Multi-user Task Offloading to Heterogeneous Processors with Communication Delay and Budget Constraints

Sowndarya Sundar*, Jaya Prakash Champati†, and Ben Liang*
*Electrical and Computer Engineering, University of Toronto, Ontario, Canada
†Information Science and Engineering, EECS, KTH Royal Institute of Technology, Stockholm, Sweden
Email: ssundar@ece.utoronto.ca, jpra@kth.se, liang@ece.utoronto.ca

Abstract—We study task scheduling and offloading in a cloud computing system with multiple users where tasks have different processing times, release times, communication times, and weights. Each user may schedule a task locally or offload it to a shared cloud with heterogeneous processors by paying a price for the resource usage. We consider four different models in this paper: (i) zero task release and communication times, (ii) non-zero task release times and zero communication times, (iii) non-zero task release times and fixed communication times, and (iv) non-zero task release times and sequence-dependent communication times. Our work aims at identifying a task scheduling decision that minimizes the weighted sum completion time of all tasks, while satisfying the users' budget constraints. We propose an efficient solution framework for this NP-hard problem. As a first step, we use a relaxation and a rounding technique to obtain an integer solution that is a constant factor approximation to the minimum weighted sum completion time. This solution violates the budget constraints, but the average budget violation decreases as the number of users increases. Thus, we develop a scalable algorithm termed Single-Task Unload for Budget Resolution (STUBR), which resolves budget violations and orders the tasks to obtain robust solutions. We prove performance bounds for the rounded solution as well as for the budget-resolved solution, for all four models considered. Via extensive trace-driven simulation for both chess and compute-intensive applications, we observe that STUBR exhibits robust performance under practical scenarios and outperforms existing alternatives. We also use simulation to study the scalability of STUBR algorithm as the number of tasks and the number of users in the system increases.

1 Introduction

Computational offloading is a key feature of cloud computing that led to the development of Mobile Cloud Computing (MCC) systems, where mobile devices offload their computational tasks to cloud resource providers [2]. Edge computing [3]–[7] is a recent advancement in MCC where finite computational/cloud resources are made available at the edge of the network or in the vicinity of the mobile users. For example, in a Mobile/Multiaccess Edge Computing (MEC) system [5]–[7], MEC servers are deployed at the cellular base stations and are shared by the mobile users. Each user pays a monetary cost for the computational resource usage.

Motivated by above systems, we study a problem of task scheduling and offloading in a cloud computing system to minimize the computational delays of the users' tasks. Several existing works study the offloading problem for single-user environment [8]–[13], multi-user environments with simplified cloud models [14]–[19], and optimize energy or makespan objectives [11], [17], [18], [20]–[25]. Several works consider a sum completion time objective but in the absence of budget constraints [26]–[32]. In this work, we study a multi-user scenario with finite-capacity user devices, a finite-capacity cloud consisting of heterogeneous servers, budget constraints for the users, and an objective to minimize weighted sum completion time.

In particular, we consider user tasks that may have different processing times, release times, communication times, and weights. A task may be executed locally on the user's device or offloaded to a server at the finite-capacity cloud. The servers at the cloud are heterogeneous processors with different speeds. The users are required to pay a certain monetary price based on the usage time of a processor at the cloud, and the price may potentially depend on the processor speed. Each user has a specific budget which determines the monetary cost that the user is willing to spend for offloading tasks to the cloud.

Our objective is to identify the task scheduling decision that minimizes the sum of weighted completion times of all tasks subject to all users' budget constraints. The problem is NP-hard since minimizing the sum of weighted completion times of jobs with release times on a single processor is NP-hard [33]. For a special case of our problem where there is a *single user* and *no budget constraint*, an efficient solution was proposed in [32]. However, extending their solution is non-trivial even for a single budget constraint. We will see later that having budget constraints for multiple users makes the problem much more challenging. Our solution approach is inspired by an interval-indexed Integer Linear Program (ILP) introduced in [32]. We exploit the structure of an approximation solution to such an ILP to solve our

- *Part of this paper has appeared in [1]. This new version contains substantial revision with new model extensions, analysis, and simulation results.*
- *This work has been funded in part by the Natural Sciences and Engineering Research Council of Canada.*

problem.

Our main contributions are summarized below:

1) We first consider the problem where all tasks are available at time zero and communication times are negligible. We formulate an interval-indexed ILP inspired by [32]. Using a relaxed LP-solution, we obtain an integer solution that is shown to provide a constant-factor approximation to the minimum weighted sum completion time. Even though this integer solution violates the budget constraints, we make an interesting observation that the average budget violation decreases with respect to the number of users.
2) Based on the above observation, we propose an algorithm termed Single Task Unload for Budget Resolution (STUBR). In addition to finding a relaxed and rounded LP-solution for the above interval-indexed ILP, STUBR resolves budget violations. We prove performance bounds for this budget-resolved solution. We then use a greedy task ordering scheme on each processor to further reduce the weighted sum completion time. We also study the computational complexity of STUBR.
3) We then extend STUBR to more practical models (a) with task release times, (b) with fixed communication times, and (c) with sequence-dependent communication times, i.e., considering a finite-capacity channel model where tasks must be sequenced and communicated. We obtain performance bounds for these cases as well.
4) Our trace-driven simulation shows that STUBR performs consistently better than the existing alternatives. It exhibits maximum performance gains of more than 50% for both chess and compute intensive applications [34] in comparison with the Greedy Weighted Shortest Processing Time (WSPT) scheme. Finally, our simulation results demonstrate that STUBR is highly scalable with respect to the number of users in the system.

The rest of the paper is organized as follows. In Section 2, we present the related work. Section 3 describes the system model and the problem formulation. In Section 4, we propose the STUBR algorithm, and present performance guarantees. In Section 5, we extend this to the problem with release times, fixed communication times, and with sequence-dependent communication times. Section 6 presents the simulation results, and we conclude the paper in Section 7.

2 Related work

The problem of computational offloading and scheduling in the mobile cloud environment has received much recent attention. Existing works that investigate this problem for a multi-user multi-task system often tend to view the cloud as a single entity. For example, in [14], the tasks were assumed to have constant processing time at the cloud and the queueing delay is considered. In [15]–[18], the remote cloud was assumed to be an infinite server. Similarly, in [19], a multi-user system with just a single multi-core MEC server was considered where the server was assumed to maintain separate buffers for separate users. Unlike these aforementioned studies, we account for the *heterogeneity* and *finite-capacity* of the cloud resource by considering a finite number of cloud processors that must be shared by all users.

Several studies have considered multiple processors or multiple resources at the cloud and focused on optimizing different objectives. Some works look to minimize energy consumption or some form of cost [17], [18], [20], [21], [24], [25]. On the other hand, [11], [22], [23] focus on minimizing makespan or response time.

However, the sum completion time is also an important performance metric, as it represents the processing delay incurred by individual tasks. The objective of *weighted* sum completion time enhances this feature further by allowing us to express the relative priorities of the tasks. The general problem of minimizing the weighted sum completion time on a single processor has been well studied [26]. Some works in the literature have also considered the same objective to schedule tasks in a multi-processor cloud environment. In [27], the authors proposed an Ant Colony Optimization based algorithm to solve this NP-hard problem. Similarly, in [28], the authors proposed Min-Min and Min-Max heuristics for this purpose. The same objective was also considered in [29], [30] for scheduling coflows in data center networks and approximation algorithms were proposed. In [31], online algorithms are presented to minimize weighted sum completion time for the concurrent open shop, coflow, and concurrent cluster models. In [32], the authors considered this objective for scheduling tasks with release times on parallel processors, and proposed an 8-approximation algorithm. Our solution approach is inspired by [32]. However, our problem, in addition to considering a weighted sum completion time objective and *task release times*, also accounts for *multiple users*, and *per-user budget constraints*, which renders our problem more challenging than those addressed in [26], [27], [29], [32].

Some existing works also consider the expense incurred by users to utilize the resources at the cloud. A majority of these works address this problem by minimizing some form of usage cost (e.g., [35]) or maximizing revenue (e.g., [36]), while very few aim to maximize the benefit to users in a cloud computing environment under budget constraints [23], [37]. In [37], the authors considered the problem of maximizing the service quality for a multi-task application with a single budget constraint. In [23], the authors investigated the case where jobs are scheduled onto virtual machines with an objective of minimizing the response time subject to budget constraints on individual tasks. However, [23], [37] consider objectives that are different from ours and do not accommodate per-user budget constraints.

Additionally, in this work, we also address the proposed problem under more generic *task communication time* models. The existing works that consider sum completion time objective for multiprocessor environments, i.e., [26], [27], [29], [32] do not consider communication time, and cannot be easily extended to accommodate a *finite-capacity communication channel* model as considered in this work.

In [1], we had considered a model with task release times and fixed communication times. We had proposed the STUBR algorithm to minimize weighted sum completion time. In this work, we additionally consider a modified channel model to accommodate sequence-dependent communication times and corresponding problem formula-

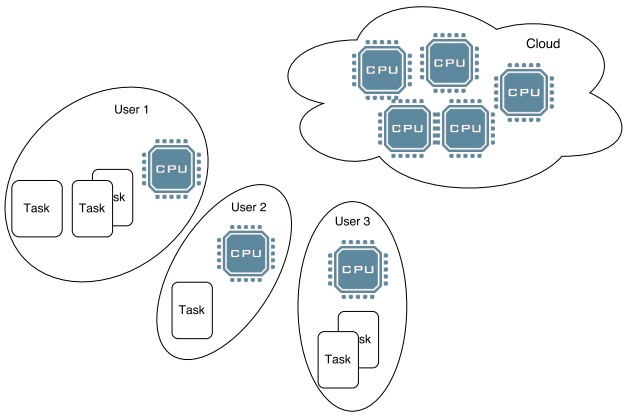


Fig. 1: Example system of 3 users and 5 cloud processors.

TABLE 1: Notations

Notation	Description
t_j	local processing time of task j
c_j	communication time of task j
t_j^R	release time of task j
w_j	weight of task j
α_{ir}	speed-up achieved by user i's tasks on processor r
β_r	cost per unit time to utilize processor r
B_i	budget of user i
$\mathcal{J}_i$	set of tasks user i wishes to execute
$\mathcal{R}$	set of all processors (local and cloud)
$\mathcal{C}$	set of cloud processors
$\mathcal{R}_i$	set of cloud processors and user i's local processor
$\mathcal{R}'$	set of machine-interval processors
N	total number of users
(τ_{l-1},τ_l)	time interval l
L	number of intervals

tion, and prove performance bounds for the same. Furthermore, we prove improved performance bounds for budget-resolved solutions for all release-time and communication time models considered, and additional simulation results to further analyze the proposed schemes.

3 System Model and Problem Formulation

In this section, we present details of the system model and problem formulation. Initially, we consider the problem of scheduling immediately available tasks to heterogeneous processors under user budget constraints. In Section 5, we extend this to the case where tasks have release times, fixed communication times, and sequence-dependent communication times.

