Task Dispatch through Online Training for Profit Maximization at the Cloud

Sowndarya Sundar and Ben Liang
Department of Electrical and Computer Engineering
University of Toronto, Ontario, Canada
{ssundar, liang}@ece.utoronto.ca

***Abstract*—We study the scheduling of tasks that arrive dynamically at a networked cloud computing system consisting of heterogeneous processors. Execution of tasks yields some profit to the cloud service provider. We intend to maximize the total profit across all tasks arriving within a time interval, subject to processor load constraints, without prior knowledge of the task arrival times or processing requirements. We propose the Task Dispatch through Online Training (TDOT) algorithm, which consists of training and exploitation phases. We provide performance bound analysis to show that TDOT can generate profit that is close to the optimum, given a suitable size for the training task set. TDOT assumes that profit can be obtained from partially completed tasks, so we further propose a modified version of TDOT, termed TDOT-G, for implementations where profit can only be obtained from fully-completed tasks. Through simulation, using Google cluster data, we compare the performance of TDOT and TDOT-G with that of greedy scheduling, logistic regression, and an offline upper-bound solution.**

I. Introduction

Computational offloading refers to the migration of tasks from local devices over a network to more resourceful servers. Such convergence between networking and computing increases the capabilities of resource-poor local devices by improving battery life or application response time.

Most existing works deal with this task offloading problem from an offline perspective by assuming all necessary task information is known in advance. However, in practice, tasks may arrive dynamically at the cloud, and task processing times may not be known in advance [1]–[12]. In this paper, we adopt an *online task model* where tasks arrive over time, in order to address this more practical problem. We consider the offloading of these tasks to multiple cloud servers. Each cloud server in our system model consists of *finite-capacity and heterogeneous* processors. As a result, the cloud servers could represent cloudlets, edge-clouds, or peer devices, in a generic and hybrid cloud computing environment.

In this paper, we address the task scheduling problem from the perspective of a cloud service provider (CSP) that obtains profit by processing user tasks. In our model, the profit obtained is a function of the task processing time and the profit generated per unit time on the scheduled processor. We aim to obtain the scheduling decision that maximizes the *total profit* across all tasks arriving within a time interval, subject to processor *load constraints*. In our online model, we do not know in advance the total number of tasks that will arrive within the time interval. We also do not know the processing times of a task until it arrives at the CSP's controller, which then dispatches the task to the scheduled cloud server for processing. In this paper, we propose polynomial-time algorithms that use a combination of learning and dual-optimization techniques to obtain effective solutions. The main contributions of this work are as follows:

- We formulate our task scheduling problem and propose the Task Dispatch through Online Training (TDOT) algorithm. It consists of two broad phases: (1) a training phase where we observe the processing times of some arriving tasks to obtain information about task characteristics, and (2) an exploitation phase where we make decisions on future tasks with the help of the information obtained. We draw inspiration from a relaxed solution to the offline problem to identify the parameters that bridge TDOT's training and exploitation phases. This algorithm assumes that profit can be obtained on a *partially-completed task*, if the processor load constraint is met before the task could complete execution.
- We derive performance bounds that quantify TDOT's effectiveness against the offline benchmark. For example, for Poisson task arrivals, we present a scenario (below Corollary 1) where TDOT achieves an expected profit that is at least half of the maximum profit achievable by any offline algorithm.
- We then propose a modified version of TDOT, namely TDOT with Greedy Scheduling (TDOT-G), for implementation in systems where profit can only be obtained from *fully-completed tasks*. We use tasks generated from Google cluster traces [13] to investigate the practical performance of the proposed algorithms. We compare it with greedy scheduling, logistic regression, and an offline upper-bound solution. We observe that TDOT and TDOT-G generally outperform all other online alternatives and achieves near-optimal performance over the non-training set of tasks.

II. Related Work

There are many existing works that deal with task scheduling problems for cloud computing environments. However, only a small portion of these address the online problem

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

where task information is not known in advance. This problem has been explored for objectives such as makespan [1]–[3], [5], and average response time [4], [6]. However, fewer papers address this problem with an objective to maximize profit/revenue or minimize cost.

