Task Offloading with Uncertain Processing Cycles

Shaoran Li[†] Chengzhang Li[†] Yan Huang[†] Brian A. Jalaian[‡] Y. Thomas Hou[†] Wenjing Lou[†]
[†] Virginia Tech, Blacksburg, VA
[‡] U.S. Army Research Laboratory, Adelphi, MD

ABSTRACT

Mobile Edge Computing (MEC) has emerged to be an integral component of 5G infrastructure due to its potential to speed up task processing and reduce energy consumption for mobile devices. However, a major technical challenge in making offloading decisions is that the number of required processing cycles of a task is usually unknown in advance. Due to this processing uncertainty, it is difficult to make offloading decisions while providing any guarantee on task deadlines. To address this challenge, we propose EPD—Energy-minimized solution with Probabilistic Deadline guarantee to task offloading problem. The mathematical foundation of EPD is *Exact Conic Reformulation* (ECR), which is a powerful tool that reformulates a probabilistic constraint for task deadline into a deterministic one. In the absence of distribution knowledge of processing cycles, we use the estimated mean and variance of processing cycles and exploit ECR to the fullest extent in the design of EPD. Simulation results show that EPD successfully guarantees the probabilistic deadlines while minimizing the energy consumption of mobile users, and can achieve significant improvement in energy saving when compared to a state-of-the-art approach.

CCS CONCEPTS

• **Networks → Network resources allocation**; **Mobile networks**; *Network performance analysis.*

KEYWORDS

Mobile Edge Computing, Processing Uncertainty, Offloading, Scheduling

1 INTRODUCTION

With the proliferation of energy-constrained user equipments (UEs) and computationally intensive applications, Mobile Edge Computing (MEC) has been proposed in recent years [3, 14]. Under MEC, computation-intensive tasks generated by UEs may be offloaded to the edge Base Station (BS) for processing in addition to local processing at the UEs. There are two benefits in offloading: (i) the UEs may conserve precious energy; (ii) the processors at the BS are expected to be much more powerful than that at the UE and thus may complete the task faster. Motivated by these benefits, infrastructure standard bodies such as 3GPP have included MEC in their 5G network standardization [2].

An important objective in offloading tasks from the UEs to the BS is to guarantee their deadlines. In other words, the outcome of these tasks must meet certain deadlines so that they can be used in time by the underlying applications. Most of the existing works that addressed task deadlines assume the required number of processing cycles of a task is known *a priori* [8, 22–24]. However, such an assumption is overly idealistic and does not reflect what happens with many real-world applications, where the required number of processing cycles of a task is independent of the input file size and is unknown until the task is completed [13, 28]. For example, in a facial recognition application, the processing time of a task is unpredictable and unknown in advance. Therefore, most of the existing offloading algorithms proposed in the literature are not applicable as they are unable to address uncertainty in task processing.

Indeed, the uncertain nature of a task's processing cycles brings in considerable challenges to making offloading decisions. The distribution of a task's processing cycles is typically unknown in advance. At best, one may derive some partial (and very limited) knowledge, such as mean, variance, or worst-case values through online measurement. There has been very limited work that addresses processing cycle uncertainty in MEC [10, 13, 21] and most of them do not offer theoretical guarantees. The state-of-the-art work that offers theoretical guarantees is the one by Eshraghi and Liang [10], which is based on *worst-case* analysis. It is well known that results following worst-case analysis tend to be very conservative.

In this paper, we address the uncertainty problem in task processing through probabilistic deadline guarantees (a.k.a chance-constrained programming (CCP) [25]). Instead of providing a hard guarantee of a task's deadline, we allow some occasional violation of a task's deadline—as long as such violation probability is kept below a given threshold (risk level). This approach is plausible for many mobile applications as occasional deadline violations can be tolerated by the highly adaptive nature of human perceptual systems [26]. In our research community, CCP has been in underlay coexistence [17, 19], OFDM scheduling [18], and video streaming [4], etc. To the best of our knowledge, there is no work that employs CCP to address processing uncertainty in MEC.

The main contributions of this paper are summarized as follows:

- To the best of our knowledge, this is the first paper that addresses the uncertainty of task processing cycles in MEC through CCP. In contrast to the state-of-the-art approach where worst-case bounds are used to handle uncertainty, we exploit the estimated mean and variance of task processing cycles and offer probabilistic deadline guarantees.
- Since the constraints for probabilistic deadlines in our problem are intractable in the absence of distribution functions, we employ a powerful tool called *Exact Conic Reformulation*

MobiHoc '21, July 26–29, 2021, Shanghai, China
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8558-9/21/07…$15.00
https://doi.org/10.1145/3466772.3467034

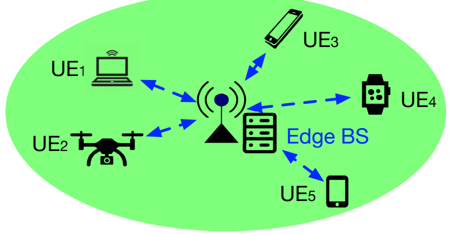


Figure 1: A reference network for MEC

(ECR) to reformulate the chance constraints (involving randomness) into deterministic constraints. In the case that only means and variances are known, ECR is "exact" in the sense that no relaxation errors are introduced in the reformulation.

- To mitigate uncertainty in a task's processing cycles via ECR, we have successfully developed an online solution called Energy-minimized solution with Probabilistic Deadline guarantee (EPD) that minimizes energy consumption at UEs with probabilistic deadline guarantees. When the system is in normal operation (not overloaded), EPD makes offloading decisions to minimize energy consumption while guaranteeing probabilistic deadlines of all tasks. When the system is overloaded (i.e., in an infeasible region where not all tasks' probabilistic deadlines can be met), EPD maximizes the number of tasks that can meet their probabilistic deadlines while attempting to minimize energy if possible.
- Through extensive simulations, we find that EPD successfully guarantees the probabilistic deadlines of tasks using only the estimated mean and variance of processing cycles. As a comparison, EPD offers a significant improvement in energy saving when compared to the state-of-the-art worst-case approach to address uncertainty in processing cycles.

2 SYSTEM DESCRIPTION

Consider the reference network for MEC shown in Fig. 1, where there is a set of UEs and an edge BS. The UEs are connected to the edge BS via wireless channels. The edge BS is equipped (or co-located) with a number of high-performance processors. Throughout this paper, we use the term "BS" to refer to joint edge BS and these co-located high-performance processors.

Consider the scenario where a task is generated by a UE and the output (from processing this task) is to be consumed by the UE with a deadline. However, the processing of a task can be done either locally or remotely. In this work, we consider binary offloading meaning that a task must be *fully* processed either locally or remotely [23]. Clearly, there are some trade-offs in time and energy between processing locally at the UE and processing at the BS:

- **Processing task locally at its UE** In this case, the total time consumption only consists of task processing time at the UE. Further, significant energy may need to be consumed by the UE's battery, depending on the required processing cycles and type of processors.
- **Offloading task to the BS** In this case, the total time consumption consists of data transfer time through wireless channel, task queuing time at the BS, and task processing time at the BS. In terms of energy consumption, the UE will consume energy for uplink and downlink data transfer but will save processing energy due to offloading.

In either case, one would hope that the output after processing a task can come back to the UE's application in time for its consumption. As discussed in Section 1, the required number of processing cycles of a task (and thus processing time) is typically unknown in advance. Due to such an uncertainty, it should be modeled as a random variable (RV). For this RV, in most cases, one does not have its distribution function. Instead, one may estimate its mean and variance through online measurements, which is relatively easy to do (comparing to obtaining its distribution function).

To cope with the uncertainty in a task's processing cycles, we employ a probabilistic deadline guarantee for each task instead of a deterministic one. This means that we will allow some occasional violation of the deadline, as long as such a violation probability is below a tolerable threshold. Note that when the violation probability is 0, we will have a hard deadline, which is a special case of our probabilistic deadline. In this sense, a probabilistic deadline guarantee is more general and better suited for practical purposes.

In the rest of this section, we will briefly describe the behaviors of the UEs and the BS, and then formally state our problem.

UE At the UE side, each UE will generate tasks from time to time. If there is no other pending task request (before its decision is made) at the BS, it shall send its task request to the BS immediately and wait for the decision. Otherwise the task will be allocated to local processing. In other words, each UE can only have one pending task request at the BS but can have multiple offloaded tasks (can be in transmission, queuing or processing) at the BS. For each task request sent to the BS, the UE will receive its decision from the BS. The delay in this decision process will be counted toward total time consumption. If the offloading decision is "local", the UE will process the task locally. Otherwise, it will transmit the task data to the BS for processing and receive the results later from the BS.

