Completion Time Minimization in Multi-user Task Scheduling with Heterogeneous Processors and Budget Constraints

Sowndarya Sundar, Jaya Prakash Champati, and Ben Liang
Electrical and Computer Engineering, University of Toronto, Ontario, Canada
Email: {ssundar, champati, 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 finite-capacity shared cloud with heterogeneous processors by paying a price for the resource usage. 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 solve an integer-relaxed problem and use 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 Single-Task Unload for Budget Resolution (STUBR) algorithm, which resolves budget violations and orders the tasks to reduce the weighted sum completion time. Our trace-driven simulation shows that STUBR exhibits robust performance under practical scenarios and outperforms several alternatives.**

I. Introduction

Computational offloading is one of the key features that led to the development of Mobile Cloud Computing (MCC) and Mobile Edge Computing (MEC) systems, where mobile devices may offload their computational tasks to cloud resource providers. Each mobile user may need to pay a monetary cost for the computational resource usage. Furthermore, as in cloudlets and fog computing, MCC and edge computing are often characterized by multiple heterogeneous helper processors that are of diverse capabilities.

Motivated by these systems, we study a problem of task scheduling and offloading in a heterogeneous cloud computing system to minimize the computational delays of multiple users' tasks. Several existing works study the offloading problem only for the single-user scenario [1]–[5], while the multi-user scenario was studied in [6]–[10]. In this work, we focus on practical constraints in the multi-user scenario including finite-capacity user devices, finite-capacity cloud consisting of heterogeneous servers, and budget limitation for the users.

We consider 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 finite-capacity cloud server. 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. 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 user 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 [11]. For a special case of our problem where there is a *single user* and *no budget constraint*, an efficient solution was proposed in [12]. However, extending their solution approach is non-trivial even for the case of a single user with budget constraint. We will see later that having budget constraints for multiple users makes the problem much more challenging.

Our main contributions are summarized below:

- For our problem, we formulate an interval-indexed ILP, inspired by [12]. 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 of the original problem, we make an interesting observation that the average budget violation decreases with respect to the number of users. Consequently, when the number of users is sufficiently large, the average cost incurred across all users meets the average budget.
- Based on the above observation, we propose the Single Task Unload for Budget Resolution (STUBR) algorithm. In addition to finding a relaxed LP-solution for the above ILP, STUBR resolves budget violations in the rounded integer solution and then uses greedy task ordering on each processor to further reduce the weighted sum completion time. We also derive the computational complexity of STUBR.
- Our trace-driven simulation shows that STUBR performs consistently better than the alternatives. It exhibits maximum performance gains of 40% for the chess application and 43.4% for compute-intensive applications [13] in

This work has been funded in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

comparison with a Weighted Shortest Processing Time (WSPT) scheme [14]. 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 II, we present the related work. Section III describes the system model and the problem formulation. In Section IV, we propose the STUBR algorithm and provide performance guarantees. Section V presents the simulation results, and we conclude the paper in Section VI.

II. 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 [7]–[10]. Unlike these 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.

For general cloud or multi-processor task scheduling problems, many studies have focused on energy consumption [1], [3], [4], [9] or makespan [15], [16]. However, in practice, 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. However, few works in the literature have considered the same objective to schedule tasks in a cloud environment [12], [17]. Both [12] and [17] assume a single user with multiple tasks. While [17] employs an ant-colony heuristic and simulation based performance evaluation, [12] further considers release times on parallel processors and presents an 8-approximation algorithm. Our solution approach is inspired by [12]. However, our problem also accounts for multiple users, individual user budget constraints, and task communication times, which renders it more challenging than those addressed in [12].

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., [5], [6]), while few aim to maximize the benefit to users under budget constraints [18], [19]. In [18], the authors considered the problem of maximizing the service quality for a multi-task application with a single budget constraint. In [19], 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. In our problem, in addition to a different optimization objective, we further consider the more general case of *multiple users* and *per-user budget constraints*, where each user may have multiple tasks.

