Delay-Sensitive Computation Partitioning for Mobile Augmented Reality Applications

Chaokun Zhang*, Rong Zheng†, Yong Cui‡, Chenhe Li†, Jianping Wu‡
*College of Intelligence and Computing, Tianjin University, Tianjin, China
†Department of Computing and Software, McMaster University, Hamilton, ON, Canada
‡Department of Computer Science and Technology, Tsinghua University, Beijing, China
zhangchaokun@tju.edu.cn, {rzheng, lic54}@mcmaster.ca, cuiyong@tsinghua.edu.cn, jianping@cernet.edu.cn

***Abstract*—Good user experiences in Mobile Augmented Reality (MAR) applications require timely processing and rendering of virtual objects on user devices. Today's wearable AR devices are limited in computation, storage, and battery lifetime. Edge computing, where edge devices are employed to offload part or all computation tasks, allows an acceleration of computation without incurring excessive network latency. In this paper, we use acyclic data flow graphs to model the computation and data flow in MAR applications and aim to minimize the makespan of processing input frames. Due to task dependencies and variable resource availability, makespan minimization is proven to be NP-hard in general. We design DPA, a polynomial-time heuristic algorithm for this problem. For special data flow graphs including chain or star, the algorithm can provide optimal solutions or solutions with a constant approximation ratio. The effectiveness of DPA has been evaluated using extensive simulations with realistic workloads and resource availability measured from a prototype implementation.**

***Index Terms*—Edge computing; Mobile augmented reality; Computation partitioning; Precedence constraint**

I. Introduction

Mobile Augmented Reality (MAR) is gaining popularity due to the wide availability of smartphones and wearable devices in the past decade [1], [2]. Reports forecast that the worldwide market size for AR is estimated to reach over USD 195 billion by 2025 [3]. In MAR applications, virtual objects are overlaid over the physical world for the users to perceive the augmented information as part of their surrounding environments. Though application-specific, the virtual objects are typically constructed based on camera, voice or other high rate sensor inputs. For instance, a face recognition app running on the smart glasses takes real-time video feeds from a front-facing camera, performs face detection and recognition, and then overlays the relevant information of people present in the field of view onto a head-up display.

However, MAR poses significant challenges to today's wearable devices. First, the majority of MAR applications are delay-sensitive [1]. Virtual objects are generally context-dependent. Excessive delays in constructing and rendering the virtual objects make them irrelevant. Second, the construction of virtual objects often relies on machine learning algorithms to extract useful information from sensor inputs. These inference tasks tend to be compute-intensive, and often require a lot of storage space to accommodate necessary models. Furthermore, MAR requires user interactions in real-time [1]. As a result, existing standalone low- to medium-end Augmented Reality (AR) devices in the market are not suitable for handling delay-sensitive and compute-intensive applications.

There has been much interest recently to offload compute-intensive tasks to edge devices, termed *Edge Computing* (EC) [4], to shorten their response time. Broadly speaking, edge devices can be cellular base stations, enterprise or home Wi-Fi access points, or even more powerful mobile phone devices. Migrating heavy tasks to the EC device can reduce the computational burden of AR devices. Compared to Mobile Cloud Computing (MCC), offloading to edge devices incurs shorter latency. Hence, it can improve the Quality of Experience (QoE) in MAR applications.

Orthogonal to where computation is done, e.g., on the AR device, on an edge device or in the cloud, another dimension in the design space of MAR is whether the computation tasks shall be entirely offloaded [5] or split between the AR device and the edge device or cloud computing facilities [6]. Mobile computation offloading can be thought of as a special case of *Mobile Computation Partitioning* (MCP) [7] – the latter provides fine-grained control and enables offloading portions of tasks onto more powerful computation units to shorten the completion time.

Existing works on MCP assume that the computation resource on offloaded sites is unlimited. This may be reasonable when the offloaded computation runs on a high-end server or a server cluster but is problematic when resource-limited edge devices are considered. Assuming unlimited resources on the edge leads to optimistic timing estimations. Moreover, many works are based on fixed partitioning and ignore resource variations such as bandwidth fluctuation and dynamic CPU frequency or voltage scaling. Resource variability may lead to excessive delays that degrade application performance or violate safety guarantees.

In this paper, we consider the problem of computation partitioning for delay-sensitive MAR applications in edge comput-

The corresponding author is Chaokun Zhang. This research is supported in part by National Key R&D Program of China under Grant 2019YFB2102400, in part by NSERC Discovery Program, in part by National Natural Science Foundation of China under Grant 61872211,61832013.

ing. Sensory inputs to an MAR application are organized in *frames* (e.g., a frame in a real-time video stream or a segment of sensor data). The computation tasks performed on each frame can be modeled as a data flow graph. In the data flow graph, tasks have precedence constraints, namely, a task cannot be executed until its predecessors are completed [8]. Given the computation workload of each task and the amount of data flow between tasks, we aim to find the optimal partitioning decision between the AR device and the EC device such that the total completion time (formally called *makespan*) of each frame is minimized subject to the availability of computation and network resources.

Our contribution in this paper is three-fold. First, we formulate the Makespan Minimization Computation Partition (M2CP) problem under precedence constraints and time-varying variables through intra-frame and inter-frame task modeling. We show that it is an NP-hard problem.

Second, we show that for precedence constraints modeled by chain or star parallel graphs, polynomial algorithms exist that either give an exact solution or solutions with a constant approximation ratio. Specifically, for applications that can be modeled as a star parallel graph, an efficient algorithm called *EFS2* is proposed and is proven to achieve an approximation ratio of $2\sqrt{3}$. For general cycle-free data flow graphs, a heuristic algorithm called *DPA* is proposed.

Third, we implement a prototype that includes a lightweight profiler and a decision-maker using the proposed algorithm. To overcome the difficulty of unknown resource demands, a simple yet effective approach is adopted based on historical data as initial inputs. We evaluate the performance of the proposed algorithm with realistic workloads and resource availability measured from the prototype implementation.

The rest of the paper is organized as follows. The related work is discussed in Section II. In Section III, we present the system model and formulate the makespan minimization computation partition problem. We show that the problem is NP-hard. In Section IV, polynomial algorithms are developed for makespan minimization for two special cases. The heuristic solution for the general data flow graphs is presented in Section V. In Section VI, we evaluate the performance of the algorithms based on the prototype. Finally, Section VII concludes the paper.

II. Related Work

Mobile computation offloading has gained much interest in the research community and the industry in recent years. The main benefit is to relieve resource competition and augment mobile devices' capabilities [7], [9], [10]. Essentially, mobile computation is to adopt whole VM migration without any partition [5], [11]–[13]. Such a coarse-grained approach is not suitable for AR applications as sensory data is in fact generated on mobile devices [14].