3.1 System Model

3.1.1 Processors and Tasks

We consider a system with N user/mobile devices. Each user $i \in \{1,\ldots,N\}$ wishes to complete a set of independent tasks, denoted by $\mathcal{J}_i$. Each user has its own unary local processor, i.e., it can execute only one task at a time. This assumption is without loss of generality, as allowing multiple tasks to share a processor simultaneously will not provide any improvement to our weighted sum completion time objective. The system includes a finite-capacity cloud consisting of a number of heterogeneous processors that run at different speeds. Let $\mathcal{C}$ be the set of cloud processors. Each processor at the cloud is assumed to be unary. Similar to the local processor scenario, this unary-capacity assumption is also without loss of generality. Additionally, one may note that multiple unary-capacity processors may be combined to create finite-capacity servers at the cloud. An example of such a system is illustrated in Figure 1.

The processing time for each task $j \in \mathcal{J}_i$ is t_j on user i's local processor. The speed-up factor for each cloud processor r is $\alpha_{ir} \geq 0$, so that the processing time for task j at processor r is $\alpha_{ir}t_j$. Each user can execute its tasks either locally or remotely at one of the cloud processors. The processing times may be obtained by applying a program profiler as shown in experimental studies such as MAUI [8], Clonecloud [38], and Thinkair [39]. In this work, we proceed assuming that such information is already given. We also consider a weight w_j associated with each task, to signify the relative urgency of certain tasks with respect to the others. We assume these weights are set apriori depending on the task priorities. In the absence of such information, the weights can just be set to 1. For notation simplicity, we further define $\mathcal{R}$ as the set of all processors (including all users' local processors), $\mathcal{R}_i$ as the set of processors to which user i can offload its tasks, i.e., its own local processor and cloud processors.

3.1.2 User Budget

The users are required to pay a certain price per unit time to use the processors at the cloud, but no price to execute tasks locally on their own device. Let β_r be the cost per unit time for executing a task on processor r. Each user i has a budget B_i that determines the total expense that the user is willing to incur for offloading tasks to the cloud.

3.1.3 Release Times and Communication Times

Each task j has a release time t_j^R, i.e., the time at which the task j becomes available at the local processor. Furthermore, each task may require some input data that needs to be communicated if the task is to be executed at the cloud. The time to transmit the input data for task j to the cloud is given by c_j. We consider two different communication models:

1) **Fixed Communication Times:** The input data for each task j can be transmitted to the cloud as soon as the task is available. Hence, the communication delay for task j is simply c_j. Hence, the overall communication delay for task j is the sum of transmission times of itself and all tasks before it. This allows us to model a communication link with a large number of channels.
2) **Sequence-dependent Communication Times:** The input data for each task cannot be transmitted as soon as the task is available. We assume that the data is transmitted to the scheduled processor one task at a time, i.e., the channel to a processor is *unary*. This allows us to model a communication link with finite capacity.

3.2 Problem Formulation

For clarity of presentation, initially we consider the case where all tasks are released and available at time zero, and the links between processors are fast enough so that the communication delay between them is negligible. This leads to the problem formulation in this section and the corresponding STUBR algorithm in Section 4. We will provide details on how they are extended to the cases non-zero task release times, fixed communication times, and sequence-dependent communication times in Section 5.

We wish to identify the task scheduling decision that minimizes the weighted sum completion time of all tasks subject to user budget constraints. We formulate the proposed problem by using an interval-indexing method proposed in [32]. Towards this end, we divide the time axis into intervals $(\tau_l - 1, \tau_l)$, where $\tau_0 = 1$ and $\tau_l = 2^{l-1}$, for $l \in 1, ..., L$, where L is the smallest integer such that

$$2^{L-1} \geq \sum_j t_j.$$

This means that 2^{L-1} is a sufficiently large time horizon for the scheduling of all given tasks since it accounts for the worst-case completion time $\sum_j t_j$. The task scheduling decision determines the processors where each task should be scheduled, as well as the order of the tasks. We define decision variables $\{x_{jrl}\}$ where $x_{jrl} = 1$ if and only if task j finishes execution on processor r in time interval $l \in \{1, \ldots, L\}$. Such an approach reduces the number of variables in our formulation in comparison with a time-indexed formulation with constant-size intervals, making it computationally tractable, with a small penalty in the precision of quantifying the optimization objective. The optimization problem is defined below.

$$\min_{\{x_{jrl}\}} \sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} w_j \sum_{r \in \mathcal{R}_i} \sum_{l=1}^{L} \tau_{l-1} x_{jrl}, \tag{1}$$

$$\text{s.t.} \sum_{l=1}^{L} \sum_{r \in R_i} x_{jrl} = 1, \quad \forall i \in \{1, \ldots, N\}, j \in \mathcal{J}_i, \tag{2}$$

$$\sum_{i=1}^{N} \sum_{j \in J_i} \alpha_{ir} t_j x_{jrl} \leq \tau_l, \quad \forall r \in \mathcal{R}, l \in \{1, \ldots, L\}, \tag{3}$$

$$\sum_{j \in \mathcal{J}_i} \sum_{l=1}^{L} \sum_{r \in \mathcal{R}_i} \beta_r \alpha_{ir} t_j x_{jrl} \leq B_i, \quad \forall i \in \{1, \ldots, N\}, \tag{4}$$

$$x_{jrl} = 0, \quad \text{if } \tau_l < \alpha_{ir} t_j, \quad \forall i \in \{1, \ldots, N\}, j \in \mathcal{J}_i, r \in \mathcal{C}, l \in \{1, \ldots, L\}, \tag{5}$$

$$x_{jrl} = 0, \quad \text{if } \tau_l < t_j, \quad \forall i \in \{1, \ldots, N\}, j \in \mathcal{J}_i, r \notin \mathcal{C}, l \in \{1, \ldots, L\}, \tag{6}$$

$$x_{jrl} = 0, \quad \text{if } B_i < \beta_r \alpha_{ir} t_j, \quad \forall i \in \{1, \ldots, N\}, j \in \mathcal{J}_i, r \in \mathcal{R}, l \in \{1, \ldots, L\}, \tag{7}$$

$$x_{jrl} \in \{0, 1\}, \quad \forall i \in \{1, \ldots, N\}, j \in \mathcal{J}_i, r \in \mathcal{R}, l \in \{1, \ldots, L\}. \tag{8}$$

The objective (1) is to minimize the weighted sum completion times of tasks across all users. Constraint (2) ensures that every task is assigned to exactly one processor and one interval. Constraint (3) enforces that for each interval l, the total load on every processor r cannot exceed τ_l. Equation (4) enforces the budget constraints for each user. Equations (5)-(7) ensure that individual tasks do not exceed the τ_l interval deadline and the budget. Constraint (8) forces the decision variables to take on binary values.

Remark 1. *One may note that τ_{l-1} is a lower bound to the completion time of a task completing in interval l, and consequently, (1) is a lower bound to the weighted sum completion time. In Sections 4 and 5, we present algorithms that provide worst-case performance guarantees in terms of constant factors above this lower-bound objective. Hence, the same algorithms also have at least the same worst-case performance guarantees with respect to the optimal weighted sum completion time.*

4 The STUBR Algorithm

In this section, we present the STUBR algorithm to solve problem (1). We then prove some guarantees and properties of this algorithm, to better understand its functionality and performance.

STUBR has the following steps:

1) Relax the integer constraints in problem (1) and obtain a relaxed solution.
2) Round this solution to obtain an integer solution that gives an objective value that is no higher than 8 times the objective value achieved by the relaxed solution, and thus is also no higher than 8 times of the optimal objective value of problem (1). While this rounded solution is expected to violate the budget, we prove that the average cost over a large number of users meets the average user budget.
3) We resolve any budget violation by strategically moving some tasks to the local device.
4) To further reduce the total weighted completion time, we note that the well-known WSPT is optimal for a single processor and jobs without release times. Hence, on each processor, we reorder the tasks allocated to it by WSPT ordering.

These steps are explained in detail in the following sections.

4.1 Relaxed Solution

For each user $i \in \{1, \ldots, N\}$, $j \in \mathcal{J}_i$, and $r \in \mathcal{R}_i$, let p_{jr} and b_{jr} be the processing times and costs for scheduling task j on processor r. For our initial plain model with no release times and communication times, we have

$$p_{jr} := \begin{cases} \alpha_{ir} t_j & \text{if } r \in \mathcal{C}, \\ t_j & \text{otherwise,} \end{cases} \tag{9}$$

$$b_{jr} := \begin{cases} \beta_r \alpha_{ir} t_j & \text{if } r \in \mathcal{C}, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

Using (9) and (10), we reformulate the optimization problem in Section 3.2, and relax the integer constraints to obtain the following linear program.

$$\min_{\{x_{jrl}\}} \sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} w_j \sum_{r \in \mathcal{R}_i} \sum_{l=1}^{L} \tau_{l-1} x_{jrl}, \tag{11}$$

$$\text{s.t.} \sum_{l=1}^{L} \sum_{r \in R_i} x_{jrl} = 1, \quad \forall i \in \{1, \ldots, N\},\ j \in \mathcal{J}_i, \tag{12}$$

$$\sum_{i=1}^{N} \sum_{j \in J_i} p_{jr} x_{jrl} \leq \tau_l, \quad \forall r \in \mathcal{R}, l \in \{1, \ldots, L\}, \tag{13}$$

$$\sum_{j \in \mathcal{J}_i} \sum_{r \in \mathcal{R}_i} \sum_{l=1}^{L} b_{jr} x_{jrl} \leq B_i, \quad \forall i \in \{1, \ldots, N\}, \tag{14}$$

$$x_{jrl} = 0, \quad \text{if } \tau_l < p_{jr}, \quad \forall i \in \{1, \ldots, N\}, \quad r \in \mathcal{R}, l \in \{1, \ldots, L\}, \tag{15}$$

$$x_{jrl} = 0, \quad \text{if } B_i < b_{jr}, \quad \forall i \in \{1, \ldots, N\}, \quad j \in \mathcal{J}_i, r \in \mathcal{R}, l \in \{1, \ldots, L\}, \tag{16}$$

$$x_{jrl} \geq 0, \quad \forall i \in \{1, \ldots, N\}, \quad j \in \mathcal{J}_i, r \in \mathcal{R}, l \in \{1, \ldots, L\}. \tag{17}$$

The above linear program can be solved efficiently in polynomial-time to obtain a relaxed solution to the problem (1). This formulation also resembles the LP-relaxed version of the problem minimizing the weighted sum completion time in a system of unrelated[1] machines, i.e. $R||\sum w_j C_j$ in the standard parallel-processor-scheduling notation as formulated in [32]. However, our formulation has additional budget constraints, (14) and (16), that need to be met for each user. It also accommodates multiple users unlike the formulation in [32]. These aspects render our formulation a more complex one requiring more sophisticated techniques, for recovering an integer solution and resolving budget overage.

4.2 Rounded Solution

In [32], the authors used a method proposed in [40] for solving the generalized assignment problem, and obtained an integer solution for their problem at hand. However, our formulation renders a further constrained version of the generalized assignment problem due to the budget constraints. We therefore extend the method proposed in [40] to obtain a rounded solution to problem (1). We study the behavior of this solution and later employ techniques to improve this solution for our problem. We also provide worst-case performance and incurred cost guarantees. Additionally, we study the behavior of the average incurred cost as the number of users increases in the system.

4.2.1 Rounding Technique

We first convert the LP-solution x_{jrl} to $x_{jr'}$, where each machine-interval pair (r, l) is viewed as a single virtual processor $r' \in \mathcal{R}'$, such that $\mathcal{R}'$ is the set of machine-interval processors. This facilitates the application of the rounding method proposed in [40] to our problem.