Some earlier works aim to optimize a revenue or cost objective by using right-sizing or appropriately turning off servers to cut cost [7]–[9]. On the other hand, [10] and [11] address the problem through intelligent scheduling. In [10], the problem of maximizing profit for a datacenter by assigning requests from clients to servers appropriately is considered, and a heuristic is proposed. However, the average processing times of requests from each client are assumed to be known apriori, with an objective to maximize the expected profit. In [11], the online data-migration problem with an objective to minimize cost incurred by a user is considered, and an approximation algorithm is proposed. However, the processing capacity of the datacenters is not accounted for, and data can be sent to only one chosen datacenter in a time slot.

In general, existing works that tackle online problems in cloud computing make certain assumptions such as a single server [3], [6], purely fluid tasks [6], [10], [11], homogeneous resources [2], [4], [5], preemptable tasks [12], or propose heuristic solutions [1], [10]. On the other hand, certain theoretical works address generic online problems such as assigning items to agents with budgets [14], [15] and scheduling jobs to machines [16], [17], providing performance guarantees for their schemes. However, these works solve a simpler problem [14], address a considerably different objective [16], [17], or make certain impractical assumptions such as equal-length jobs [16] or a single-processor system [17]. Some of the techniques we apply are similar to those proposed in [15], but unlike [15], we (i) have no prior knowledge of the total number of tasks, (ii) propose an algorithm to obtain feasible task scheduling decisions, (iii) propose a modified algorithm for implementations where profit can only be obtained from fully-completed tasks, and (iv) assess the performance of the proposed schemes through both performance bound analysis and trace-based simulation.

In summary, we consider a novel system model comprising multiple finite-capacity and heterogeneous cloud resources in addition to the online task model. We address the scheduling problem for tasks arriving within a time interval with an objective to maximize profit subject to load constraints. To the best of our knowledge, this leads to an interesting problem that has not been studied in existing literature.

III. System Model and Problem Statement

A. Cloud Processors and Online Task Arrival

We consider a CSP with K broadly defined cloud servers (CSs), which can be, for example, remote cloud servers, mobile edge hosts, or cloudlets. Each CS k has P_k processors, which may not be identical. Tasks arrive at the CSP's controller at an average rate of λ tasks per unit time over a duration of length T. The role of the controller is to dispatch the tasks to the CSs for execution. The processing

TABLE I: Notations

Notation	Description
t_{jrk}	processing time for task j on processor r at CS k
p_{rk}	profit obtained per unit time on processor r at CS k
T	time duration of the system
L	maximum load on each processor
λ	task arrival rate
P_k	total number of processors at CS k
K	total number of CSs

requirements, e.g. number of cycles or processing time, for each task j on processor r in CS k is given by $t_{jrk}, \forall r, k$, and is known only once the task arrives at the controller.

B. Profit Maximization

We assume that the work of processor r in CS k generates profit p_{rk} per unit time, which may account for multiple contributing factors such as the revenue from user payment and the cost of maintaining the processor. Then, the profit obtained by executing task j having processing requirements t_{jrk} on processor r in CS k is given by $p_{rk}t_{jrk}$. By intelligently scheduling this task, we may maximize the profit we gain from it. In this paper, we aim to identify a task dispatching decision that maximizes the total profit across all tasks arriving within duration T. It is important to note that since tasks arrive dynamically, we have no prior knowledge of the exact number of tasks that arrive within duration T, nor their processing requirements on any processor.

Let M be the random number of tasks that arrive within duration T. We define the scheduling decision variables as $x_{jrk} = 1$, when task j is scheduled to processor r in cloud server k, and 0 otherwise. We consider a given load constraint L on each processor[1], so that

$$\sum_{j=1}^{M} t_{jrk}x_{jrk} \le L, \quad \forall k \in \{1,\dots,K\}, r \in \{1,\dots,P_k\}. \tag{1}$$

Additionally, each task is executed at most once:

$$\sum_{r=1}^{P_k}\sum_{k=1}^{K} x_{jrk} \le 1, \quad \forall j \in \{1,\dots,M\}. \tag{2}$$