BS At the BS side, the BS only makes offloading decisions periodically. This period (denoted as T) can be set by the system operator and can be as low as one Transmission Time Interval (TTI) (in 5G transmission terminology). When the BS makes offloading decisions for the task requests for current T, it considers the current state at the BS, the probabilistic deadline guarantees of tasks, and the objective function. Therefore, in addition to deciding which tasks are to be offloaded, the BS needs to tentatively schedule how the newly offloaded tasks are to be processed in the BS processors. Then the BS sends the decisions to the UEs and exchanges task data with the UEs that have offloading tasks. Note that the tentative task schedule is subject to change during run-time due to processing uncertainty. Such a schedule change must guarantee the probabilistic deadlines of all offloaded tasks.

As for the processors at the BS, we assume they have the same processing speed. We further assume that each task can only be assigned to one processor [7]. There is a shared pool for queued tasks (whose processing has not yet started) at the BS and a queued task can be scheduled to any processor. Once a task starts its processing in a particular processor, it will be processed with the full capacity of the processor without interruption or migration until it finishes (success) or the processing time reaches its maximum cut-off time (forcibly stopped). We note that without cut-off times, a task can run infinitely since we donâĂŹt assume any upper bound on a taskâĂŹs number of processing cycles. Clearly, such cut-off times should be optimized by the BS.

Problem Statement and Technical Challenges In this paper, we are interested in minimizing energy consumption of the UEs while offering probabilistic deadline guarantees for tasks. Specifically, we aim to design an online algorithm that (i) decides what subset of task requests should be offloaded to the BS at the beginning of each time interval T, and (ii) schedules how the offloaded tasks are to be processed, i.e., in which processor and in what order.

There are several technical challenges in this problem. First, it is mathematically challenging to offer a probabilistic guarantee of each task when we only have the knowledge of mean and variance (with no knowledge of the distribution) of a task's required processing cycles. For the special case when the number of processing cycles of a task is given (deterministic), it has been shown in [12, 16] that the corresponding schedulability problem (non-preemption scheduling on multiprocessor) is NP-hard. Since this deterministic schedulability problem is a special case of our problem (involving RVs with unknown distributions), our problem is also NP-hard.

Second, due to uncertainty in the number of required processing cycles, the processing schedule of all the tasks on each processor (at the BS) is only tentative at best—it is subject to further re-scheduling (i.e., change of service order, move to a different processor), depending on the actual number of processing cycles used by a task on the processor. Such "knowledge-only-till-the-end" uncertainty in task processing time brings additional dynamics and challenges in the design of an online algorithm.

3 MATHEMATICAL MODELING AND ANALYSIS

Since the offloading decision is made at the beginning of an interval, we will focus our attention on such a time instance (say τ). Denote M as the number of processors at the BS. Denote $\mathcal{S}$ as the set of tasks that were offloaded to the BS from the previous interval(s) but still have not been finished (i.e., in transmission, queuing, or processing). A task in $\mathcal{S}$ could be in the process of uplink transfer to the BS, waiting at the BS's processor queue, being processed, or in the process of downlink transfer to its UE. For any task in $\mathcal{S}$, its offloading decision (made in previous intervals) cannot be changed. This is important in practice as one would not expect the BS to go back and forth to change its offloading decisions and confuse the UEs. Mathematically, $\mathcal{S}$ represents the current state of unfinished tasks in the system at time τ whose offloading decisions were already made in the past.

Denote $\mathcal{N}$ as the set of N new task requests received from the UEs in the previous interval. Clearly, $\mathcal{N}$ and $\mathcal{S}$ are disjoint. Our goal is to select a subset of tasks from $\mathcal{N}$ for offloading to the BS to minimize energy consumption while meeting the probabilistic deadlines of all tasks.

3.1 Local Processing

If task i is to be processed locally (i.e., no offloading), then the total amount of time consumption (denoted as t_i^{UE}) consists of two parts: (i) the waiting time from the generation time of task i, to the reception of a decision (denoted as t_i^{dec}), and (ii) the local processing time.

Clearly, t_i^{dec} can be easily calculated. At time τ, t_i^{dec} has already passed since the generation time of task i, and thus it should be subtracted from task i's deadline. As for the local processing time of task i, it depends on the required processing cycles of task i (denoted as C_i) and the processing speed (in cycles/sec) of the UE's processor (denoted as $F_{\text{U(i)}}$, where U(i) denotes the UE that generated task i). We have

$$t_i^{\text{UE}} = t_i^{\text{dec}} + \frac{C_i}{F_{\text{U(i)}}} \ . \tag{1}$$

Note that C_i is a RV with unknown distribution at time τ. The exact value of C_i is only known upon the completion of task i. But as discussed in Section 2, we assume its mean (denoted as $\overline{C_i}$) and variance (denoted as V_i) can be estimated.

As for energy consumption, denote e_i^{UE} as the energy consumed at $U(i)$ to process task i. Then following the most common energy consumption model (see e.g., [15, 30]), e_i^{UE} can be modelled by[1]

$$e_i^{\text{UE}} = \alpha_{\text{U(i)}} \cdot C_i \cdot F_{\text{U(i)}}^2 \ , \tag{2}$$

where $\alpha_{\text{U(i)}}$ is a system parameter depending on UE $U(i)$. Note that e_i^{UE} is proportional to C_i and quadratic to $F_{\text{U(i)}}$.

3.2 Offloading to BS

If task i is to be offloaded to the BS for processing, then the total amount of time consumption (denoted as t_i^{BS}) consists of five parts: (i) decision time (denoted as t_i^{dec}), which is the same with that in (1), (ii) uplink transfer time (denoted as t_i^{up}), (iii) waiting time of task i at the BS before its processing (denoted as w_i^{proc}), (iv) task processing time at the BS (denoted as t_i^{proc}), (v) downlink transfer time for the outcome of task i (denoted as t_i^{down}). We have

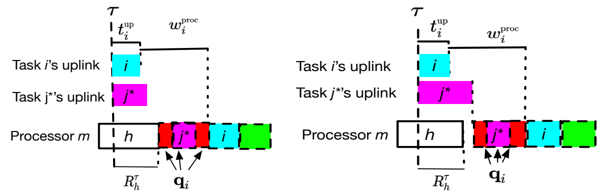
$$t_i^{\text{BS}} = t_i^{\text{dec}} + t_i^{\text{up}} + (w_i^{\text{proc}} + t_i^{\text{proc}}) + t_i^{\text{down}} \ . \tag{3}$$

t_i^{up} and t_i^{down} are the uplink and downlink transfer times for task i, respectively from $U(i)$ to the BS and vice versa, We assume both t_i^{up} and t_i^{down} are given (or their upper bounds can be estimated in advance) in our problem. In the following paragraphs, we discuss w_i^{proc} and t_i^{proc}, which are directly related to C_i's and are the main focus in our modeling and analysis.

3.2.1 w_i^{proc}. At decision time τ, we need to estimate the waiting time (w_i^{proc}) for task i if it is to be offloaded to the BS. Denote m as the processor at the BS to which task i will be offloaded at time τ. To ensure that the processing time of a task is under control, we introduce a *cut-off time* for each task at time τ. Denote R_i^τ as the cut-off time of task i at time τ. Starting from time τ, if task i's processing is not completed by time $\tau + R_i^\tau$, task i will be forcibly terminated. The setting of cut-off times is critical to provide probabilistic deadline guarantees and is part of our design of EPD. In Section 4, we will show how to construct R_i^τ for task i if it is to be offloaded to the BS. With the cut-off times, we can establish two upper bounds for w_i^{proc} based on two different reference tasks.

An Upper Bound based on Preceding Task Denote task k as the preceding task of task i with a cut-off time R_k^τ. This means task k is scheduled immediately before task i on processor m. If

[1] A more complex energy model may be employed as it does not affect the proposed solution procedure.



(a) Case (i): $t_{j^*}^{\text{up}} \le R_h^\tau$ (b) Case (ii): $t_{j^*}^{\text{up}} > R_h^\tau$

Figure 2: An upper bound of w_i^{proc} based on HOL task

task i is received before R_k^τ, i.e., $t_i^{\text{up}} < R_k^\tau$, w_i^{proc} is upper bounded by $R_k^\tau - t_i^{\text{up}}$. Otherwise, task i can be processed upon reception without waiting (i.e., $w_i^{\text{proc}} = 0$). Thus, w.r.t. task k (preceding task), we have

$$w_i^{\text{proc}} \le \text{UB}_1 = \max\{R_k^\tau - t_i^{\text{up}}, 0\} , \tag{4}$$

where UB_1 denotes the upper bound for w_i^{proc} based on the preceding task of task i.

An Upper Bound based on Head-of-line (HOL) Task Denote task h as the HOL task of task i with a cut-off time R_h^τ. This means task k is currently under processing on processor m, as shown in Fig. 2. The remaining processing time of task h is a RV whose mean and variance are unknown (since the distribution of task h's processing time is unknown), and is smaller than R_h^τ.