The rounding technique lists the tasks in non-increasing order of $p_{jr'}$, for $r' \in \mathcal{R}'$, and constructs a bipartite fractional matching. A fractional matching between task nodes and machine nodes assigns each task node partially to multiple machine nodes, and all allocated fractions for a particular task node should sum up to 1. Let $f(v_{r's}, u_j)$ denote the fractional matching between task nodes u_j, for $j \in \mathcal{J}_i$, $i \in \{1, \ldots, N\}$, and machine nodes $v_{r's}$, for $r' \in \mathcal{R}'$, $s \in \{1, \ldots, k_{r'}\}$, and $k_{r'} = \lceil \sum_j x_{jr'} \rceil$. This is constructed in accordance with the following:

$$x_{jr'} = \sum_{s:(v_{r's}, u_j) \in \mathcal{E}} f(v_{r's}, u_j), \quad \forall r' \in \mathcal{R}', j \in \mathcal{J}_i, i = \{1, \ldots, N\}, \tag{18}$$

$$\sum_{j:(v_{r's}, u_j) \in \mathcal{E}} f(v_{r's}, u_j) = 1, \quad \forall r' \in \mathcal{R}', s = \{1, \ldots, (k_{r'} - 1)\}, \tag{19}$$

where $\mathcal{E}$ is the set of edges of the bipartite graph. This fractional matching is then converted to a minimum cost integer matching where each task is assigned to a single machine node.

For our problem, this would be equivalent to a weighted sum completion time integer matching. We call this integer solution $\overline{\mathbf{x}} = \{\overline{x_{jrl}}\}$. This integer matching solution, however, is likely to violate the interval deadline τ_l constraints as well as the user budget constraints, since the relaxed solution that meets these constraints has been rounded. We now analyze the extent to which these constraints could be violated, and the resulting performance guarantee.

4.2.2 Interval Deadline Violation and Performance Guarantee

Lemma 1. *With the rounded solution, the total processing time of all tasks for every $r' \in \mathcal{R}' = (r, l)$, and interval $l \in \{1, \ldots, L\}$ cannot be worse than $2\tau_l$, i.e., constraint (13) is violated by at most τ_l.*

Proof. For each machine node $v_{r's}$, let the maximum possible processing time be

$$p_{r's}^{\max} = \max_{j:(v_{r's}, u_j) \in \mathcal{E}} p_{jr'}, \tag{20}$$

and minimum possible processing time be

$$p_{r's}^{\min} = \min_{j:(v_{r's}, u_j) \in \mathcal{E}} p_{jr'}. \tag{21}$$

Consequently, $p_{r's}^{\min} \geq p_{r'(s+1)}^{\max}$, since tasks are allocated in non-increasing order of $p_{jr'}$ while constructing the fractional bipartite matching. Along the lines of the proof in [40], we have for each $r' \in \mathcal{R}'$,

$$\sum_{s=2}^{k_{r'}} p_{r's}^{\max} \leq \sum_{s=1}^{k_{r'}-1} p_{r's}^{\min} \tag{22}$$

$$\leq \sum_{s=1}^{k_{r'}-1} \sum_{j:(v_{r's}, u_j) \in \mathcal{E}} p_{jr'} f(v_{r's}, u_j) \tag{23}$$

$$\leq \sum_{s=1}^{k_{r'}} \sum_{j:(v_{r's}, u_j) \in \mathcal{E}} p_{jr'} f(v_{r's}, u_j) \tag{24}$$

$$= \sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} p_{jr'} x_{jr'} \leq \tau_l. \tag{25}$$

Furthermore, $p_{r'1}^{\max} \leq \tau_l, \forall r'$, from (15). Hence, we have

$$\sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} p_{jr} \overline{x_{jrl}} = \sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} p_{jr'} \overline{x_{jr'}} \tag{26}$$

1. In the mode of unrelated machines, the processing times of a task on any two machines are independent.

$$\leq \sum_{s=1}^{k_{r'}} p_{r's}^{\max} \leq 2\tau_l, \quad \forall r \in \mathcal{R}, l = \{1, \ldots, L\}. \tag{27}$$

□

We derive an approximation ratio for the integer matching $\bar{\mathbf{x}}$ which is presented in the following theorem.

Theorem 2. *The objective value of the rounded solution obtained from the integer matching $\overline{\mathbf{x}}$ cannot be worse than 8 times the optimal objective of problem (1).*

Proof. We define new intervals $\overline{\tau_l} := 2^{l+1}$, $\forall l = \{1, \ldots, L\}$. From Lemma 1, we can see every task j that was scheduled in the lth interval will be completed by time $\bar{\tau}_l$. This is because $\overline{\tau_l} - \overline{\tau_{l-1}} \leq 2\tau_l$, and from Lemma 1, we know that the total processing time for the tasks assigned to the lth interval does not exceed $2\tau_l$. Let the contribution to the objective by task j be given by O_j^{relax} and O_j^{round} in the relaxed solution and the rounded solution, respectively. If task j is scheduled to complete in interval l, we have

$$O_j^{\text{relax}} = w_j \tau_{l-1} \tag{28}$$
$$= w_j 2^{l-2}. \tag{29}$$

Similarly, for the rounded solution, we have

$$O_j^{\text{round}} \leq w_j \overline{\tau_l} \tag{30}$$
$$\leq w_j 2^{l+1} \tag{31}$$
$$\leq w_j 2^{l-2} 2^3 \tag{32}$$
$$\leq 8 O_j^{\text{relax}}. \tag{33}$$

This implies that

$$\sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} O_j^{\text{round}} \leq \sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} 8 O_j^{\text{relax}}. \tag{34}$$

We see that the rounded objective value is at most 8 times the relaxed solution, and hence, at most 8 times the optimal objective of problem (1) since the relaxed solution by definition returns an objective value that is below the optimal objective. □

4.2.3 Multiple Users and Incurred Cost Guarantees

Since our problem also accounts for multiple users and budget constraints, we wish to evaluate the performance of this rounded solution with respect to these parameters.

Theorem 3. *With the rounded solution, the sum of the incurred cost of all users cannot be worse than $(|\mathcal{R}'| + 1)$ times the sum of user budgets.*

Proof. Let $b_{r's}^{\max}$ and $b_{r's}^{\min}$ be the maximum and minimum possible costs at machine node (r',s), respectively. For each processor r', we have $p_{r's}^{\min} \geq p_{r'(s+1)}^{\max}$, as explained in 1. Consequently, we have $b_{r's}^{\min} \geq b_{r'(s+1)}^{\max}$ for our model from (9) and (10). Then, we have

$$\sum_{i=1}^{N} \sum_{r' \in \mathcal{R}'} \sum_{s=2}^{k_{r'}} b_{r's}^{\max} \leq \sum_{i=1}^{N} \sum_{r' \in \mathcal{R}'} \sum_{s=1}^{k_{r'}-1} b_{r's}^{\min} \tag{35}$$
$$\leq \sum_{i=1}^{N} \sum_{r' \in \mathcal{R}'} \sum_{s=1}^{k_{r'}-1} \sum_{j:(v_{r's}, u_j) \in \mathcal{E}} b_{jr'} f(v_{r's}, u_j) \tag{36}$$
$$\leq \sum_{i=1}^{N} \sum_{r' \in \mathcal{R}'} \sum_{s=1}^{k_{r'}} \sum_{j:(v_{r's}, u_j) \in \mathcal{E}} b_{jr'} f(v_{r's}, u_j) \tag{37}$$
$$= \sum_{i=1}^{N} \sum_{r' \in \mathcal{R}'} \sum_{j \in \mathcal{J}_i} b_{jr'} x_{jr'} \leq \sum_{i=1}^{N} B_i. \tag{38}$$

Hence, if we take out the tasks allocated to machine nodes $v_{r'1}$ for every $r' \in \mathcal{R}'$, the remaining tasks have a sum cost that is less than the sum of user budgets. There are at most $|R'|$ such tasks. Furthermore, for each $r' \in \mathcal{R}'$ and j such that ($v_{r'1}$,u_j) $\in \mathcal{E}$, we know from (16) that

$$b_{jr'} \leq b_{r'1}^{\max} \leq \sum_{i=1}^{N} B_i. \tag{39}$$

Hence we have

$$\sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} \sum_{r \in \mathcal{R}} \sum_{l=1}^{L} b_{jr} \overline{x_{jrl}} = \sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} \sum_{r' \in \mathcal{R}'} \sum_{l=1}^{L} b_{jr'} \overline{x_{jr'}} \tag{40}$$
$$\leq \sum_{i=1}^{N} \sum_{r' \in \mathcal{R}'} \sum_{s=1}^{k_{r'}} b_{r's}^{\max} \leq (|\mathcal{R}'| + 1) \sum_{i=1}^{N} B_i. \tag{41}$$

□

Remark 2. *We see from the above at most one task on each interval-processor violates the sum of the user budgets. Consequently, we can find a subset of at most $|\mathcal{R}'|$ tasks that violate the sum of the user budgets.*

The following conclusions follow directly from Theorem 3.

Corollary 4. *If b_{jr} is independent of task j, let $b_r = b_{jr}$. We further define $S = \{r \in R' : \exists\, j, x_{jr} = 1\}$. Then, we have*

$$\sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} \sum_{r \in \mathcal{R}} \sum_{l=1}^{L} b_r \overline{x_{jrl}} \leq \sum_{i=1}^{N} B_i + \sum_{r \in \mathcal{S}} b_r. \tag{42}$$

Corollary 5. *If C_i is the incurred cost for user i,*

$$\frac{1}{N} \sum_{i=1}^{N} C_i \leq \frac{1}{N} \sum_{i=1}^{N} B_i + \frac{1}{N} |\mathcal{R}'| B^{max}, \tag{43}$$

where $B^{max} = \max_i B_i$, and for the specific case from Corollary 4,

$$\frac{1}{N} \sum_{i=1}^{N} C_i \leq \frac{1}{N} \sum_{i=1}^{N} B_i + \frac{1}{N} \sum_{r \in \mathcal{S}} b_r. \tag{44}$$

If we increase the number of users N, the total processing time increases, and consequently, the number of intervals L increases. But we note that since the interval size increases exponentially, the number of intervals L only increases logarithmically. Additionally, the number of processors $|\mathcal{R}|$ is fixed. This implies that $|\mathcal{R}'| = L|\mathcal{R}|$ increases more slowly in comparison with N. Hence, we can see that as $N \to \infty$, the second term on the right-hand side of (43) approaches zero, leading to Corollary 6.

Corollary 6. *As $N \to \infty$, the average cost incurred across all users meets the average budget.*

Thus, the average user cost performance improves as the number of users in the system increases. This property indicates that the proposed algorithm is highly scalable and is a suitable choice for multi-user systems.

4.3 Dealing with Budget Violation

Even if the budget constraints are met on average, the budget constraints for each individual user could still be violated. In cases where the users expect strict budget constraints, we need to identify a technique by which this rounded solution can be modified to ensure that each user's budget is met, while not significantly affecting the weighted sum completion time. Since there is no budget constraint on executing tasks on a user's local device, we propose the following technique to move certain tasks to the local device in the event of a budget violation:

1) Check if budget is violated for user i.
2) If so, sort all its offloaded tasks, $\{j \in \mathcal{J}_i \mid \overline{x_{jrl}} = 1, \forall r \in \mathcal{C}\}$, in non-decreasing order of $w_j t_j$. We do this as we expect a task with smaller weight and smaller local processing time does lesser damage to the weighted sum completion time objective when transferred to the local device.
3) Start with the first task (with least $w_j t_j$) and schedule it on the local device. Update the incurred cost of user i by subtracting the previously incurred cost of this task.
4) If incurred cost now meets the budget, stop. If not, repeat Steps 2 and 3 until the budget is met.
5) Repeat for all users.
6) Once every user meets its budget, we apply our modified WSPT (presented in Section 4.4) on all processors.