We aim to maximize the profit of the CSP. Hence, we formulate an optimization problem as follows:

$$\begin{aligned}
\underset{\{x_{jrk}\}}{\text{maximize}} \quad & \sum_{j=1}^{M}\sum_{k=1}^{K}\sum_{r=1}^{P_k} p_{rk}t_{jrk}x_{jrk} && (3)\\
\text{subject to} \quad & (1)-(2), \\
& x_{jrk} \in \{0,1\} \quad \forall j \in \{1,\dots,M\}, \\
& \qquad k \in \{1,\dots,K\}, r \in \{1,\dots,P_k\}. && (4)
\end{aligned}$$

Here, objective (3) is to maximize the total profit across all tasks, CSs, and processors, while we ensure that the maximum load L is well-utilized by packing these tasks efficiently.

[1]The load constraint can be generalized to be processor dependent, i.e., L_{rk}. See Section IV-C for a discussion on this.

Remark 1. *The multiple cloud servers in our model allow us to differentiate between groups of processors. For example, all the processors in a particular CS may incur a different cost from another, and we may have profits $p_{rk} = p_k, \forall r, k$. However, one may also visualize this model as just processors with different profit rates.*

In the *offline* version of the problem, the number of tasks and the task processing times are known in advance. On the other hand, the *online* nature of the proposed problem is more challenging due to the lack of prior information. Consequently, we propose a polynomial-time online algorithm that uses training to identify appropriate scheduling decisions.

IV. Task Dispatch through Online Training

In this section, we first obtain an optimal solution to the binary-relaxed offline problem for performance benchmarking and to gain insights into the online algorithm construction. We then propose the TDOT algorithm for online task scheduling, and provide a performance bound with respect to the relaxed offline solution.

A. Offline Solution through Lagrange Relaxation

We relax the binary constraints (4) in the offline version of problem (3). The dual of this problem is then given by

$$\underset{\{u_{rk}\geq 0, v_j \geq 0\}}{\text{minimize}} \quad \sum_{k=1}^{K}\sum_{r=1}^{P_k} u_{rk}L + \sum_{j=1}^{M} v_j \tag{5}$$

$$\text{subject to} \quad u_{rk}t_{jrk} + v_j \geq p_{rk}t_{jrk}, \quad \forall j \in \{1, \dots, M\},$$
$$r \in \{1, \dots, P_k\}, k \in \{1, \dots, K\}, \tag{6}$$

where u_{rk} are v_j are Lagrange multipliers corresponding to constraints (1) and (2) respectively.

Constraint (6) implies that an optimal solution must satisfy

$$v_j = \max_{r,k} \left(p_{rk} - u_{rk}\right) t_{jrk}, \forall j \in \{1, \dots, M\}. \tag{7}$$

In other words, given optimal $\mathbf{u} = \{u_{rk}, \forall r, k\}$, we should assign each task j to the processor given by $\text{argmax}_{r,k} \left(p_{rk} - u_{rk}\right) p_{rk} t_{jrk}$.

Thus, the dual problem can be rewritten as follows:

$$\underset{\{\mathbf{u}\geq 0\}}{\text{minimize}} \quad D(\mathbf{u}) \tag{8}$$

where
$$D(\mathbf{u}) = \sum_{k=1}^{K}\sum_{r=1}^{P_k} u_{rk}L + \sum_{j=1}^{M} \max_{r,k} \left(p_{rk} - u_{rk}\right) t_{jrk}. \tag{9}$$

This solution is optimal for the binary-relaxed, offline version of problem (3), and is an upper bound to the optimal online solution. We call this solution OPT and use it in Section IV-C to define the performance bound.

B. Online Scheduling with Partial-Task Profit Taking

Now we consider the online problem where tasks arrive dynamically. We neither know the total number of tasks arriving within duration T nor the processing times of the tasks in advance. Hence, we need to dynamically learn about the processing time characteristics, i.e., optimal $\mathbf{u}$ values defined in (9). The proposed TDOT algorithm utilizes a technique that initially performs training to learn from arriving tasks, and then uses the information to produce profit on the remaining set of tasks. TDOT assumes that profit can also be obtained from *partially-completed tasks* within the load constraints. In other words, if only a part of the task scheduled to a processor can be completed before the load L is met, then we retain the partial profit generated due to the execution of that task upto that point. This assumption is eliminated in Section V, i.e., we consider a model where profits can only be obtained from fully-completed tasks. The TDOT algorithm consists of two phases and involves a user-defined parameter $0 < \epsilon < 1$.