Unlike the offloading method, MCP provides a fine-grained approach with better performance. For ease of implementation and use, many MCPs use a fixed partitioning scheme [6], [15], [16]. This requires developers to have a good understanding of computation and communication requirements to make sound decisions. Furthermore, the fixed partition tends to be sub-optimal in the presence of device heterogeneity and varying resource availability.

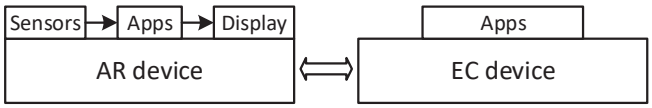


Fig. 1. Basic framework of an MAR system

The extended approach is to dynamically determine where each task should be executed [17]–[22]. MAUI [17] is arguably the first work that considers such an approach. In MAUI, the partitions are manually tagged and computation tasks are serialized. CloneCloud [18] uses an analyzer to give a feasible partition. However, it involves a code migration approach [19] that is not compatible with MAR apps based on data flow characteristics. Odessa [20] first proposes the data flow based partitioning approach for interactive perception applications that can be naturally adapted to MAR. Structure of data flow hence should be considered in dynamic partitioning problem. In this dimension, MAUI [17] processes tasks in sequential and CloneCloud [18] and Hermes [21] process tasks based on a tree structure, while our MCP algorithm involves general precedence constraints. Additionally, several computation partitioning approaches are devised for different edge computing applications [23], [24] instead of MAR. The work in [22] makes the unrealistic assumption that the resources on the EC device are unlimited.

III. Models and Problem Formulation

In this section, we introduce the data flow programming model and the problem formulation.

A. Data Flow Programming Model

Figure 1 shows the basic framework of an MAR system that supports computation partitioning. Here, the EC device can be a mobile phone or more powerful devices such as Wi-Fi or cellular access points. Unlike high performance clusters in data centers, EC devices are more constrained in available resources. Transferring data between the AR and EC devices incurs extra latency. Therefore, it is necessary to balance the computation workload between the AR device and the EC device.

MAR applications can be modeled by data flow graphs. The vertices in the graph are processing steps called *tasks* and the edges are connectors that represent the data dependencies between tasks. Tasks within an application employ a shared-nothing model: they share no state and interact only through connectors. This programming model allows programmers to express coarse-grained application parallelism while hiding much of the complexity of parallel and distributed programming from application developers. The data flow programming model is a natural fit for MAR since it can easily capture a series of operations on streaming data and facilitate the partitioning of computation across multiple devices [20].

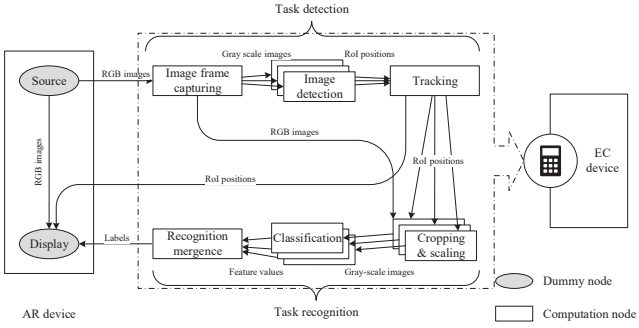


Fig. 2. The data flow graph of a recognition app

In this paper, we assume that the data flow graph of a target MAR application is represented by a Directed Acyclic Graph (DAG), $G = (V, A)$. This is not a restrictive assumption as a data flow graph with recurrent edges can be rolled out into a "feed-forward" graph. In graph G, each node $i \in V$ is associated with a task and its demand of CPU resource is given by p_i measured in the number of CPU cycles. An arc $a = \langle i, j \rangle$ where $\{a | i, j \in V, \text{ and } a \in A\}$ represents the data path from task i to task j. The amount of data from task i to task j is given by $c_{i,j}$. If task i and j reside on different devices, $c_{i,j}$ is associated with a communication cost. The precedence constraints are modeled in the DAG as directed edges. Denote the direct predecessor set of task i as Γ_i^-, where $\Gamma_i^- = \{h | \langle h, i \rangle \in A\}$. Therefore, task i cannot be scheduled until all its direct predecessors in Γ_i^- are completed. Similarly, the direct successor set of task i is denoted by Γ_i^+, where $\Gamma_i^+ = \{j | \langle i, j \rangle \in A\}$. We introduce two dummy nodes [22], a *Source* and a *Display* node that respectively connects to the first task and all nodes without successors in G (as shown in Figure 2). For MAR applications, both the Source and Display nodes reside on the AR device. The source node takes input data (e.g., sensing data) whereas results from tasks in the last task shall be displayed locally the AR device.

An example data flow model and its DAG representation are given in Figure 2 for a typical recognition app. In this example, the source corresponds to a camera and the sink is a head-up display on the AR device. In this application, the input data consists of video frames from a camera. Each frame will be processed following the same data flow graph. The tasks in the data flow graph differ in computation complexity. For instance, detection and tracking are relatively lightweight, whereas recognition is compute-intensive due to data retrieval. In the graph, data transferred along the edges includes large-scale images such as original RGB images, grayscale images, and Regions of Interest (RoIs) of possible objects, as well as some feature values (e.g., Local Binary Patterns Histogram (LBPH) for face recognition [25] and Scale-Invariant Feature Transform (SIFT) for object recognition [26]).

B. Problem Formulation

Let x_i denote the binary partition decision of task i. $x_i = 0$ indicates that the task runs on the AR device (or locally); otherwise, the task is executed on the EC device (or remotely) if $x_i = 1$. For task i, let t_i^S and t_i^C denote its starting time and completion time, respectively. The processing time of task i is hence $t_i^P = t_i^C - t_i^S$. Let f_L and f_R be the processing speeds on the AR and EC devices, which are determined by the reciprocal of the clock cycle (Hz). Both speeds can be profiled online [27]. In general, it is reasonable to assume that EC devices have a high processing power and thus $f_L < f_R$. Let b be the available bandwidth in bit per second (bps). In addition to partitioning, we are also interested in deciding the two time-varying variables, i) $y_i(t)$ the amount of CPU resource allocated to task i at time t, and ii) $z_{i,j}(t)$ the bandwidth allocated to transfer data from task i to j if the two tasks reside on different devices at time t.

Intra-frame Task Modeling: The precedence constraints among the tasks can be characterized by,

$$\begin{cases} t_i^S \geq \max\limits_{h \in \Gamma_i^-} \left(t_h^C + |x_h - x_i| \, t_{h,i}^T \right) & \Gamma_i^- \neq \emptyset \\ t_i^S = 0 & \Gamma_i^- = \emptyset \end{cases} \quad \forall i \in V. \quad (1)$$

In other words, task i cannot start until all its predecessor tasks finish, and data from its predecessors have been transferred to the device where task i is executed.

