Capacity = 1,000 Mbps
Device A Require = 11MIPS Capacity = 100MIPS
Device B Require = 0MIPS Capacity = 1,000MIPS
Device C Require = 1MIPS Capacity = 10,000MIPS

Figure 2: Scheduling of Visual Fog Computing

As illustrated in Fig. 1, task parallelism exploits the depth of a pipeline for running simultaneous tasks; data parallelism uses data locality. The scheduling problem of visual fog computing is to determine the best scheduling with respect to a certain objective, e.g., the least total bandwidth consumption, the least power consumption and the shortest turnaround time.

Fig. 2 depicts an example of scheduling in visual fog computing. Fig. 2a is a *task dependency graph* comprising resource requirements over tasks and links between intermediate tasks. Tasks ε and σ are artifacts for problem formulation, whose characteristics are as follows.

- Task ε is a *producer* task that must run on a *source* device, i.e., a camera, where the data come from.
- Task σ is a *consumer* task that must run on a *sink* device, i.e., the cloud, where results may store or a user may consume the data.

Fig. 2b is a *device connectivity graph* comprising resource capacity over devices and links between intermediate devices. Fig. 2c-e show possible schedules X, Y and Z given the various resource constraints and link constraints.

In this paper, we proposed the first global scheduler for visual fog computing taking into account task dependency and device connectivity based on the fact that research confirms that the task runtime can be effectively predicted by resource types and quantities (Curino et al. 2014; Delimitrou and Kozyrakis 2013; Delimitrou and Kozyrakis 2014; Ferguson et al. 2012). We, essentially, assumed devices and their connectivity and tasks and their dependency can be represented as (DAGs); then we prove its hardness and provide practical, efficient solutions.

3 Problem Formulation

In this section, we start with describing the scheduling problem of visual fog computing, hereinafter may be referred to as `VFC`, by defining the two conceptual graphs representing devices and their connectivity, and tasks and their dependency, respectively. Then, we will show how the objective and additional constraints are formalized.

Device connectivity graph is defined as a DAG $G_{\mathcal{D}} = (V_{\mathcal{D}}, E_{\mathcal{D}})$. Each $v \in V_{\mathcal{D}}$ represents a *device* and each edge

$(u, v) \in E_{\mathcal{D}}$ represents the *connectivity* between device u to device v. Specifically, $v \in V_{\mathcal{D}}$ is a source device if $\deg^+(v) = 0$, and is a sink device if $\deg^-(v) = 0$, where $\deg^+(\cdot)$ and $\deg^-(\cdot)$ denote in-degree and out-degree of a vertex, respectively.. We denote $V_{\mathcal{D}}^+$ and $V_{\mathcal{D}}^-$ as the collections of source devices and sink devices, respectively, and $E_{\mathcal{D}}^+$ and $E_{\mathcal{D}}^-$ as the collections of edges start with a source device and end with a sink device, respectively.

Task dependency graph $G_{\mathcal{J}} = (V_{\mathcal{J}}, E_{\mathcal{D}}) = \cup_{j=1}^{J} G_{\mathcal{J}}^j$ where J is the number of *disjoint* subgraphs – jobs. Each subgraph $G_{\mathcal{J}}^j = (V_{\mathcal{J}}^j, E_{\mathcal{D}}^j)$ is a linear (or path) graph having exact one (entry) producer task and one (exit) consumer task; any intermediate tasks are neither producer or consumer tasks. For each subgraph $G_{\mathcal{J}}^j$, $|\{\deg^+(v) = 0 | v \in V_{\mathcal{J}}^j\}| = |\{\deg^-(v) = 0 | v \in V_{\mathcal{J}}^j\}| = 1$. Each $v \in V_{\mathcal{J}}$ represents a *task* and $(u, v) \in E_{\mathcal{J}}$ represents the *dependency* between task u and task v. Specifically, $v \in V_{\mathcal{J}}$ is a producer task if $\deg^+(v) = 0$, and is a consumer task if $\deg^-(v) = 0$. We denote $V_{\mathcal{J}}^+$ and $V_{\mathcal{J}}^-$ as the collections of producer and consumer tasks, respectively, and $E_{\mathcal{J}}^+$ and $E_{\mathcal{J}}^-$ as the collections of edges start with a producer task and end with a consumer task, respectively.

In VFC, in additional to the aforementioned two graphs, the mappings of producer tasks and consumer tasks are predetermined, which can be expressed as $\tilde{g}^+ : V_{\mathcal{J}}^+ \to V_{\mathcal{D}}^+$ and $\tilde{g}^- : V_{\mathcal{J}}^- \to V_{\mathcal{D}}^-$, respectively. Then, a VFC instance can be expressed in the following form:

$$\left(G_{\mathcal{D}}, G_{\mathcal{J}}, \tilde{g}^+, \tilde{g}^-, c_V, c_E, r_V, r_E\right) \tag{1}$$

where c_V and c_E are capacity functions over $G_{\mathcal{D}}$, and r_V and r_E are requirement functions over $G_{\mathcal{D}}$ which will be defined later in Sec. 3.1.