III. System Model and Problem Formulation

A. System Model

1) Tasks: We consider a system with N user devices, possibly mobile. Each user $i \in \{1, \ldots, N\}$ wishes to complete a finite set of independent tasks, denoted by $\mathcal{J}_i$. Each task j is released or becomes available for processing at time t_j^R. The processing time for each task $j \in \mathcal{J}_i$ on user i's local processor is bounded and denoted by t_j. Additionally, each task may require input data that need to be communicated if the task is to be executed at the cloud. Towards this end, we assume communication times c_j for task j. We also consider a weight w_j associated with each task. The inclusion of these weights allows a more generic model that assigns different priorities to users and/or tasks. In this work, we proceed assuming that such information is already given.

2) Processors: The system includes a finite-capacity cloud consisting of a number of heterogeneous processors that run at different speeds. Each processor at the cloud is assumed to be unary, 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 unary processor simultaneously will not provide any improvement to the sum completion time objective. In addition to the cloud processors, each user has its own unary local processor. The user can execute its tasks either locally or remotely at one of the cloud processors.

The speed-up factor for each cloud processor r is α_{ir}, so that the processing time for task j at processor r is $\alpha_{ir} t_j$. Let $\mathcal{R}_i$ denote the set of processors to which user i can offload its tasks, i.e., its own local processor and cloud processors. Let $\mathcal{C}$ be the set of cloud processors, and $\mathcal{R}$ be the set of all processors (including all users' local processors).

3) 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.

B. Problem Formulation

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 [12]. Towards this end, we divide the time axis into intervals indexed by $l \in \{1, \ldots, L\}$, where L is the smallest integer such that

$$2^{L-1} \geq \max_j t_j^R + \sum_j (t_j + c_j).$$

This means that 2^{L-1} is a sufficiently large time horizon for the scheduling of all given tasks since it accounts for the largest release time $\max_j t_j^R$ and worst-case completion time $\sum_j (t_j + c_j)$. Let $\tau_0 = 1$ and $\tau_l = 2^{l-1}$, for all $l \in \{1, \ldots, L\}$. Each interval l corresponds to the time slot (τ_{l-1}, τ_l). 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 $l \in \{1, \ldots, L\}$. As shown in [12], 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.

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

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

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

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

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

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

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

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

The objective (1) is to minimize the weighted sum completion times of tasks across all users by utilizing interval bound τ_{l-1} for tasks that finish in interval l. 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. This optimization problem can be solved by a central scheduler, for example, residing at the cloud.

IV. The STUBR algorithm

The aforementioned problem is NP-hard since a special case of this problem, involving just a single user with no budget constraints, is NP-hard [12]. Consequently, in this section, we present the polynomial-time STUBR algorithm, as a heuristic solution to problem (1). We then prove some guarantees and properties of this algorithm, to better understand it's 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 optimal objective value of problem (1). While this rounded solution is expected to violate the budget constraints, we prove that for a large number of users, the average cost incurred meets the average user budget.
3) We resolve any budget violation by strategically moving some tasks to the local device.
4) WSPT ordering is optimal for our objective for a single processor and jobs without release times [14]. Hence, on each processor, we reorder the tasks allocated to it by utilizing a modified version of WSPT ordering to further improve the total weighted completion time.

These steps are explained in detail in the following sections.

A. Relaxed Solution

For each user $i \in \{1, \ldots, N\}$, $j \in \mathcal{J}_i$, and $r \in \mathcal{R}_i$, let p_{jr}, t_{jr}^R, and b_{jr} be the processing times, release times and costs for scheduling task j on processor r. Then we have

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

$$t_{jr}^R := \begin{cases} t_j^R + c_j & \text{if } r \in \mathcal{C}, \\ t_j^R & \text{otherwise,} \end{cases} \tag{10}$$

$$b_{jr} := \begin{cases} \beta_r \alpha_{ir} t_j & \text{if } r \in \mathcal{C}, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

We may reformulate the optimization problem in Section III-B with p_{jr}, r_{jr}, and b_{jr} for simplicity, and relax the integer constraints to obtain a linear program. This linear program can be solved efficiently in polynomial-time to obtain a relaxed solution to the proposed problem. This formulation also resembles the LP-relaxed version of the problem minimizing the weighted sum completion time in a system of unrelated machines with release times formulated in [12]. However, our formulation has additional budget constraints, (4) and (7), that need to be met for each user. It also accommodates multiple users unlike the formulation in [12]. These aspects render our formulation a more complex one requiring more sophisticated techniques, for recovering an integer solution and resolving budget overage.