The constraints on the CPU resources on the AR and EC devices are given by,

$$\sum_{i \in V} (1 - x_i) y_i(t) \leq f_L \quad \forall t \geq 0, \quad (2)$$

and

$$\sum_{i \in V} x_i y_i(t) \leq f_R \quad \forall t \geq 0, \quad (3)$$

where $y_i(t)$ satisfies,

$$\begin{cases} y_i(t) = 0 & t \notin \left[t_i^S, t_i^C \right) \\ y_i(t) \geq 0 & t \in \left[t_i^S, t_i^C \right) \end{cases} \quad \forall i \in V. \quad (4)$$

Furthermore, the CPU resource allocation for task i over time should be equal to the demand of task i, that is,

$$\int_{t_i^S}^{t_i^C} y_i(t) dt = p_i \quad \forall i \in V. \quad (5)$$

Similar to CPU constraints (2) and (3), the bandwidth (denoted by $z_{i,j}(t)$) allocated to each task pair is constrained by the total available bandwidth,

$$\sum_{\langle i,j \rangle \in A} z_{i,j}(t) \leq b \quad \forall t \geq 0, \quad (6)$$

where $z_{i,j}(t)$ satisfies,

$$\begin{cases} z_{i,j}(t) = 0 & t \notin \left[t_i^C, t_i^C + t_{i,j}^T \right) \\ z_{i,j}(t) \geq 0 & t \in \left[t_i^C, t_i^C + t_{i,j}^T \right) \end{cases} \quad \forall \langle i, j \rangle \in A, \quad (7)$$

where $t_{i,j}^T$ is the transmission time from task i to task j.

Additionally, the data transferred from task i to j if they are executed on different devices satisfies,

$$\int_{t_i^C}^{t_i^C + t_{i,j}^T} z_{i,j}(t) dt = |x_i - x_j| c_{i,j} \quad \forall \langle i, j \rangle \in A. \quad (8)$$

Fig. 3. A chain data flow graph

Inter-frame Task Modeling: Let τ be the deadline for each frame. As the result of high temporal locality in video streams, it is often sufficient to process only one of several consecutive frames (called Group Of Pictures, or GOP) [28]. We introduce n as the number of such consecutive frames. The delay constraint can thus be expressed as,

$$\max_{i \in V} t_i^C \leq n\tau. \tag{9}$$

Let $t_i^{C'}$ and x_i' be the completion time and the partition decision of task i in the previous frame. If the corresponding device is running a task of the previous frame before the current task start, the current task j will be discarded because of the limited resource. Thus, we have,

$$t_j^S \geq t_i^{C'}, \text{ if } x_i' = x_j. \tag{10}$$

That is, the constraint holds only if task j in the current frame and task i in the previous frame are running on the same device.

Makespan Minimization: We are now in the position to formulate the Makespan Minimization with Computation Partitioning (M2CP) problem. We have,

$$\begin{aligned} \textbf{M2CP}: \quad \min \quad & \max_{i \in V} t_i^C && (11)\\ \text{s.t.} \quad & (1)-(10) \\ & x_i \in \{0,1\}, \forall i \in V && (12)\\ & x_{Source} = 0, x_{Display} = 0. && (13) \end{aligned}$$

Clearly, M2CP is a Mixed Integer Programming (MIP) problem with both binary and continuous variables. In fact, in Section IV, we show that M2CP is NP-hard even for simple data flow graphs such as star (or parallel) tasks. Therefore, M2CP is NP-hard for general DAG data flow graphs.

IV. Polynomial Algorithms for Special Data Flow Graphs

In this section, we first simplify the problem under sequential scheduling policies. Under this assumption, we turn to two special cases for M2CP with precedence constraints: chain data flow (Figure 3) and star data flow (Figure 4). We show that even for a simple data flow graph as in Figure 4, M2CP is NP-hard. However, we show that a polynomial-time algorithm exists for this case with a constant approximation ratio. Solutions to the chain and the star precedence are subsequently used as building blocks in devising a heuristics to M2CP for general data flow graphs.

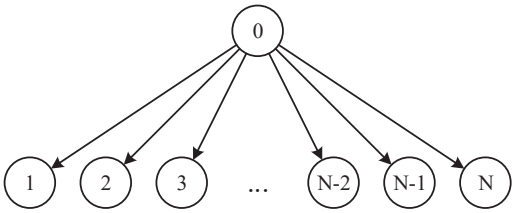


Fig. 4. A data flow graph with a root task and multiple successors (called *star*)

A. Simplification under Sequential Scheduling Policies

In the formulation of M2CP, we allow multiple tasks to share the CPU resources and network resources on both AR and EC devices. Two types of parallelism can be exploited to improve data throughput of AR applications, namely, data parallelism and pipeline parallelism [20]. However, both forms of parallelism would not benefit the makespan of processing a single frame [29]. In fact, sequential scheduling policies are known to minimize makespan on a single resource. Therefore, in the subsequent discussion, we limit the allocation of each resource to only one task at a time[1]. As a result, (2)–(4) can be simplified as,

$$y_i(t) = \begin{cases} 0 & t \notin \left[t_i^S, t_i^C\right) \\ (1-x_i)f_L + x_i f_R & t \in \left[t_i^S, t_i^C\right) \end{cases} \quad \forall i \in V, \tag{14}$$

and (6)–(7) can be simplified as,

$$z_{i,j}(t) = \begin{cases} 0 & t \notin \left[t_i^C, t_i^C + t_{i,j}^T\right) \\ b & t \in \left[t_i^C, t_i^C + t_{i,j}^T\right) \end{cases} \quad \forall \langle i,j \rangle \in A. \tag{15}$$

As can be seen from (14) and (15), in theory, the optimal case is that $y_i(t)$ and $z_{i,j}(t)$ are only related to the frequency f_L or f_R and bandwidth b, instead of time. We hence reduce (5) and (8) to

$$t_i^P = t_i^C - t_i^S = (1-x_i)\frac{p_i}{f_L} + x_i \frac{p_i}{f_R} \quad \forall i \in V, \tag{16}$$

and

$$t_{i,j}^T = |x_i - x_j| \frac{c_{i,j}}{b} \quad \forall \langle i,j \rangle \in A. \tag{17}$$

Under the sequential scheduling policy, we now consider the two special cases.