Denote $\mathbf{q}_i$ as the set of tasks tentatively scheduled on processor m before task i but excluding task h. Clearly, w_i^{proc} is influenced by task h and tasks in $\mathbf{q}_i$: their uplink transfer times and processing times. To find an upper bound of w_i^{proc}, we need to find the task with the maximum remaining uplink transfer time at time τ among tasks in $\mathbf{q}_i \cup \{i\}$. Denote task j^* as the task with maximum remaining uplink transfer time at time τ from tasks in $\mathbf{q}_i \cup \{i\}$, i.e.,

$$j^* = \arg \max_{j \in \mathbf{q}_i \cup \{i\}} t_j^{\text{up}} . \tag{5}$$

When calculating w_i^{proc}, we consider two cases depending on the relationship between $t_{j^*}^{\text{up}}$ and R_h^τ, shown in Fig. 2.

Case (i): $t_{j^*}^{\text{up}} \le R_h^\tau$. This means task j^* (pink in Fig. 2(a)) arrives at the BS before processor m completes processing task h. Based on the definition of j^* in (5), all tasks in $\mathbf{q}_i \cup \{i\}$ should arrive before time $\tau + R_h^\tau$. Then processor m will first finish the remaining processing for task h, followed by tasks in $\mathbf{q}_i$ without any gap. We have

$$w_i^{\text{proc}} \le R_h^\tau + \sum_{q \in \mathbf{q}_i} t_q^{\text{proc}} - t_i^{\text{up}} . \tag{6}$$

Case (ii): $t_{j^*}^{\text{up}} > R_h^\tau$. This means task j^* (pink in Fig. 2(b)) arrives at the BS after processor m completes processing task h. Again, processor m will first finish the remaining processing for task h, followed by tasks in $\mathbf{q}_i$. Unlike case (i), there may be some idle time at processor m following some tasks' processing if the later tasks have not arrived. Nevertheless, such a gap cannot be longer than $t_{j^*}^{\text{up}} - R_h^\tau$ since all tasks must arrive by time $\tau + t_{j^*}^{\text{up}}$ (based on the definition of j^*), i.e., the white gaps among the tasks in $\mathbf{q}_i$ in Fig. 2(b). Thus, we have the following upper bound for w_i^{proc}:

$$w_i^{\text{proc}} \le t_{j^*}^{\text{up}} + \sum_{q \in \mathbf{q}_i} t_q^{\text{proc}} - t_i^{\text{up}} . \tag{7}$$

Combining (6) and (7) from both cases, we have

$$w_i^{\text{proc}} \le \text{UB}_2 = \max\{R_h^\tau, t_{j^*}^{\text{up}}\} + \sum_{q \in \mathbf{q}_i} t_q^{\text{proc}} - t_i^{\text{up}} , \tag{8}$$

where UB_2 is another upper bound of w_i^{proc} based on the HOL task of task i.

Since it is unknown which upper bound ((4) or (8)) is tighter theoretically, we calculate both and set

$$w_i^{\text{proc}} = \min\{\text{UB}_1, \text{UB}_2\} . \tag{9}$$

3.2.2 t_i^{proc}. The processing time of task i at a BS's processor (t_i^{proc}) depends on the required processing cycles of task i (C_i) and the processing speed of the processors at the BS (denoted as F_{BS}). So t_i^{proc} is given by

$$t_i^{\text{proc}} = \min\left\{\frac{C_i}{F_{\text{BS}}},\ R_i^\tau - t_i^{\text{up}} - w_i^{\text{proc}}\right\} , \tag{10}$$

where R_i^τ is the cut-off time of task i at time τ.

As for the energy consumption of $U(i)$, denote e_i^{BS} as the energy consumption of $U(i)$ if task i is offloaded to the BS. Denote $p_{\text{U(i)}}^{\text{up}}$ and $p_{\text{U(i)}}^{\text{down}}$ as the power consumption of $U(i)$ during uplink transmission and downlink reception respectively. In general, $p_{\text{U(i)}}^{\text{up}}$ includes static circuit power consumption and transmit power of $U(i)$, while $p_{\text{U(i)}}^{\text{down}}$ only includes static circuit power consumption. We have

$$e_i^{\text{BS}} = p_{\text{U(i)}}^{\text{up}} \cdot t_i^{\text{up}} + p_{\text{U(i)}}^{\text{down}} \cdot t_i^{\text{down}} . \tag{11}$$

3.3 Objective

Our overall objective is to minimize energy consumption of the UEs while guaranteeing each task's deadline with a given probability. For energy consumption, denote e_i as the actual energy consumption of task i. If task i is offloaded to the BS, then its energy consumption is $e_i = e_i^{\text{BS}}$; otherwise, its energy consumption is $e_i = e_i^{\text{UE}}$, i.e.,

$$e_i = \begin{cases} e_i^{\text{BS}} & \text{if task } i \text{ is offloaded to BS,} \\ e_i^{\text{UE}} & \text{otherwise.} \end{cases} \tag{12}$$

Denote t as the elapsed time of the MEC system from time 0. Over $[0, t]$, the number of decision intervals is $\lfloor t/T \rfloor$. Denote $\boldsymbol{Q}^t$ as the set of task requests generated all the UEs over time $[0, t]$ and $Q^t = |\boldsymbol{Q}^t|$. Then our objective is to minimize E, the average energy consumption per task over $[0, t]$ when $t \to \infty$, i.e.,

$$E = \lim_{t \to \infty} \frac{1}{Q^t} \sum_{i \in \boldsymbol{Q}^t} e_i . \tag{13}$$

In terms of deadline feasibility, denote D_i as the deadline of task i and $\epsilon_{\text{U(i)}}$ as the deadline violation probability (i.e., exceeding D_i) that $U(i)$ can tolerate. $\epsilon_{\text{U(i)}}$ is also known as the *risk level* for $U(i)$. Then we would like to have

$$\mathbb{P}\left\{t_i^{\text{BS}} \le D_i\right\} \ge 1 - \epsilon_{\text{U(i)}} \ \text{ if task } i \text{ is offloaded to BS,} \tag{14a}$$

$$\mathbb{P}\left\{t_i^{\text{UE}} \le D_i\right\} \ge 1 - \epsilon_{\text{U(i)}} \ \text{ otherwise.} \tag{14b}$$

where $\mathbb{P}\{\cdot\}$ denotes probability. t_i^{UE} and t_i^{BS} are defined in (1) and (3) respectively.

Clearly, the motivation of task offloading is either to ensure deadline feasibility or energy saving, or both[2]. In general, deadline

[2]If a task can meet its probabilistic deadline at the UE and costs less energy compared to offloading, this task request should not be sent to the BS.

feasibility of each task is more important than energy saving of its UE. Therefore, a good design should try to guarantee the tasks' deadlines first and then minimize the energy consumption of the UEs. However, when the new task request rate becomes excessively high or the tasks' deadlines are too strict to be met, the feasibility of deadlines from some tasks will not be satisfied. This is the overload scenario and must be addressed in our algorithm design.

4 EPD – AN ENERGY-MINIMIZED SOLUTION WITH PROBABILISTIC DEADLINE GUARANTEE

In this section, we present an online solution to address the offloading problem discussed in Section 3.3. For ease of reference, we call our proposed solution "EPD", an abbreviation for Energy-minimized solution with Probabilistic Deadline guarantee. EPD consists of two components: (i) a scheduling algorithm executed periodically at the beginning of every time interval T, and (ii) a schedule updating algorithm executed upon the completion of a task within each interval. We present these two components in Section 4.1 and 4.2, respectively.

4.1 Periodic Scheduling Per Interval

As discussed in 3.3, we need to prioritize deadline feasibility over energy saving. The main idea of this component is to first separate $\mathcal{N}$ into two disjoint subsets: $\mathcal{N}^{\text{UE}}$ and $\mathcal{N}^{\cancel{\text{UE}}}$ based on whether a task can or cannot meet their probabilistic deadlines if processed locally (i.e., whether or not (14b) can be satisfied). Clearly, $\mathcal{N}^{\text{UE}} \cap \mathcal{N}^{\cancel{\text{UE}}} = \emptyset$ and $\mathcal{N}^{\text{UE}} \cup \mathcal{N}^{\cancel{\text{UE}}} = \mathcal{N}$. For tasks in $\mathcal{N}^{\cancel{\text{UE}}}$, they must be offloaded to the BS to meet their probabilistic deadlines, and thus we will maximize the number of offloaded tasks from $\mathcal{N}^{\cancel{\text{UE}}}$ for deadline feasibility before allocating tasks in $\mathcal{N}^{\text{UE}}$. Then for tasks in $\mathcal{N}^{\text{UE}}$, at least local execution guarantees their probabilistic deadlines, and thus we will further minimize the energy consumption by offloading tasks in $\mathcal{N}^{\text{UE}}$. These correspond to the three steps of the periodic scheduling per interval in EPD, as detailed below.