B. Rounded Solution

In this section, we produce an initial integer solution, by extending a technique used in [12], and applying the rounding technique proposed in [20] to our relaxed solution. 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.

1) Rounding technique: We first label each processor-interval pair (r, l) as a single virtual processor r', and denote the collection of such virtual processors as $\mathcal{R}'$. We then convert the LP-solution x_{jrl} to $x_{jr'}$ for each $r' \in \mathcal{R}'$. We now summarize the rounding method proposed in [20], and its application to our problem. The rounding technique consists of two basic steps: (1) ordering tasks, and (2) constructing a bipartite matching. The first step lists the tasks in non-increasing order of $p_{jr'}$, for $r' \in \mathcal{R}'$. It may be noted that $p_{jr'} = p_{jr}$, since the processing time for a task only depends

on the processor it is executed on and not the interval it is executed in.

In the second step, to construct a bipartite matching, we first define 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 $k_{r'} = \lceil \sum_j x_{jr'} \rceil$. We require a fractional matching between the task nodes and machine nodes, denoted by $f(v_{r's},u_j)$, which assigns each task partially to multiple machine nodes, such that all allocated fractions for a particular task node should sum up to 1. Let $\mathcal{E}$ be the set of edges in the bipartite graph representing this fractional matching. The fractional matching is constructed in accordance with the following:

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

where $\mathcal{S}' = \{s : (v_{r's}, u_j) \in \mathcal{E}\}$, and

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

where $\mathcal{J}' = \{j : (v_{r's}, u_j) \in \mathcal{E}\}$ is the set of tasks connected to machine node $v_{r's}$. 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 $\overline{\mathbf{x}} = \{\overline{x_{jrl}}\}$.

2) Interval deadline violation and performance guarantee: We first establish a lemma that provides a bound on the violation of the interval deadline constraints, and then use it to obtain a performance guarantee on the objective. We present the results here and omit the proofs due to page limitation.

Lemma 1. *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 (3) is violated by at most τ_l.*

Theorem 1. *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).*

3) Multiple users and incurred cost guarantees: One must note that the above rounded solution still violates the user budget constraints. The following theorem quantifies the amount of budget violation.

Theorem 2. *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.*

The following conclusions follow directly from Theorem 2.

Corollary 1. *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*

$$\sum_{i=1}^{N} \sum_{j \in \mathcal{J}_i} \sum_{r \in \mathcal{R}} \sum_{l=1}^{L} b_r \overline{x_{jrl}} \leq \sum_{i=1}^{N} B_i + \sum_{r \in \mathcal{S}} b_r. \tag{12}$$

Corollary 2. *If C_i is the incurred cost for user i,*

$$\frac{1}{N}\sum_{i=1}^{N} C_i \leq \frac{1}{N}\sum_{i=1}^{N} B_i + \frac{1}{N}|\mathcal{R}'|B^{max}, \tag{13}$$

where $B^{max} = \max_i B_i$, and for the specific case from Corollary 1,

$$\frac{1}{N}\sum_{i=1}^{N} C_i \leq \frac{1}{N}\sum_{i=1}^{N} B_i + \frac{1}{N}\sum_{r \in \mathcal{S}} b_r. \tag{14}$$

From this corollary, we obtain the following property of the rounded solution.

Corollary 3. *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.

C. 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 it does not cost a user to execute tasks on its 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, $\{j \in \mathcal{J}_i : \overline{x_{jrl}} = 1, \forall r \in \mathcal{C}, l \in \{1,\ldots,L\}\}$, 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 to do less 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 the incurred cost now meets the budget, stop. If not, repeat Steps 2 and 3 until user i's budget is met.
5) Repeat for all users.