B. Chain Data Flow Graph

With the chain precedence, tasks have to be scheduled one after another. Therefore, the scheduling decision is straightforward. The only decision variables in M2CP are the partition variables x_i's. It has been proven in [30], [31] that the optimal partition can be found in polynomial time. In particular, given non-zero network transfer delay, it can be shown that if task j and k ($j \leq k$) are executed on the EC device, all tasks between j and k on the chain shall be executed on the device

[1]In the simplified formulation, parallelism still exists between the AR and EC devices, and between computation and communication as they correspond to different resources.

is in the optimal partitioning as well. As a result, the minimum makespan and the optimal partition can be computed as,

$$\min_{j\le k}\sum_{i<j}\frac{p_i}{f_L}+\frac{c_{j-1,j}}{b}+\sum_{i\in[j,k]}\frac{p_i}{f_R}+\frac{c_{k,k+1}}{b}+\sum_{i>k}\frac{p_i}{f_L}, \quad (18)$$

where $c_{0,1}=c_{N,N+1}=0$.

C. Star Data Flow Graph

In the star data flow graph in Figure 4, tasks 1 – N share a common predecessor. We call the *root* task 0. After the completion of task 0, successor tasks can be scheduled in parallel on different devices. In absence of communication costs, M2CP is equivalent to the $Q2||C_{max}$ problem, where parallel tasks are scheduled on two uniform processors at different speeds and the problem asks to minimize the makespan C_{max}. $Q2||C_{max}$ is known to be NP-hard [32]. Thus, M2CP is NP-hard as well. Both the Longest Processing Time (LPT) schedule algorithm [33] and the list schedule algorithm [34] provide constant approximation ratios algorithms to $Q2||C_{max}$. However, when the network bandwidth is finite between the AR and the EC devices, the total time to execute a task (including both the communication time if any and the computation time) on different devices becomes "unrelated" (in the sense that the total time the task takes on different devices is no longer proportional to the processor speeds). This makes M2CP with star data flow similar to $R2||C_{max}$, i.e., minimizing makespan for parallel tasks on unrelated processors.

In [35], the Efficiency First Scheduling (EFS) algorithm was proposed to determine the schedule of parallel tasks on M unrelated processors. EFS first computes the efficiency of task i on processor $p, p\in M$. Each processor keeps a list of all tasks in the non-increasing order of their efficiency. It further maintains the workload w_p of tasks assigned to processor p so far. A processor is set to be inactive if either it has no more unassigned task in its list or the remaining unassigned tasks all have efficiencies lower than a threshold. A task is scheduled on the active processor p with the least w_p. EFS is a polynomial time algorithm and has been proven to attain a constant approximation ratio [35].

Scheduling parallel tasks with star precedence constraints and communication costs are different from $R2||C_{max}$ in two aspects. First, the time to complete a task on a device depends on which device its precedent task is executed. If both are on the same device, the makespan is solely the computation time. Otherwise, the communication time needs to be considered. Second, communication and computation can happen concurrently for different tasks. Thus, the makespan of assigned tasks on the EC device is not naively the sum of all computation time. For example, consider task 0 is scheduled on the AR device while task 1 and 2 are currently assigned to the EC device. The makespan on the EC device is given by

$$\frac{c_{0,1}}{b}+\max(\frac{p_1}{f_R},\frac{c_{0,2}}{b})+\frac{p_2}{f_R}. \quad (19)$$

The EFS algorithm [35] was proposed to determine the schedule of parallel tasks on the unrelated processors. We next propose EFS2, an adaption of EFS, for task scheduling on two uniform devices with star precedence and communication costs.

Let $\mu_{i,L}$ and $\mu_{i,R}$ be the processing time of the ith task on the AR device (denoted as "L") and the EC device (denoted as "R") ($i\in V$), respectively. Therefore,

$$\begin{cases}\mu_{i,L}=\dfrac{p_i}{f_L}+|x_i-x_j|\dfrac{c_{i,j}}{b}\\ \mu_{i,R}=\dfrac{p_i}{f_R}+|x_i-x_j|\dfrac{c_{i,j}}{b}\end{cases} \quad i\in V,\langle i,j\rangle\in A. \quad (20)$$

In Figure 4, both $\mu_{i,L}$ and $\mu_{i,R}$ ($i=1,2,\ldots,N$) are uniquely determined once the partition of its predecessor is known. Denote the efficiency of the ith task on the AR and EC devices as $e_{i,L}$ and $e_{i,R}$, respectively,

$$\begin{cases}e_{i,L}=\dfrac{\min\{\mu_{i,L},\mu_{i,R}\}}{\mu_{i,L}}\\ e_{i,R}=\dfrac{\min\{\mu_{i,L},\mu_{i,R}\}}{\mu_{i,R}}\end{cases} \quad i\in V. \quad (21)$$

Let Γ_R (Γ_L) be the set of tasks currently assigned to the EC (AR) device. Denote the workload of the tasks in Γ_R (Γ_L) by w_R (w_L). If task 0 runs on the AR device, after assigning a new task j to either device, the workloads are updated as,

$$\begin{cases}w_L &= w_L+\frac{p_j}{f_L}(1-x_j)\\ w_R &= \max\left(w_R,\sum_{i\in\Gamma_R}\frac{c_{0,i}}{b}\right)+\frac{p_j}{f_R}x_j\end{cases}. \quad (22)$$

Similarly, if task 0 runs on the EC device, after assigning a new task j to either device, the workloads are updated as,

$$\begin{cases}w_L &= \max\left(w_L,\sum_{i\in\Gamma_L}\frac{c_{0,i}}{b}\right)+\frac{p_j}{f_L}(1-x_j)\\ w_R &= w_R+\frac{p_j}{f_R}x_j\end{cases}. \quad (23)$$

The recursion in the second equation in (22) (or in the first equation in (23)) is due to the fact that transmission times are additive, and the computation of a task can proceed only when the required data has been transferred from the AR device.

Algorithm 1 summarizes the proposed EFS2 algorithm. It extends the EFS algorithm by consideration of communication time to transfer data between the root task and the successor tasks if assigned to different devices. AM, an assignment function, indicates the partition decision. In EFS2, tasks are assigned to an active device with the least workload (w_R or w_L) thus far. The threshold parameter η controls the percentage of tasks running on the AR device. If η is set too high, most tasks will run on the EC device. In the algorithm, we choose $\eta=\frac{\sqrt{3}}{2}$ as explained in Theorem 1. It is easy to show that the computation complexity of EFS2 is $O(N^2)$ since sorting requires $O(N\log N)$ time and the calculation of w_L or w_R requires $O(N)$. We can further show that EFS2 has a constant approximation ratio. Formally, let Π^* and Π be the schedules by the optimal and the EFS2 scheduling. The makespans of Π^* and Π are $t^C_{\Pi^*}$ and t^C_{Π}, respectively. We can show that,

Theorem 1. *The EFS2 algorithm for tasks with star precedence constraint and communication costs satisfies,*

$$t^C_\Pi \le 2\sqrt{3}t^C_{\Pi^*}, \tag{24}$$

if and only if the threshold of the active device satisfies $\eta = \frac{\sqrt{3}}{2}$.