4.1.1 *Finding $\mathcal{N}^{\text{UE}}$ and $\mathcal{N}^{\cancel{\text{UE}}}$.*

To find $\mathcal{N}^{\text{UE}}$ and $\mathcal{N}^{\cancel{\text{UE}}}$, we need to check whether or not (14b) can be satisfied for all $i \in \mathcal{N}$.

Denote d_i as the remaining portion of task i's deadline at time τ, i.e., $d_i = D_i - t_i^{\text{dec}}$. Substituting (1) into (14b), we have $i \in \mathcal{N}^{\text{UE}}$ if

$$\mathbb{P}\left\{\frac{C_i}{F_{\text{U(i)}}} \le d_i\right\} < 1 - \epsilon_{\text{U(i)}} \ . \tag{15}$$

Recall that C_i is a RV with known mean $\overline{C_i}$ and variance V_i but unknown distribution. The constraint in (15) is called a chance constraint. Due to a lack of distribution knowledge, a key step in CCP is to reformulate the intractable chance constraint into a deterministic constraint. There are a number of methods to perform this reformulation (substitution) such as Chebyshev inequalities, Markov inequalities [11], conditional value-at-risk [27], Bernstein Approximation [6] and *Exact Conic Reformulation* (ECR) [17]. Among them, ECR is the state-of-the-art approach to handle chance constraint (15), which we summarize as follows.

Theorem 1. *Given RVs $\boldsymbol{\xi} = [\xi_1, \xi_2, \cdots, \xi_n]^T$ with known mean $\overline{\boldsymbol{\xi}} = \left[\overline{\xi_1}, \overline{\xi_2}, \cdots, \overline{\xi_n}\right]^T$ (a $n \times 1$ column vector) and covariance matrix $\mathbf{R} = \mathbb{E}\left\{(\boldsymbol{\xi} - \overline{\boldsymbol{\xi}})(\boldsymbol{\xi} - \overline{\boldsymbol{\xi}})^T\right\}$ (a $n \times n$ matrix), a deterministic vector $\mathbf{b} = [b_1, b_2, \cdots, b_n]^T$ (a $n \times 1$ column vector), the risk level ϵ, and a constant z, then we have*

$$\mathbb{P}_{\boldsymbol{\xi} \sim (\overline{\boldsymbol{\xi}}, \mathbf{R})}\left\{\mathbf{b}^T \boldsymbol{\xi} \le z\right\} \ge 1 - \epsilon \ , \tag{16}$$

if and only if

$$\mathbf{b}^T \overline{\boldsymbol{\xi}} + \sqrt{\frac{1-\epsilon}{\epsilon}} \sqrt{\mathbf{b}^T \mathbf{R} \mathbf{b}} \le z \ . \tag{17}$$

In (16), $\boldsymbol{\xi} \sim (\overline{\boldsymbol{\xi}}, \mathbf{R})$ means $\boldsymbol{\xi}$ has mean $\overline{\boldsymbol{\xi}}$, covariance $\mathbf{R}$, and unknown distribution. Under such an assumption, the reformulation of (16) in Theorem 1 is called "exact" because no relaxation errors are introduced in this reformulation [17]. In other words, there exist some distributions of $\boldsymbol{\xi}$ such that the equalities hold and the threshold violation probability is exactly ϵ. For other distributions of $\boldsymbol{\xi}$, the threshold violation probability is smaller than ϵ. Further, ECR makes no assumption of the RVs in $\boldsymbol{\xi}$ such as boundaries or independence and thus it can handle more general cases.

Based on Theorem 1, (15) will hold *if and only if*

$$\frac{1}{F_{\text{U(i)}}} \cdot \overline{C_i} + \sqrt{\frac{1-\epsilon_{\text{U(i)}}}{\epsilon_{\text{U(i)}}}} \cdot \frac{1}{F_{\text{U(i)}}} \cdot \sqrt{V_i} \le d_i \ , \tag{18}$$

which is equivalent to

$$\overline{C_i} + K_{\text{U(i)}} \sqrt{V_i} \le d_i F_{\text{U(i)}} \ , \tag{19}$$

where we define $K_{\text{U(i)}}$ as $K_{\text{U(i)}} = \sqrt{\frac{1-\epsilon_{\text{U(i)}}}{\epsilon_{\text{U(i)}}}}$.

Clearly, each term in (19) is given and there is no RV. It is easy to determine whether or not (19) holds for task i. Further, (19) implies that task i can meet its probabilistic deadline after being processed for $\overline{C_i} + K_{\text{U(i)}}\sqrt{V_i}$ cycles at $U(i)$. Then $U(i)$ can choose to whether continue processing task i or terminate it to save energy, both guarantee the probabilistic deadline of task i.

Using (19) as the criteria to determine whether or not task i belongs to $\mathcal{N}^{\text{UE}}$ or $\mathcal{N}^{\cancel{\text{UE}}}$, we have

$$\begin{aligned} \mathcal{N}^{\text{UE}} &= \left\{i \in \mathcal{N} : \overline{C_i} + K_{\text{U(i)}}\sqrt{V_i} \le F_{\text{U(i)}} d_i\right\} \ , \\ \mathcal{N}^{\cancel{\text{UE}}} &= \left\{i \in \mathcal{N} : \overline{C_i} + K_{\text{U(i)}}\sqrt{V_i} > F_{\text{U(i)}} d_i\right\} \ . \end{aligned} \tag{20}$$

4.1.2 *Offloading tasks in $\mathcal{N}^{\cancel{\text{UE}}}$.*

The goal of this step is to maximize the number of tasks in $\mathcal{N}^{\cancel{\text{UE}}}$ to be offloaded to the BS so that their probabilistic deadlines (as well as those already at the BS) can be satisfied. This step is the core of EPD's periodic scheduling. There are two questions we need to address in this step.

- Q1: First, when choosing a task from $\mathcal{N}^{\cancel{\text{UE}}}$ to offload to the BS, we need a metric to decide which task we should consider first, second, and so forth, i.e., we need an order among the tasks in $\mathcal{N}^{\cancel{\text{UE}}}$ for this step.
- Q2: Second, we need to have a rigorous mathematical criterion to determine whether a selected task from $\mathcal{N}^{\cancel{\text{UE}}}$ can be offloaded to the BS while satisfying all probabilistic deadlines (its own and those already at the BS). This is the main challenge of our problem.

In the following paragraphs, we present the solutions for Q1 and Q2, and then describe how this step works.

Q1: Ordering Tasks in $\mathcal{N}^{\text{UE}}$ For Q1, we introduce a metric v_i for each task $i \in \mathcal{N}^{\text{UE}}$, which is motivated by (19) as follows:

$$v_i = \frac{\overline{C_i} + K_{\text{U(i)}}\sqrt{V_i}}{F_{\text{BS}} d_i} \quad (i \in \mathcal{N}^{\text{UE}}) \ . \tag{21}$$

Then we will always pick a task i with the smallest v_i among the remaining tasks in $\mathcal{N}^{\text{UE}}$, i.e., $i = \arg\min_{j \in \mathcal{N}^{\text{UE}}} v_j$. Note that the metric v_i favors those tasks with small mean $\overline{C_i}$, small variance V_i and large deadline d_i. This is rather intuitive as tasks with these attributes are more likely to be offloaded to the BS while meeting their deadlines, which corresponds to our objective of this step (maximize the number of offloaded tasks in $\mathcal{N}^{\text{UE}}$). Interestingly, in the special case when $V_i = 0$ (i.e., C_i is deterministic), v_i is the "utilization factor" that has been used in real-time scheduling with deterministic processing times [20].

Q2: Criterion for Offloading a Task For Q2, suppose we have picked a task i from $\mathcal{N}^{\text{UE}}$, we need to decide which one (if any), among the M processors at the BS, can accommodate task i. For simplicity, we will consider the M processors sequentially (i.e., from $m = 1$ to M based on its index) and examine one at a time to see if it can accommodate task i. Once we find a processor that meets our requirement, we will choose this processor for task i and update $\mathcal{S}$ (the set of offloaded tasks at the BS) with $\mathcal{S} = \mathcal{S} \cup \{i\}$. This sequential fitting of processors is also called *first-to-fit* in multiprocessor scheduling and works well in practice [5]. If task i cannot fit any processor, the BS will declare that task i offloading is infeasible and send decision "local" to its corresponding UE $U(i)$.

When considering task i w.r.t. a particular processor m, it will be tentatively placed on processor m based on Earliest Deadline First (EDF) since EDF is optimal for single processor scheduling. Then we need to determine whether or not this allocation is feasible such that all offloaded tasks (including task i) meet their probabilistic deadline guarantees. That is, task i can be offloaded to processor m if and only if for any task $j \in \mathcal{S} \cup \{i\}$,

$$\mathbb{P}\left\{t_j^{\text{up}} + w_j^{\text{proc}} + t_j^{\text{proc}} + t_j^{\text{down}} \le d_j\right\} \ge 1 - \epsilon_{\text{U(j)}} \ . \tag{22}$$

Clearly, if task j is scheduled on the $(M-1)$ processors other than m, its time consumption is not affected when task i is offloaded to processor m. Thus, (22) trivially hold for such a task j.

