2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application

Real-Time Schedulability Analysis for Overloaded Primary-to-Secondary Processor Systems

Mitchell Duncan, *Wayne State University*
Fatima Raadia, *Wayne State University*
Syeda Tanjila Atik, *Wayne State University*
Marco Brocanelli, *Wayne State University*
Nathan Fisher, *Wayne State University*

***Abstract*—Heterogeneous computing has emerged as a powerful tool for modern highly-connected systems to efficiently distribute computing resources. Yet, many systems with real-time constraints may require expensive over-provisioning to meet their requirements. To address this problem, a real-time system may be designed to be overloaded on its primary processor, but be made viable through the support of a powerful secondary processor. In such a system, workload is scheduled on the primary processor by default, but some jobs that would cause a deadline miss can be offloaded, through a transfer of the necessary data, to the secondary processor. We propose an earliest-deadline-first (EDF) based scheduling framework to demonstrate schedulability conditions of such a system, including the costs incurred by offloading. This represents a first stage towards a complete understanding of this complex class of systems and their dynamics.**

I. Introduction

Modern computing is rapidly expanding into multiple new and interdisciplinary domains. Real-time computing in particular is finding itself relevant in many fresh areas, like heterogeneous and distributed computer resources. As more fields create demands for computation, heterogeneous computing systems are emerging as a flexible solution for managing the variety of requirements from many complex systems. Notably, some of the systems generating these demands, e.g., navigation tasks of autonomous vehicles, drones, and autonomous mobile robots, have safety-critical aspects that must satisfy certain timing constraints. In general, an on-board primary-secondary processor setup enables computational offloading when the primary processor is *overloaded*, i.e., a given taskset is unchedulable on the primary processor. For example, jobs can be offloaded from the primary little core (energy-efficient but weak) to the secondary big core (powerful but power-hungry) in *big.LITTLE* architectures when deadlines cannot be met only with the little core. Another example is when jobs are offloaded from the on-board primary processor to a remote powerful edge server through the on-board network processor, i.e., the secondary processor.

In this work we address the schedulability of online decisions for offloading permitting the secondary processor's resources to be utilized only when absolutely needed, which allows real-time developers to safely use these powerful resources without fear of excessive network traffic or wasteful energy usage on their local devices. Specifically, we determine an offloading policy that minimizes the number of offloaded jobs to the secondary processor and then provide a schedulability analysis for the following general scenario: a multiprocessor system comprised of a primary local processor and a secondary local processor that either executes jobs in place of the primary processor or offloads data to powerful remote servers.

Our work utilizes a variety of existing concepts and tools. Prior studies in similar scenarios do not take real-time constraints into account (Goldstein [1], Lv et al. [2]), do not select jobs for offloading based on preserving primary schedulability (Schönberger et al. [3]), or are based on choosing tasks to offload offline [4], [5]. *However, to the best of our knowledge, there is no prior work that guarantees the schedulability of sporadic real-time tasks on such a heterogeneous system using demand-based real-time analysis while minimizing the amount of offloaded demand as a central goal.*

In summary, this paper makes the following contributions:

- A real-time analysis of primary-secondary processor systems running constrained deadline sporadic tasksets;
- An analysis on the amount of demand offloaded to the secondary processor;
- An extensive evaluation of offloaded demand estimation and schedulability ratio comparison for a variety of task systems compared to a runtime simulated system.

The rest of the paper is organized as follows. Section II describes the related work. Section III details our system model. Section IV describes our scheduling framework and how we address deadline-aware offloading. Section V details how we construct a schedulability analysis for this framework and derive demand-based bounds. Section VI shows our experimental results and Section VII concludes the paper.

II. Related Work

The general topic of computational offloading has been well studied in the past. A good survey of the field is provided by Mach and Becvar [6]. Some existing studies on real-time computation offloading focus on specific application such as object recognition and tracking [7] or assume, as default model, that all tasks will have a remote execution component such as Schönberger et al. [3]. Contrary to such solutions, our paper does not focus on a specific application but analyzes a system that executes jobs at the primary processor by default, and on the secondary only when necessary. Both Liu et al. [5] and Toma et al. [4] focus on demand-based edge offloading, but use offline decisions to determine the set of tasks to offload (and all jobs of selected tasks are offloaded). Our system makes job level decisions, and does so online (i.e., at runtime); thus, the objectives of the approaches are orthogonal.

979-8-3503-1993-4/22/$31.00 ©2022 IEEE
DOI 10.1109/HPCC-DSS-SmartCity-DependSys57074.2022.00289

Our system model may be viewed as composed of heterogeneous multi-processors (e.g., Raravi et al. [8]). Most of these papers involve balanced assignment of tasks to the processors, which is different from our case study where the secondary processor receives jobs only when the primary processor cannot schedule them. *To the best of our knowledge, this is the first work studying the schedulability of tasks in a primary-secondary system by including the scheduling of job data transmissions.* In particular, considering the online job offloading policy described in Section IV, we design the proposed offline schedulability test by analyzing the amount of demand offloaded to the secondary processor in Section V.

III. System Model

Our long-term goal (beyond this paper) is to develop general schedulability tests for online job-level offloading policies for a system of n tasks scheduled across a primary processor, and a general set of secondary processors. However, due to the high complexity of the problem and given the lack of previous work focusing on such online offloading policies with strict real-time analysis, in this paper we make the following assumptions that allow us to simplify the problem and make the first step towards the above goal:

1) The system has one primary processor that executes task τ_i with a Worst-Case Execution Time (WCET) C_{i_P}, and one secondary processor that executes them with a WCET C_{i_S};
2) The primary processor uses preemptive EDF scheduling;
3) The secondary processor is capable of using either preemptive or non-preemptive EDF;
4) System tasks are modeled by the constrained-deadline sporadic task model;
5) A system-wide scaling factor $\gamma \in (0, 1]$ can be used to describe the non-equal relationship between the primary and secondary processors' execution time as follows: $C_{i_S} = \gamma * C_{i_P}$ for $\tau_i \in \tau$.

The first assumption restricts us to one offloading processor, and requires it to be well analyzed and suitable for real-time applications. The second and third assumptions are requirements for these processors, but not restrictive in many applications. The fourth assumption is a typical real-time system model while the last one assumes a uniform scaling factor in WCET of tasks from primary to secondary processor. Our model is a slight variation on this standard, with an additional term to address the presence of both processors. Each task τ_i is of the form:

$$\tau_i =< C_{i_P}, C_{i_S}, T_i, D_{i_P}, D_{i_S} >$$

where C_{i_P} and C_{i_S} are the WCET of a task on the primary and secondary processor, respectively. If the secondary processor is a network processor, C_{i_S} will denote the transfer time. Note, we are aware that execution times may vary at runtime. However, having bounded transfer time and execution times is fundamental for real-time analysis [9] since real-time guarantees cannot be provided if applications cannot bound their runtime. T_i is the shortest time between the arrival of two jobs of the same task. Each task has a relative deadline D_{i_P} that must be met if a job executes on the primary platform, and another relative deadline D_{i_S} if a job executes on the secondary platform. In both cases, these represent the offset of a job's arrival to it's absolute deadline. In general, we assume that a job's secondary deadline may be more restrictive than its primary deadline, so $D_{i_S} \leq D_{i_P}$ We plan to remove the above first assumption in future work. Note that this system model can be used on most on-board primary-secondary processor setups. For example, if computation is offloaded to an edge server, the primary processor offloads jobs through the network (secondary) processor by subtracting the worst-case response time of the job on the edge server from the original deadline D_{i_P} to find D_{i_S}, thus including possible delays due to shared edge resources. The updated deadline is then used to conduct the schedulability analysis. We will evaluate this restrictive setup in the evaluation section. In addition, we plan to expand our model to consider schedulability also of tertiary processors, e.g., an edge server processor, in our future work.