Due to the page limitation, we only give a proof sketch here.

Proof. Let z be the task that finishes the last in EFS2. It has,

$$t^C_\Pi = \mu_{z,A_\Pi(z)} + t^S_z. \tag{25}$$

Next, with Lemma 2 and Lemma 6 in [35], it can be proved that the inequalities for the approximation analysis of each part in (25) hold, namely

$$\mu_{z,A_\Pi(z)} \le \frac{1}{\eta} t^C_{\Pi^*}, \tag{26}$$

and

$$t^S_z \le \left(2\eta + \frac{1}{2\eta}\right) t^C_{\Pi^*}. \tag{27}$$

According to (25), (26) and (27), we conclude that when $2\eta = \sqrt{3}$, t^C_Π satisfies the optimal value $2\sqrt{3}t^C_{\Pi^*}$. □

V. Solution Approach to M2CP for General Data Flow Graphs

As discussed in Section IV, M2CP is NP-hard. In this section, we develop a heuristic solution approach to general data flow graphs modeled as DAGs. We use the polynomial algorithms for chain and star graphs as building blocks. The proposed Dynamic Planning Algorithm (DPA) is illustrated in Algorithm 2. Let $\mathcal{F}, \mathcal{E}, \mathcal{S}$ be the sets for tasks whose schedules have been decided, tasks that are eligible to be scheduled, namely the tasks whose precedent tasks have completed, and tasks whose schedules to be determined next. *CurrentTime* is used to keep track of the progress of the virtual schedule. It scans the DAG from the source node to the sink node and identifies groups of tasks to be scheduled next. As units of scheduling, we focus on groups of tasks that either follow a chain or can be scheduled in parallel (without precedence constraints). The computation complexity of DPA for each frame is related to that of the two base algorithms. Hence it is $O(MN^2)$ if we process the M frames with N tasks each frame.

Whether a task belongs to a chain or a star depends the schedule in place and its execution. Take the example in Figure 5. Intuitively, task 7, task 8, and task 9 can be treated as star tasks since they are of the same shortest path distance from the source node. However, from a scheduling viewpoint, this is not always the case and is highly dependent on the amount of computation and data transfer of the previous tasks and their assignment. For example, consider the case when task 5 and task 6 are executed remotely and their sibling task 4 is executed locally. Suppose that the two tasks running on the remote device are finished before task 4. The idle EC device will decide that task 8 and 9 can be scheduled as a star data flow while task 7 is not yet eligible.

Algorithm 1 EFS2 algorithm

Input: $N, \check{t}^P_i, \hat{t}^P_i, t^T_{i,j}, i \in V, \langle i,j\rangle \in A$;
Output: AM;

Compute each $e_{i,L}$ and $e_{i,R}$ according to (21), $i \in [1, N]$;
Create a list of the tasks $i = 1, 2, \cdots, N$ sorted in non-decreasing order of each $e_{i,L}$ and $e_{i,R}$;
Designate all tasks as "unassigned" and both devices as "active";
while not all tasks are assigned **do**
  Find the minimal value between w_L and w_R which device is denoted as $M, M \in \{L, R\}$;
  Find the next unassigned task i on M's list of tasks;
  if i does not exist or $e_{i,L} < \eta$ **then**
    Mark device M as "inactive";
  else
    $AM(i) = M$;
    Designate task i as being "assigned";
    $t^S_i = w_M$;
    Update w_M according to (22) or (23);
  end if
end while

Algorithm 2 DPA algorithm

Input: All tasks $i, i \in V$
Output: $makespan, x_i, i \in V$

$CurrentTime = 0$;
$\mathcal{F} = \{\mathbf{0}\}$; $\mathcal{E} = \Gamma^+_{\mathbf{0}}$; $\mathcal{S} = \emptyset$;
while $\mathcal{E} \neq \emptyset$ **do**
  Compute t^S_i and t^C_i, $i \in \mathcal{E}$;
  if $|\mathcal{E}| == 1$ **then**
    Mark task i as a member in chain;
    $\mathcal{E} = \Gamma^+_i$; $\mathcal{S} \leftarrow i$;
  else
    Sort tasks in $\mathcal{E}$;
    for each task $j \in \mathcal{E}$ **do**
      if $t^S_j < \max_{i\in\mathcal{S}} t^C_i$ **then**
        $\mathcal{S} \leftarrow i$;
      end if
    end for
    if there is only one non-member task k in $\mathcal{S}$ **then**
      Mark task k as a member in chain; $\mathcal{E} = \Gamma^+_k$;
    end if
  end if
  if $\mathcal{S}$ is a chain **then**
    Call the chain algorithm [30] for the chain data flow;
  else
    Call the EFS2 algorithm for the star data flow;
  end if
  Update all variables accordingly;
end while
$makespan = CurrentTime$;

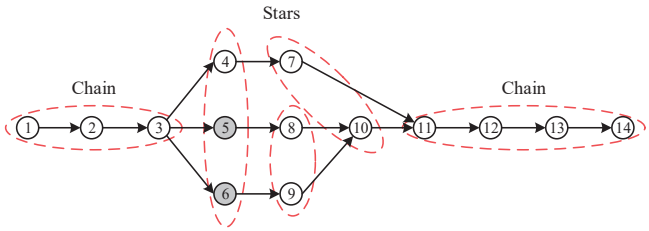


Fig. 5. An example that DPA divides the general DAG into the chain and/or the star data flow graphs as building blocks

VI. Evaluation

We have implemented the proposed algorithms on the Android platform as a demonstration. Evaluations are done using both data collected from the real AR and EC devices using simulations.

A. Prototype Implementation

The partitioning decision module consists of two parts: the profiler and the decision-maker. Both of them run on separate threads on each device. The profiler is lightweight since it just logs the available CPU cycles and network bandwidth. The profile information is sent by piggy-backing transfered data to the other device for synchronization. The decision-maker uses the profiles to make partition decisions (a binary vector) following the DPA algorithm. Its overhead is based on the complexity of DPA and is negligible when the number of tasks is not too large [36]. At the beginning of each task, the system determines where the task is to run based on the result of the decision-maker. If the respective decision variable is 0, it invokes the task on the AR device; or on the EC device, otherwise.