On the other hand, if task j is scheduled on processor m, its time consumption could be affected and thus we need to check whether (22) holds or not. Such tasks scheduled on processor m include: (i) tasks scheduled before task i, which include HOL task h and tasks in $\mathbf{q}_i$, (ii) task i, (iii) tasks scheduled after task i. In the following paragraphs, we will discuss tasks in the order of (i), (ii), and (iii).

(i) Here, task j is the task scheduled before task i (i.e., HOL task h and $j \in \mathbf{q}_i$ in either Fig. 2(a) or Fig. 2(b)). Since task j completes its processing before task i, task j's waiting time in the system (w_j^{proc}) will not be affected by task i. Therefore, the previous deadline guarantee of task j remains unchanged, i.e., (22) still holds.

(ii) Now we consider task i. Based on the analysis in Section 3, t_i^{up} and t_i^{down} in (22) are given constants solely depending on task i. But the calculation of w_i^{proc} and t_i^{proc} involves RVs C_i and C_j ($j \in \mathbf{q}_i$), which we only have knowledge of their means and variances. This is the major challenge here and we will again resort to the reformulation technique ECR in Theorem 1.

Based on (9), we consider two cases for w_i^{proc}, depending on whether $\text{UB}_1 \le \text{UB}_2$ or $\text{UB}_1 > \text{UB}_2$.

Case 1: $\text{UB}_1 \le \text{UB}_2$. In this case, $w_i^{\text{proc}} = \text{UB}_1$. Substituting (4) into (22), we have

$$\mathbb{P}\left\{\max\{R_k^\tau, t_i^{\text{up}}\} + t_i^{\text{proc}} + t_i^{\text{down}} \le d_i\right\} \ge 1 - \epsilon_{\text{U(i)}} \ , \tag{23}$$

where task k is the task immediately preceding task i on processor m. Since $t_i^{\text{proc}} \le \frac{C_i}{F_{\text{BS}}}$ (from (10)), a sufficient condition for (23) to hold is

$$\mathbb{P}\left\{\max\{R_k^\tau, t_i^{\text{up}}\} + \frac{C_i}{F_{\text{BS}}} + t_i^{\text{down}} \le d_i\right\} \ge 1 - \epsilon_{\text{U(i)}} \ .$$

Based on Theorem 1, the above chance constraint (involving RV $C_i \sim (\overline{C_i}, V_i)$) can be equivalently reformulated as

$$\max\{R_k^\tau, t_i^{\text{up}}\} + \frac{\overline{C_i}}{F_{\text{BS}}} + \frac{K_{\text{U(i)}}\sqrt{V_i}}{F_{\text{BS}}} + t_i^{\text{down}} \le d_i \ . \tag{24}$$

Thus, if (24) holds, task i can meet its probabilistic deadline (22).

Further, based on (24), we can construct the cut-off time of task i as follows:

$$R_i^\tau|_{w_i^{\text{proc}}=\text{UB}_1} = \max\{R_k^\tau, t_i^{\text{up}}\} + \frac{\overline{C_i}}{F_{\text{BS}}} + \frac{K_{\text{U(i)}}\sqrt{V_i}}{F_{\text{BS}}} \ . \tag{25}$$

The physical meaning of $R_i^\tau|_{w_i^{\text{proc}}=\text{UB}_1}$ is that it sets a hard cut-off time for task i's processing time if task i is to meet its deadline with a probability at least $1 - \epsilon_{\text{U(i)}}$.

Case 2: $\text{UB}_1 > \text{UB}_2$. In this case $w_i^{\text{proc}} = \text{UB}_2$. Substituting (8) into (22), we have

$$\mathbb{P}\left\{\max\{R_h^\tau, t_{j^*}^{\text{up}}\} + \sum_{q \in \mathbf{q}_i} t_q^{\text{proc}} + t_i^{\text{proc}} + t_i^{\text{down}} \le d_i\right\} \ge 1 - \epsilon_{\text{U(i)}} \ , \tag{26}$$

where task h is the HOL task on processor m.

Since $t_q^{\text{proc}} \le \frac{C_q}{F_{\text{BS}}}$ (from (10)) for $q \in \mathbf{q}_i$ and $t_i^{\text{proc}} \le \frac{C_i}{F_{\text{BS}}}$, a sufficient condition for (26) to hold is

$$\mathbb{P}\left\{\max\{R_h^\tau, t_{j^*}^{\text{up}}\} + \sum_{q \in \{\mathbf{q}_i, i\}} \frac{C_q}{F_{\text{BS}}} + t_i^{\text{down}} \le d_i\right\} \ge 1 - \epsilon_{\text{U(i)}} \ .$$

Since C_i and C_q's are independent RVs, their covariance matrix ("$\mathbf{R}$" in Theorem 1) is a diagonal matrix with V_q (for $q \in \mathbf{q}_i$) and V_i as the diagonal elements. Based on Theorem 1, the above chance constraint can be reformulated equivalently as

$$\max\{R_h^\tau, t_{j^*}^{\text{up}}\} + \sum_{q \in \{\mathbf{q}_i, i\}} \frac{\overline{C_q}}{F_{\text{BS}}} + \frac{K_{\text{U(i)}}}{F_{\text{BS}}} \cdot \sqrt{\sum_{q \in \{\mathbf{q}_i, i\}} V_q} + t_i^{\text{down}} \le d_i \ . \tag{27}$$

Thus, if (27) holds, task j can meet its probabilistic deadline (22). Similarly, based on (27), we can construct R_i^τ as follows:

$$R_i^\tau|_{w_i^{\text{proc}}=\text{UB}_2} = \max\{R_h^\tau, t_{j^*}^{\text{up}}\} + \sum_{q \in \{\mathbf{q}_i, i\}} \frac{\overline{C_q}}{F_{\text{BS}}} + \frac{K_{\text{U(i)}}}{F_{\text{BS}}} \cdot \sqrt{\sum_{q \in \{\mathbf{q}_i, i\}} V_q} \ . \tag{28}$$

Combining the two conditional cut-off times in (25) and (28), the cut-off time of task i is

$$R_i^\tau = \min\left\{R_i^\tau|_{w_i^{\text{proc}}=\text{UB}_1}, R_i^\tau|_{w_i^{\text{proc}}=\text{UB}_2}\right\} \ . \tag{29}$$

In summary, task i can meet its probabilistic deadline if

$$R_i^{\mathcal{T}} + t_i^{\text{down}} \le d_i \ . \tag{30}$$

(iii) If task j is scheduled after task i on processor m (green aera in either Fig. 2(a) or Fig. 2(b)), the whole decision process is the same as in (ii) for task i except a notation change for task j.

With the above solutions to Q1 and Q2, this step works as follows. We first order the tasks in $\mathcal{N}^{\cancel{\text{UE}}}$ based on (21) in ascending order. Then we iteratively pick a task i from $\mathcal{N}^{\cancel{\text{UE}}}$ and update $\mathcal{N}^{\cancel{\text{UE}}}$ with $\mathcal{N}^{\cancel{\text{UE}}} \backslash \{i\}$. For each picked task i, we perform the procedure in Q2 to examine if task i can be offloaded to the BS and send the decisions to the UEs. This step stops when $\mathcal{N}^{\cancel{\text{UE}}} = \emptyset$.

4.1.3 Offloading Tasks in $\mathcal{N}^{UE}$. In the previous paragraphs, we have shown how to decide which tasks in $\mathcal{N}^{\cancel{\text{UE}}}$ can be offloaded to the BS. Now we move on to consider tasks in $\mathcal{N}^{\text{UE}}$ to further minimize energy consumption.

We follow a similar approach as we did when offloading tasks in $\mathcal{N}^{\cancel{\text{UE}}}$. The only difference here is the metric used in deciding which task in $\mathcal{N}^{\text{UE}}$ we should consider first, second, and so forth. Instead of (21), we introduce the following metric (since our objective now is solely to minimize energy):

$$\eta_i = \overline{e}_i^{\text{UE}} - e_i^{\text{BS}} \quad (i \in \mathcal{N}^{\text{UE}}) \ , \tag{31}$$

where $\overline{e}_i^{\text{UE}}$ is the mean of e_i^{UE} (see (2)) and e_i^{BS} is defined in (11). η_i is the potential energy saving on average if task i is offloaded to the BS. So EPD will iteratively choose the task with the highest η_i from $\mathcal{N}^{\text{UE}}$, which corresponds to our objective in this step (minimize energy consumption).