1) Training: We observe the first $\lfloor \epsilon\lambda T \rfloor$ arriving tasks, denoted by $\mathcal{A} = \{1, \dots, \lfloor \epsilon\lambda T \rfloor\}$. For each task $j \in \mathcal{A}$, we record its computing requirement and hence $t_{jrk}, \forall r, k$. These tasks may be arbitrarily scheduled. For simplicity, we may ignore for now the profit earned from these tasks and set $x_{jrk} = 0, \forall j \in \{1, \dots, \lfloor \epsilon\lambda T \rfloor\}, r, k$, which is shown later not to affect our derivations of the competitive ratios for TDOT.

If we allocate only ϵL load to $\mathcal{A}$, then we can write the dual problem objective (9) purely for $\mathcal{A}$ as follows:

$$D(\mathbf{u}, \mathcal{A}) = \sum_{k=1}^{K}\sum_{r=1}^{P_k} u_{rk}\epsilon L + \sum_{j\in\mathcal{A}} \max_{r,k} \left(p_{rk} - u_{rk}\right) t_{jrk}. \tag{10}$$

Since the dual of an LP is also an LP, we can use any existing LP solver to efficiently obtain $\mathbf{u}^* = \text{argmin}_{\mathbf{u}\geq 0} D(\mathbf{u}, \mathcal{A})$.

2) Exploitation: Let $\mathcal{A}^c$ denote all tasks arriving after task $\lfloor \epsilon\lambda T \rfloor$. Now for each arriving task $j \in \mathcal{A}^c$, we apply weights $\mathbf{u}^*$ to obtain the scheduling decision as follows. We set

$$(r', k') = \underset{r,k}{\text{argmax}} \left(p_{rk} - u^*_{rk}\right) t_{jrk}, \tag{11}$$

if the task can be scheduled on r' in CS k' without violating $(1-\epsilon)L$. Otherwise, we schedule just a fraction of the task that can be scheduled while meeting $(1-\epsilon)L$. This is because TDOT assumes we may obtain profit from partially-completed tasks as well. We achieve this by defining load variable l_{rk} for every r and k. If task j satisfies $t_{jr'k'} < (1-\epsilon)L - l_{r'k'}$, we set $x_{jr'k'} = 1$; else we set $x_{jr'k'} = \frac{(1-\epsilon)L - l_{r'k'}}{t_{jrk}}$. We then update the variables, $l_{r'k'} = l_{r'k'} + x_{jr'k'}t_{jr'k'}$. We stop at the end of duration T.

After the above two phases, we obtain scheduling decision $x_{jrk}, \forall j, r, k$, and the resulting profit can be calculated by $\sum_{j=1}^{M}\sum_{k=1}^{K}\sum_{r=1}^{P_k} p_{rk}t_{jrk}x_{jrk}$.

Remark 2. *TDOT uses single-shot learning, which is unlike a reinforcement learning approach, often used to solve multi-armed bandit problems, that iteratively explores and exploits, for each incoming task, for example. Single-shot learning is more efficient to implement, and can be more effective if tasks arriving within duration T have similar characteristics. As shown below, it also has a performance bound unlike reinforcement learning.*

C. Performance Bound Analysis

The TDOT algorithm, despite its simple premise of training and exploitation, obtains performance that is close to the optimum. In this section, we present a performance bound for expected profit produced by TDOT in comparison to the upper bound *offline* solution. We have not included proof details here due to lack of space.