Now we wish to understand the impact of moving tasks to the local device to meet the budget on the performance.

Theorem 7. *The objective value of the final solution is at most $2^{\lceil \log_2(2+\frac{1}{a})\rceil+2}$ times the optimal solution, where $a = \min_{i,r} \alpha_{ir}$ is the minimum value of speed-up factor in the system.*

Proof. We know, from Remark 2, that for every interval l, at most one task from every cloud processor needs to be moved to the local device, and this task has a maximum processing time of τ_l. We also know, from Lemma 1, that total processing time on a local processor for the tasks assigned to the lth interval does not exceed $2\tau_l$. Furthermore, from (15), the processing time of a task scheduled to finish in interval l cannot exceed τ_l. Thus, after moving a task belonging to user i from cloud processor r to user i's local processor, the total processing time on the local processor will be $(2 + \frac{1}{\alpha_{ir}})\tau_l$, in the worst case. Since this task that we move back may belong to any user, this value will be at most $(2 + \frac{1}{a})\tau_l$, as $a = \min_{i,r} \alpha_{ir}$ is the minimum value of speed-up factor. In other words, we now have

$$\sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} p_{jr} \overline{x_{jrl}} \leq (2+\frac{1}{a})\tau_l, \quad \forall r \in \mathcal{R}, l = \{1, \ldots, L\}. \quad (45)$$

We need to redefine $\overline{\tau_l}$ defined in Theorem 2 such that every task that is assigned to the lth interval may be run entirely within the interval $(\overline{\tau_{l-1}}, \overline{\tau_l})$. In other words, we need

$$\overline{\tau_l} - \overline{\tau_{l-1}} \leq (2 + \frac{1}{a})\tau_l. \quad (46)$$

Towards this end, we set $x = \log_2\left(2 + \frac{1}{a}\right) + 1$, and $\overline{\tau_l} = 2^x \tau_l = 2^x 2^{l-1} = 2^{x+l-1}$.

We now get, for every task j,

$$\frac{O_j^{\text{round}}}{O_j^{\text{relax}}} \leq w_j \frac{\overline{\tau_l}}{\tau_{l-1}} \leq 2^{x+1} \leq 2^{\log_2\left(2+\frac{1}{a}\right)+2} \quad (47)$$

Thus, the objective value of the final solution is at most $2^{\lceil \log_2(2+\frac{1}{a})\rceil+2}$ times the optimal solution. □

4.4 WSPT Ordering

From the above, we obtain a scheduling decision for every task that specifies on which processor the task should be executed. Some processors will be assigned multiple tasks. We know that the WSPT ordering is optimal for the weighted sum completion time objective for a single processor and jobs without release times [41]. Thus, we perform a WSPT ordering on the tasks allocated to a particular processor to further improve our objective value as follows:

1) **Step 1:** Obtain the task scheduling decision, i.e., the processor on which each task should be scheduled.
2) **Step 2:** On each processor $r \in \mathcal{R}$, order the scheduled tasks in the non-decreasing order of $\frac{p_{jr}}{w_j}$. This ensures that tasks with smaller weights and longer completion times (without accounting for wait times) are scheduled earlier.
3) **Step 3:** Modify the task completion times correspondingly, and obtain the new objective value.

The STUBR pseudocode is outlined in Algorithm 1 below.

Algorithm 1 STUBR algorithm pseudocode

Input: Processing times t_j for every task j, speed-up factor α_r and cost β_r for every processor r, budgets B_i for every user i.
Output: Scheduling decision for every task j.

Solve (11) to obtain relaxed solution $\{x_{jrl}\}$ for every task j on processor r in interval l.
Set $(r, l) \rightarrow r'$, for every r, l.
Construct bipartite fractional matching $f(v_{r's}, u_j)$ between task nodes u_j and machine nodes $v_{r's}$, for $r' \in \mathcal{R}'$, $s \in \{1, \ldots, k_{r'}\}$, and $k_{r'} = \lceil \sum_j x_{jr'} \rceil$, in accordance with conditions (18) and (19).
Convert fractional matching to integer matching solution $\overline{\mathbf{x}} = \{\overline{x_{jrl}}\}$ using technique in [40]. This solution is at most 8 times the optimal.
for all user i **do**
  if budget is violated **then**
    Set offloaded tasks in non-decreasing order of $w_j t_j \rightarrow \mathcal{O}$.
    for all task $o \in \mathcal{O}$ **do**
      Schedule task o back on the user's local device and modify $x_{or'}$, for all r' accordingly.
      Update incurred cost to user.
      if incurred Cost $< B_i$ **then**
        break
      end if
    end for
  end if
end for
for all processor r **do**
  Order the scheduled tasks in the non-decreasing order of $\frac{p_{jr}}{w_j}$.
end for

4.5 Feasibility and Complexity Analysis

It can be readily noted that the STUBR algorithm provides a feasible solution. In other words, the user budgets are

TABLE 2: STUBR Steps and Solutions

Step	Technique	Resulting Solution
1	Relax the integer constraints in the interval-indexed formulation	Lower bound to optimum
2	Round the fractional values by applying technique from [40]	(1) Constant-factor approximation; (2) Violates budget constraints; (3) Average budget violation decreases with increase in number of users.
3	Budget resolution by moving some tasks to local devices	Approximation; Dependent on minimum value of speed-up factor
4	Re-order tasks on each processor using WSPT or modified WSPT (Skip for sequence-dependent communication times)	Heuristic with improved performance

always met, and all the tasks are always scheduled. Thus, in the worst case with extremely tight budgets, the algorithm will execute all tasks locally.

The time complexity of STUBR is dominated by the LP-solving step (in Section 4.1) and the rounding step (in Section 4.2) that involves finding the weighted sum completion time bipartite matching. An LP can be solved in $O(n^{3.5})$ time where n is the number of variables [42]. For our problem, this would imply that the time complexity for solving the LP is $O((P|\mathcal{R}|L)^{3.5})$, where $P = \sum_{i=1}^{N} |\mathcal{J}_i|$ is the total number of tasks. On the other hand, bipartite matching can be solved in cubic time in the number of vertices by utilizing the Hungarian algorithm, proposed in [43]. If $P > |\mathcal{R}|$, the time complexity of this step is $O(P^3)$. Thus, we see that the overall worst-case time complexity of STUBR is $O((P^2L)^{3.5})$.

We present the steps and the resulting solution following each step in Table 2.

5 STUBR Extensions

In this section, we consider the models with release times, fixed communication times, and sequence-dependent communication times, introduced in Section 3.1.3.

5.1 With Task Release Times

STUBR can also be applied to solve the problem of scheduling tasks with release times. We reformulate problem (11) to incorporate release times t_j^R for every task j as follows.

$$\min_{\{x_{jrl}\}} \sum_{i=1}^{N} \sum_{j\in\mathcal{J}_i} w_j \sum_{r\in\mathcal{R}_i} \sum_{l=1}^{L} \tau_{l-1} x_{jrl}, \tag{48}$$

$$\text{s.t.} \sum_{l=1}^{L} \sum_{r\in R_i} x_{jrl} = 1, \quad \forall i \in \{1,\dots,N\},\ j \in \mathcal{J}_i, \tag{49}$$

$$\sum_{i=1}^{N} \sum_{j\in J_i} p_{jr} x_{jrl} \le \tau_l, \quad \forall r \in \mathcal{R}, l \in \{1,\dots,L\}, \tag{50}$$

$$\sum_{j\in\mathcal{J}_i} \sum_{r\in\mathcal{R}_i} \sum_{l=1}^{L} b_{jr} x_{jrl} \le B_i, \quad \forall i \in \{1,\dots,N\}, \tag{51}$$

$$x_{jrl} = 0, \quad \text{if } \tau_l < t_j^R + p_{jr}, \quad \forall i \in \{1,\dots,N\}, \; r \in \mathcal{R}, l \in \{1,\dots,L\}, \tag{52}$$

$$x_{jrl} = 0, \quad \text{if } B_i < b_{jr}, \quad \forall i \in \{1,\dots,N\}, \; j \in \mathcal{J}_i, r \in \mathcal{R}, l \in \{1,\dots,L\}, \tag{53}$$

$$x_{jrl} \ge 0, \quad \forall i \in \{1,\dots,N\}, \; j \in \mathcal{J}_i, r \in \mathcal{R}, l \in \{1,\dots,L\}. \tag{54}$$

On applying the same rounding method proposed in Section 4.2, we can easily see that Lemma 1 is satisfied for this case as well. Additionally, we can also extend the results in Theorem 2 to prove the following.

Theorem 8. *The objective value of the rounded solution obtained from the integer matching $\overline{\mathbf{x}}$ cannot be worse than 8 times the optimal objective of problem (48).*

Proof. Every task that is assigned to the lth interval may be run entirely within the interval $(\overline{\tau_{l-1}}, \overline{\tau_l})$, where $\overline{\tau_l} := 2^{(l+1)}$. This is because $\overline{\tau_l} - \overline{\tau_{l-1}} \le 2\tau_l$, and from Lemma 1, we know that the total processing time for the tasks assigned to the lth interval does not exceed $2\tau_l$. Additionally, every task j that is assigned to the lth interval will have been released by $\overline{\tau_{l-1}}$ because $\overline{\tau_{l-1}} > \tau_l > t_j^R + p_{jr} > t_j^R$. Thus, similar to Theorem 2, we see that the rounded objective value is at most 8 times the relaxed solution, and hence, at most 8 times the optimal objective of problem (48) since the relaxed solution by definition returns an objective value that is below the optimal objective. □

We can also see that Theorem 3 and the corresponding corollaries are satisfied for this case. We apply the budget resolution technique proposed in Section 4.3 and can easily see that Theorem 7 can be proved for this case as well. Additionally, we also apply a modified WSPT similar to that in Section 4.4 that we call m-WSPT ordering, by accommodating task release times. We do this by scheduling tasks in the non-decreasing order of $\frac{t_j^R + p_{jr}}{w_j}$ in Step 2.