4.2 Schedule Updates within an Interval

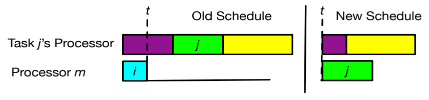
In this subsection, we present the second component of EPD: schedule updates within an interval. This algorithm runs upon completion of a task, either finishing its processing before its cut-off time or being terminated due to cut-off time. The goal of this component is to ensure all M processors are fully utilized as much as possible.

Consider task i is completed on processor m at time t. There are three cases due to the departure of task i, as shown in Fig. 3:

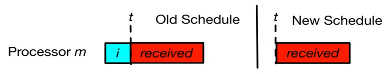
Case (i): Task i is the last task scheduled on processor m, as shown in Fig. 3(a). Instead of leaving processor m idle, we should move another task, say j, with the earliest deadline from a queue of another processor. After this move-over, task j starts its processing at time t, which is earlier than its previous schedule. Further, the scheduled processing times for all other offloaded tasks are either earlier or the same. Thus, all tasks can meet their probabilistic deadlines. The cut-off times of task j and other tasks on task j's original processor will need to be updated based on (29).

Case (ii): Task i is not the last task scheduled on processor m and the task scheduled after task i has already been received at the BS, as shown in Fig. 3(b). In this case, we immediately start to process this task at time t and no update is needed.

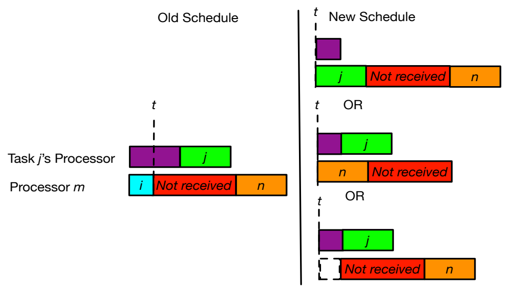
Case (iii): Task i is not the last task scheduled on processor m but the task scheduled immediately after task i has not been received at the BS yet, as shown in Fig. 3(c). In this case, instead of leaving processor m idle, we should try to find a task among those already at the BS and immediately move it over to processor m for processing.



(a) Case (i): Task i is the last task on processor m



(b) Case (ii): Task scheduled after task i on processor m has arrived



(c) Case (iii): Task scheduled after task i on processor m has not arrived

Figure 3: Possible scenarios for schedule updates.

Unlike (i), we must make sure that the tasks scheduled on processor m can still meet their probabilistic deadlines (22).

To do this, we iteratively examine all received tasks (candidates) based on EDF, and check whether or not all offloaded tasks' probabilistic deadlines are still satisfied if the candidate task immediately starts processing on processor m. If we find that such a task exists, it will be processed immediately on processor m. Note that at time t (old schedule), this chosen task could be tentatively scheduled on another processor (task j in Fig. 3(c)) or on processor m (task n in Fig. 3(c)). Further, we need to update the cut-off times of tasks scheduled on processor m and the cut-off times of the tasks on the original processor of this task based on (29). If such a task cannot be found, processor m will have to be put in idle until its next scheduled task arrives and no update of the task schedule is needed.

4.3 Complexity Analysis

In this section, we analyze the complexity of EPD.

For periodic scheduling at the beginning of each time interval T, there are at most N new task requests from $\mathcal{N}$. For a specific task request i, it has at most M possible allocations. Given a tentative allocation of task i, we need to check whether all offloaded tasks in $\mathcal{S} \cup \{i\}$ can meet their probabilistic deadlines (22), and there are no more than $S + N$ tasks in $\mathcal{S} \cup \{i\}$. For a specific task in $\mathcal{S} \cup \{i\}$, say task j, the complexity depends on the calculation of (27). Though there are two sums of multiple items in (27): $\sum_{q \in \{\mathbf{q}_j, j\}} \overline{C_q}$ and $\sum_{q \in \{\mathbf{q}_j, j\}} V_q$, they can be calculated from the result when checking the task preceding task j, which has $O(1)$ complexity. Thus, the complexity of periodic scheduling at the beginning of each time interval is $O(NM(S+N))$.

For scheduling updates within an interval (upon completion of a task at a processor), the core step is finding a new task to process

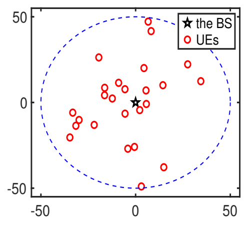


Figure 4: A cellular network with 25 UEs

if necessary. This step has at most $S + N$ candidate tasks. For each candidate task, we need to check the probabilistic deadlines (22) for at most $S + N$ tasks with a complexity of $O(S + N)$. Thus, the complexity of scheduling updates within an interval from EPD is $O((S + N)^2)$.

In summary, both components of EPD have polynomial complexity and we will show that their running times are negligible compared to interval T in Section 5.

5 NUMERICAL RESULTS

In this section, we evaluate EPD through simulations. We will focus on percentages of tasks offloaded, average energy consumption per task (i.e., E in (13)), and deadline violation probability.

5.1 Settings

We consider a cellular network with a BS at the center and 25 UEs ($N \leq 25$) randomly distributed around the BS, as shown in Fig. 4. The radius of the cell is assumed to be 50 meters, following the settings in [9, 24]. The wireless channel between a UE and the BS is characterized by a path-loss model from [1]: $\beta_{U(i)} = 38 + 30 \times \log_{10} r_{U(i)}$, where $\beta_{U(i)}$ and $r_{U(i)}$ are the path-loss (in dB) and distance between $U(i)$ and the BS (in meters), respectively.

The uplink and downlink data rates of $U(i)$ are calculated based on $\beta_{U(i)}$, minimum bandwidth, transmit power of $U(i)$ and the BS, and thermal noise using Shannon formula. Without loss of generality, we assume the channel bandwidth for both uplink and downlink is 100 MHz. So the minimum bandwidth for each UE is 4 MHz, which occurs when all 25 UEs are active. The transmit powers of UEs and the BS are set to 0.1 W and 1 W respectively. The thermal noise is set to 10^{-10} W. The static circuit power consumption at the UEs is set to 0.05 W [29].

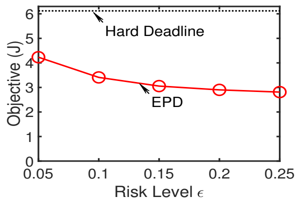
We consider 5 processors ($M = 5$) and each one has a processing speed of $F_{BS} = 4.3 \times 10^9$ cycles/sec (e.g., Intel core i9-7900X). At each UE, there is only 1 processor and its processing speed is assumed to be $F_{U(i)} = 1.8 \times 10^9$ cycles/sec (e.g., Qualcomm Snapdragon 855). The parameter α_i in (2) is set to 10^{-27}.

For each generated task i at its UE, denote L_i^{up} and L_i^{down} as the file sizes (in number of bits) before and after its processing. L_i^{up} is given for each specific simulation. Without loss of generality, we generate L_i^{down} as 0.5 ∼ 1.5 times of L_i^{up} based on a uniform distribution. Then the upper bound of uploading and downloading time can be calculated based on the file sizes and data rates respectively. The probabilistic deadline guarantee for each task (i.e., its deadline and risk level) will be given for each specific simulation.

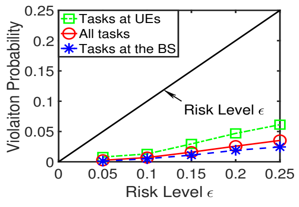
To simulate task processing uncertainty, C_i is generated by $C_i = \omega_i L_i^{up}$ where ω_i is a RV, called "processing density" of task i meaning the required computation cycles per bit for task i. Without loss of generality, we assume ω_i's follow the same distribution, i.e., the same kind of task. In the simulation, we generate ω_i based on certain distributions with given mean $\overline{C_i}/L_i^{up}$ and variance $V_i/(L_i^{up})^2$. The distribution used here is solely for simulation purposes. EPD does not know the distribution and the value of C_i beforehand and only knows its actual value upon the completion of task i.

Table 1: Percentage of tasks offloaded in EPD

Risk Level ϵ	0.05	0.1	0.15	0.2	0.25
Offload Percentage (%)	55.84	60.75	62.96	64.84	66.26



(a) Average energy consumption



(b) Deadline violation probability

Figure 5: Performance of EPD under different ϵ's

5.2 A Case Study

We use a case study to examine the internal mechanics of EPD and its performance behavior. L_i^{up} is randomly generated from 32 KB to 1 MB based on uniform distribution, which covers a wide range of file sizes. We assume each UE generates tasks following a Poisson process with a rate of $\lambda = 1/3$ per second. The periodic interval for EPD scheduling is 200 ms, i.e., $T = 200$ ms. We simulate the MEC system for 1 hour, which contains 18,000 scheduling intervals.