IV. Demand Based Offloading Policy

Our scheduling methodology is based on an extension of EDF scheduling. A released job has priority assigned to its deadline and earlier deadlines have higher priority. An arriving job of higher priority will either preempt a job of lower priority or be offloaded. For the primary system, we assume a preemptive constrained-deadline EDF scheduling condition based on the work of Baruah et al. [10] to determine if offloading will be necessary:

$$\forall L > 0,\ L \geq DBF(L) = \sum_{i=1}^{n} \left(\left\lfloor \frac{L + T_i - D_{i_P}}{T_i} \right\rfloor \right) C_{i_P}$$

If this condition is satisfied, then we have a trivial case since no offloading is necessary. If this condition is not met, then at some point offloading may need to occur to maintain primary processor schedulability. In this case, we must determine a methodology for choosing what jobs to offload and when to do so. We refer to this as an offloading policy and define it for our system in the following section. It runs each time a job arrives and tests short term primary schedulability, then offloads only those arriving jobs that would cause a deadline to be missed on the primary processor. Note that this is not the only policy that could be defined, and the rest of our analysis hinges upon the behavior of this algorithm with the rest of our model. We plan to explore other policies and their scheduling properties in our future work.

A. The Online Offloading Policy

Our scheduling methodology considers only the primary and secondary processors, and can be treated as a two processor heterogeneous EDF system that first attempts to schedule jobs on the primary processor. If not possible, it schedules them on the secondary processor. This is, to the best of our knowledge, a novel scheduling environment that requires some theoretical investigation. In particular, *can we bound the amount of work sent to the secondary processor as part of offloading?* This

Algorithm 1 Offload(J_a,S,t)

Input J_a: a job just arrived. S: current set of active jobs that have been admitted to primary processor prior to J_a's arrival. t: current time instant.

1: **for** each job J_i in $S \cup J_a$, in earliest deadline order **do**
2: Let $C_{i_P}^{observed}$ be the amount of time J_i has spent executing on the primary processor
3: Let $C_{i_P}^{left} = C_{i_P} - C_{i_P}^{observed}$
4: $t = t + C_{i_P}^{left}$
5: **if** $t >$ absolute deadline of job J_i **then**
6: Offload J_a and terminate.
7: **end if**
8: **end for**
9: Admit J_a to the primary processor schedule

involves a discussion of what a *critical instant* means for this system. For a traditional uniprocessor system this critical instant for a job J_i is when all tasks release a job at the same time[1]. It is the scenario that creates the largest response time for that job. In the uniprocessor case, using the critical instant allows the construction of a worst-case schedule to determine whether a job J_i could potentially miss its deadline. However, the case considered in this paper is substantially different because we consider that the primary processor is overloaded. We then have two questions. *How to know when a sequence of jobs will cause an offload?* In addition, *how to measure the offloaded demand?* Both of these questions are tied to the offloading algorithm that we propose. We have adopted an offloading policy where the system admits jobs until it is overloaded. Once this happens, the offending job, i.e. the one that caused the system to become overloaded, is offloaded. We present our offloading policy in Algorithm 1, which is called at the arrival of each job into the system. It takes three arguments: J_a, the just arrived job with relative deadline D_{a_P} and WCET C_{a_P}; S: the current set of active jobs that have been admitted to the primary processor prior to J_a's arrival; t: the current time instant. If multiple jobs arrive simultaneously, they are processed in an arbitrary order. Lines 1-8 process each job of the existing schedule S plus J_a in order of their deadlines. Line 2 defines $C_{i_P}^{observed}$, the amount of time that a job has spent executing on the primary processor. We assume the system can track this for each job. Line 3 then subtracts $C_{i_P}^{observed}$ from C_{i_P} to get $C_{i_P}^{left}$, the job's worst-case remaining execution time. Line 4 advances the current time instant by $C_{i_P}^{left}$ and Line 5 determines whether a deadline has been missed. If so, then the addition of J_a may cause a deadline miss on the primary processor, thus it is offloaded and Algorithm 1 terminates its execution. Otherwise, if the for loop completes and the algorithm reaches Line 9, no deadline miss is possible. Thus, J_a is admitted to the primary processor.

[1]Unless otherwise specified, the index i for a job does NOT imply that the job was generated by task i.

B. Offline Analysis Overview

The described offloading policy ensures that the primary system is schedulable, since S was schedulable before the arrival of J_a, and that the offloading infrastructure that may be shared among different devices is not unnecessarily congested. However, this is not the only possible offloading policy. A system could offload before its primary processor is overloaded and it could offload more jobs than it needs to. We will consider different policies in our future work. For an interval of length L, the demand that could be offloaded is analyzed. This is done by calculating the G^* function - which is an upper bound on the offloaded demand over an interval (Section V). If the maximum offloaded demand G^* is schedulable on the secondary processor, then there is no deadline miss for this interval. Successively larger intervals are tested until L is equal to the hyperperiod H, i.e., the analysis terminates and determines the taskset to be schedulable, or until G^* is unschedulable on the secondary processor for some interval L, i.e., the taskset is potentially unschedulable. The hyperperiod H is a safe upper bound; it is omitted for space and will appear in the extended version of the paper.

V. Offloaded Demand Analysis

We now begin the construction of a schedulability test by examining the total amount of *offloaded demand*, i.e., the total amount of jobs' primary execution time that may be offloaded to the secondary processor over a given interval.

A. Offloaded Demand Analysis

1) Outline: To determine a sufficient condition for schedulability, we determine a necessary condition for unschedulability. Throughout this section, we use the following conventions: Sets of objects use calligraphy letters such as a set of jobs, $\mathcal{J}$; Relative terms use a capital letter indexed with a subscript such as some job a's WCET C_a; Absolute terms use lowercase letters with a subscript such as a time instant t_n. Due to the complex interactions between tasks and the processors, there are a few of steps we must go through before reaching a schedulability test:

1) In Section V-A2, we define the conditions for the primary processor to offload an arriving job (Claim 1).
2) Using Claim 1 we show that each job offloading is caused by demand intervals and that intervals of consecutive offloaded jobs are contiguous (Lemma 1). Then, based on the findings of Lemma 1, we re-define the conditions for a job to be offloaded using the intervals and total demand (Lemma 2).
3) In Section V-A3 we generalize this condition away from job-specific terms to quantify what primary processor conditions are required for offloading, regardless of the arriving job (Lemma 3). This lets us describe, for a given interval, which of its subintervals must be busy (Lemma 4). We also define two new forms of demand (accepted and induced).
4) In Section V-A4, we find an upper bound on how much accepted demand can be carried between contiguous

intervals (Lemma 5). This lets us prove bounds on the amount of offloaded demand that can be introduced to the secondary processor from the primary processor in a given interval (Theorem 1).

5) Finally, Section V-B proposes a function $G^*(L)$ in Theorem 2, which performs the bounding of offloaded demand in a fast and easily computable way.