3.1 The Decision Problem

A *schedule* that maps an edge in $G_{\mathcal{J}}$ to a simple path (hereinafter we may refer to simple path as path) in arbitrary lengths in $G_{\mathcal{D}}$, which can be expressed as:

$$f : E_{\mathcal{J}} \to \bigcup_{k=0}^{|V_{\mathcal{D}}|} (v_i)_{i=1}^{k} \tag{2}$$

where $(v_i)_{i=1}^{k}$ represents an arbitrary length-k path. A schedule f implies vertex-to-vertex mapping; we omit details for space consideration. To facilitate formulation, we define *task schedule* as:

$$g : V_{\mathcal{J}} \to V_{\mathcal{D}} \tag{3}$$

that maps tasks over devices. We also define partial order relation $\preceq_{\mathcal{J}}$ over task dependency graph $G_{\mathcal{J}}$, and $\preceq_{\mathcal{D}}$ over device connectivity graph $G_{\mathcal{D}}$, which can be expressed as:

$$u \preceq_{\mathcal{J}} v \iff v \text{ is reachable from } u, \forall u, v \in V_{\mathcal{J}} \tag{4}$$

$$u \preceq_{\mathcal{D}} v \iff v \text{ is reachable from } u, \forall u, v \in V_{\mathcal{D}} \tag{5}$$

Note that task schedule g is not a dual of schedule f; the latter is more expressive. Specifically, there may exist multiple paths between any arbitrary pairs of devices in $G_{\mathcal{D}}$; a schedule not only select a collection of devices to run a collection of tasks but also select a collection of links between devices for data transmission among tasks. A schedule is subject to the following mapping constraints:

$$\tilde{g}^+, \tilde{g}^- \subseteq g \tag{6}$$

$$(v_i)_{i=1}^{k} : \text{path} \iff \oplus_{i=1}^{k-1} f((v_i, v_{i+1})) : \text{path}, \forall v_i \in V_{\mathcal{J}} \tag{7}$$

where $\oplus$ is the path joining operator that concatenates two paths; $(u_1, u_2) \oplus (v_1, v_2)$ is (u_1, u_2, v_2) if $u_2 = v_1$; if v_1 is reachable from u_2, it becomes $(u_1, u_2, \cdots, v_1, v_2)$, and undefined otherwise. Eq. 7 implies:

The Single Choice Constraint A task must be mapped to one and only device (i.e., g is a function) – $f(e_1)$ ends at where $f(e_2)$ starts if and only if e_1 ends at where e_2 starts, for all $e_1, e_2 \in E_{\mathcal{J}}$. All simple paths being mapped from $G_{\mathcal{J}}^j$ must form a simple path from a source device to a sink device in $G_{\mathcal{D}}$

The Ordering Constraint The ordering of tasks along a job in $G_{\mathcal{J}}$ must remain in their mapped devices in $G_{\mathcal{D}}$, which can be expressed as follows:

$$u \preceq_{\mathcal{J}} v \implies g(u) \preceq_{\mathcal{D}} g(v) \tag{8}$$

A *feasible* schedule is subject to *device and network constraints*. An *optimal* schedule is a feasible schedule that minimizes an *objective*. Throughout this paper, we will use overall network utilization as the objective, while extension to other objectives, e.g., overall device (resource) utilization is straightforward. The network utilization is the total network used given the particular schedule f.

The device constraint is defined in a way that tasks mapped to a device shall not go over the device's capacity; likewise, the network constraint is defined in a way that links in the task dependency graph mapped to a link in the device connectivity graph must not go over the link's capacity. We will first define capacity function and requirement function in $G_{\mathcal{D}}$ and $G_{\mathcal{J}}$, respectively, and define the device constraint and the network constraint.

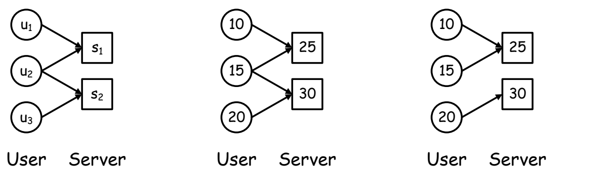
The Device & Network Constraints We first define the capacity function and requirement function in $G_{\mathcal{D}}$ and $G_{\mathcal{J}}$, respectively. The capacity functions are defined over vertices and edges, respectively, in $G_{\mathcal{D}}$. It is the maximum amount of resource a device or a link between a device to its succeeding device can accommodate for a task in $G_{\mathcal{J}}$.

- Device capacity function $c_V : V_{\mathcal{D}} \to \mathbb{R}$ represents device capacity, e.g., CPU MIPS.
- Link capacity function $c_E : E_{\mathcal{D}} \to \mathbb{R}$ represents the capacity of data transmission from one device to another, e.g., bandwidth capacity in Mbps.

The requirement functions are defined over vertices and edges, respectively, in $G_{\mathcal{J}}$. In other words, it is the required amount of resource a task or a link between a task to its succeeding task would require from a device in $G_{\mathcal{D}}$.

- Task requirement function $r_V : V_{\mathcal{J}} \to \mathbb{R}$ represent task requirement, e.g., CPU MIPS.
- Link requirement function $r_E : E_{\mathcal{J}} \to \mathbb{R}$ represents the requirement of data transmission from one task to another, e.g., bandwidth requirement in Mbps.