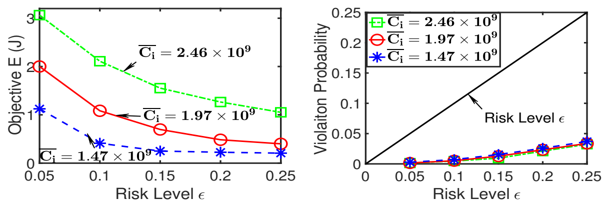
For simulation purpose, we assume ω follows Gamma distribution [10]. A Gamma distribution is usually denoted as $\Gamma(k, \theta)$, where k is the shape parameter and θ is the scale parameter [11]. We set $k = 2$ and $\theta = 400$ in this subsection. Further, to cover more general cases, we set the deadline of task i, D_i, randomly between 2 to 5 times of the average local processing time (i.e., $\overline{C_i}/F_{U(i)}$) following a uniform distribution. The reason we chose 2 as lower bound and 5 as upper bound is to avoid overly strict and overly loose probabilistic deadlines respectively (both lead to trivial cases).

For comparison, we also show the results from state-of-the-art worst-case optimization [10], where the deadline of each task is set based on the empirical upper bound of ω_i and is a hard deadline (i.e., no violation is allowed). To the best of our knowledge, this is the only existing solution that provides deadline guarantees when addressing task processing uncertainty. Further, since an offline optimal solution cannot be found due to formidable complexity, we will only compare EPD with this worst-case optimization.

As shown in Fig. 5(a), the objective value E by EPD is always smaller than the worst-case approach (with hard deadline guarantee). Even for $\epsilon = 0.05$, the per task energy saving is already 31%. When ϵ is increased to 0.25, EPD saves up to 54% in energy consumption when compared to hard deadline guarantee. When ϵ increases from 0.05 to 0.25, we see that the objective value E from EPD decreases monotonically, which is expected since more tasks can be offloaded to the BS when ϵ increases. This can be validated by the percentages of offloaded tasks under each ϵ, as shown in

Table 2: Percentage of offloaded tasks under different $\overline{C_i}$'s

Risk level ϵ	0.05	0.1	0.15	0.2	0.25
$\overline{C_i} = 2.46 \times 10^9$	61.94	73.79	81.01	84.28	86.92
$\overline{C_i} = 1.97 \times 10^9$	68.35	83.43	89.52	92.74	94.01
$\overline{C_i} = 1.47 \times 10^9$	76.99	92.03	95.41	95.95	96.29



(a) Average energy consumption (b) Deadline violation probability

Figure 6: Performance of EPD under different $\overline{C_i}$'s

Table 1. As ϵ increases from 0.05 to 0.25, the percentage of tasks that are offloaded to the BS increases from 55.84% to 66.26%. For the case of hard deadline guarantee, only 45.95% tasks are offloaded to the BS, which is smaller than that by EPD under all risk levels.

Fig. 5(b) shows the deadline violation probability of EPD under different risk levels. In addition to showing violation probability for all tasks, we also break down the violation probabilities between offloaded tasks and tasks processed at the UEs (excluding those tasks from $\mathcal{N}^{\text{UE}}$ but rejected by the BS for offloading). There are two observations from Fig. 5(b):

First, the violation probabilities are always under the target risk levels (regardless of whether the tasks are offloaded to the BS or processed locally). This affirms our desired probabilistic guarantee for the tasks. The gap between the risk levels and the actual threshold violation probabilities is because the used distribution (Gamma) is not a worst-case distribution that maximizes the deadline violation probability, as discussed in Section 4.1.

Second, the violation probabilities from locally processed tasks are higher than those offloaded to the BS. This is because the use of R_h^τ (HOL task h) in (28) introduces relaxation at the BS side but no relaxation is introduced at the UE side.

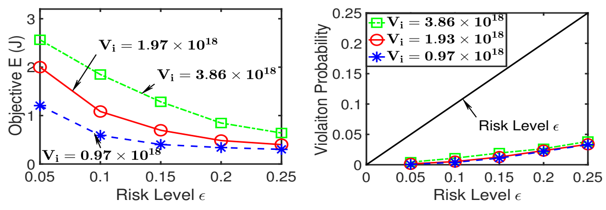
Now we show the running time of EPD on a Xeon E5-1650v2 CPU using MATLAB R2017b. The running time of periodic scheduling in EPD has a mean of 0.2951 ms and a standard deviation of 0.2763 ms. The running time of scheduling updates within an interval is even smaller, with a mean of 0.0330 ms and a standard deviation of 0.0966 ms. From our simulation, the maximum number of tasks completed on a processor within one interval is no more than 5. So the amount of time used to update task schedule on the same processor is around 0.1650 ms. Therefore, the running time of both components in EPD are negligible compared to the interval $T = 200$ ms and EPD meets the real-time processing environment.

5.3 Impact of C_i

In this section, we study the impact of C_i on EPD's performance. For ease of controlling C_i's mean ($\overline{C_i}$) and variance (V_i) in the simulation study, we set $L_i^{\text{up}} = 300$ KB for all tasks in this study, which is a typical file size of a 1440×1080 JPEG 24 bit figure. This will allow the randomness of C_i to be controlled solely by changing the distributions of one random variable ω_i.

Table 3: Percentage of offloaded tasks under different V_i's

Risk Level ϵ	0.05	0.1	0.15	0.2	0.25
$V_i = 3.86 \times 10^{18}$	65.01	71.32	80.03	86.82	89.89
$V_i = 1.93 \times 10^{18}$	68.35	83.43	89.52	92.74	94.01
$V_i = 0.97 \times 10^{18}$	81.71	91.25	94.17	95.22	95.73



(a) Average energy consumption (b) Violation probability

Figure 7: Performance of EPD with different V_i's

5.3.1 *Impact of* $\overline{C_i}$.

We use three Gamma distributions for C_i to study the impact of $\overline{C_i}$. Specifically, we set ω_i in (5.1) to $\Gamma(3.125, 320)$, $\Gamma(2, 400)$, and $\Gamma(1.125, 533.33)$, respectively. Based on (5.1), we obtain $C_i \sim \Gamma(3.125, 7.86 \times 10^8)$, $C_i \sim \Gamma(2, 9.83 \times 10^8)$ and $C_i \sim \Gamma(1.125, 13.11 \times 10^8)$. The variance of all three distributions of C_i's is 1.93×10^{18} while the means of $\overline{C_i}$'s in the three distributions are: 2.46×10^9, 1.97×10^9 and 1.47×10^9.

To focus on the impact of $\overline{C_i}$, we use the same deadline for all tasks. Since a task's average processing time on a UE is 1.37 seconds (with a standard deviation of 0.77 seconds) when $\overline{C_i} = 2.46 \times 10^9$, we set $D_i = 3$ seconds for all tasks, which is also $1\backslash\lambda$ coincidentally.

The percentages of offloaded tasks are shown in Table 2. The performance w.r.t. ϵ under different $\overline{C_i}$'s is shown in Fig. 6.

As shown in Table 2, under a given risk level ϵ (each column), the offloading percentages increases with decreasing $\overline{C_i}$. This is because under the same deadline, when $\overline{C_i}$ decreases, tasks are becoming "smaller" and more tasks are offloaded to the BS, thus saves more energy. This is corroborated in Fig. 6(a), where the objective curve for $\overline{C_i} = 2.46 \times 10^9$ is the highest and the one for $\overline{C_i} = 1.47 \times 10^9$ is the lowest. Note that in Fig. 6(a), for a given ϵ, the increase in E is not proportional to the increase in $\overline{C_i}$. For example, at $\epsilon = 0.15$, when $\overline{C_i}$ increases 25% from 1.97×10^9 to 2.46×10^9, the objective E increases from 2.00 to 3.06, a 35% increase. This is because the average energy consumption is higher when a task is processed locally than offloaded to the BS. So when both factors (offloading percentage and difference in energy consumption at the UE vs. the BS) are taken into considerations, the ratio of increase in E is faster than that in $\overline{C_i}$. In general, under the same deadline D_i and risk level ϵ, a higher $\overline{C_i}$ leads to a lower offloading percentage and a higher energy consumption per task (objective E).

Fig. 6(b) shows the violation probabilities of all tasks. As shown in Fig. 6(b), the violation probabilities are all smaller than the risk level, which validates that EPD guarantees the probabilistic deadline of tasks. Further, the curves overlap with each other, which suggests the actual violation probabilities are rather independent of $\overline{C_i}$.

5.3.2 *Impact of* V_i.