2) Job-Specific Offloading Conditions: Our analysis begins by determining the general conditions for a job to be offloaded:

Claim 1. *A job J_a can only be offloaded if it being accepted into the primary processor's schedule would cause J_a or a previously accepted job to miss its deadline.*

Proof Sketch: The for loop in Lines 1-8 of Algorithm 1 are an admission schedulability test for the primary processor. In particular, the offloading action in Line 6 is only reached if a job will miss its deadline, or cause a previously accepted job to miss its deadline, by the statements of Lines 2-5. So a job that misses its deadline must do so after being sent to the secondary processor. □

Since by construction a job admitted to the primary processor will not miss its deadline, our schedulability analysis begins by assuming a job on the secondary processor has missed its deadline, and then "works backwards" to find the necessary sequence of arrivals and offloads from the primary processor that cause this deadline miss. Assume there is a deadline miss on the secondary processor at time t_m. We assume w.l.o.g. that t_m is the first deadline miss, i.e., the interval $[0, t_m)$ contains no deadline miss. Denote J_m as the job that misses its deadline. J_m has arrival time a_m and absolute deadline $d_m = t_m$. Denote the last idle instant (on the secondary processor) before t_m as t_0. We may ignore all jobs on the secondary processor that complete before t_0 without affecting the deadline miss at t_m. There is therefore a set of jobs that are continuously active in the interval $[t_0, t_m)$. From this set of continuously active jobs, we want to consider only the minimum set of jobs which "span" the entire interval of $[t_0, t_m)$ and adopt a concept from [11] called *spanning chain* of jobs. In essence, it describes how, for a busy interval, you can find a subset of jobs such that at any time point at least one, but at most only two of them are active on the processor. For details, we direct the interested reader to the cited paper. We will index these spanning-chain jobs by their absolute deadline on the secondary processor. J_0 has the earliest absolute deadline after t_0, then J_1, and so on, until J_m, which does not complete by its deadline at t_m. For each $J_i : 0 \leq i \leq m$, the term a_i (resp., d_i) represents the absolute arrival time (resp., deadline) of J_i. To satisfy the properties of the spanning chain, we require the following:

1) $\forall 1 < i \leq m : a_{i-1} < a_i \leq d_{i-1}$
2) $\forall 1 < i < m : d_{i-1} < a_{i+1}$

By construction, this spanning chain of jobs $\mathcal{J} = J_0, J_1, J_2, ..., J_m$ entirely covers the secondary processor busy interval over $[t_0, t_m)$.

As the rest of this section involves the contributions of various jobs to various forms of processor demand, we specify three different demand values in terms of primary execution time (i.e., C_{i_P} values) that must be generated for the jobs of $\mathcal{J}$ to be offloaded to the secondary processor:

- $\mathbb{D}_{total}$: The traditional notion of demand, the sum of the demand of all jobs released and having a deadline in some interval;
- $\mathbb{D}_{primary}$: The demand of all jobs that are not offloaded and thus execute on the primary processor over some interval;
- $\mathbb{D}_{offload}$: The demand of all jobs that have been selected to be sent to the secondary processor.

We now consider the demand intervals on the primary processor for each job $J \in \mathcal{J}$ that has been offloaded. We can use this to construct the interval of primary demand that was sufficient to offload a given job in $\mathcal{J}$ i.e., $J_k, 0 \leq k \leq m$. This interval is calculated based on the active set of jobs S during the execution of Algorithm 1 w.r.t. J_k. Let the set of jobs in the system upon each J_k's arrival be $\mathcal{S}_k$. Denote the earliest arrived job in $\mathcal{S}_k$ as J_k^α, and its arrival time as a_k^α. Denote the job that *would* miss its deadline should J_k be accepted as J_k^β, and its deadline as d_k^β. Trivially, $a_k^\alpha \leq a_k$. For determining when a job may obstruct the execution of another, we use the notion of *interference*:

Definition V.1. A job J_x *interferes* with a job J_y iff J_x executes on the same processor as J_y after the release of, and before the completion of J_y.

Since the primary processor is scheduled with EDF, J_k cannot interfere with jobs in $\mathcal{S}_k$ with deadline earlier than d_k. Therefore, $d_k^\beta \geq d_k$. This defines the interval $[a_k^\alpha, d_k^\beta)$, which we refer to as the primary demand interval of J_k. Note that the interval itself may not be entirely busy, as J_k is offloaded. Considering the primary demand intervals of each offloaded job J_k, we now demonstrate the following lemma:

Lemma 1. *The primary demand intervals of offloaded jobs are contiguous: $\forall k < m$, $d_k^\beta \geq a_{k+1}^\alpha \geq a_k^\alpha$*

Proof of Lemma 1. Let the primary demand interval of job J_k be $[a_k^\alpha, d_k^\beta)$. By construction of the spanning-chain for $[t_0, t_m)$, we have that $d_k \geq a_{k+1}$. Therefore, $d_k^\beta \geq d_k \geq a_{k+1} \geq a_{k+1}^\alpha \implies d_k^\beta \geq a_{k+1}^\alpha$. Additionally, by the spanning chain $a_{k+1} \geq a_k$. The earliest job that is active when J_{k+1} arrives cannot have strictly earlier arrival than the earliest job that interferes with J_k; thus, we also have $a_{k+1}^\alpha \geq a_k^\alpha$. This concludes the proof. □

Therefore, the total demand interval that must be investigated for the deadline miss at d_m is the union of all the primary demand intervals of jobs of $\mathcal{J}$, which, by Lemma 1, is a single contiguous interval $[a_0^\alpha, d_m^\beta)$ whose length we denote as $L = d_m^\beta - a_0^\alpha$. Importantly, since t_0 is the last idle instant before t_m, the earliest arrival of any J_k on the secondary processor occurred at t_0. Since $a_k^\alpha \leq a_k$, it must also be true that this $a_k^\alpha \leq t_0$. Similarly, since $d_m^\beta \geq d_m = t_m$, the secondary processor interval $[t_0, t_m)$ is contained entirely within $[a_0^\alpha, d_m^\beta)$. We know that all jobs released by the system

execute either on the primary or secondary processors. That is, over $[a_0^\alpha, d_m^\beta)$, the following must hold: $\mathbb{D}_{total}(L) = \mathbb{D}_{primary}(L) + \mathbb{D}_{offload}(L)$. The total demand follows from the processor demand-bound equation for constrained-deadline sporadic task sets, consisting of all jobs released within an interval from all tasks in the system, multiplied by their execution lengths, i.e., $\mathbb{D}_{total}(L) \leq DBF(L)$.

We may now upper bound the offloaded demand by upper bounding total demand and lower bounding primary demand. First, we determine a safe lower bound on $\mathbb{D}_{primary}$ within an interval of length L. To bound this demand, we consider the condition upon which a job may get offloaded. Since the primary processor is scheduled according to EDF, the necessary condition for a job J_k to be offloaded is that the amount of already accepted demand over the interval $[a_k, d_k)$ must exceed the amount that J_k brings:

Definition V.2. *Accepted Demand* with respect to a time a and interval $[a, b)$ is the potential remaining execution of jobs that arrive at or before a, are accepted on the primary processor by Algorithm 1 and have deadlines at or before b.

Denote by $\widehat{D}_k$ the length of the interval from the arrival of J_k to the deadline that would be missed on the primary processor. That is, $\widehat{D}_k = (d_k^\beta - a_k)$. Remember that d_k^β may itself be d_k, in which case J_k itself could not fit on the primary processor, or it could be a later deadline, in which case a job that had already been accepted to run on the primary processor could miss its deadline. Note that either way, $\widehat{D}_k \geq D_{k_P}$, the primary relative deadline of J_k. As a result:

Lemma 2. *J_k is offloaded if and only if at the arrival of job J_k, the amount of accepted demand over the interval $[a_k, a_k + \widehat{D}_k)$ exceeds $\widehat{D}_k - C_{k_P}$.*

Proof of Lemma 2. First, assume that J_k is offloaded. Then by Claim 1, we have seen that trying to schedule it on the primary processor would cause a deadline miss at $a_k + \widehat{D}_k$. Since we are scheduling with EDF, then necessarily it must have caused the accepted demand within $[a_k, a_k + \widehat{D}_k)$ to exceed $\widehat{D}_k$. Since J_k brings C_{k_P} demand, the amount previously scheduled there must have been at least $\widehat{D}_k - C_{k_P}$. This proves the first direction. Now assume there is less than $\widehat{D}_k - C_{k_P}$ accepted demand over $[a_k, a_k + \widehat{D}_k)$. There is at least C_{k_P} idle time on the primary processor over that interval. J_k can be scheduled in these idle instances without missing a deadline, or causing any already accepted job to miss a deadline. This proves the second direction. □