5.2 With Fixed Communication Times

We can further extend the solution in Section 5.1 to the case where every task j has release time t_j^R and communication time c_j. This is equivalent to defining a new *per processor* release time t_{jr}^R, to be the release times for scheduling task j on processor r, as follows:

$$t_{jr}^R := \begin{cases} t_j^R + c_j & \text{if } r \in \mathcal{C}, \\ t_j^R & \text{otherwise,} \end{cases} \tag{55}$$

Thus, the new version of problem (11) becomes

$$\min_{\{x_{jrl}\}} \sum_{i=1}^{N} \sum_{j\in\mathcal{J}_i} w_j \sum_{r\in\mathcal{R}_i} \sum_{l=1}^{L} \tau_{l-1} x_{jrl}, \tag{56}$$

$$\text{s.t.} \sum_{l=1}^{L} \sum_{r\in R_i} x_{jrl} = 1, \quad \forall i \in \{1,\dots,N\},\ j \in \mathcal{J}_i, \tag{57}$$

$$\sum_{i=1}^{N}\sum_{j\in J_i} p_{jr}x_{jrl} \le \tau_l, \quad \forall r \in \mathcal{R}, l \in \{1,\dots,L\}, \quad (58)$$

$$\sum_{j\in\mathcal{J}_i}\sum_{r\in\mathcal{R}_i}\sum_{l=1}^{L} b_{jr}x_{jrl} \le B_i, \quad \forall i \in \{1,\dots,N\}, \quad (59)$$

$$x_{jrl} = 0, \quad \text{if } \tau_l < t_{jr}^R + p_{jr}, \quad \forall i \in \{1,\dots,N\}, \; r \in \mathcal{R}, l \in \{1,\dots,L\}, \quad (60)$$

$$x_{jrl} = 0, \quad \text{if } B_i < b_{jr}, \quad \forall i \in \{1,\dots,N\}, \; j \in \mathcal{J}_i, r \in \mathcal{R}, l \in \{1,\dots,L\}, \quad (61)$$

$$x_{jrl} \ge 0, \quad \forall i \in \{1,\dots,N\}, \; j \in \mathcal{J}_i, r \in \mathcal{R}, l \in \{1,\dots,L\}. \quad (62)$$

We can see that the fixed communication times are incorporated in constraint (60). On applying the same rounding method proposed in Section 4.2, we can easily see that Lemma 1 is satisfied for this case as well. Additionally, we can also extend the results in Theorem 8 to prove the following.

Theorem 9. *The objective value of the rounded solution obtained from the integer matching $\overline{\mathbf{x}}$ cannot be worse than 8 times the optimal objective of problem (56).*

Proof. The proof is similar to the proof of Theorem 8, except that we note that every task j that is assigned to the lth interval, i.e., $(\overline{\tau_{l-1}}, \overline{\tau_l})$ will have been released by $\overline{\tau_{l-1}}$ because $\overline{\tau_{l-1}} > \tau_l > t_{jr}^R + p_{jr} > t_{jr}^R$. □

We see that Theorem 3, the corresponding corollaries, as well as Theorem 7 can be proved for this case as well. Additionally, we can accommodate both task release times and communication times by scheduling tasks in the non-decreasing order of $\frac{t_{jr}^R + p_{jr}}{w_j}$ in Step 2 of m-WSPT proposed in Section 4.4.

5.3 With Sequence-dependent Communication Times

5.3.1 Modified Channel Model and Problem Formulation

In a more practical model of a communication channel with finite channel capacity, the input data is communicated to the scheduled processor one task at a time. To extend STUBR to this more complicated scenario, we introduce the following new decision variables:

$$x_{jrpl} := \begin{cases} 1 & \text{task } j \text{ is communicated at interval } p \\ & \text{to processor } r \text{ and executed at interval } l, \\ 0 & \text{otherwise,} \end{cases} \quad (63)$$

We define communication times for a task j on a processor r as

$$c_{jr} := \begin{cases} c_j & \text{if } r \in \mathcal{C}, \\ 0 & \text{otherwise,} \end{cases} \quad (64)$$

It should be noted that, under this channel model, the release time of a task j at the local device is t_j^R, but the release time of the task at a cloud processor is now determined by the sequence in which tasks are communicated to this processor.

5.3.2 Relaxed Solution

Incorporating (63), (64), and the consideration of sequence-dependent communication times into the first step of STUBR, we first solve the following optimization problem to obtain an LP-relaxed solution.

$$\min_{\{x_{jrpl}\}} \sum_{i=1}^{N}\sum_{j\in\mathcal{J}_i} w_j \sum_{r\in\mathcal{R}_i}\sum_{l=1}^{L}\tau_{l-1}\sum_{p=1}^{L} x_{jrpl}, \quad (65)$$

$$\text{s.t.} \sum_{l=1}^{L}\sum_{r\in R_i}\sum_{p=1}^{l} x_{jrpl} = 1, \quad \forall i \in \{1,\dots,N\}, j \in \mathcal{J}_i, \quad (66)$$

$$\sum_{i=1}^{N}\sum_{j\in J_i}\sum_{p=1}^{l} p_{jr}x_{jrpl} \le \tau_l, \quad \forall r \in \mathcal{R}, l \in \{1,\dots,L\}, \quad (67)$$

$$\sum_{i=1}^{N}\sum_{j\in J_i}\sum_{l=p}^{L} c_j x_{jrpl} \le \tau_p, \quad \forall r \in \mathcal{C}, p \in \{1,\dots,L\}, \quad (68)$$

$$\sum_{j\in\mathcal{J}_i}\sum_{r\in\mathcal{R}_i}\sum_{l=1}^{L}\sum_{p=1}^{L} b_{jr}x_{jrl} \le B_i, \quad \forall i \in \{1,\dots,N\}, \quad (69)$$

$$\sum_{l=1}^{L} x_{jrpl} = 0, \quad \text{if } \tau_p < t_j^R + c_j, \forall i \in \{1,\dots,N\}, \; j \in \mathcal{J}_i, r \in \mathcal{C}, p \in \{1,\dots,L\}, \quad (70)$$

$$\sum_{p=1}^{L} x_{jrpl} = 0, \quad \text{if } \tau_l < t_j^R + p_{jr}, \; \forall i \in \{1,\dots,N\}, j \in \mathcal{J}_i, r \notin \mathcal{C}, l \in \{1,\dots,L\}, \quad (71)$$

$$x_{jrpl} = 0, \quad \text{if } \tau_l < \tau_{p-1} + p_{jr} \text{ and } \tau_p > t_j^R + c_j, \; \forall i \in \{1,\dots,N\}, j \in \mathcal{J}_i, r \in \mathcal{C}, \; p \in \{1,\dots,L\}, l \in \{1,\dots,L\}, \quad (72)$$

$$x_{jrpl} = 0, \quad \text{if } B_i < b_{jr}, \quad \forall i \in \{1,\dots,N\}, \; j \in \mathcal{J}_i, r \in \mathcal{R}, l \in \{1,\dots,L\}, p \in \{1,\dots,L\}, \quad (73)$$

$$x_{jrpl} \ge 0, \quad \forall i \in \{1,\dots,N\}, \; j \in \mathcal{J}_i, r \in \mathcal{R}, l \in \{1,\dots,L\}, \quad (74)$$

Constraint (96) enforces that for each interval p the total load on the channel cannot exceed τ_p. Equations (98) and (99) ensure that individual tasks do not exceed the τ_l and τ_p separately. Constraint (100) ensures that a task cannot be communicated in p and executed in l even if the task can be communicated by τ_p but it cannot be executed by τ_l.

5.3.3 Rounded Solution

We convert the LP-solution $x_{jrpl}, \forall j, r$ to

$$y_{jr'} = \sum_{p=1}^{L} x_{jrpl} \quad (75)$$

where each (r, l) pair is viewed as a single virtual processor r', and

$$z_{j\hat{r}} = \sum_{l=1}^{L} x_{jrpl} \quad (76)$$

where each (r, p) pair is viewed as a single virtual processor $\hat{r}$. We then apply the rounding technique proposed in Section 4.2 to both $y_{jr'}$ and $z_{j\hat{r}}$ separately.

Lemma 10. *With the rounded solution, the total processing time of all tasks for every $r' \in \mathcal{R}'$ and interval $l \in \{1, \ldots, L\}$ cannot be worse than $2\tau_l$, i.e., constraint (95) is violated by at most τ_l.*

Proof. From inequality (24) of Lemma 1 and using constraint (95), we can see that for each $r' \in \mathcal{R}'$,

$$\sum_{s=2}^{k_{r'}} p_{r's}^{\max} \leq \sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} p_{jr'} y_{jr'} \tag{77}$$

$$\leq \sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} \sum_{p=1}^{L} p_{jr'} y_{jr'} \leq \tau_l. \tag{78}$$

Furthermore, $p_{r'1}^{\max} \leq \tau_l, \forall r'$, from (99), (100) and (102). Hence, we have

$$\sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} \sum_{p=1}^{L} p_{jr} \overline{x_{jrpl}} \leq 2\tau_l, \quad \forall r \in \mathcal{R}, l = \{1, \ldots, L\}. \tag{79}$$

□

Lemma 11. *With the rounded solution, the total communication time of all tasks for every $\hat{r} \in \hat{\mathcal{R}}$ and interval $p \in \{1, \ldots, L\}$ cannot be worse than $2\tau_p$, i.e., constraint (96) is violated by at most τ_p.*

Proof. For each machine node $v_{r's}$, let the maximum possible communication time be

$$c_{\hat{r}s}^{\max} = \max_{j:(v_{\hat{r}s}, u_j) \in \mathcal{E}} c_j, \tag{80}$$

and minimum possible communication time be

$$c_{\hat{r}s}^{\min} = \min_{j:(v_{\hat{r}s}, u_j) \in \mathcal{E}} c_j. \tag{81}$$

From inequality (24) of Lemma 1 and using constraint (96), we can see that for each $\hat{r} \in \hat{\mathcal{R}}$,

$$\sum_{s=2}^{k_{\hat{r}}} p_{\hat{r}s}^{\max} \leq \sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} c_{j\hat{r}} z_{j\hat{r}} \tag{82}$$

$$\leq \sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} \sum_{l=1}^{L} c_{j\hat{r}} z_{j\hat{r}} \leq \tau_p. \tag{83}$$

Furthermore, $c_{\hat{r}1}^{\max} \leq \tau_p, \forall \hat{r}$, from (98) and (102). Hence, we have

$$\sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} \sum_{l=1}^{L} c_j \overline{x_{jrpl}} \leq 2\tau_p, \quad \forall r \in \mathcal{R}, p = \{1, \ldots, L\}. \tag{84}$$

□

Theorem 12. *The objective value of the rounded solution obtained from the integer matching $\overline{\mathbf{x}}$ cannot be worse than 32 times the optimal objective of problem (93).*

Proof. We define new communication intervals $\overline{\tau_p} := 2^{(p+1)} = 4\tau_p$, similar to Theorem 2. We can easily show using Lemma 11 that every task j assigned to finish communication in the pth interval may be run entirely within interval $(\overline{\tau_{p-1}}, \overline{\tau_p})$. Thus, the actual release time of these tasks at the cloud processors is $\overline{\tau_p}$.

We also define new execution intervals $\overline{\tau_l} := 2^{(l+3)}$. We can easily show using Lemma 10 that every task j assigned to finish communication in the lth interval may be run entirely within interval $(\overline{\tau_{l-1}}, \overline{\tau_l})$. Every task j that is assigned to the lth interval will have been released by $\overline{\tau_{l-1}}$ because

$$\overline{\tau_{l-1}} = 8\tau_l > 8\tau_{p-1} + 4p_{jr} > 8\tau_{p-1} > 4\tau_p \tag{85}$$

If task j is scheduled to complete in interval l, we have

$$O_j^{\text{relax}} = w_j 2^{l-2} \tag{86}$$

Similarly, for the rounded solution, we have

$$O_j^{\text{round}} \leq w_j 2^{l+3} \tag{87}$$

$$\leq w_j 2^{l-2} 2^5 \tag{88}$$

$$\leq 32 O_j^{\text{relax}}. \tag{89}$$

This implies that

$$\sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} O_j^{\text{round}} \leq \sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} 32 O_j^{\text{relax}}. \tag{90}$$

Thus, we see that the rounded objective value is at most 32 times the relaxed solution, and hence, at most 32 times the optimal objective of problem (56) since the relaxed solution by definition returns an objective value that is below the optimal objective. □

Theorem 3 and the corresponding corollaries on budget violation can be trivially extended for this case since the costs are only dependent on task processing times.