Recall that f defines an edge-to-path mapping from an edge $e = (u_1, u_2) \in G_{\mathcal{J}}$ to an arbitrary length path $(v_1, v_2, \cdots, v_k)$ in $G_{\mathcal{D}}$. This manifests the nature of data transmission in VFC. Specifically, if the length of path $f(e)$ is zero, i.e., $k = 1$, $r_E(e)$ doesn't implicate any links in $G_{\mathcal{D}}$ since $f(e)$ doesn't contain any edge in $G_{\mathcal{D}}$. In other words, data transmission within a device is essentially omitted since inter-process or inter-thread communication is way faster than inter-device communication. Otherwise, if the length



(a) Users & servers (b) Req. & capacity (c) A solution to (b)

Figure 3: A CC instance

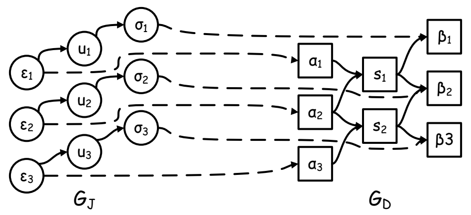


Figure 4: Reducing a CC Instance to a VFC Instance

of path $f(e)$ is greater than or equal to one, i.e., $k \geq 2$, it implies task u_1 and u_2 run on v_1 and v_k, respectively, i.e., $g(u_1) = v_1$ and $g(u_2) = v_k$, and the link requirement $r_E(e)$ remains constant on all edges $\{(v_i, v_{i+1})\}_{i=1}^{k-1}$ along path $f(e)$.

A feasible f must satisfy the device constraint and the network constraint which can be expressed, respectively, as follows:

$$\left(\sum_{v \in V_\mathcal{J}} [\![u = g(v)]\!] r_V(v)\right) \leq c_V(u), \forall u \in V_\mathcal{D} \quad (9)$$

$$\left(\sum_{e \in E_\mathcal{J}} [\![d \in f(e)]\!] r_E(e)\right) \leq c_E(e), \forall d \in E_\mathcal{D} \quad (10)$$

where Eq. 10 implies that when multiple consecutive tasks mapped to one device, the network overhead do not add to any links between devices.

3.2 The Optimization Problem

The optimization problem of VFC is to find a feasible $f^\star$ that minimizes an objective. In this paper, we defined the objective as minimizing total network utilization, which can be expressed as follows.

$$\sum_{d \in E_\mathcal{D}} \sum_{e \in E_\mathcal{J}} [\![d \in f(e)]\!] r_E(e) \quad (11)$$

However, it is straightforward to extend to others, like minimizing sink device resource utilization or maximizing source device resource utilization.

4 NP-Completeness

In this section, we prove that the decision problem of VFC is NP-complete. In order to prove NP-completeness, we need to show that (1) polynomial time verification – a feasible schedule can be validated in polynomial time, and (2) NP-hardness – VFC can be reduced from an NP-complete problem in polynomical time.

It is straightforward to derive an algorithm to verify the feasibility of a schedule by evaluating Eqs. 6–10 in polynomial time. To prove the NP-hardness, we construct a polynomial time reduction from the cloud computing problem CC (Weinman 2011) to VFC, where CC is reduced from PARTITION and 3-SAT. We first briefly review the definition of CC before we proceed to the detail of the reduction to VFC.

4.1 The Cloud Computing Problem

Given a set of users with their resource requirements and a set of servers with their resource capacities, the goal of CC is to assign each user to a server while preventing the total resource used on a server from exceeding the capacity of the server. Moreover, each user has a designated subset of *reachable* servers, where servers not belonging to the subset cannot be assigned to the user.

An instance of CC can be represented in the following form:

$$(\mathbf{U}, \mathbf{S}, \{\mathbf{S}_n\}_n, \{c_m\}_m, \{r_n\}_n) \quad (12)$$

where $\mathbf{U} = \{u_n\}_{n=1}^N$ is the set of resource requirements from N users, $\mathbf{S} = \{s_m\}_{m=1}^M$ is the set of resource capacities of M servers, $\mathbf{S}_n \subseteq \mathbf{S}$ represents the collection of reachable servers by user n, $c_m \in \mathbb{N}$ is the resource capacity of server s_m and $r_n \in \mathbb{N}$ is the resource requirement of user u_n. Fig. 3a and Fig. 3b illustrate a CC instance, and the corresponding requirements and capacities. The left hand side nodes represent three users and their resource requirements; the right hand side nodes represent two servers and their resource capacities. The edges in between represent reachable servers for each user.

The goal of CC is a to find a mapping $\mathbf{\Phi} : \mathbf{U} \rightarrow \mathbf{S}$ such that $\mathbf{\Phi}(u_n) \in \mathbf{S}_n, \forall u_n \in \mathbf{U}$ while satisfying the server capacity constraint, which can be expressed as:

$$\left(\sum_{n=1}^{N} [\![\mathbf{\Phi}(u_n) = s_m]\!]\, r_n\right) \leq c_m, \forall s_m \in \mathbf{S} \quad (13)$$

where $[\![\cdot]\!]$ is the indicator function. Fig. 3c shows a feasible mapping in of the CC instance in Fig. 3b.

4.2 Reduction from CC to VFC

A reduction from CC to VFC implies (1) every CC instance can be transformed into a VFC instance, and (2) a solution to the VFC instance can be transformed back as a solution to the corresponding CC instance.

The reduction is about constructing, based on a CC instance, a task dependency graph $G_\mathcal{J}$ that represents $\mathbf{U}$ and a device connectivity graph that represents both $\mathbf{S}$ and $\{\mathbf{S}_n\}_{n=1}^N$.

A CC Instance ⇒ A VFC Instance Recall that a VFC instance is represented as the tuple in Eq. 1. We first let $r_E(e) = 0, \forall e \in E_\mathcal{J}$ and $c_E(e) = 0, \forall e \in E_\mathcal{D}$, and will show how to formalize $G_\mathcal{D}$, $G_\mathcal{J}$, $\tilde{g}^+$, $\tilde{g}^-$, c_V and r_V.