3) Generalization of Offloading Conditions: In order to remove the dependency of the results of Lemma 2 from specific job characteristics, we define the term *density* by $\delta_k = \frac{C_{k_P}}{D_{k_P}}$. We also define δ_{max}, the maximum density of any task in the system: $\delta_{max} := \{\delta_k | 1 \leq i \leq n, \delta_k \geq \delta_i\}$. We use density to translate Lemma 2 into the following interval-based offloading condition:

Lemma 3. *The accepted demand scheduled over some interval with job J_k being offloaded, $[a_k, a_k + \widehat{D}_k)$, is lower-bounded by $\widehat{D}_k(1 - \delta_{max})$*

Proof of Lemma 3. Let a job J_k arrive at time a_k, and the accepted demand already scheduled over $[a_k, a_k + \widehat{D}_k)$ exceed $\widehat{D}_k - C_{k_P}$ as per Lemma 2. Necessarily, $\widehat{D}_k - C_{k_P} \geq \widehat{D}_k - \frac{\widehat{D}_k}{D_{k_P}} C_{k_P} = \widehat{D}_k(1-\delta_k) \geq \widehat{D}_k(1-\delta_{max})$. Since by assumption the accepted demand over $[a_k, a_k + \widehat{D}_k)$ exceeds $\widehat{D}_k - C_{k_P}$ it also exceeds $\widehat{D}_k(1 - \delta_{max})$. □

The result of Lemma 3 is that, for any interval checked by Algorithm 1, $(1 - \delta_{max})$ is the minimum busy percentage required for an offload. In other words, for every job in $\mathcal{J}$, their intervals are busy for at least $(1 - \delta_{max})$ percentage of time. However, it is not straightforward to simply take $L(1 - \delta_{max})$ as the lower bound on the full $[a_0, d_m)$ interval. This is because Lemma 3 is discussing an interval with exactly one offload. To do this we must make an argument on the minimum busy percentage of related, non-overlapping subintervals that make up this broader interval, which can be unioned together to find the lower bound of $[a_0, d_m)$. We then define a term to discuss the accepted demand that must occur within a smaller interval, but does not contribute to the demand of that smaller interval. Technically, you may accept demand from a job without actually executing all of it, as jobs may under-run their WCETs. However, as we are doing a worst-case analysis, we will assume that all jobs execute for their full WCET runtimes. This leads to the term *Induced Demand* to describe how smaller intervals within larger intervals must necessarily have some execution in them that does not correspond directly to traditional ideas of demand:

Definition V.3. *Induced Demand* for an interval $[a, b) \subset [a, c)$ is the amount of the accepted demand at time a of $[a, c)$ that is executed within $[a, b)$ due to the work conserving scheduler.

Consider the interval of interest for a job that is offloaded, J_k. Such a job arrives at a_k, and at the time of its arrival, we can find $\widehat{D}_k$. Lemma 3 tells us that over $[a_k, a_k + \widehat{D}_k)$ the interval has at least $(1 - \delta_{max})$ of accepted demand. Since we know the arrival of the next offloaded job a_{k+1} occurs within this interval, let us divide it into two portions: $[a_k, a_{k+1})$ and $[a_{k+1}, a_k + \widehat{D}_k)$. Lemma 4 lets us determine how potentially busy the subinterval $[a_k, a_{k+1})$ is, based on our knowledge about $[a_k, a_k + \widehat{D}_k)$ from Lemma 3:

Lemma 4. *Let w be the accepted demand at a_k for $[a_k, a_k + \widehat{D}_k)$. Then, the induced demand for interval $[a_k, a_{k+1})$ where $a_k \leq a_{k+1} \leq a_k + \widehat{D}_k$ is $\min(w, a_{k+1} - a_k)$*

Proof of Lemma 4. Assume $w \leq (a_{k+1} - a_k)$. Thus, the accepted demand tha may be scheduled over $[a_k, a_k + \widehat{D}_k)$ will have to be scheduled entirely over $[a_k, a_{k+1})$, which is verified because of the work-conserving scheduler assumption. As a result, the amount of induced demand in $[a_k, a_{k+1})$ is w as well. If $w > (a_{k+1} - a_k)$, then there is more accepted demand over $[a_k, a_k + \widehat{D}_k)$ than can fit in $[a_k, a_{k+1})$. Again since we

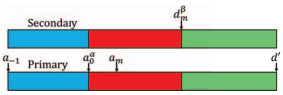

Fig. 1. The three regions defined by $a_{-1}, a_0^\alpha, d_m^\beta$ and d' on the Primary and Secondary Processors

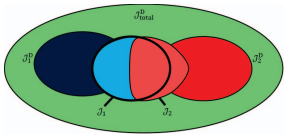

Fig. 2. Relations between sets of jobs. Note that $\mathcal{J}_1$ and $\mathcal{J}_2$ overlap, but neither is a subset of the other, and portions of each exist outside of both $\mathcal{J}_2^D$ and $\mathcal{J}_1^D$

have a work conserving scheduler and all jobs included in the accepted demand have already arrived by a_k, the interval $[a_k, a_{k+1})$ may be continuously busy (i.e., no idle instants before completion of the execution of the accepted demand), and its induced demand is $(a_{k+1} - a_k)$. □

4) DBF-Based Offloaded Demand: The result of Lemma 4 is that $[a_k, a_{k+1})$ must be busy for at least $(1 - \delta_{max})$ of its length. Since this is true for any k, this allows us to claim that $[a_0, a_1)$ is $(1 - \delta_{max})$ busy, as is $[a_1, a_2)$, up to $[a_{m-1}, a_m)$. Thus, the interval $[a_m, a_m + \widehat{D}_m)$ is itself $(1 - \delta_{max})$ busy. Since these intervals are constructed to be non-overlapping, the entire interval from $[a_0, d_m^\beta)$ is $(1-\delta_{max})$ busy. Unfortunately, calculating work offline is a difficult proposition. This is because arrival patterns of sporadic tasks are unpredictable and jobs may run under their WCETs. So we finally seek to transform this condition into one based on demand bound function calculations. To do so, we must consider the primary demand intervals that can contribute work to the $[a_0^\alpha, d_m^\beta)$ interval. For this we must consider all the jobs generated over a broader interval $[a_{-1}, d')$. a_{-1} is the last idle instant on the primary processor before a_0^α, and d' is the latest possible absolute deadline of any job executing in the system over $[a_0^\alpha, d_m^\beta)$. d' is defined by $d' = d_m^\beta + D_{max}$, with D_{max} being the largest relative deadline of any task in the system. Since all jobs that execute within $[a_0^\alpha, d_m^\beta)$ must be released at or after a_{-1}, and have a deadline before d', $DBF(d' - a_{-1})$ is a safe upper bound on the accepted demand within $[a_0^\alpha, d_m^\beta)$. On the other hand, this upper bound is highly pessimistic. To reduce the level of pessimism, in the rest of the sections we analyze several subintervals of $[a_{-1}, d')$ that we can more tightly bound.

Figure 1 details the important time points within this analysis as well as separating the three important demand regions we will analyze across both processors. Our goal is the tightest possible upper bound on the demand of the middle region, $[a_0^\alpha, d_m^\beta)$ on the secondary processor, through analysis on the demand of all three regions on the primary processor. The left region begins from a full idle point for the system, a_{-1}, where a job arrives on the primary processor. It ends with the arrival of the first job to be offloaded since a_{-1} at a_0^α, which marks the beginning of the second region, as well as a busy period on the secondary processor. This second region contains the arrival a_m of the job that will eventually miss its deadline and ends at its deadline d_m^β. The third region extends for an additional D_{max} time units and contains primary processor work that may have begun in the prior two regions. We start our analysis with the left region $[a_{-1}, a_0^\alpha)$.