B. Setup

Environment: A pair of Recon Jet smart glasses [37] are used as an AR device and a Huawei P30 phone running an Android operating system serves as an EC device. The maximum CPU frequencies of the Recon Jet smart glasses and the smartphone are 1.0GHz and 2.6GHz, respectively. The maximum transmission bandwidth of the Bluetooth radio used to communicate between the two devices is 6.4Mbps. We set the deadline for the app to run at 120ms and the number of GOP to 25.

Data Flow Graph: For evaluation, we consider a data flow graph at two different granularities as shown in Figure 2.

- A two-stage data flow consisting of a *detection* task that detects every object (including suspected objects) in the image and a *recognition* task that recognizes the objects from an existing database.
- A fine-grained data flow, where the *detection* and the *recognition* tasks are further divided into 5 and 7 fine-grained tasks according to the data flow graph in Figure 2, respectively.

Baselines: In addition to DPA, three algorithms have been implemented for comparison.

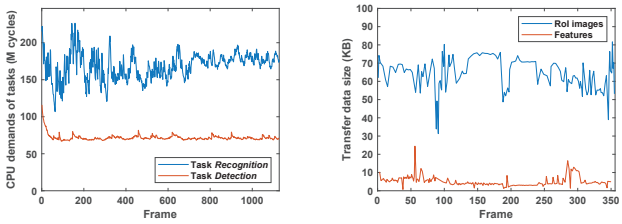
- *FS*: The partition decision follows that of DPA but the scheduling of tasks on the AR and EC devices assumes fair sharing of CPU resources. Network bandwidth is allocated proportionally according to the amount of data transferred between the communicating tasks. The way to allocate resources is the same as that of [22].
- *Cluster*: Partitioning is decided by a heuristic clustering method in [30]. It first converts a DAG data flow graph to a tree graph by pruning links with lighter costs. Next, computation partitioning is done by first sorting the sum cost of the communication and the computation on the EC device, and then distributing the load roughly equally between the two devices. Since no specific is provided in [30] how the tasks on each device are scheduled, we assume the same mechanism is adopted as in DPA.
- *Optimal*: For the star data flow graph with limited tasks, we find the optimal makespan by solving M2CP directly using exhaustive search.

Partitioning Schemes: We also evaluate three naive schemes for comparing our dynamic partitioning algorithm, namely, *Local* where all tasks are executed in the AR device, *Remote* where all tasks are offloaded to the EC device, and *Fixed* where the *detection* and *recognition* tasks are executed on the AR and the EC devices, respectively.

Profiling: As inputs to M2CP, we first record the response time of each task by measuring the running time of the CPU core. The computation demand of each task is then estimated by the product of the response time and the CPU frequency of the associated device. The reason for not using the directly measured response time is because it is related to the computation capability of each device.

In practice, however, we may not know the demand of a task before its execution is completed when the task execution time is input sensitive. Alternatively, for online profiling, we use historical demands to predict the computation demand of the current frame. This approach is reasonable because the computation demands of neighboring frames are likely to be similar. We first use the approach mentioned at the beginning to record computation demands. During online profiling afterward, the computation demand of the current task ($\bar{p}_i$) is updated using $\bar{p}_i = \alpha p_i + (1-\alpha)\bar{p}'_i$, where $\bar{p}'_i$ is the computation demand from the previous frames. The result is shown in Figure 6(a), where parameter α is empirically set to be $\frac{7}{8}$ in the experiments. In the figure, the computation demand of task *detection* is between $60-80$M cycles, while the demand of task *recognition* is between $100-200$M cycles.

Similarly, the result of the data demand is shown in Figure 6(b). In the figure, it shows the data size of task *detection* and task *recognition*. It can be further divided into two groups, Grayscale RoI images, typically, $50-70$KB per frame, and small amount of transferred data between tasks, such as object positions and feature values, around $0-10$KB per frame. In



(a) Computation demand profiling (b) Data demand profiling

Fig. 6. An example of task demands profiling

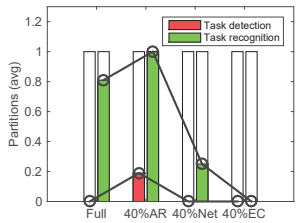


Fig. 7. Partitions for the two-stage data flow graph

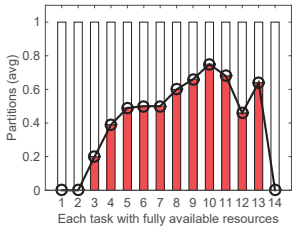


Fig. 8. Partitions for the fine-grained data flow graph

addition, the data transferred from the source and the detection tasks are 320×240 RGB images at 225KB per frame.

C. Evaluation Results

Computation Partitioning: For the two-stage partitioning, in functionality evaluation, we vary the resource availability on AR, EC and the network as detailed in Table I. In all scenarios, DPA achieves the optimal makespan. Figure 7 shows the percentage of the *detection* and *recognition* tasks running on the EC device. Zero percent means the task always runs on the AR device, whereas one hundred percent means it always runs on the EC device. As the resource demand of each task varies from frame to frame, with fixed resource availability, the outcome of computation partition can still be different. From Figure 7, we observe that for most frames, *detection* tasks run on the AR device. However, as the available resource on the AR device reduces to 40%, the *detection* tasks are offloaded to the EC device instead among 20% of frames, while the *recognition* tasks for all frames are executed on the EC device in this case. The small percentage is because the *detection* task is lightweight in CPU demands but has a high data demand (RGB raw images from the camera as input). Reducing the availability of EC computation resource to 40% has a more dramatic effect on the partitioning, where both tasks are now running on the AR device. This is because the prolonged computation time together with the latency incurred by transferring data over the network makes offloading less desirable. The effect of reducing network resources is more complicated. In this case, the latency in transferring the RoI blocks of images and the results after recognition increases. As a result, fewer *recognition* tasks run on the EC device.

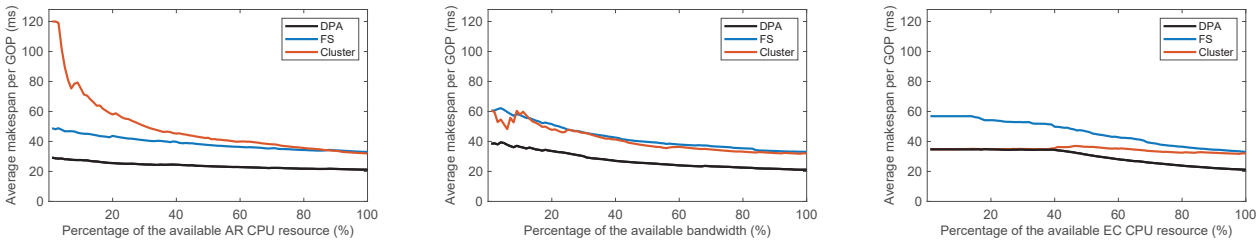
Furthermore, the partitions of the fine-grained data flow with the 100% available resource are shown in Figure 8. In the figure, each frame contains 14 tasks. The setting is the same as that of the two-stage data flow. However, the partitioning result is different from that in Figure 7. Task 2 always executes on the AR device due to a large RGB image transfer delay and a relatively small computation demand. Less than 50% of frames belonging to task $3-6$ execute on the EC device. The percentage of remote executions increases for the *recognition* tasks, but it is still lower than 80% compare to Figure 7. We also observe that the star tasks in Figure 8 have different partition outcomes (e.g., task $3-5$) due to workload balancing. Interestingly, it is preferable to run task 13 remotely comparing to task 12. Similarly, two of three predecessors of task 13 (i.e., task 10 and task 11) tend to execute on the EC device while task 12 executes on the AR device due to workload balancing.