D. Modified 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 [14]. Thus, we perform a modified version of WSPT ordering on the tasks allocated to a particular processor to further improve our objective value. In order to accommodate the task release times, we apply our modified WSPT as follows:

1) Obtain the task scheduling decision, i.e., the processor on which each task should be scheduled.

2) On each processor $r \in \mathcal{R}$, order the scheduled tasks in the non-decreasing order of $\frac{t^R_{jr}+p_{jr}}{w_j}$. This ensures that tasks with smaller weights and longer completion times (without accounting for wait times) are scheduled earlier.
3) Modify the task completion times correspondingly, and obtain the new objective value.

E. Feasibility and Complexity Analysis

It can be readily noted that the STUBR algorithm provides a feasible solution. In other words, the user budgets are 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 IV-A) and the rounding step (in Section IV-B) that involves finding the weighted sum completion time fractional matching. For a fixed number of decision variables, a linear program can be solved in $O(n)$ time where n is the number of constraints. For our problem, this implies that solving the LP is upper bounded by $O(P|\mathcal{R}|L)$, where $P = \sum_{i=1}^{N} |\mathcal{J}_i|$ is the total number of tasks. On the other hand, the bipartite matching can be solved in cubic time in the number of vertices by utilizing the Hungarian algorithm [21]. If $P > |\mathcal{R}|$, the time complexity of this step would be upper bounded by $O(P^3)$. Thus, we see that the overall worst-case time complexity of STUBR is $O(P^3 + P^2L)$.

V. Trace-driven Simulation

In this section, we investigate the performance of STUBR, using trace-driven simulation. In [13], the authors conducted experiments on several applications, and provided task characteristics in terms of input data, computation need, and arrival rates. Additionally, they considered different mobile devices with varying computational capacities. We use traces corresponding to (1) chess application and (2) compute-intensive application from this paper. We obtain the necessary information from the traces as follows:

1) We take the computation need and input data given in [13] as mean, and allow a maximum of ± 50 % variation. In other words, we randomly pick values from a uniform distribution in (0.5 mean, 1.5 mean).
   a) We calculate the mean local processing time t_j of the tasks as $\frac{\text{computation need of task (in MFLOPs)}}{\text{computation capacity of device (in MFLOPS)}}$.
   b) We calculate the mean communication time as $\frac{\text{input data (in MBytes)}}{\text{20 Mbps}}$, where the available data rate assigned to each user is 20 Mbps from [13].
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, and utilize the CVX programming package to solve our linear programs.

We use the following targets for comparison with STUBR algorithm:

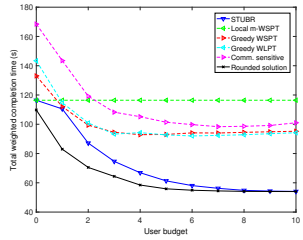
- *Rounded solution*: This is the solution obtained from Section IV-D, without dealing with budget violation. We also perform the modified WSPT, proposed in Section IV-D, on this solution. Hence, this solution may violate the user budget and can be viewed as a lower bound to STUBR.
- *Greedy WSPT*: All tasks are sorted in the non-decreasing order of t_j/w_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.
- *Greedy Weighted Longest Processing Time (WLPT)*: Same as Greedy WSPT except tasks are sorted in the non-increasing order of $t_j w_j$ for all j.
- *Local m-WSPT*: All tasks are scheduled locally, and sorted based on the modified WSPT proposed in Section IV-D. This would illustrate the benefits of offloading using our algorithm.
- *Comm. sensitive*: Same as Greedy WSPT except tasks are sorted in the non-decreasing order of c_j for all j. This method tries to offload the tasks that have shorter communication times thereby decreasing the overall contribution of communication time to the objective.

In Figures 1a and 1b, we consider three Galaxy S5 users [13] and the chess application. 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.