Jobs that arrive and have deadlines within $[a_{-1}, a_0^\alpha)$ cannot execute during $[a_0^\alpha, d_m^\beta)$ or $[d_m^\beta, d')$, and are not offloaded. This quantity can be bound by $DBF(a_0^\alpha - a_{-1})$. Other jobs that arrive in $[a_{-1}, a_0^\alpha)$ may execute partially in $[a_{-1}, a_0^\alpha)$ and $[a_0^\alpha, d_m^\beta)$, entirely in $[a_0^\alpha, d_m^\beta)$, partially in $[a_0^\alpha, d_m^\beta)$ and $[d_m^\beta, d')$, entirely in $[d_m^\beta, d')$ or over all three regions. In order to characterize the effect on $[a_0^\alpha, d_m^\beta)$ these jobs may have, we use Lemma 5 to quantify the amount of accepted demand that executes in $[a_0^\alpha, d_m^\beta)$, but that comes from jobs that do not contribute to the total demand of $[a_0^\alpha, d_m^\beta)$:

Lemma 5. *An upper bound on the accepted demand that executes in $[a_0^\alpha, d_m^\beta)$ but does not contribute to the demand of $[a_0^\alpha, d_m^\beta)$ is given by*[2]*:*

$$\left[\sum_{i=1}^{n} C_{i_P} - [(a_0^\alpha - a_{-1}) - DBF(a_0^\alpha - a_{-1})]_0\right]_0$$

Proof of Lemma 5. To begin, we can take all the execution within $[a_0^\alpha, d_m^\beta)$ as a trivial upper bound. Then for each task i in the system, we can subtract a lower bound on the accepted demand executed over $[a_{-1}, a_0^\alpha - T_i)$ from the $DBF(d' - a_{-1})$. This is effectively a lower bound on the work from jobs that do not contribute to the demand of $[a_0^\alpha, d_m^\beta)$ and do not execute in $[a_0^\alpha, d_m^\beta)$. Only the last job released of each task in $[a_{-1}, a_0^\alpha)$ can actually carry execution into $[a_0^\alpha, d_m^\beta)$. If this last job was released before $a_0^\alpha - T_i$ for each task i, than any execution after a_0^α would imply a deadline miss. The interval is busy, so there must be at least $a_0^\alpha - a_{-1}$ execution present, which is a lower bound. All tasks could contribute at most $DBF(a_0^\alpha - a_{-1})$ to that interval exclusively. $DBF(a_0^\alpha - a_{-1})$ is an upper bound. The remaining work consists of jobs that may carry execution into $[a_0^\alpha, d_m^\beta)$. Thus, $[(a_0^\alpha - a_{-1}) - DBF(a_0^\alpha - a_{-1})]_0$ is the amount of existing execution and consists of jobs that may execute into $[a_0^\alpha, d_m^\beta)$. Thus, $[(a_0^\alpha - a_{-1}) - DBF(a_0^\alpha - a_{-1})]_0$ is a lower bound on the amount of those jobs that have arrival in $[a_{-1}, a_0^\alpha)$ but deadlines after a_0^α and must execute in $[a_{-1}, a_0^\alpha)$. To get an upper bound on the execution of these jobs, we note that the total amount of execution of tasks that carry-in work into $[a_0^\alpha, d_m^\beta)$ is at most one job of each task, which is upper bounded by: $\sum_{i=1}^{n} C_{i_P}$. Thus, $[\sum_{i=1}^{n} C_{i_P} - [(a_0^\alpha - a_{-1}) - DBF(a_0^\alpha - a_{-1})]_0]_0$ gives an upper bound on the amount of execution that could be carried into $[a_0^\alpha, d_m^\beta)$ without contributing to its demand. □

The results of Lemmas 3 and 5 allow us to lower bound the amount of primary execution during $[a_0^\alpha, d_m^\beta)$, which consists of demand from $[a_0^\alpha, d_m^\beta)$ by considering the lower bound on total execution within $[a_0^\alpha, d_m^\beta)$ (Lemma 3), and subtracting the upper bound of demand that may execute in $[a_0^\alpha, d_m^\beta)$, but does not contribute to its demand (Lemma 5). Then, since the total demand across both processors is upper bounded by $DBF(d_m^\beta - a_0^\alpha)$, Theorem 1 subtracts that lower bound to get the overall upper bound on offloaded demand in $[a_0^\alpha, d_m^\beta)$:

[2]We use $[x]_0$ to denote the value taken is the larger of x and 0

Theorem 1. *The offloaded demand over the interval $[a_0^\alpha, d_m^\beta)$ is upper bounded by*

$$G(a_{-1}, a_0^\alpha, d_m^\beta) = \Bigg\lceil DBF(d_m^\beta - a_0^\alpha) - \Bigg\lceil (1-\delta_{max})(d_m^\beta - a_0^\alpha) - \Bigg\lceil \sum_{i=1}^{n} C_{i_P} - [a_0^\alpha - a_{-1} - DBF(a_0^\alpha - a_{-1})]_0 \Bigg\rceil_0 \Bigg\rceil_0 \Bigg\rceil_0 \quad (1)$$

Proof of Theorem 1. We use job sets to prove Theorem 1. In particular, we indicate with $|\mathcal{J}|$ the cardinality of the number of jobs in a set, $\|\mathcal{J}\|$ the total execution time of a set of jobs $\mathcal{J}$ on the primary processor (also called set norm), and $\bar{\mathcal{J}}$ the complement of a set $\mathcal{J}$, which contains all jobs not in the set. From Figure 2's notation, we define the following five sets of jobs:

- $\mathcal{J}^D_{total}$, jobs that contribute to the primary demand of $[a_{-1}, d')$;
- $\mathcal{J}^D_1$, jobs that contribute to the total demand of $[a_{-1}, a_0^\alpha)$;
- $\mathcal{J}^D_2$, jobs that contribute to the total demand of $[a_0^\alpha, d_m^\beta)$;
- $\mathcal{J}_1$, jobs that have some primary execution in $[a_{-1}, a_0^\alpha)$;
- $\mathcal{J}_2$, jobs that have some primary execution in $[a_0^\alpha, d_m^\beta)$.

We can see that $\mathcal{J}^D_2 \setminus \mathcal{J}_2$ is the set of jobs that contribute to the demand of $[a_0^\alpha, d_m^\beta)$, but do not execute on the primary processor in $[a_0^\alpha, d_m^\beta)$. These jobs are the *offloaded* demand of $[a_0^\alpha, d_m^\beta)$. Our goal then, is to upper bound $\left\|\mathcal{J}^D_2 \setminus \mathcal{J}_2\right\|$. Following the definition of set operations, we can define this upper bound as follows: $\left\|\mathcal{J}^D_2\right\| - \left\|\mathcal{J}^D_2 \cap \mathcal{J}_2\right\| \geq \left\|\mathcal{J}^D_2 \setminus \mathcal{J}_2\right\|$. We cannot take $\left\|\mathcal{J}^D_2\right\| - \|\mathcal{J}_2\|$ directly, as $\mathcal{J}_2$ may contain jobs that are not in $\mathcal{J}^D_2$. An upper bound on $\left\|\mathcal{J}^D_2\right\|$ and a lower bound on $\left\|\mathcal{J}^D_2 \cap \mathcal{J}_2\right\|$ will provide the safe upper bound we need. Upper bounding $\|\mathcal{J}_2\|$ is simple, by using the demand bound function: $DBF(d_m^\beta - a_0^\alpha) \geq \left\|\mathcal{J}^D_2\right\|$. We know by some basic properties of set operations that $\mathcal{J}^D_2 \cap \mathcal{J}_2 = \mathcal{J}_2 \setminus (\mathcal{J}_2 \cap \bar{\mathcal{J}}^D_2)$ and its norm: $\left\|\mathcal{J}_2 \setminus \mathcal{J}_2 \cap \bar{\mathcal{J}}^D_2\right\| \geq \|\mathcal{J}_2\| - \left\|\mathcal{J}_2 \cap \bar{\mathcal{J}}^D_2\right\|$. Therefore, if we lower bound $\|\mathcal{J}_2\|$ and upper bound $\left\|\mathcal{J}_2 \cap \bar{\mathcal{J}}^D_2\right\|$, we can lower bound $\left\|\mathcal{J}_2 \setminus \mathcal{J}_2 \cap \bar{\mathcal{J}}^D_2\right\|$. Our lower bound of primary demand to have an offload for $\|\mathcal{J}_2\|$ is given by Lemma 3: $(1-\delta_{max})(d_m^\beta - a_0^\alpha) \leq \|\mathcal{J}_2\|$. Our upper bound for $\left\|\mathcal{J}_2 \cap \bar{\mathcal{J}}^D_2\right\|$ is given by Lemma 5: $[\sum_{i=1}^{n} C_{i_P} - (a_0^\alpha - a_{-1} - DBF(a_0^\alpha - a_{-1}))]_0$.