5.3.4 Dealing with Budget Violation

We apply the same budget resolution technique proposed in Section 4.3 in order to ensure that individual user budgets are met. Consequently, along the lines of Theorem 7, we can prove the following.

Theorem 13. *The objective value of the final solution is at most $2^{\lceil \log_2(2+\frac{1}{a}) \rceil + 3}$ times the optimal solution, where $a = \min_{i,r} \alpha_{ir}$ is the minimum value of speed-up factor in the system.*

Proof. Similar to Theorem 7, we can show that after moving a task belonging to user i back to the local device, the local total processing time will be at most $(2 + \frac{1}{a})\tau_l$, where $a = \min_{i,r} \alpha_{ir}$ is the minimum value of speed-up factor. In other words, we now have, using (95),

$$\sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} \sum_{p=1}^{L} p_{jr} \overline{x_{jrpl}} \leq (2 + \frac{1}{a})\tau_l, \quad \forall r \in \mathcal{R}, l = \{1, \ldots, L\}. \tag{91}$$

We need to redefine $\overline{\tau_l}$ defined in Theorem 12 such that every task that is assigned to the lth interval is available for processing by $\overline{\tau_{l-1}}$ and may be run entirely within the interval $(\overline{\tau_{l-1}}, \overline{\tau_l})$. Towards this end, we set $x = \log_2\left(2 + \frac{1}{a}\right) + 1$, and $\overline{\tau_l} = 2^{x+l}$.

We now get, for every task j,

$$\frac{O_j^{\text{round}}}{O_j^{\text{relax}}} \leq w_j \frac{\overline{\tau_l}}{\tau_{l-1}} \leq 2^{x+2} \leq 2^{\log_2\left(2+\frac{1}{a}\right)+3}. \tag{92}$$

Thus, the objective value of the final solution is at most $2^{\lceil \log_2(2+\frac{1}{a}) \rceil + 3}$ times the optimal solution. □

Remark 3. *Even though the proven worst-case bounds look large, from the trace-driven simulation results in Section 6, we see that the performance of STUBR in practice is no worse than 4 times the relaxed solution and consequently the optimal, for all models and cases considered.*

We do not apply the modified WSPT ordering here as the release times on processors depend on the sequence of tasks transmitted, and we cannot trivially extend this technique to improve performance for this case.

For all of the extensions described in this section, using similar arguments as in Section 4.5, it can be verified that the STUBR algorithm always provides a feasible solution and has worst-case time complexity $O((P^2L)^{3.5})$.

5.4 Online Implementation

We next address the online problem where tasks arrive randomly over time. In other words, the release times of the tasks are not known in advance, and we assume that we know the processing and communication times, i.e., t_j and c_j of a task j only once it has been released. Toward this goal, we will modify the offline STUBR algorithm and use it as a recurring component of our online solution. We use the case of sequence-dependent communication times as example, but the proposed method can be modified to suit other scenarios.

5.4.1 STUBR Subroutine

The online implementation is an iterative algorithm that runs an offline subroutine at each iteration l, where each iteration corresponds to each computation interval. The offline subroutine tries to identify the subset of tasks that maximizes the total weight of the released but unscheduled tasks. For each user $i \in \{1,\dots,N\}$ and iteration l, let $\mathcal{T}_{il}$ be the released but unscheduled tasks and $B_{il} \leq B_i$ be the budget allocated. The following optimization gives us an LP-relaxed solution to this problem for a particular interval l:

$$\min_{\{x_{jrps}\}} \sum_{i=1}^{N} \sum_{j\in\mathcal{T}_{il}} w_j \sum_{r\in\mathcal{R}_i} \tau_{l-1} \sum_{p=1}^{L} x_{jrpl}, \tag{93}$$

$$\text{s.t.} \sum_{r\in R_i} \sum_{p=1}^{l} x_{jrpl} = 1, \quad \forall i \in \{1,\dots,N\}, j \in \mathcal{T}_{il}, \tag{94}$$

$$\sum_{i=1}^{N} \sum_{j\in\mathcal{T}_{il}} \sum_{p=1}^{l} p_{jr} x_{jrpl} \leq \tau_l, \quad \forall r \in \mathcal{R}, \tag{95}$$

$$\sum_{i=1}^{N} \sum_{j\in\mathcal{T}_{il}} c_j x_{jrpl} \leq \tau_p, \quad \forall r \in \mathcal{C}, p \in \{1,\dots,L\}, \tag{96}$$

$$\sum_{j\in\mathcal{T}_{il}} \sum_{r\in\mathcal{R}_i} \sum_{p=1}^{L} b_{jr} x_{jrpl} \leq B_{il}, \quad \forall i \in \{1,\dots,N\}, \tag{97}$$

$$x_{jrpl} = 0, \quad \text{if } \tau_p < t_j^R + c_j, \forall i \in \{1,\dots,N\}, \\ j \in \mathcal{T}_{il}, r \in \mathcal{C}, p \in \{1,\dots,L\}, \tag{98}$$

$$\sum_{p=1}^{L} x_{jrpl} = 0, \quad \text{if } \tau_l < t_j^R + p_{jr}, \\ \forall i \in \{1,\dots,N\}, j \in \mathcal{T}_{il}, r \notin \mathcal{C}, \tag{99}$$

$$x_{jrpl} = 0, \quad \text{if } \tau_l < \tau_{p-1} + p_{jr} \text{ and } \tau_p > t_j^R + c_j, \\ \forall i \in \{1,\dots,N\}, j \in \mathcal{T}_{il}, r \in \mathcal{C}, \\ p \in \{1,\dots,L\}, \tag{100}$$

$$x_{jrpl} = 0, \quad \text{if } B_{il} < b_{jr}, \quad \forall i \in \{1,\dots,N\}, \\ j \in \mathcal{T}_{il}, r \in \mathcal{R}, p \in \{1,\dots,L\}, \tag{101}$$

$$x_{jrpl} \geq 0, \quad \forall i \in \{1,\dots,N\}, j \in \mathcal{T}_{il}, r \in \mathcal{R}. \tag{102}$$

To this relaxed solution, we apply the rounding technique proposed in Section 5.3.3, and deal with budget violation as proposed in Section 5.3.4. This gives us a subset of tasks to schedule during iteration l. We expect that this would help increase the $\sum_j w_j \tau_{l-1}$ term for interval l in the original objective.

5.4.2 Implementation Steps

The main steps of our online implementation are as follows:

1) Divide the time horizon into intervals (τ_{l-1}, τ_l), as defined in Section 4.
2) Set $B_{i1} = B_i$ for every user i.
3) At the end of each interval l, i.e., time instant τ_l, identify the set of tasks $\mathcal{T}_{il}$ for every user i.
4) Run the STUBR subroutine (given in Section 5.4.1) to identify the subset of tasks that maximizes the total weight of the tasks.
5) Mark these subset of tasks as scheduled.
6) Calculate the resulting cost C_i, and set $B_{i(l+1)} = B_{il} - C_i$ for every user i.
7) Repeat Steps 3-6 until all tasks are scheduled.

The above online algorithm terminates once all tasks have been scheduled and there is no more task arrival. Intuitively, it greedily maximizes the weight of tasks assigned to earlier intervals, resulting in smaller weighted sum completion time.

6 Trace-driven Simulation

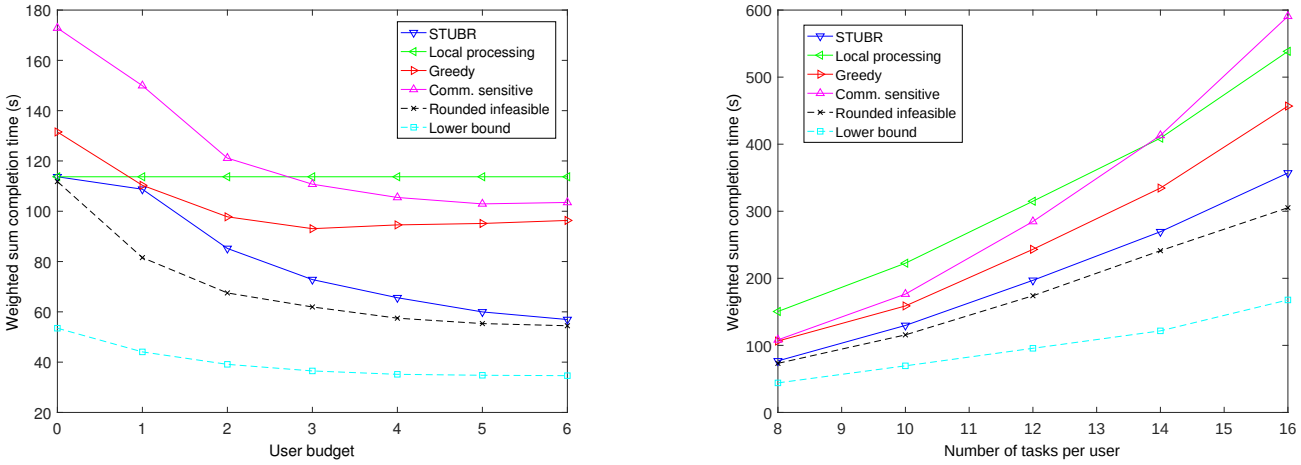
In addition to the worst-case bounds derived in Sections 4 and 5, in this section, we investigate the performance of STUBR, using trace-driven simulation. We study the effect of user budget and number of tasks on algorithm performance. We evaluate STUBR for the model with release times and fixed communication times described in Section 5.2, and for the model with sequence-dependent release times and communication times described in Section 5.3.

6.1 Traces and Parameter Setting

In [34], the authors conducted experiments on different applications, and provided task characteristics in terms of input data, computation need, and arrival rates. We provide these details in Table 3, for the chess and compute-intensive

TABLE 3: Task Characteristics

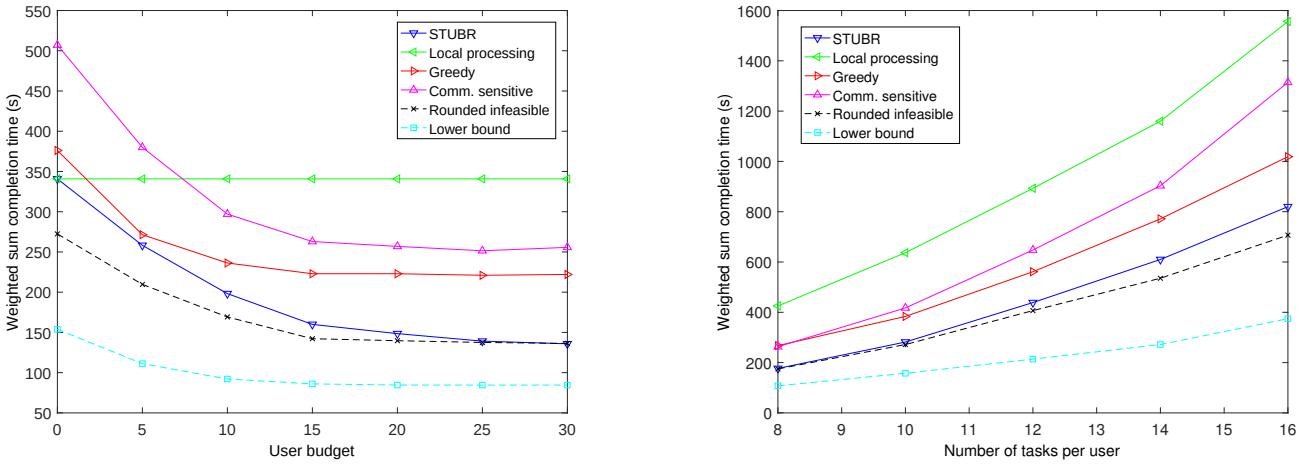
Application	Input	Computation	Arrival Rate
Chess	2 MBytes	10 MFLOPs	1 task/sec
Compute-intensive	8 MBytes	100 MFLOPs	0.5 task/sec



(a) Effect of user budget

(b) Effect of the number of tasks per user

Fig. 2: For chess application on Galaxy S5.