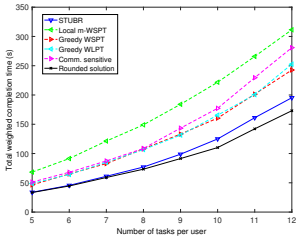
For Figure 1a, we consider users with equal budget, and constant number of tasks $|\mathcal{J}_1| = 5$, $|\mathcal{J}_2| = 5$, and $|\mathcal{J}_3| = 10$. We see that as the user budget increases, the weighted sum completion time decreases as expected. We also see that the STUBR curve appears to plateau 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 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, STUBR approaches the rounded solution.

In Figure 1b, 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.

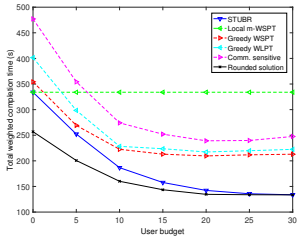
In Figures 1c and 1d, we consider three Nexus 10 users [13] running the compute-intensive application. We consider the same five-processor simulation setup as that of Figures 1a and 1b. For Figure 1c, we consider constant number of tasks



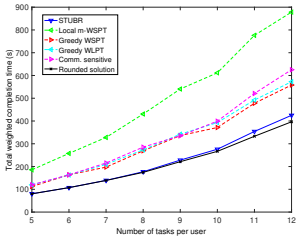
(a) Effect of user budget for chess application on Galaxy S5.



(b) Effect of the number of tasks per user for chess application on Galaxy S5.



(c) Effect of user budget for compute-intensive application on Nexus 10.



(d) Effect of the number of tasks per user for compute-intensive application on Nexus 10.

Fig. 1: Comparison between STUBR and alternatives.

$|\mathcal{J}_1| = 5$, $|\mathcal{J}_2| = 5$, and $|\mathcal{J}_3| = 10$. For Figure 1d, we set $B_1 = B_2 = B_3 = 20$. We again see that STUBR provides superior performance and scales well.

VI. Conclusion

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 cloud resource usage, and each user has a budget constraint. 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. The underlying rounded solution is shown to have the interesting property that at most a single task on each virtual processor needs to be removed to satisfy the total budget constraint regardless of the number of users in the system. This renders our proposed algorithm highly scalable with respect to the number of users in the system. Through simulation using real-world application traces, we have observed that STUBR is scalable and substantially outperform the existing alternatives especially for larger systems.

References

[1] 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)*, pp. 49–62, 2010.

[2] 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)*, pp. 190–194, 2013.

[3] B. Y.-H. Kao and B. Krishnamachari, "Optimizing mobile computational offloading with delay constraints," in *Proc. IEEE Global Communication Conference (Globecom)*, pp. 8–12, 2014.

[4] 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)*, pp. 1451–1455, 2016.

[5] S. Pandey, K. K. Gupta, A. Barker, and R. Buyya, "Minimizing cost when using globally distributed cloud services: A case study in analysis of intrusion detection workflow application," Cloud Computing and Distributed Systems Laboratory, University of Melbourne, Australia, Tech. Rep, 2009.

[6] 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)*, pp. 443–450, 2015.

[7] 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.

[8] 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.

[9] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading decision and resource allocation for multi-user multi-task mobile cloud," in *Proc. IEEE International Conference on Communications (ICC)*, pp. 1–6, 2016.

[10] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *arXiv preprint arXiv:1702.00892*, 2017.

[11] 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*, pp. 111–118, 1997.

[12] 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.

[13] 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)*, pp. 9–16, 2015.

[14] W. E. Smith, "Various optimizers for single-stage production," *Naval Research Logistics*, vol. 3, no. 1-2, pp. 59–66, 1956.

[15] W. Lin, C. Liang, J. Z. Wang, and R. Buyya, "Bandwidth-aware divisible task scheduling for cloud computing," *Software: Practice and Experience*, vol. 44, no. 2, pp. 163–174, 2014.

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

[17] 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.

[18] 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*, pp. 304–307, 2010.

[19] 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)*, pp. 67–78, 2013.

[20] 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.

[21] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics*, vol. 2, no. 1-2, pp. 83–97, 1955.