Let $S(\mathbf{u}^*, \mathcal{A}^c)$ be the profit obtained by TDOT on the non-training set $\mathcal{A}^c$. We next provide in Lemma 1 a conditional performance bound on $S(\mathbf{u}^*, \mathcal{A}^c)$ with respect to the upper bound OPT. The following definitions are necessary. We define $R(\mathbf{u}^*)$ as the profit obtained in the *absence* of load constraints by applying weights $\mathbf{u}^*$ to the entire set of M tasks, and $R_{rk}(\mathbf{u}^*)$ as the contribution of processor r in CS k to $R(\mathbf{u}^*)$. We further define $R_{rk}(\mathbf{u}^*, \mathcal{A})$ similarly to $R_{rk}(\mathbf{u}^*)$, except over just the set of tasks $\mathcal{A}$.

Lemma 1. *For any given M number of tasks that arrive within duration T, if we have*

$$\sum_{k=1}^{K}\sum_{r=1}^{P_k} |R_{rk}(\boldsymbol{u}^*, \mathcal{A}) - \epsilon R_{rk}(\boldsymbol{u}^*)| \leq \epsilon^2 \sqrt{\frac{\lambda T}{M}} \max\{OPT, R(\boldsymbol{u}^*)\}, \quad (12)$$

then $S(\boldsymbol{u}^*, \mathcal{A}^c) \geq (1 - \epsilon - \epsilon\sqrt{\frac{\lambda T}{M}})OPT$.

This lemma states that if $\mathbf{u}^*$ produces an unconstrained profit on the entire set of tasks that is proportionally close to that on the training set $\mathcal{A}$ for each (r, k), then we obtain a performance bound on the profit on the non-training set, i.e., $S(\mathbf{u}^*, \mathcal{A}^c)$ for a given M.

Lemma 1 is used in our main theorem below, for which we need to define $P_{\max} = \max_k P_k$, and $c_{\max} = \max_{r,k} p_{rk}$.

Theorem 1. *If $\frac{OPT}{c_{max}} \geq KP_{max}\frac{\ln(K^2P_{max}^2/\epsilon)}{\epsilon^3}$, then we have $\mathbb{E}_M[\mathbb{E}[S(\boldsymbol{u}^*, \mathcal{A}^c)|M]] \geq \left(1 - 2\epsilon - \epsilon\sqrt{\lambda T}\mathbb{E}_M\left[\frac{1}{\sqrt{M}}\right]\right) OPT.$*

Proof. By bounding the probability of condition (12) being met, and using Lemma 1, we can prove a performance bound on the expected profit produced on $\mathcal{A}^c$. $\square$

Note that the conclusion of Theorem 1 gives us a bound on the expected performance of TDOT.

Remark 3. *The condition on $\frac{OPT}{c_{max}}$ in Theorem 1 is easily met for all practical scenarios as this is a ratio, of total profit across all tasks and processors to the profit per unit time on a single processor, which is generally a large value.*

Furthermore, by Jensen's inequality, we have $\mathbb{E}_M\left[\frac{1}{\sqrt{M}}\right] \leq \sqrt{\mathbb{E}_M[\frac{1}{M}]}$. Using this in Theorem 1, we now have

$$\mathbb{E}_M[\mathbb{E}[S(\mathbf{u}^*, \mathcal{A}^c)|M]] \geq \left(1 - 2\epsilon - \epsilon\sqrt{\lambda T \mathbb{E}_M\left[\frac{1}{M}\right]}\right) \text{OPT} \geq \left(1 - 2\epsilon - \sqrt{\epsilon\mathbb{E}_M\left[\frac{\epsilon\lambda T}{M}\right]}\right) \text{OPT}. \quad (13)$$

Thus, we can see that the profit performance gap depends on $\mathbb{E}_M\left[\frac{\epsilon\lambda T}{M}\right]$, which is the expected proportion of tasks in the training set. Furthermore, using a lower bound on $\mathbb{E}_M\left[\frac{1}{M}\right]$ if M is Poisson [18], we have the following corollary.

Corollary 1. *Assume the condition of Theorem 1 is met. If M has a Poisson distribution with mean λT, we have*

$$\mathbb{E}_M[\mathbb{E}[S(\boldsymbol{u}^*, \mathcal{A}^c)|M]] \geq \left(1 - 2\epsilon - \epsilon\sqrt{\frac{(3 + \lambda T)(1 - e^{-\lambda T})}{\lambda T}}\right) OPT. \quad (14)$$

This corollary allows precise numerical calculation. As an example, if $\lambda = 0.1$, $T = 1000s$, and we choose $\epsilon = 0.15$, then $\mathbb{E}_M[\mathbb{E}[S(\mathbf{u}^*, \mathcal{A}_c)] \geq 0.5$ OPT.