(a) Effect of user budget

(b) Effect of the number of tasks per user

Fig. 3: For compute intensive application on Nexus 10.

applications considered. Additionally, they consider different mobile devices with varying computational capacities. We use these traces from this paper in order to test our proposed algorithm as follows:

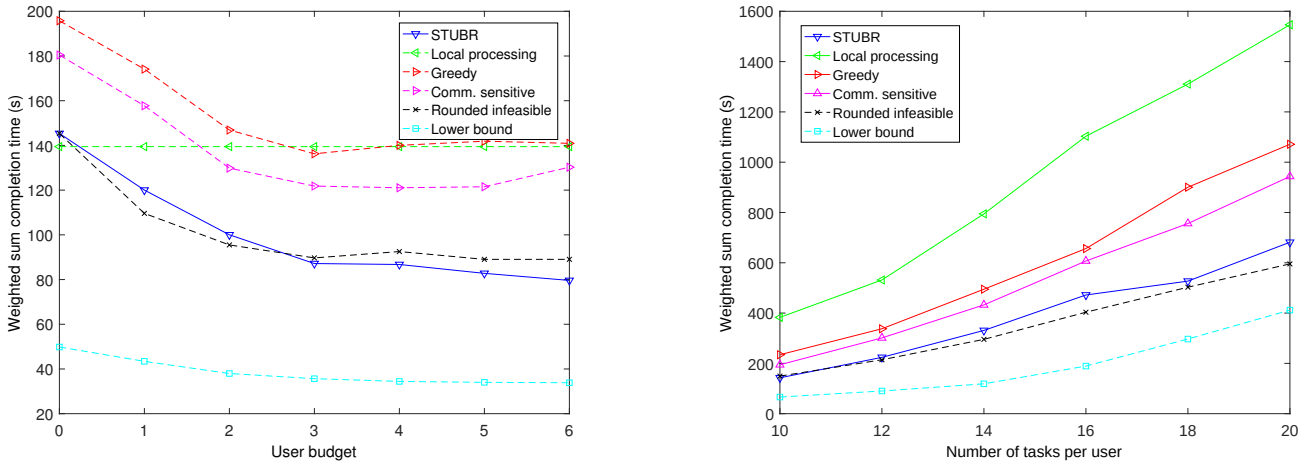
1) We take the computation need and input data given in [34] as mean, and allow a maximum of ± 50 % variation. In other words, we randomly pick these values from a uniform distribution in (50% mean, 150% mean).
   a) We calculate the mean local processing time t_j of tasks.
   b) We calculate the mean communication time of a task by dividing the input data (in MBytes) by the available data rate, which is 20 Mbps from [34].
2) We pick the release time values from a uniform distribution in the range $(0, \frac{\text{number of tasks}}{\text{arrival rate (in task/sec)}})$.
3) We pick the task weights from a uniform distribution in the range (0,1).
4) We run multiple randomized iterations (for different values of input data and computation) for each parameter setting, and take the average among them to plot each point on the graph.

We run our simulation using MATLAB R2016b, and utilize the CVX programming package to solve our linear programs.

6.2 Comparison Targets

We use the following targets for comparison with STUBR algorithm:

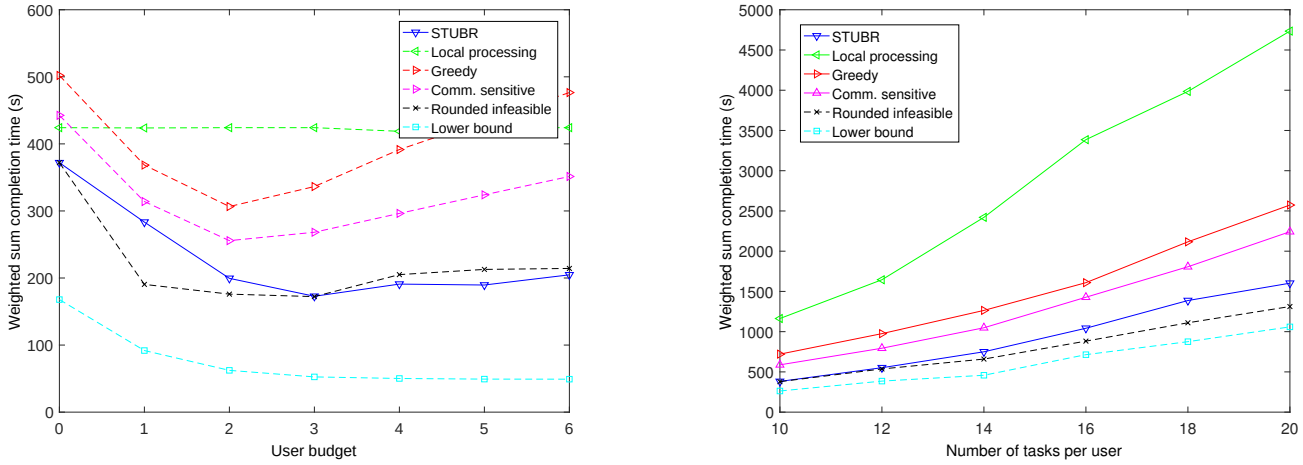
- *Lower bound*: This is the relaxed solution obtained from Section 4.1.
- *Rounded infeasible*: This is the solution obtained from Section 4.2, without dealing with budget violation. We also perform a modified WSPT, proposed in Section 4.4, on this solution. Hence, this solution has an objective value that is a constant times the optimal, but may violate the user budgets.
- *Greedy*: All tasks are sorted in the non-decreasing order of weighted local processing time $\frac{t_j}{w_j}$ for all j, and each task is scheduled in this order onto the



(a) Effect of user budget

(b) Effect of the number of tasks per user

Fig. 4: For chess application on Galaxy S5.



(a) Effect of user budget

(b) Effect of the number of tasks per user

Fig. 5: For compute intensive application on Nexus 10.

processor where it meets its user's budget and has the fastest processing time.

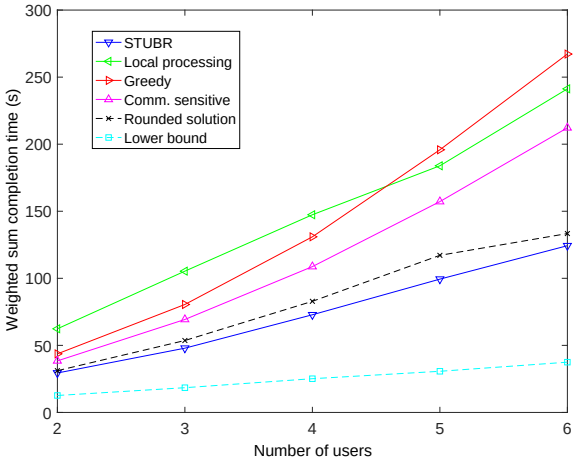
- *Local processing*: All tasks are scheduled locally, and ordered in the non-decreasing order of weighted local processing time $\frac{t_j^R + t_j}{w_j}$ for all j. This would illustrate the benefits of offloading using our algorithm.
- *Comm. sensitive*: All tasks are sorted in the non-decreasing order of communication c_j for all j, and each task is scheduled in this order onto the processor where it meets its user's budget and has the fastest processing time. This method of sorting tries to offload the tasks that have shorter communication times thereby decreasing the overall contribution of communication time to the objective.

6.3 For Release Times and Fixed Communication Times

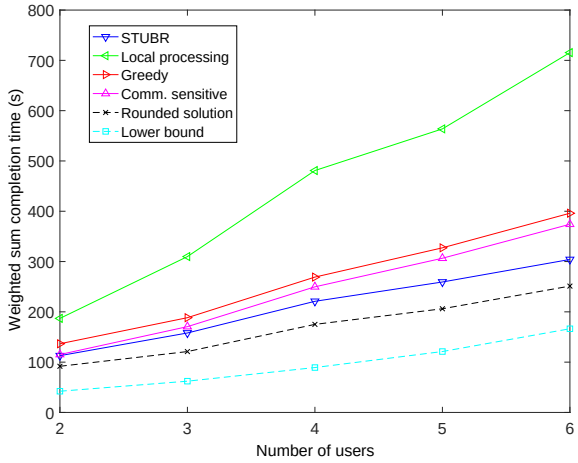
In this section, we evaluate STUBR for the model with release times and fixed communication times described in Section 5.2, for chess and compute-intensive applications presented in [34].

In Figures 2a and 2b, we consider three Galaxy S5 users and chess applications. We consider a five-processor cloud with speed-up factors $\alpha_{i1} = 0.5$, $\alpha_{i2} = \alpha_{i3} = 0.1$, and $\alpha_{i4} = \alpha_{i5} = 0.2$ for every user i. We set the processor prices as $\beta_1 = 0.5$, $\beta_2 = \beta_3 = 3$, and $\beta_4 = \beta_5 = 2$. This parameter setting ensures that the users will have to pay a higher price to use a faster processor.

For Figure 2a, we consider users with equal budget, and constant number of tasks $|\mathcal{J}_1| = 5$, $|\mathcal{J}_2| = 5$, and $|\mathcal{J}_3| = 10$. This allows us to study the impact of user budget on the weighted sum completion time and algorithm performance. We see that as the user budget increases, the weighted sum completion time decreases as expected. We also see that the STUBR curve appears to plateau beyond a particular value of budget that is large enough to offload all tasks to the fastest processors. On the other hand, for tighter values of budget, we see that the STUBR curve coincides with the local execution curve. Additionally, we also note that the gap



(a) For chess application on Galaxy S5.



(b) For compute intensive application on Nexus 10.

Fig. 6: Effect of number of users

between STUBR and the rounded solution decreases with increasing budget until eventually the STUBR curve meets the rounded solution curve. This illustrates that the amount of budget violation decreases with increasing budget, and consequently, the STUBR solution approaches the rounded solution.

In Figure 2b, we observe the impact of the number of tasks per user, for user budgets $B_1 = B_2 = B_3 = 5$. The total weighted completion time increases with increasing number of tasks per user (and total number of tasks) as expected. We see that the performance gap between STUBR and other schemes increases with increasing number of tasks, indicating that STUBR is more scalable.