To formulate CC in VFC we first turn $\mathbf{U}$ into $G_\mathcal{J} = (V_\mathcal{J}, E_\mathcal{J})$. Each CC user u_n is represented as a task in $V_\mathcal{J}$; in addition, a producer task ϵ_n and a consumer task σ_n are added for each user u_n. Each user's corresponding VFC task and their corresponding producer task and consumer task form a path graph of $(\epsilon_n, u_n, \sigma_n)$. Therefore, $V_\mathcal{J}$ and $E_\mathcal{J}$ can be expressed as follows:

$$V_\mathcal{J} = \cup_{n=1}^N \{\epsilon_n, v_n, \sigma_n\} \quad (14)$$

$$E_\mathcal{J} = \cup_{n=1}^N \{(\epsilon_n, v_n), (v_n, \sigma_n)\} \quad (15)$$

where $r_V(u_n) = r_n$ and $r_V(\epsilon_n) = r_V(\sigma_n) = 0$ for all $u_n \in \mathbf{U}$.

Then, we represent $\mathbf{S}$ and $\{\mathbf{S}_n\}_{n=1}^N$ in $G_\mathcal{D} = (V_\mathcal{D}, E_\mathcal{D})$.

Likewise, each CC server s_m is represented as a device in $V_{\mathcal{D}}$. To represent CC reachability $\mathbf{S}_n$, a source device α_n and a sink device β_n are added for each user u_n; user u_n's source device α_n and sink device β_n are connected to a device s_m, if $s_m \in \mathbf{S}_n$, and form a path of (α_n, s_m, β_n) in $G_{\mathcal{D}}$. Therefore, $V_{\mathcal{D}}$ and $V_{\mathcal{D}}$ can be expressed as follows:

$$V_{\mathcal{D}} = \cup_{n=1}^{N}\{\alpha_n, \beta_n\} \cup \{s_m\}_{m=1}^{M} \tag{16}$$

$$E_{\mathcal{D}} = \cup_{n=1}^{N}\{(\alpha_n, s_m), (s_m, \beta_n) | s_m \in \mathbf{S}_n\} \tag{17}$$

where $c_V(s_m) = c_m$ for all $s_m \in \mathbf{S}$ and $c_V(\alpha_n) = c_V(\beta_n) = 0$ for all $u_n \in \mathbf{U}$.

Finally, we let $\tilde{g}^{+}(\epsilon_n) = \alpha_n$ and $\tilde{g}^{-}(\sigma_n) = \beta_n$ for all user n. Fig. 4 illustrates the reduced VFC instance from the CC instance in Fig. 3a. The left hand side along with the solid lines shows the task dependency graph $G_{\mathcal{J}}$, and the right hand side along with the solid lines shows the device connectivity graph $G_{\mathcal{D}}$; the dotted lines show the mapping constraints $\tilde{g}^{+}$ and $\tilde{g}^{-}$.

A Solution in VFC $\Rightarrow$ A Solution in CC Observe that CC reachability $\mathbf{S}_n$ is manifested in $G_{\mathcal{D}}$. The cardinality of $\mathbf{S}_n$ is translated to the number of paths from α_n to β_n. Specifically, if server s_m is reachable by user u_n, there exists a path from α_n to β_n through s_m, and vice versa.

A CC solution can be obtained from task schedule g in VFC, which can be expressed as follows:

$$\mathbf{\Phi}(u_n) = g(u_n), \forall u_n \in \mathbf{U} \tag{18}$$

It is straightforward to verify the server capacity constraint over CC since $r_V(u_n) = r_n$ for all $u_n \in \mathbf{U}$, $c_V(s_m) = c_m$ for all $s_m \in \mathbf{S}$. The device constraint of VFC (Eq. 9) implies the server capacity constraint of CC (Eq. 13).

5 Proposed Solutions

In this section, we show how VFC can be formulated as an integer linear programming problem ILP, and, thereby, solved with state-of-the-art ILP solvers. An ILP formulation is essentially a reduction from VFC to ILP. In addition to a naive formulation in ILP, we also show a novel *dual* formulation in ILP that can be solved much more efficiently.

We first briefly review the definition of ILP, and propose two VFC solutions in ILP formulations in task perspective and device perspective, respectively. Finally, we characterize and compare the two solutions.

5.1 Integer Linear Programming

Integer linear programming is a non-convex optimization problem that had been well-studied in the literature (Dantzig, Orden, and Wolfe 1955; Schrijver 1998). Despite the fact that ILP is in NP-hard, many efficient solvers have been developed and deployed in real-world applications.

We describe the well-known 0-1 (or binary) integer linear programming, which will be later on used for formulating VFC, whose decision problem is also in NP-complete. An ILP instance can be expressed as:

$$\text{minimize} \quad \mathbf{c}^{\top}\mathbf{x} \tag{19}$$

$$\text{subject to} \quad \mathbf{A}\mathbf{x} \leq \mathbf{b} \tag{20}$$

$$\mathbf{C}\mathbf{x} = \mathbf{d} \tag{21}$$

$$\text{and} \quad \mathbf{x} \in \{0,1\}^{K} \tag{22}$$

where K is the length of $\mathbf{x}$, Eq. 19 is the objective term, Eq. 20 is the inequality constraint, Eq. 21 is the equality constraint and Eq. 22 is the binary constraint.