Since the relation between the Lemma 3 and Lemma 5 can be difficult to reason about, we take the maximum of their difference and zero to avoid situations of adding demand incorrectly. Additionally, since offloaded demand can never be negative, we apply this maximum again on the entire expression. We recap by showing the whole series of steps, from basic upper bound, to final equation:

$$\begin{aligned}
&\left\|\mathcal{J}^D_2 \setminus \mathcal{J}_2\right\| \leq \left\|\mathcal{J}^D_2\right\| - \left\|\mathcal{J}^D_2 \cap \mathcal{J}_2\right\| = \\
&= \left\|\mathcal{J}^D_2\right\| - \left\|\mathcal{J}_2 \setminus (\mathcal{J}_2 \cap \bar{\mathcal{J}}^D_2)\right\| \\
\leq\ &\left\|\mathcal{J}^D_2\right\| - \left(\|\mathcal{J}_2\| - \left\|\mathcal{J}_2 \cap \bar{\mathcal{J}}^D_2\right\|\right) \leq DBF(d_m^\beta - a_0^\alpha) - \\
&- \left(\|\mathcal{J}_2\| - \left\|\mathcal{J}_2 \cap \bar{\mathcal{J}}^D_2\right\|\right) \\
\leq\ &\Bigg\lceil DBF(d_m^\beta - a_0^\alpha) - \Bigg\lceil (1-\delta_{max})(d_m^\beta - a_0^\alpha) - \\
&- \Bigg\lceil \sum_{i=1}^{n} C_{i_P} - [a_0^\alpha - a_{-1} - DBF(a_0^\alpha - a_{-1})]_0 \Bigg\rceil_0 \Bigg\rceil_0 \Bigg\rceil_0
\end{aligned}$$

Thus, Equation 1 successfully upper bounds the offloaded demand in $[a_0^\alpha, d_m^\beta)$ □

B. Simplification of the Offloaded Demand Bound

Theorem 1 gives a good analytical understanding of what is the upper bound of offloaded demand for a generic taskset to have a deadline miss, but it suffers from practical drawbacks. In particular, determining the actual value of a_{-1} is only possible through a full schedule simulation, which could be impractical to compute. We therefore propose the $G^*(L)$ function, which gives a safe upper bound on Theorem 1 that can be easily computed offline. Theorem 1 starts from simple upper bound of $DBF(d_m^\beta - a_0^\alpha)$ and performs reductions on this to get a less pessimistic, but still valid, upper bound. G^* relaxes some of these reductions to get a bound that is not as tight as Theorem 1, but is fast to compute:

Theorem 2. *Given any interval of length L, $G^*(L)$ upper bounds the $G(a_{-1}, a_0^\alpha, d_m^\beta)$ function, $\forall a_{-1},\ a_0^\alpha, d_m^\beta \in \mathbb{R}_{\geq 0}$ such that $a_{-1} \leq a_0^\alpha$ and $d_m^\beta = a_0^\alpha + L$:*

$$G^*(L) = \Bigg\lceil \sum_{\tau_i \in \tau : T_i \leq L} C_{i_P} \left\lfloor \frac{L + T_i - D_{i_P}}{T_i} \right\rfloor - \Bigg\lceil (1-\delta_{max})L - \sum_{\tau_i \in \tau} C_{i_P} \Bigg\rceil_0 \Bigg\rceil_0$$

Proof of Theorem 2. We seek to show that $G^*(L)$ upper bounds the $G(a_{-1}, a_0^\alpha, d_m^\beta)$ function from Theorem 1. We assert $L = d_m^\beta - a_0^\alpha$, and show that the bound holds regardless of choice of a_{-1}. The bound is proven in parts, noting that the $G(a_{-1}, a_0^\alpha, d_m^\beta)$ function can be broken into three major terms. The first two terms are the same in G and G^*. In fact, for the first term we have that: $DBF(d_m^\beta - a_0^\alpha) = DBF(L) = \sum_{\tau_i \in \tau : T_i \leq L} C_{i_P} \left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor$. Note that using the reduced set of tasks $\tau_i \in \tau : T_i \leq L$ for the sum rather than the full taskset does not reduce the DBF value. This is because tasks with $T_i > L$ cannot have a job be released and have a deadline in the interval, and thus contribute no demand. For the second term, adopted from Lemma 3, we have that: $(1-\delta_{max})(d_m^\beta - a_0^\alpha) = (1-\delta_{max})L$. The last term adopted from Lemma 5, i.e., $\left\lceil \sum_{i=1}^{n} C_{i_P} - [a_0^\alpha - a_{-1} - DBF(a_0^\alpha - a_{-1})]_0 \right\rceil_0$, is clearly smaller than $\sum_{i=1}^{n} C_{i_P}$. As the second terms are equal, and the third term in $G^*(L)$ is equal to or larger than the one in G, their difference is either equal or smaller. Therefore the amount subtracted from the DBF term in $G^*(L)$ is no larger than the amount subtracted in G. Thus, $G^*(L) \geq G(a_{-1}, a_0^\alpha, d_m^\beta)$. □

The $G^*(L)$ function is non-negative, is defined for all non-negative integer values of L and provides an upper bound on the amount of demand that can be offloaded over any interval of length L. When $\gamma = 1$, $C_{i_P} = C_{i_S}$ for $\tau_i \in \tau$, then the amount of offloaded demand from the primary processor exactly equals the demand upon the secondary processor. Therefore, we can use G^* as an upper bound of demand in the standard EDF uniprocessor schedulability condition (e.g., if the secondary processor was fully preemptive, a sufficient condition for secondary processor schedulability would be $G^*(L) \leq L,\ \forall L > 0$).

We are now ready to move from our analysis of offloaded demand to deriving a general offline schedulability test for the system ($\forall\ \gamma \in (0, 1]$). We begin first with some definitions.

Definition V.4. Consider any L-length busy interval $[t, t+L)$ on the non-preemptive secondary processor that satisfies the following properties:

1) a job with secondary relative deadline $D_{i_S} \leq L$ arrives at time t;
2) the secondary processor is continuously backlogged over $[t, t + L)$ with offloaded jobs that arrive and have deadline in $[t, t + L)$;
3) Under non-preemptive scheduling, there is at most one lower priority job with an absolute deadline after $t + L$ that begins execution prior to t and continues its execution within the interval $[t, t + L)$. This job necessarily must have $D_{i_S} > L$; and
4) No job misses a deadline in $[t, t + L)$ (note a deadline miss is permitted at $t + L$).