In Figures 3a and 3b, we consider three Nexus 10 users running compute-intensive applications (as described in [34]). We consider the same five-processor simulation setup as that of Figures 2a and 2b. For Figure 3a, we consider constant number of tasks $|\mathcal{J}_1| = 5$, $|\mathcal{J}_2| = 5$, and $|\mathcal{J}_3| = 10$. For Figure 3b, we set $B_1 = B_2 = B_3 = 20$. We again see that STUBR provides superior performance and scales well.

6.4 For Sequence-dependent Communication Times

We now consider the model with sequence-dependent release times and communication times described in Section 5.3. In Figures 4 and 5, we use the same simulation setting from Section 6.3 for the modified channel model and STUBR proposed in Section 5.3.

Interestingly, we observe that STUBR performs better than even the rounded solution for some samples. This happens because moving tasks to the local device may significantly reduce task completion times in some cases because of the reduction of sequence-dependent release/communication times, particularly when these dominate the processing times. Furthermore, STUBR outperforms all other alternatives for both chess and compute-intensive applications. In fact, the performance gap between STUBR and other alternatives is even greater than for the fixed communication case. It is also interesting to note that the communication sensitive scheme performs better than the greedy scheme for this sequence-dependent communication model since the communication times now contribute more to the overall objective. In some cases, we see that greedy and communication sensitive schemes may even increase with increase in user budget, because of the naive nature of these schemes that causes the initial tasks to use up all of the budget and do not take release times into account while making scheduling decisions. We again see that STUBR scales well with increase in number of tasks per user.

In Figure 6, we study the impact of number of users on algorithm performance. We consider the same five-processor simulation setup as that of Figures 4 and 5, for user budgets $B_i = 5$ and number of tasks $|\mathcal{J}_i| = 5$, for every user i. We see that STUBR still outperforms all other alternatives for both chess and compute-intensive applications, for the entire range of number of users considered. We restrict it to a maximum of six users for the purposes of simulation, but we can see that STUBR scales well with increasing number of users. Additionally, we see that the gap between STUBR and other alternatives increases as the number of users increases, which further validates the relative superiority of STUBR.

7 Conclusion and Future Work

We have studied a multi-user computational offloading problem, for a system consisting of a finite-capacity cloud with heterogeneous processors. The offloaded tasks incur monetary cost for the cloud resource usage and each user has a budget constraint. We have formulated the problem of weighted-sum-completion-time minimization subject to the user budget constraints. We have formulated a problem to minimize the weighted sum completion time subject to the user budget constraints. The proposed STUBR algorithm relaxes, rounds, and resolves budget violations, and it sorts the tasks to obtain an effective solution. We have also obtained interesting performance bounds for both the underlying rounded solution and the budget-resolved solution for different release-time and communication-time models. Through simulation using real-world application traces, we have observed that STUBR is scalable and substantially

outperforms the existing alternatives especially for larger systems.

A possible future research direction is to account for explicit task dependencies in the formulation. Additionally, the consideration of multiple types of computing resource in task scheduling is a challenging but important problem [44], [45]. For example, the memory requirements of the tasks and the memory capacity of the processors may be considered. Additionally, we may consider different pricing and budget schemes as an extension to the linear scheme considered in this paper.

REFERENCES

[1] S. Sundar, J. P. Champati, and B. Liang, "Completion time minimization in multi-user task scheduling with heterogeneous processors and budget constraints," in *Proc. IEEE/ACM International Symposium on Quality of Service (IWQoS), Short Paper*, 2018.

[2] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 84–106, Jan 2013.

[3] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. MCC workshop on Mobile cloud computing*, 2012.

[5] E. G. Specification, "Mobile edge computing (mec); framework and reference architecture," *ETSI GS MEC 003 V1.1.1*, vol. 3, 2016.

[6] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[7] B. Liang, "Mobile edge computing," in *Key Technologies for 5G Wireless Systems*, V. W. S. Wong, R. Schober, D. W. K. Ng, and L.-C. Wang, Eds., Cambridge University Press, 2017.

[8] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proc. ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2010.

[9] W. Zhang, Y. Wen, and D. O. Wu, "Energy-efficient scheduling policy for collaborative execution in mobile cloud computing," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, 2013.

[10] Y.-H. Kao and B. Krishnamachari, "Optimizing mobile computational offloading with delay constraints," in *Proc. IEEE Global Communication Conference (Globecom)*, 2014.

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

[12] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Transactions on Mobile Computing*, 2017.

[13] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2016.

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

[15] V. Cardellini, V. D. N. Personé, V. Di Valerio, F. Facchinei, V. Grassi, F. L. Presti, and V. Piccialli, "A game-theoretic approach to computation offloading in mobile cloud computing," *Mathematical Programming*, vol. 157, no. 2, pp. 421–449, 2016.

[16] M.-H. Chen, B. Liang, and M. Dong, "Multi-user multi-task offloading and resource allocation in mobile cloud systems," *arXiv preprint arXiv:1803.06577*, 2018.

[17] M.-H. Chen, M. Dong, and B. Liang, "Resource sharing of a computing access point for multi-user mobile cloud offloading with delay constraints," *IEEE Transactions on Mobile Computing*, 2018.

[18] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, 2017.

[19] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.

[20] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, 2016.

[21] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, 2018.

[22] S. K. Panda and P. K. Jana, "Efficient task scheduling algorithms for heterogeneous multi-cloud environment," *The Journal of Supercomputing*, vol. 71, no. 4, pp. 1505–1533, 2015.

[23] M. Mao and M. Humphrey, "Scaling and scheduling to maximize application performance within budget constraints in cloud workflows," in *Proc. International Symposium on Parallel & Distributed Processing (IPDPS)*, 2013.

[24] J. Yan, S. Bi, and Y. A. Zhang, "Optimal offloading and resource allocation in mobile-edge computing with inter-user task dependency," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–8.

[25] Z. Kuang, Y. Shi, S. Guo, J. Dan, and B. Xiao, "Multi-user offloading game strategy in ofdma mobile cloud computing system," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 12, pp. 12 190–12 201, 2019.

[26] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*, 1st ed. New York, NY, USA: Cambridge University Press, 2011.

[27] C. Mateos, E. Pacini, and C. G. Garino, "An ACO-inspired algorithm for minimizing weighted flowtime in cloud-based parameter sweep experiments," *Advances in Engineering Software*, vol. 56, pp. 38–50, 2013.

[28] Z. Zhou and H. Zhigang, "Task scheduling algorithm based on greedy strategy in cloud computing," *The Open Cybernetics & Systemics Journal*, vol. 8, no. 1, pp. 111–114, 2014.

[29] Z. Qiu, C. Stein, and Y. Zhong, "Minimizing the total weighted completion time of coflows in datacenter networks," in *Proc. ACM Symposium on Parallelism in Algorithms and Architectures*, 2015.

[30] B. Tian, C. Tian, H. Dai, and B. Wang, "Scheduling coflows of multi-stage jobs to minimize the total weighted job completion time," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, 2018.

[31] S. Khuller, J. Li, P. Sturmfels, K. Sun, and P. Venkat, "Select and permute: An improved online framework for scheduling to minimize weighted completion time," *Theoretical Computer Science*, vol. 795, pp. 420–431, 2019.

[32] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, "Scheduling to minimize average completion time: Off-line and on-line approximation algorithms," *Mathematics of Operations Research*, vol. 22, no. 3, pp. 513–544, 1997.

[33] P. Crescenzi and V. Kann, "Approximation on the web: A compendium of np optimization problems," in *Proc. International Workshop on Randomization and Approximation Techniques in Computer Science*, 1997.

[34] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *IEEE Proc. International Conference on Cloud Computing (CLOUD)*, 2015.

[35] Y. Kim, J. Kwak, and S. Chong, "Dual-side dynamic controls for cost minimization in mobile cloud computing systems," in *Proc. International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, 2015.

[36] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 726–738, 2019.

[37] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," in *Proc. ACM International Symposium on High Performance Distributed Computing*, 2010.

[38] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proc. ACM Conference on Computer Systems*, 2011.

[39] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, 2012.
[40] D. B. Shmoys and É. Tardos, "An approximation algorithm for the generalized assignment problem," *Mathematical Programming*, vol. 62, no. 1-3, pp. 461–474, 1993.
[41] W. E. Smith, "Various optimizers for single-stage production," *Naval Research Logistics*, vol. 3, no. 1-2, pp. 59–66, 1956.
[42] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proc. ACM Symposium on Theory of Computing*, 1984.
[43] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics*, vol. 2, no. 1-2, pp. 83–97, 1955.
[44] W. Wang, B. Liang, and B. Li, "Multi-resource fair allocation in heterogeneous cloud computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 10, pp. 2822–2835, 2014.
[45] W. Wang, B. Li, B. Liang, and J. Li, "Multi-resource fair sharing for datacenter jobs with placement constraints," pp. 1003–1014, 2016.

Sowndarya Sundar Sowndarya Sundar is currently a Research Scientist at Ford Greenfield Labs, working on continual learning and computer vision problems for connected vehicles. She completed her PhD in Electrical and Computer Engineering at the University of Toronto, Canada, where she worked on the design and analysis of algorithms to solve optimization problems in cloud computing. She obtained her Masters of Applied Sciences degree from the University of Toronto, and Bachelor of Engineering degree from Anna University, India. She has been a recipient of multiple research scholarships including the Natural Sciences and Engineering Research Council of Canada postgraduate scholarships, Queen Elizabeth II Graduate Scholarships in Science & Technology, and Ontario Graduate Scholarships.

Jaya Prakash Champati Jaya Prakash Champati is currently a post-doctoral researcher from the division of Information Science and Engineering, EECS, KTH Royal Institute of Technology, Sweden. In September 2020, he will be joining IMDEA Networks Institute, Spain, as an Assistant Professor. His general research interest is in the design and analysis of algorithms for scheduling problems that arise in networking and information systems. Currently, his focus is on the analysis and optimization of delay and freshness in networked systems to support emerging applications from Cyber-Physical Systems (CPS), Internet of Things (IoT), and edge computing systems. He finished his PhD in Electrical and Computer Engineering, University of Toronto, Canada in 2017. He obtained his master of technology degree from Indian Institute of Technology (IIT) Bombay, India, and bachelor of technology degree from National Institute of Technology Warangal, India. Prior to joining PhD, he worked at Broadcom Communications, where he was part of developing LTE MAC layer. He was a recipient of the best paper award at IEEE National Conference on Communications, 2011.

Ben Liang Ben Liang received honors-simultaneous B.Sc. (valedictorian) and M.Sc. degrees in Electrical Engineering from Polytechnic University (now the engineering school of New York University) in 1997 and the Ph.D. degree in Electrical Engineering with a minor in Computer Science from Cornell University in 2001. He was a visiting lecturer and post-doctoral research associate at Cornell University in the 2001 - 2002 academic year. He joined the Department of Electrical and Computer Engineering at the University of Toronto in 2002, where he is now Professor and L. Lau Chair in Electrical and Computer Engineering. His current research interests are in networked systems and mobile communications. He is an associate editor for the IEEE Transactions on Mobile Computing and has served on the editorial boards of the IEEE Transactions on Communications, the IEEE Transactions on Wireless Communications, and the Wiley Security and Communication Networks. He regularly serves on the organizational and technical committees of a number of conferences. He is a Fellow of IEEE and a member of ACM and Tau Beta Pi.