In the following sections, we will describe two realizations of VFC in ILP forms, which we will be refered to as ILP-T and ILP-D, respectively. The high level idea is as follows:

- $\mathbf{x}$ to present the collection of possible schedule f
- Objective term (Eq. 19) to present total network utilization (Eq. 11)
- Constraints (Eq. 20–21) to present device and network constraints (Eqs. 9–10) and mapping constraints (Eqs. 6–7)

5.2 ILP-T: Task Perspective Formulation

A first attempt at solving VFC is to represent all possible assignments from $E_{\mathcal{J}}$ to $E_{\mathcal{D}}$ in $\mathbf{x}$. The key idea is to represent sub-schedule $f^{j} : G_{\mathcal{J}}^{j} \rightarrow G_{\mathcal{D}}$ in $\mathbf{x}_j$ for each subgraph $G_{\mathcal{J}}^{j} \in G_{\mathcal{J}}$, where $\mathbf{x} = (\mathbf{x}_1^{\top} \cdots \mathbf{x}_J^{\top})^{\top}$. A feasible schedule can be realized with the equality constraint in ILP for which only one entry in $\mathbf{x}_j$ is non-zero.

The Objective A sub-schedule f^{j} for each job $G_{\mathcal{J}}^{j}$ is formulated separately. Denote ϵ_j and σ_j the producer task and the consumer task of the j-th job, respectively, and let $\alpha_j = g(\epsilon_j)$ and $\beta_j = g(\sigma_j)$. We define $x_{p,q}^{j}$ a binary variable representing the q-th possible assignment along the p-th path from α_j to β_j. Therefore, the objective can be expressed as:

$$\mathbf{c}^{\top}\mathbf{x} = \sum_{j=1}^{J}\sum_{p=1}^{P(j)}\sum_{q=1}^{Q(j,p)} c_{p,q}^{j} x_{p,q}^{j} \tag{23}$$

where $P(j)$ is the number of paths from α_j to β_j, $Q(j,p)$ is the number of possible assignments from α_j to β_j along the p-th path, and $c_{p,q}^{j}$ is the total network utilization for $x_{p,q}^{j}$. Essentially, $x_{p,q}^{j}$ represents the use of the q-th assignment along the p-th path for job j.

The Single Choice Constraint To ensure each $\mathbf{x}_j$ has only one non-zero entry, we define the following equality constraint:

$$\sum_{p=1}^{P(j)}\sum_{q=1}^{Q(j,p)} x_{p,q}^{j} = 1, 1 \leq j \leq J \tag{24}$$

The Ordering Constraint This is automatically satisfied since ILP-T expands all possible assignments in the formulation.

The Device & Network Constraints We denote, given $x_{p,q}^{j}$, $R_V^{j}(v,p,q)$ the total resource requirement for each device $v \in V_{\mathcal{D}}$ and $R_E^{i}(e,p,q)$ the total link requirement for each link $e \in E_{\mathcal{D}}$. Therefore, we can formulate the device and network constraints as:

$$\left(\sum_{j=1}^{J}\sum_{p=1}^{P(j)}\sum_{q=1}^{Q(j,p)} R_V^{j}(v,p,q)\, x_{p,q}^{j}\right) \leq c_V(v),\ \forall v \in V_{\mathcal{D}} \tag{25}$$

$$\left(\sum_{j=1}^{J}\sum_{p=1}^{P(j)}\sum_{q=1}^{Q(j,p)} R_E^{j}(e,p,q)\, x_{p,q}^{j}\right) \leq c_E(e),\ \forall e \in E_{\mathcal{D}} \tag{26}$$

The ILP-T formulation reflects the nature of task-to-device assignment, and enumerates all possible assignments in $\mathbf{x}$. In order to turn all constraints into linear forms, ILP-T essentially expands all possibilities in $\mathbf{x}$. It works fairly well when the number of jobs and the number of devices are no more than a few thousands. However, $P(j)$ grows significantly when there are multiple paths from α_j to β_j; even worse, $Q(j,p)$ grow exponentially, which can be expressed as follow:

$$Q(j,p) = \binom{|V_{\mathcal{J}}^j| + |P(j)| - 3}{|V_{\mathcal{J}}^j| - 2} \tag{27}$$

The outstanding cardinality of $\mathbf{x}$ can make ILP converge rather slowly.

While ILP-T maps directly from f in VFC to $\mathbf{x}$ in ILP by enumerating all possible paths for each job j, the dimension of $\mathbf{x}$ grows exponentially with increased complexity in both graphs $V_{\mathcal{J}}$ and $V_{\mathcal{D}}$. In the next section, we present a novel formulation ILP-D of VFC in ILP that exploits the duality of f – the cardinality of $\mathbf{x}$ can remain moderate; VFC can be solved much more efficiently. A detailed comparison between ILP-T and ILP-D will be discussed in Sec. 5.4.

5.3 ILP-D: Device Perspective Formulation

The ILP-D formulation is based on a simple perhaps critical observation that there exists a *reverse* schedule h for each feasible schedule f; essentially, h is a one-to-many *relation* between $E_{\mathcal{D}}$ and $E_{\mathcal{J}} \cup \phi$, which can be expressed as:

$$h : E_{\mathcal{D}} \times \{E_{\mathcal{J}} \cup \phi\} \tag{28}$$

In addition, h can be essentially represented with a collection of reverse sub-schedule h^j, which are functions (cf. h is not a function) mapping a device back to a task in a per job manner:

$$h^j : E_{\mathcal{D}} \to \{E_{\mathcal{J}}^j \cup \phi\} \qquad 1 \le j \le J \tag{29}$$

$$h(d) = \cup_{j=1}^{J} h^j(d) \qquad \forall d \in E_{\mathcal{D}} \tag{30}$$

where J is the number of jobs (disjoint subgraphs) in $G_{\mathcal{J}}$. The reverse schedule h is a *dual* of f; the fact that h is a dual of f allows us to reformulate VFC in ILP with reverse schedule h in $\mathbf{x}$, which lead to a much more efficient runtime efficiency for the reduced number of variables in ILP-D.