Corollary 2. *For $\lambda \to \infty$, (14) reduces to*

$$\mathbb{E}_M[\mathbb{E}[S(\boldsymbol{u}^*, \mathcal{A}^c)|M]] \geq (1 - 3\epsilon)\, OPT. \quad (15)$$

When the task arrival rate is high, there are enough tasks for training so that ϵ can be set small. In this case, Corollary 2 suggests that TDOT can perform close to an optimal offline algorithm.

Remark 4. *Instead of a single load constraint L, we can consider processor-dependent load constraints $L_{rk}, \forall r, k$, and rewrite equation (1) as follows:*

$$\sum_{j=1}^{M} t_{jrk}x_{jrk} \leq L_{rk}, \forall r \in \{1, \ldots, P_k\}, k \in \{1, \ldots, K\}.$$

We note that all results can be trivially extended to this case.

Remark 5. *Our performance bound is computed purely based on the profit from the non-training set of tasks $\mathcal{A}^c$, but it is compared against the profit of an upper-bound offline algorithm that considers the entire set of tasks. Consequently, any additional profit we obtain on training set $\mathcal{A}$ is a bonus and further improves profit performance. We note that the value of ϵ we choose splits the tasks into sets $\mathcal{A}$ and $\mathcal{A}^c$, and consequently impacts profit performance. We study the effect of ϵ on the total profit in Section VI.*

D. Complexity Analysis

An LP can be solved in $O(n^{3.5}B)$ time where n is the number of variables and B is the number of bits in the input [19]. Thus, for a given M, the dual minimization during the training phase of TDOT can be done in $O((\epsilon MP)^{3.5}B)$ time where $P = \sum_{k=1}^{K} P_k$ is the total number of processors. On the other hand, the time complexity of the exploitation phase is $O((1 - \epsilon)M)$. Thus, the time complexity of TDOT is dominated by LP-solving in the training phase.

Remark 6. *We note that $|\mathcal{A}| = \epsilon M$, and hence, the above complexity is equivalent to $O((|\mathcal{A}|P)^{3.5}B)$. This is usually small since the number of training tasks, i.e., $|\mathcal{A}|$, is much smaller relative to M.*

V. Modified Algorithm Without Partial-Task Profit Taking

The TDOT algorithm proposed in the previous section assumes that we may obtain profit on partially-completed tasks, if load constraint on the scheduled processor is met. Hence, we propose a variant, namely TDOT with Greedy scheduling or TDOT-G, for scenarios where profit can be obtained only for tasks that have fully completed execution while meeting the load constraints. This algorithm consists of the same two broad phases as that of TDOT, namely, the training phase, and the exploitation phase.

In this version, if an incoming task cannot be scheduled on the maximum profit processor, we try to schedule it on the second maximum profit processor, and then the third maximum profit processor, and so on. We expect this technique to result in better practical performance than simply discarding a task that cannot be scheduled on the maximum profit processor as we greedily try to ensure that the current task is at least executed on some processor, which will produce some profit. The following are the steps of this algorithm.

- **Step 1:** Observe the processing times of the set of the first $\lfloor \epsilon \lambda T \rfloor$ arriving tasks, $\mathcal{A}$.
- **Step 2:** Find weights $\mathbf{u*} = \text{argmin}_{\mathbf{u}} D(\mathbf{u}, \mathcal{A})$.
- **Step 3:** For each incoming task j, we initialize $\mathcal{P}$ to be the total set of processors.
  - **Step 3a:** Schedule the task to processor $(r', k') = \text{argmax}_{r,k \in \mathcal{P}} (p_{rk} - u^*_{rk}) t_{jrk}$, if $(1-\epsilon)L$ is not violated on processor r' in CS k'. If $(1-\epsilon)L$ is violated on processor r' in CS k', go to Step 3b. This violation is checked by using load variables $l_{rk}, \forall r, k$, similar to TDOT.
  - **Step 3b:** Remove (r', k') from $\mathcal{P}$ and repeat Step 3a unless $\mathcal{P}$ is empty, i.e., the task cannot be scheduled on any processor.
- **Step 4:** Stop at the end of duration T.