We use three Gamma distributions for C_i to study the impact of V_i. We set ω_i in (5.1) to $\Gamma(1, 800)$, $\Gamma(2, 400)$, and $\Gamma(4, 200)$, respectively. Based on the (5.1), we obtain $C_i \sim \Gamma(1, 19.66 \times 10^8)$, $C_i \sim \Gamma(2, 9.83 \times 10^8)$, and $C_i \sim \Gamma(4, 4.92 \times 10^8)$. The mean of all three distributions of C_i's ($\overline{C_i}$'s) is 1.97×10^9 while the variances of C_i's (V_i's) in these three distributions are: 3.86×10^{18}, 1.93×10^{18}, and 0.97×10^{18}. The deadline of tasks are $D_i = 3$ seconds.

The percentages of tasks offloaded are shown in Table 3. The performance w.r.t. ϵ under different V_i's is shown in Fig. 7.

As shown in Table 3, under a specific risk level ϵ (each column), the offloading percentages increase with a decreasing V_i. This is corroborated in Fig. 7(a), where the objective curve for $V_i = 3.86 \times 10^{18}$ is the highest and the one for $V_i = 0.97 \times 10^{18}$ is the lowest. The reason is that when V_i increases, we need to allocate more processing cycles per task in order to meet its probabilistic deadline guarantee. As a result, fewer tasks can be offloaded to the BS, and thus the objective E increases. Further, in Fig. 7(a), for a given ϵ, the increase in E is not proportional to the increase in V_i. For example, when V_i increases 100% from 1.93×10^{18} to 3.86×10^{18}, the objective E increases 34%, 82% and 37% at $\epsilon = 0.05,\ 0.15$ and 0.25 respectively. This is because the average energy consumption is independent of V_i and the percentages of offloaded tasks are not increasing proportionally to V_i.

Fig. 7(b) shows the violation probabilities of all tasks. As shown in this figure, the violation probabilities are all smaller than the risk levels, which validates that EPD guarantees the probabilistic deadlines. Similar to Fig. 6(b), the overlap of three curves suggests that deadline violation probabilities are rather independent of V_i.

6 CONCLUSIONS

The lack of knowledge (or uncertainty) of a task's processing cycles poses a fundamental challenge to making offloading decisions in MEC, especially when there is a need to guarantee tasks' deadlines. In this paper, we introduced a novel approach by guaranteeing a task's probabilistic deadline via chance-constrained programming. Our approach only uses the estimated mean and variance of the unknown processing cycles and does not assume any knowledge of their distributions or worst-case bounds. By employing a powerful tool called Exact Conic Reformulation (ECR), we were able to reformulate probabilistic chance constraints into deterministic ones without any relaxation errors. Based on ECR, we developed a solution called Energy-minimized solution with Probabilistic Deadline guarantee (EPD) that minimizes the energy consumption at the UEs while guaranteeing probabilistic deadlines of all tasks when the system is not overloaded. When the system is overloaded and not all tasks' deadlines can be guaranteed, EPD maximizes the number of tasks that can meet their probabilistic deadlines while attempting to minimize energy as much as possible. Simulation results show that EPD can offer a significant improvement in energy saving when compared to the state-of-the-art approach based on upper bounds.

7 ACKNOWLEDGMENTS

This research was supported in part by ONR under MURI Grant N00014-19-1-2621, Virginia Commonwealth Cyber Initiative (CCI), and Virginia Tech Institute for Critical Technology and Applied Science.

REFERENCES

[1] 3GPP. *TR 36.931: Radio Frequency (RF) requirements for LTE Pico Node B*, June 2018. Version 15.0.0. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2589.

[2] 3GPP. *TR 23.501: System architecture for the 5G System (5GS)*, Aug. 2020. Version 16.5.1. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144.

[3] Abbas, N., Zhang, Y., Taherkordi, A., and Skeie, T. Mobile edge computing: A survey. *IEEE Internet of Things Journal 5*, 1 (2017), 450–465.

[4] Atawia, R., Abou-Zeid, H., Hassanein, H. S., and Noureldin, A. Joint chance-constrained predictive resource allocation for energy-efficient video streaming. *IEEE J. Selected Areas in Commun. 34*, 5 (2016), 1389–1404.

[5] Baker, T. P. A comparison of global and partitioned EDF schedulability tests for multiprocessors. In *Proc. ACM RTNS 2005*. Paris, France, May 2005.

[6] Ben-Tal, A., and Nemirovski, A. Selected topics in robust convex optimization. *Mathematical Programming 112*, 1 (2008), 125–158.

[7] Cao, J., Yang, L., and Cao, J. Revisiting computation partitioning in future 5G-based edge computing environments. *IEEE Internet of Things Journal 6*, 2 (2018), 2427–2438.

[8] Chen, J., and Ran, X. Deep learning with edge computing: A review. *Proceedings of the IEEE 107*, 8 (2019), 1655–1674.

[9] Chen, X., Jiao, L., Li, W., and Fu, X. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans. Networking 24*, 5 (2015), 2795–2808.

[10] Eshraghi, N., and Liang, B. Joint offloading decision and resource allocation with uncertain task computing requirement. In *Proc. IEEE INFOCOM*, pp. 1414–1422. Paris, France, May 2019.

[11] Feller, W. *An Introduction to Probability Theory and Its Applications*, vol. 2. John Wiley & Sons. Ch. 1, 2008.

[12] Fisher, N., and Baruah, S. The partitioned multiprocessor scheduling of non-preemptive sporadic task systems. In *Proc. ACM RTNS*, pp. 99–108. Poitiers, France, May 2006.

[13] Gao, W., Li, Y., Lu, H., Wang, T., and Liu, C. On exploiting dynamic execution patterns for workload offloading in mobile cloud applications. In *Proc. IEEE ICNP*, pp. 1–12. Raleigh, NC, USA, Oct. 21-24, 2014.

[14] Kekki, S., Featherstone, W., Fang, Y., Kuure, P., Li, A., Ranjan, A., Purkayastha, D., Jiangping, F., Frydman, D., Verin, G., et al. MEC in 5G networks. *ETSI white paper 28* (2018), 1–28.

[15] Kwak, J., Kim, Y., Lee, J., and Chong, S. Dream: Dynamic resource and task allocation for energy minimization in mobile cloud systems. *IEEE J. Selected Areas in Commun. 33*, 12 (2015), 2510–2523.

[16] Leung, J. Y., and Pinedo, M. Minimizing total completion time on parallel machines with deadline constraints. *SIAM J. Computing 32*, 5 (2003), 1370–1388.

[17] Li, S., Huang, Y., Li, C., Jalaian, B. A., Hou, Y. T., and Lou, W. Coping uncertainty in coexistence via exploitation of interference threshold violation. In *Proc. ACM MobiHoc*, pp. 71–80. Catania, Italy, July 2019.

[18] Li, W. W.-L., Zhang, Y. J., So, A. M.-C., and Win, M. Z. Slow adaptive OFDMA systems through chance constrained programming. *IEEE Trans. Signal Processing 58*, 7 (2010), 3858–3869.

[19] Liu, Z., Chan, K. Y., Ma, K., Guan, X., et al. Chance-constrained optimization in D2D-based vehicular communication network. *IEEE Trans. Vehicular Technology 68*, 5 (2019), 5045–5058.

[20] López, J. M., Díaz, J. L., and García, D. F. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Systems 28*, 1 (2004), 39–68.

[21] Lyu, X., and Tian, H. Adaptive receding horizon offloading strategy under dynamic environment. *IEEE Commun. Letters 20*, 5 (2016), 878–881.

[22] Mach, P., and Becvar, Z. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun. Surveys & Tutorials 19*, 3 (2017), 1628–1656.

[23] Mao, Y., You, C., Zhang, J., Huang, K., and Letaief, K. B. A survey on mobile edge computing: The communication perspective. *IEEE Commun. Surveys & Tutorials 19*, 4 (2017), 2322–2358.

[24] Mao, Y., Zhang, J., and Letaief, K. B. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE J. Selected Areas in Commun. 34*, 12 (2016), 3590–3605.

[25] Nemirovski, A., and Shapiro, A. Convex approximations of chance constrained programs. *SIAM J. Optimization 17*, 4 (2007), 969–996.

[26] Painter, T., and Spanias, A. Perceptual coding of digital audio. *Proceedings of the IEEE 88*, 4 (2000), 451–515.

[27] Rockafellar, R. T., Uryasev, S., et al. Optimization of conditional value-at-risk. *Journal of Risk 2* (2000), 21–42.

[28] Shmoys, D. B., Wein, J., and Williamson, D. P. Scheduling parallel machines on-line. *SIAM J. Computing 24*, 6 (1995), 1313–1331.

[29] Sun, H., Zhou, F., and Hu, R. Q. Joint offloading and computation energy efficiency maximization in a mobile edge computing system. *IEEE Trans. Vehicular Technology 68*, 3 (2019), 3052–3056.

[30] Zhang, W., Wen, Y., Guan, K., Kilper, D., Luo, H., and Wu, D. O. Energy-optimal mobile cloud computing under stochastic wireless channel. *IEEE Trans. Wireless Commun. 12*, 9 (2013), 4569–4581.