The Objective To formulate $\mathbf{x}$ for reverse edge-to-edge mapping of h, we define $\mathbf{x}$ as follows:

$$\mathbf{x} = \{x_{d,e}^j | d \in E_{\mathcal{D}}, e \in E_{\mathcal{J}}^j\}_{j=1}^{J} \tag{31}$$

where $x_{d,e}^j$ represents whether or not $h^j(d) = e$. Therefore, the objective can be defined as follows:

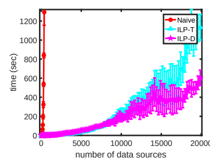
$$\mathbf{c}^\top \mathbf{x} = \sum_{j=1}^{J} \sum_{d \in E_{\mathcal{D}}} \sum_{e \in E_{\mathcal{J}}^j} r_E(e) x_{d,e}^j \tag{32}$$

The ILP-D formulation explicitly model the edge-to-edge mapping from $E_{\mathcal{D}}$ to $E_{\mathcal{J}}$ in $\mathbf{x}$, cf. ILP-T expands all possible paths for each pair of source device and sink device for each job.

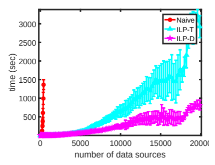
The Single Choice Constraint A device has at most one outward edge mapped by f for each job, which can be expressed as:

$$\sum_{\substack{d^+ = u \\ d \in E_{\mathcal{D}}}} \sum_{e \in E_{\mathcal{J}}^j} x_{d,e}^j \le 1, \ \forall u \in V_{\mathcal{D}} \tag{33}$$

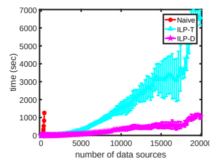
where $\cdot^+$ and $\cdot^-$ denote the tail and the head of an arbitrary edge; namely, $e^+ = u$ and $e^- = v$ if $e = (u,v)$. In addition, every device, which is neither a source device or a sink device, must have one inward edge and one outward edge or have none of them, which can be expressed as:



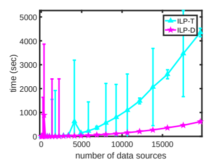
(a) Tree depth of 3, task length of 3



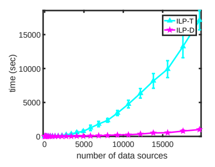
(b) Tree depth of 3, task length of 4



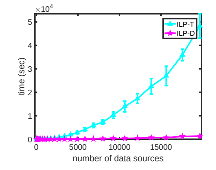
(c) Tree depth of 3, task length of 5



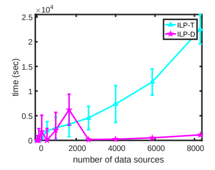
(d) Tree depth of 4, task length of 3



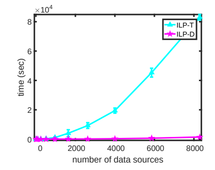
(e) Tree depth of 4, task length of 4



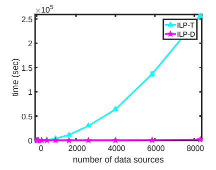
(f) Tree depth of 4, task length of 5



(g) Tree depth of 5, task length of 3



(h) Tree depth of 5, task length of 4



(i) Tree depth of 5, task length of 5

Figure 5: Quantitative Results with Realistic Settings

$$\sum_{e \in E_{\mathcal{J}}^j} \left(\sum_{\substack{d^+ = u \\ d \in E_{\mathcal{D}}}} x_{d,e}^j - \sum_{\substack{d^- = u \\ d \in E_{\mathcal{D}}}} x_{d,e}^j \right) = 0, \ \forall u \in V_{\mathcal{D}} \setminus (V_{\mathcal{D}}^+ \cup V_{\mathcal{D}}^-) \tag{34}$$

whereas a source device and a sink device must have exact one outward edge and one inward edge, respectively, which can be expressed as:

$$\sum_{\substack{d^+ \in V_{\mathcal{D}}^+ \\ d \in E_{\mathcal{D}}}} \sum_{e \in E_{\mathcal{J}}^j} x_{d,e}^j = \sum_{\substack{d^- \in V_{\mathcal{D}}^- \\ d \in E_{\mathcal{D}}}} \sum_{e \in E_{\mathcal{J}}^j} x_{d,e}^j = 1 \tag{35}$$

The Ordering Constraint The tasks along a job must be mapped in order along a path in the device connectivity graph, as described in Eq. 8, which can be expressed as:

$$\sum_{\substack{d^- = u \\ d \in E_{\mathcal{D}}}} \sum_{e \in E_{\mathcal{J}}^j} \mathbf{I}(e) x_{d,e}^j \le \sum_{\substack{d^+ = u \\ d \in E_{\mathcal{D}}}} \sum_{e \in E_{\mathcal{J}}^j} \mathbf{I}(e) x_{d,e}^j, \forall u \in V_{\mathcal{D}} \tag{36}$$

where $\mathbf{I} : E_{\mathcal{J}} \to \mathbb{R}$ is an arbitrary function that satisfies the property: $\mathbf{I}(e_1) \le \mathbf{I}(e_2) \Leftrightarrow e_1^+ \preceq_{\mathcal{J}} e_2^+ \Leftrightarrow e_1^- \preceq_{\mathcal{J}} e_2^-$ for all $e_1, e_2 \in E_{\mathcal{J}}$. The above implies that Eq. 8 holds by using an arbitrary function $\mathbf{I}$ to preserve the total ordering of $G_{\mathcal{J}}^j$.