The *critical offloaded demand* $Q(L)$ for any such L-length busy interval is the minimum offloaded demand from the primary processor that could generate such a busy interval.

Definition V.5. When jobs are executed on the secondary processor non-preemptively, some low priority jobs may cause an additional blocking time, $B(L)$, that can be bounded by the following function from George et al. [12]:

$$B(L) = \max_{\tau_j \in \tau} \{C_{j_S} | D_{j_S} > L\} \tag{2}$$

We use the convention that $B(L)$ equals 0 when L exceeds or equals $\max_{\tau_j \in \tau}\{D_j\}$.

For uniform secondary processor times, the critical offloaded demand over any L-length interval can be obtained by: $Q(L) > \frac{L - B(L)}{\gamma}$. From the above definitions, we can easily state a generic schedulability test for our system:

Theorem 3. *The both primary-executed and offloaded jobs of task system τ will meet all their deadlines if: $G^*(L) < Q(L),\ \forall L > 0$.*

Proof of Theorem 3. First, observe that Algorithm 1 offloads any job whose induced demand over some interval will cause the demand to exceed the available processing time on the primary processor. To show that the offloaded jobs will meet their deadline upon the secondary processor, we prove this statement via the contrapositive; that is, if τ misses a deadline on the secondary processor, we will show that $G^*(L') \geq Q(L')$ for some $L' > 0$. Assume that the secondary processor misses a deadline at time t_f. According to the usual demand-based analysis for non-preemptive EDF on a processor [12], there exists a time $t_{-1} (< t_f)$ that is the last idle instant with respect to jobs with deadline at or before time t_f; in other words, the processor is either idle right before t_{-1} or executing a job with deadline strictly later than t_f (in which case it can non-premptively block for at most $B(t_f - t_{-1})$). Therefore, by construction of this last idle instant, the secondary processor is continuously backlogged in the interval $[t_{-1}, t_f)$ with jobs that arrive at or after t_{-1} but have deadline at or before t_f. Therefore, the total secondary processor demand of the offloaded jobs in the $[t_{-1}, t_f)$ strictly exceeds $t_f - t_{-1} - B(t_f - t_{-1})$. Let $L' = t_f - t_{-1}$; by definition of the Q function, we have that the offloaded demand from the primary processor exceeds or equals $Q(L')$.

Note that the interval $[t_{-1}, t_f)$ exactly corresponds to interval $[a_0^\alpha, d_m^\beta)$ in the proof of Theorem 1. Thus, there must also exist an $a_{-1} (\leq a_0^\alpha)$, which is the last idle time on the primary processor with respect to interval $[a_0^\alpha, d_m^\beta)$. Thus, since the primary processor demand offloaded from this interval exceeds or equals $Q(L')$, we must have $G(a_{-1}, a_0^\alpha, d_m^\beta) \geq Q(L')$. By Theorem 2, $G^*(L') \geq Q(L')$, completing the theorem. □

VI. Evaluation

We now analyze the performance of our offline method by using both synthetically-generated taskset data and a realistic practical taskset collected from our prototype autonomous mobile robot executing computer vision tasks, which is the primary processor. The local network processor is the secondary processor, which transmits the job to a powerful edge server. The deadlines are adjusted by subtracting the worst-case response time of an offloaded job on the edge server.

A. Experimental Setup

Our evaluation was done through a python simulator A modified EDF scheduler was made to run jobs from 500 tasksets of different primary utilization using Algorithm 1 to select jobs for offloading. We run these tests on a desktop with an Intel core-i9 3.7GHz processor and 128GB DRAM. To generate the tasksets, we have adopted the UUniFast algorithm [13]. For the primary processor the utilization of a task is defined as $U_i = \frac{C_{i_P}}{T_i}$. T_i values are randomly generated from the interval [10ms, 70ms]. We also define a common scaling factor γ and then use it to calculate the secondary processor execution time for each task. To maintain integer task parameters, we then take floor values of primary and secondary processor execution times. Due to this rounding, the overall utilization of any generated task system may differ slightly from the target UUniFast utilization. Due to the exponential-time complexity of the simulator (proportional to the value of the hyperperiod), we chose to limit generated taskset hyperperiod sizes to keep experiment runtimes manageable. For our synthetic tasksets we restricted them to 50,000, and permitted hyperperiods up to 100,000 for our practical examples.

We consider two performance metrics to analyze the theoretical bounds provided in this paper: *offloaded demand ratio* (ODR) and *schedulability ratio* (SR). We calculate these for primary utilizations ranging from 110% to 200% for ODR and 110% to 290% for SR, using γ values from 0.2 to 0.6. ODR is calculated as the highest observed ratio between the value of the G^* function (Theorem 2) and the actual offloaded demand given by our simulator for all interval lengths $1 \leq L \leq H$. If the ODR is 1, G^* perfectly estimates offloaded demand. Values larger than 1 imply an over-estimation by G^*. SR is

defined as the percentage of generated tasksets that are found schedulable.

It is important to note that the results from the simulator may be *optimistic* since it is looking at the offloaded demand of one arrival sequence, which may not be the worst-case; i.e., the simulation may classify a task system as schedulable, even though a different arrival sequence may lead to a deadline miss. Nonetheless, it is useful for demonstrating the usefulness of the G^* function when compared to the optimistic results reported by the simulator. Furthermore, while both the simulation and G^* evaluations can take exponential time, in practice the G^* is significantly faster, provides an answer on whether the system is schedulable (over all possible legal job arrivals), and is not significantly more conservative than the optimistic simulation results (as we will see in the next subsections).

B. Results with Synthetic Tasksets

Figure 3a shows the cumulative distribution function (CDF) of the ODR for primary utilization varying from 110% to 200% over 500 tasksets at each utilization. For example, at utilization 120%, around 60% of the tasksets have an ODR of 2.6 or less. We observe that when primary utilization is 110%, the offloaded demand estimation given by G^* function never exceeds approximately 8.7 times the offloaded demand found from the simulation, and is never less than 1.5. As the primary utilization increases, the observed ODR values tend to decrease and approach 1. For a primary utilization of 200%, the smallest observed ODR is around 1.4. As primary utilization increases, the G^* function becomes more accurate, and ODR decreases towards 1. G^* improves at high utilizations as the primary processor tends to be more busy and offloads more jobs. As G^* is an upper-bound, the closer the system performs to the worst-case of offloading all jobs, the more accurate G^*'s estimations are. Lower primary utilizations are less likely to encounter a worst-case and G^*'s estimation will be more conservative.

Figure 3b shows the difference in SR between the simulator and the offline analysis. We tested primary utilizations from 110% to 290% and for each utilization considered secondary processor utilization based on three values of γ, 0.3, 0.4, and 0.5. Note that we did not include results for cases where primary utilization is lower than 100%, as such cases are trivially schedulable by preemptive EDF. For example, at γ=0.3 the simulator schedulability ratio ranges from 100% schedulable at 110% primary utilization down to 48% schedulable for 290% primary utilization. Correspondingly, our offline schedulability analysis based on uniform secondary processors (o labels in figure) provides an SR ranging from 89% to 17% for similar tasksets. Overall, the offline analysis had SR which perfectly matched the simulator results, or found about half as many tasksets schedulable as the simulator until either $\gamma = 0.5$, or utilization is well past 2. Understandably, the SR of our analysis drops dramatically for γ values above 0.5, and we omit them here for brevity. Nonetheless, these results show that our offline analysis is able to provide some schedulability guarantees even at very high primary utilization. In particular, with $\gamma = 0.3$, many tasksets of 290% utilization are schedulable, as a fully utilized primary processor may offload at most 190% utilization to the secondary processor, which is then reduced by the γ factor to effectively 63.3% utilization, often schedulable, even by non-preemptive EDF.