TABLE I
RESOURCE AVAILABILITY SETTING FOR THE DATA FLOW GRAPH

Labels	CPU res. on AR	Bandwidth	CPU res. on EC
Full	100%	100%	100%
40% AR	40%	100%	100%
40% Net	100%	40%	100%
40% EC	100%	100%	40%

Performance with Variable Resources: In this set of experiments, we vary the availability of one resource (AR, EC or network) and fix the rest to evaluate the performance of different algorithms on the DAG data flow graph. Due to its computation complexity, we are unable to determine the optimal solutions using exhaustive search.

Figure 9(a) shows the makespan when the available computation resource of the local AR device varies from 1% to 100%. DPA achieves the smallest makespan. As expected, as the available computation resource increases, makespan decreases for all three algorithms. We found in the figure, Cluster tends to assign more tasks on the AR device and therefore incurs longer makespan when the resource on the AR device is less. In some cases, under Cluster, the app stops working since the makespan exceeds the deadline. Figure 9(b) shows the makespan achieved when the available network bandwidth varies from 1% to 100%. Ideally, more available network bandwidth should imply shorter makespan. However, we observe when the network bandwidth is small, the relation is not monotone with all three algorithms. This is because smaller network bandwidth tends to put more tasks on the AR device by reducing the communication latency, which in turn increases the computation time. The trade-off between communication and computation is more pronounced in this scenario. However, as more network bandwidth is available, we indeed observe close to monotonic behaviors. Again, in this case, DPA, despite being sub-optimal (as evident from its non-monotonicity) outperforms the other two algorithms. Finally, in Figure 9(c), the computation resource on the EC device varies from 1% to 100%. Up to until around 60% resource availability on the EC device, the makespans achieved



(a) Under the AR workload (b) Under the bandwidth workload (c) Under the EC workload

Fig. 9. Comparisons of different algorithms on the fine-grained data flow graph

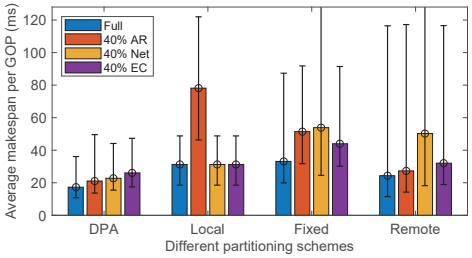


Fig. 10. Comparisons of different partitioning schemes

by Cluster and DPA are comparable and are roughly constant. This is because all tasks run on the AR devices. However, as more resource is available on the EC device, some tasks are offloaded to the EC device. From the figure, the makespan of Cluster rises after 60% since the algorithm tries to balance the load and may transfer network-intensive tasks to EC. DPA is more adept at utilizing the resource on the EC device and achieves a lower makespan compared to the other algorithms.

Effects of Different Partitioning Schemes: We evaluate the performance of different partitioning schemes under different resource availability. The fine-grained data flow graph is utilized with resource availability shown in Table I. The results are shown in Figure 10. From the figure, DPA can achieve the best performance regardless of resource availability. Moreover, DPA degrades gracefully even when available resources drop significantly at 30% increment in latency in the worst case. In contrast, when the amount of available resources decreases, other partitioning schemes perform poorly. The *Local* and *Remote* partitioning schemes are susceptible to the computation resources on AR and EC while the *Fixed* partitioning scheme is affected by all resources as shown in the figure. In particular, when the network resource available drops below 40%, executions using *Fixed* and *Remote* can no longer meet the deadline. Note that the performance of *Local* is not affected by the varying availability of network or EC resources since the execution is on the local device only.

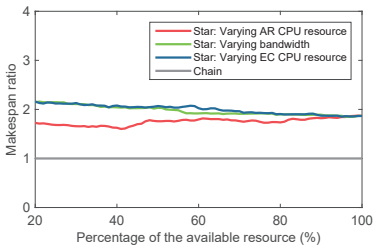
Impacts of Precedent Constraints: In this set of experiments, we evaluate the makespans achieved by DPA, FS and Cluster algorithms for chain and star data flow graphs. The makespans achieved by FS and Cluster algorithms are normalized by that of DPA. In the experiments, we vary the CPU resource availability on the AR and EC devices and the bandwidth availability from 20% to 100% and investigate their impact on the partitioning decisions.

From Figure 11(a) and 11(b), all algorithms perform identically for the chain data flow regardless of resource availability. This is expected due to the strict precedence constraints. For the star data flow, on the other hand, we see from the figures that both algorithms have $40\% - 60\%$ higher makespans. Comparing the two figures, it can be observed that lower resource availability on the AR device has more negative impacts on the performance of Cluster. Under the low available CPU resource of AR, to minimize the makespan, almost all the partitions should be uploaded onto the EC device. However, Cluster initializes by putting all partitions on the AR device and prunes edges in the data flow graph to make it a tree. It then greedily tries to balance the loads between the AR and EC devices. As a result, some tasks remain to be executed on the AR device resulting in a longer makespan.

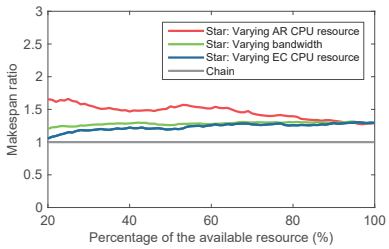
Empirical Validation of Theorem 1: We compare the performance of the proposed EFS2 algorithm with the optimal solution in handling star tasks modeled by a star data flow graph. Figure 12 shows the ratio between the makespans achieved by the EFS2 algorithm and the optimal solution to problem M2PC for different frames. The theoretical bound in Theorem 1 is included as the red line. It is clear from the figure that the makespans attained by EFS2 is close to that of the optimal and are always lower than the upper bound.