The Device & Network Constraints The network constraint is straightforward in ILP-D, which can be formulated as follows:

$$\left(\sum_{j=1}^{J} \sum_{e \in E_{\mathcal{J}}^j} r_E(e) x_{d,e}^j \right) \le c_E(d), \forall d \in E_{\mathcal{D}} \tag{37}$$

The device constraint, on the other hand, is trickier in the ILP-D formulation since edge-to-edge mapping is explicitly modeled in $\mathbf{x}$. We define $\mathbf{R} : E_{\mathcal{J}} \to \mathbb{R}$ the *aggregated* requirement function that sum up the task requirements all the way until the tail of e from its corresponding producer task, which can be expressed as:

$$\mathbf{R}(e) = \sum_{\substack{u \preceq_{\mathcal{J}} e^+ \\ u \in V_{\mathcal{J}}}} r_V(u), \forall e \in E_{\mathcal{J}} \tag{38}$$

The difference between aggregated requirements of a device's inward edge and outward edge uniquely defines the actual jobs running on the device. Hence, we can define the device constraint as follows:

$$\sum_{j=1}^{J} \sum_{e \in E_{\mathcal{J}}^j} \mathbf{R}(e) \left(\sum_{d^+ = u} x_{d,e}^j - \sum_{d^- = u} x_{d,e}^j \right) \le c_V(u), \forall u \in V_{\mathcal{D}} \tag{39}$$

where each e in the inner summation is either all zero or has exact one positive term and one negative term whose difference defines the actual task or tasks run on device u. The novel use of $\mathbf{R}$ makes possible extracting device-to-task mapping from reverse edge-to-edge mapping h.

5.4 Discussion

We proposed two formulations ILP-T and ILP-D for solving VFC using ILP solvers – ILP-T maps schedule f to $\mathbf{x}$ whereas ILP-D maps dual schedule h to $\mathbf{x}$. The novelty of ILP-D is to explicitly model the edge-to-edge relation between $E_{\mathcal{D}}$ and $E_{\mathcal{J}}$ in $\mathbf{x}$ while being able to represent the various non-linear constraints in linear forms, especially Eq. 36 and Eq. 39.

	ILP-T	ILP-D
$\lvert\mathbf{x}\rvert$	$\sum_{j=1}^{J}\sum_{p=1}^{P(j)} Q(j,p)$	$\lvert E_{\mathcal{J}}\rvert \times \lvert E_{\mathcal{D}}\rvert$
Eq. 20	$\lvert V_{\mathcal{D}}\rvert + \lvert E_{\mathcal{D}}\rvert$	$(J+1) \times \lvert V_{\mathcal{D}}\rvert + \lvert E_{\mathcal{D}}\rvert$
Eq. 21	J	$2 \times J \times \lvert V_{\mathcal{D}}\rvert$

Table 1: ILP-T vs. ILP-D

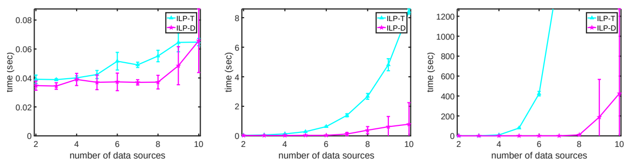
The characteristics of the two formulations are summarized in Tab. 1. As $P(j)$ and $Q(j,p)$ grow extremely fast, both in combination can make ILP-T computationally intractable in real world settings. On the other hand ILP-D, by exploiting the duality of schedule, keeps the cardinality of $\mathbf{x}$ polynomial, and, in effect, can be solved much more efficiently.

6 Experimental Results

This section evaluates our solutions for feasibility and scalability, where the former is through experiments against realistic visual fog settings and the latter is through generalized visual fog settings. Both of them are with carefully crafted simulated data so that there only exist non-trivial solutions. All experiments were run on Intel Xeon CPU E5-4657L v2 @ 2.4GHz using MATLAB's Optimization Toolbox.

6.1 Realistic Visual Fog Settings

Realistic visual fog settings are with tree structure where the root vertex is the only sink device and the leaf vertices are source devices. The root vertex manifests the concept of cloud computing where a cluster of interconnected servers are abstracted. Complete b-nary tree is assumed where all



(a) DAG depth of 3, task length of 3 (b) DAG depth of 4, task length of 4 (c) DAG depth of 5, task length of 5

Figure 6: Quantitative Results with Generalized Settings

non-leaf vertices share a same branching factor b. Each setting repeats for 10 times and assumes a tree depth and a task length while increasing branching factor of the tree until there are more than 20,000 source devices, i.e., leaf vertices.