C. Results from Realistic Practical Tasksets

Here we test the proposed offline analysis based on a real-world applications. We use our prototype HydraOne [14], which is an autonomous mobile robot that can execute navigation and objective tasks. The navigation task is a Machine Learning (ML) model trained to periodically update the robot speed and direction based on camera frames. The objective tasks are other ML models that do not directly affect the navigation decisions but give the robot useful purpose, such as recognizing faces, detecting holes in a fence, recognizing objects etc. Unfortunately, our real prototype does not implement a real time operating system, so we could not run tests directly on it. Thus, we use the simulator described in Section VI-A to schedule various tasksets generated using data profiled from our real robot. Specifically, each navigation task executes approximately 0.1M instructions, while objective tasks, depending on the model, execute between 0.5M and 1.5M instructions. The CPU speed is profiled to execute on average 50 Million Instructions per Second (MIPS) for these tasks. For network operations, we profile the size of a typical camera frame to be in the interval $4\text{Mb} - 32\text{Mb}$ and assume an average 5G networking speed of 1,000Mbps [15].

Using these profiled data, we use an example scenario of four objective tasks running concurrently to a navigation task. In order to generate different tasksets, we use the following procedure: 1) randomly choose the number of instructions for each of the 4 objective tasks in the range $0.5\text{M} - 1.5\text{M}$, 2) calculate the primary processor execution time of each task based on the profiled CPU speed, 3) choose the target utilization and calculate the periods of each objective and navigation tasks. Similar steps are used to generate the secondary processor related data of the taskset. We generate 25 different tasksets and average the ODR and SR for each target utilization. Note, due to long running times we could not use 500 tasksets as in previous section. In addition, the large period range of these tasksets leads to large hyperperiods, thus here we show results for only $\gamma = 0.5$. We plan to design a more efficient method for interval selection in our future work.

Figure 3c shows how the ODR changes for different primary utilization considering both synthetic and practical tasksets. We observe that the offloading behavior and G^* function behaves similarly whether testing synthetic or practical tasksets. In both cases G^* becomes more accurate for primary utilization increases with the ODR approaching 1 (red horizontal line). For synthetic tasksets the worst-case observed ODR is 3.40 while the realistic tasksets have a significantly lower value of 2.36. As primary utilization increases, both types of tasksets follow a similar trend, suggesting applicability of the synthetic results to real systems. Figure 3d shows the comparison of SR between the simulator and the offline analysis tested with

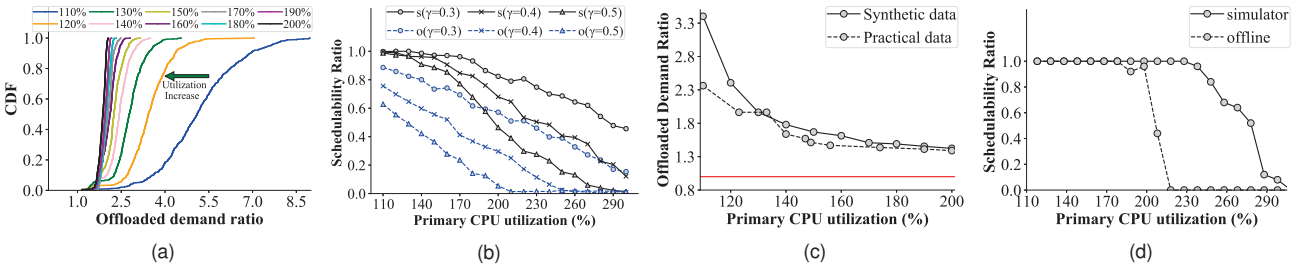

Fig. 3. (a) CDF of ODRs for varied primary utilizations. (b) SR for different primary utilizations and γ values for simulated system (s) and offline analysis (o) with uniform secondary processor execution times. (c) ODR with synthetic and practical tasksets and (d) SR for different primary utilization of the practical taskset with uniform secondary processor execution times.

the practical tasksets. We can see that when the primary utilization is lower than 200%, all of the tasksets are found schedulable by the simulator, and nearly all by the offline analysis as well. However, when utilization increases, the ratio tends to decrease for the same reason described earlier. While the simulator finds surprising schedulability even up to 270% utilization, the offline analysis drops sharply above 200%. These experiments suggest that the results for the synthetic tasksets are comparable to those of realistic cases. In addition, our offline analysis provides reasonably good guarantees of schedulability for realistic tasksets.

VII. Conclusions

While both offloading and real-time systems have been well studied in the literature, works using them together are less common. In this paper we have laid the groundwork for a scheduling framework that utilizes heterogeneous processors to assist with hard real-time systems when the primary processor is overloaded, in which case jobs can be offloaded to meet deadlines. While we take a number of simplifying assumptions, we believe that this is an important step in utilizing the high degree of connectivity in modern computing in a novel and powerful way. Our experimental results show that even with our conservative assumptions, task sets above even 200% primary utilization become schedulable, potentially lowering expensive over-provisioning in such systems. Future work will involve tightening of the offline analysis, consideration of multiple offloading units, other scheduling algorithms, network losses, and further generalizations.

Acknowledgment

This research was supported in part by the US National Science Foundation under grant nos. CCF-2118202, CNS-2038609, IIS-1724227, and CNS-1948365.

References

[1] O. Goldstein, "Real-time cost-aware machine learning at the edge," Ph.D. dissertation, University of California, Los Angeles, 2021.

[2] B. Lv, C. Yang, X. Chen, Z. Yao, and J. Yang, "Task offloading and serving handover of vehicular edge computing networks based on trajectory prediction," *IEEE Access*, 2021.

[3] L. Schönberger, G. von der Brüggen, K.-H. Chen, B. Sliwa, H. Youssef, A. K. R. Venkatapathy, C. Wietfeld, M. ten Hompel, and J.-J. Chen, "Offloading Safety- and Mission-Critical Tasks via Unreliable Connections," in *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, Dagstuhl, Germany, 2020.

[4] A. Toma, J.-J. Chen, and W. Liu, "Computation offloading for sporadic real-time tasks," in *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, 2014.

[5] W. Liu, J.-J. Chen, A. Toma, T.-W. Kuo, and Q. Deng, "Computation offloading by using timing unreliable components in real-time systems," in *Proceedings of the 51st Annual Design Automation Conference*, ser. DAC '14. New York, NY, USA: Association for Computing Machinery, 2014.

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

[7] Y. Nimmagadda, K. Kumar, Y.-H. Lu, and C. S. G. Lee, "Real-time moving object recognition and tracking using computation offloading," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[8] G. Raravi, B. Andersson, V. Nélis, and K. Bletsas, "Task assignment algorithms for two-type heterogeneous multiprocessors," *Real-Time Systems*, vol. 50, no. 1, 2014.

[9] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, may 2008.

[10] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-time systems*, vol. 2, no. 4, 1990.

[11] N. Fisher and S. Baruah, "The global feasibility and schedulability of general task models on multiprocessor platforms," in *19th Euromicro Conference on Real-Time Systems (ECRTS'07)*, 2007.

[12] L. George, N. Rivierre, and M. Spuri, "Preemptive and non-preemptive real-time uniprocessor scheduling," in *INRIA, INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE*, 2006.

[13] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, 2005.

[14] L. Liu, J. Chen, M. Brocanelli, and W. Shi, "E2m: an energy-efficient middleware for computer vision applications on autonomous mobile robots," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019.

[15] D. Xu, A. Zhou, X. Zhang, G. Wang, X. Liu, C. An, Y. Shi, L. Liu, and H. Ma, "Understanding operational 5g: A first measurement study on its coverage, performance and energy consumption," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020.