VII. Conclusion

In this paper, we investigated delay-sensitive computation partitioning for MAR applications in edge computing. Modeled as a data flow graph, the makespan minimization problem is NP-hard due to the complex dependency among tasks and variable resource constraints in both computation and communication. We designed DPA, a polynomial-time algorithm for this problem. For special data flow graphs modeled as a chain or a star, the algorithm can provide optimal solutions or solutions with a constant approximation ratio. Evaluations under realistic settings demonstrated the effectiveness of DPA and that it outperformed other heuristic policies.



(a) Makespan of FS (normalized)



(b) Makespan of Cluster (normalized)

Fig. 11. Comparisons of FS, Cluster and DPA in chain and star with different resource availability

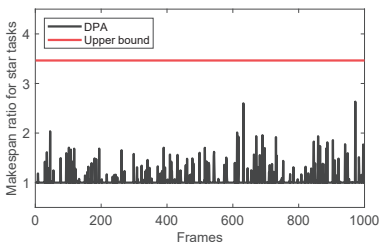


Fig. 12. Makespan ratio for star tasks, DPA vs. Optimal

REFERENCES

[1] T. Braud, F. H. Bijarbooneh, D. Chatzopoulos, and P. Hui, "Future networking challenges: The case of mobile augmented reality," in *Proc. IEEE ICDCS'17*, Jun. 2017, pp. 1796–1807.

[2] T. Starner, "The challenges of wearable computing: Part 1," *IEEE MICRO*, vol. 21, no. 4, pp. 44–52, 2001.

[3] S. Liu. (2019, Dec.) Augmented reality (AR) market size worldwide in 2017, 2018 and 2025. [Online]. Available: https://www.statista.com/statistics/897587/world-augmented-reality-market-value/

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

[5] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *ACM Mobicom'19*, Los Cabos, Mexico, Oct. 2019.

[6] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. ACM ASPLOS'17*, Xi'an, China, Apr. 2017, pp. 615–629.

[7] K. Kumar, J. Liu, and Y.-H. Lu, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[8] C. Zhang, Y. Cui, R. Zheng, J. E, and J. Wu, "Multi-resource partial-ordered task scheduling in cloud computing," in *Proc. IEEE IWQoS'16*, Jun. 2016.

[9] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, 2013.

[10] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: Architecture, applications, and approaches," *Wiley Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[11] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE TWC*, vol. 12, no. 9, pp. 4569–4581, 2013.

[12] S. Osman, D. Subhraveti, G. Su, and J. Nieh, "The design and implementation of Zap: A system for migrating computing environments," in *Proc. USENIX OSDI'02*, Dec. 2002, pp. 361–376.

[13] M. Liu and Y. Liu, "Price-based distributed offloading for mobile-Edge computing with computation capacity constraints," *IEEE Wireless Communications Letters*, vol. 7, no. 3, pp. 420–423, 2018.

[14] M. Halpern, Y. Zhu, and V. J. Reddi, "Mobile cpu's rise to power: Quantifying the impact of generational mobile cpu design trends on performance, energy, and user satisfaction," in *Proc. IEEE HPCA'16*, Barcelona, Spain, Mar. 2016, pp. 64–76.

[15] R. Newton, S. Toledo, L. Girod, S. Madden, and H. Balakrishnan, "Wishbone: Profile-based partitioning for sensornet applications," in *Proc. USENIX NSDI'09*, Apr. 2009, p. 14.

[16] G. C. Hunt and M. L. Scott, "The Coign automatic distributed partitioning system," in *Proc. USENIX OSDI'99*, Feb. 1999, pp. 187–200.

[17] E. Cuervo, A. Balasubramanian, and D. ki Cho, "MAUI: Making smartphones last longer with code offload," in *Proc. ACM Mobisys'10*, Jun. 2010, pp. 49–62.

[18] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *ACM Computer systems'11*, Salzburg, Austria, Apr. 2011, pp. 301–314.

[19] C. Wang and Z. Li, "Parametric analysis for adaptive computation offloading," in *PLDI'04: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation*, Jun. 2004, pp. 119–130.

[20] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling interactive perception applications on mobile devices," in *Proc. ACM MobiSys'11*, Jun. 2011, pp. 43–56.

[21] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Transactions on Mobile Computing*, vol. 16, no. 11, pp. 3056–3069, 2017.

[22] L. Yang, J. Cao, S. Tang, T. Li, and A. T. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," in *Proc. IEEE CLOUD'12*, Jun. 2012, pp. 794–802.

[23] J. Cao, L. Yang, and J. Cao, "Revisiting computation partitioning in future 5G-based edge computing environments," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2427–2438, 2019.

[24] L. Yang, B. Liu, J. Cao, Y. Sahni, and Z. Wang, "Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds," in *IEEE CLOUD'17*, Honolulu, CA, USA, Jun. 2017.

[25] T. Ahonen, A. Hadid, and M. Pietikainen, "Face recognition with local binary patterns," in *Proc. Springer ECCV'04*, May 2004, pp. 469–481.

[26] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *IJCV*, vol. 60, no. 2, pp. 91–110, 2004.

[27] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE Infocom'12*, Mar. 2012, pp. 945–953.

[28] W. Zhang, B. Han, and P. Hui, "On the networking challenges of mobile augmented reality," in *ACM VR/AR Network'17*, Los Angeles, CA, Aug. 2017, pp. 24–29.

[29] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *Proc. ACM SIGCOMM'14*, Chicago, IL, Aug. 2014, pp. 431–442.

[30] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *Proc. IEEE INFOCOM Workshop on Mobile Cloud Computing*, Apr. 2014, pp. 352–357.

[31] Y. Zhang, H. Liu, L. Jiao, and X. Fu, "To offload or not to offload: An efficient code partition algorithm for mobile cloud computing," in *Proc. IEEE CLOUDNET'12*, Nov. 2012, pp. 80–86.

[32] P. Brucker, *Scheduling algorithms*, 5th ed. Berlin: Springer, 2007.

[33] T. Gonzalez, O. H. Ibarra, and S. Sahni, "Bounds for LPT schedules on uniform processors," *SIAM Journal on Computing*, vol. 6, no. 1, pp. 155–165, 1977.

[34] Y. Cho and S. Sahni, "Bounds for list schedules on uniform processors," *SIAM Journal on Computing*, vol. 9, no. 1, pp. 91–103, 1980.

[35] E. Davis and J. M. Jaffe, "Algorithms for scheduling tasks on unrelated processors," *ACM JACM*, vol. 28, no. 4, pp. 721–736, 1981.

[36] W. M. Johnston, J. R. P. Hanna, and R. J. Millar, "Advances in dataflow programming languages," *ACM CSUR*, vol. 36, no. 1, pp. 1–34, 2004.

[37] (2017) Recon Instruments. [Online]. Available: https://engage.reconinstruments.com/