Fig. 5 compares ILP-T, ILP-D and a naive baseline that iterates through all possibilities, with respect to the execution time to find an optimal schedule. As one can see from Fig. 5a-c, the naive approach is computationally intractable in any realistic settings; ILP-T and ILP-D on the other hand, are scalable even when there are thousands of source devices. The performance difference between ILP-T and ILP-D is also significant. This experiment not only shows the feasibility of ILP-T and ILP-D, but also shows the advantage of the ILP-D formulation where its number of variables in $\mathbf{x}$ is significantly fewer than with ILP-T.

6.2 Generalized Visual Fog Settings

The generalized settings are with DAG structure of device graph $G_{\mathcal{D}}$ where there can be multiple paths between a pair of source device and sink device. This manifests the multi-path nature of networking. To make the problem even more challenging, we simulate $G_{\mathcal{D}}$ with complete multi-partite graphs where each independent vertex set in $G_{\mathcal{D}}$ shares a same cardinality. Likewise, each setting repeats for 10 times and assumes a DAG depth (number of indepedent vertex sets) and a task length while increasing the cardinality of all vertex sets up until 10.

Fig. 6 compares ILP-T and ILP-D, with respect to the execution time to find an optimal schedule. As can be seen in Fig. 6c, ILP-T cannot scale well even with no more than ten devices. Both experiments, in combination, concluded that ILP-D is much more scalable in both realistic and generalized settings.

7 Conclusion

In this paper, we formulated and addressed the scheduling problem of visual fog computing. To the best of our knowledge, we are the first to prove its NP-completeness and proposed practical efficient solutions under the theoretic foundation of ILP with the novel trick in ILP-D that represents non-linear constraints in linear forms. The ample experimental results showed that our methods are feasible and scalable. The next step is to extend the solutions to more practical settings – (1) compound resource types upon devices and tasks and along links between devices and dependencies between tasks, and (2) just-in-time scheduling and runtime scheduling. We also plan to use distributed ILP solvers for improved efficiency of scheduling. The next step also includes studying of approximation algorithms with theoretic guarantee for further improving the computational complexity, e.g., relaxation with randomized rounding (Raghavan and Tompson 1987).

References

[Bonomi et al. 2012] Bonomi, F.; Milito, R.; Zhu, J.; and Addepalli, S. 2012. Fog computing and its role in the internet of things. In *the first edition of the MCC workshop on Mobile cloud computing*.

[Botta et al. 2016] Botta, A.; de Donato, W.; Persico, V.; and Pescape, A. 2016. Integration of cloud computing and internet of things: a survey. 56:684–700.

[Curino et al. 2014] Curino, C.; Difalla, D. E.; Douglas, C.; Krishnan, S.; Ramakrishnan, R.; and Raio, S. 2014. Reservation-based scheduling: If you're late don't blame us! In *Proceedings of the ACM Symposium on Cloud Computing*.

[Dantzig, Orden, and Wolfe 1955] Dantzig, G. B.; Orden, A.; and Wolfe, P. 1955. Generalized simplex method for minimizing a linear form under linear inequality restraints. 5:183–195.

[Deelman et al. 2009] Deelman, E.; Gannon, D.; Shields, M.; and Taylor, I. 2009. Workflows and e-science: An overview of workflow system features and capabilities. *Future generation computer systems* 25(5):528–540.

[Delimitrou and Kozyrakis 2013] Delimitrou, C., and Kozyrakis, C. 2013. Qos-aware scheduling in heterogeneous datacenters with paragon. 31.

[Delimitrou and Kozyrakis 2014] Delimitrou, C., and Kozyrakis, C. 2014. Quasar: Resource-efficient and qos-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*.

[Ferguson et al. 2012] Ferguson, A. D.; Bodik, P.; Kandula, S.; Boutin, E.; and Fonseca, R. 2012. Jockey: guaranteed job latency in data parallel clusters. In *Proceedings of the 7th ACM European Conference on Computer Systems*.

[IHS 2015] 2015. Top video surveillance trends for 2015. Technical report.

[Kumar et al. 2013] Kumar, K.; Liu, J.; Lu, Y.-H.; and Bhargava, B. 2013. A survey of computation offloading for mobile systems. *Mobile Networks and Applications* 18(1):129–140.

[Radulescu and Van Gemund 2000] Radulescu, A., and Van Gemund, A. J. 2000. Fast and effective task scheduling in heterogeneous systems. In *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th*, 229–238. IEEE.

[Raghavan and Tompson 1987] Raghavan, P., and Tompson, C. D. 1987. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. 7:365374.

[Sakellariou and Zhao 2004] Sakellariou, R., and Zhao, H. 2004. A hybrid heuristic for dag scheduling on heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 111. IEEE.

[Satyanarayanan et al. 2009] Satyanarayanan, M.; Bahl, P.; Caceres, R.; and Davies, N. 2009. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing* 8(4).

[Schrijver 1998] Schrijver, A. 1998. *Theory of linear and integer programming*. John Wiley and Sons.

[Topcuoglu, Hariri, and Wu 2002] Topcuoglu, H.; Hariri, S.; and Wu, M.-y. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems* 13(3):260–274.

[Tumanov et al. 2016] Tumanov, A.; Zhu, T.; Park, J. W.; Kozuch, M. A.; Harchol-Balter, M.; and Ganger, G. R. 2016. Tetrisched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In *The European Conference on Computer Systems (EuroSys)*.

[Weinman 2011] Weinman, J. 2011. Cloud computing is np-complete. Technical report.

[Zheng and Sakellariou 2013] Zheng, W., and Sakellariou, R. 2013. Stochastic dag scheduling using a monte carlo approach. 73(12):1673–1689.