The overall complexity of TDOT-G is still dominated by the LP-solving step, and given by $O((|\mathcal{A}|P)^{3.5}B)$ as shown in Section IV-D.

VI. Simulation Results

We investigate the performance of our proposed algorithms with extensive simulation, using Google cluster traces with practical parameter values.

A. Comparison Targets

We use the following comparison targets to evaluate the performance of TDOT and TDOT-G:

- *Logistic Regression (LR)*: We use

$$r', k' = \underset{r,k}{\text{argmax}} \, (p_{rk} - u^*_{rk}) \, t_{jrk}$$

as the training labels for each task j in the training set $\mathcal{A}$. We then perform multi-class classification using logistic regression [20] to obtain the label for each non-training task, and schedule the task to the corresponding processor as long as $(1-\epsilon)L$ is not violated. Else, we discard the task.

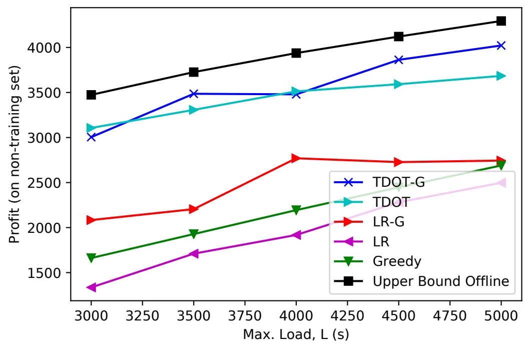

Fig. 1: Effect of max. load L on non-training set profit

- *Greedy Algorithm*: We schedule each arriving task j to processor r in CS k that maximizes $p_{rk}t_{jrk}$ as long as the total load on the processor does not exceed the load constraint. Else, we discard the task.
- *Logistic Regression - Greedy (LR-G)*: Similar to LR, but instead of discarding the task when the processor assigned by LR violates $(1-\epsilon)L$ load, we use a technique similar to TDOT-G. We schedule task j to processor $(r', k') = \text{argmax}_{r,k \in \mathcal{P}} \, p_{rk}t_{jrk}$, if $(1-\epsilon)L$ is not violated on processor r' in CS k'. If $(1-\epsilon)L$ is violated, we remove the processor from $\mathcal{P}$ and repeat until the task is scheduled or all processors are exhausted.
- *Upper Bound Offline*: Solve formulation (5) to obtain an upper-bound.

Based on whether we plot profit on just the non-training set $\mathcal{A}^c$ or the overall set of tasks, we modify these comparison targets accordingly. In Section VI-D, we ensure every comparison target obtains profit on the training set as well, for fair comparison. These details are given in Section VI-D.

B. Simulation Setup and Task Times

We consider two different CSs with two processors each. The profits are set to $p_{11} = 0.5$, $p_{12} = 0.7$, $p_{21} = 0.3$, and $p_{22} = 0.3$. We set default system duration $D = 3000$ s, maximum load $L = 3500$ s, and $\epsilon = 0.2$. We use the task events information from Google cluster data [13] to obtain the task arrival times, and compute average task per unit time $\lambda = \frac{1}{\text{average inter-arrival time}}$ from these values. We consider Poisson task arrival at the controller, so that the total number of tasks that arrive within duration T is a Poisson random variable with mean λT.

We also use the task usage information from [13], i.e., task start times and end times, to obtain task processing times. We set mean task processing time m_j = (task end time - task start time). We then consider processors with different relative speeds, $\alpha_{11} = 1$, $\alpha_{12} = 2$, $\alpha_{21} = 1.5$, and $\alpha_{22} = 1.5$, to obtain varied processing times on different processors. In the implementation of TDOT, while picking processor $(r', k') = \text{argmax}_{r,k} (p_{rk} - u^*_{rk}) \, t_{jrk}$, we randomly tie-break if there are multiple processors that give us the maximum value within a tolerance of 0.001.

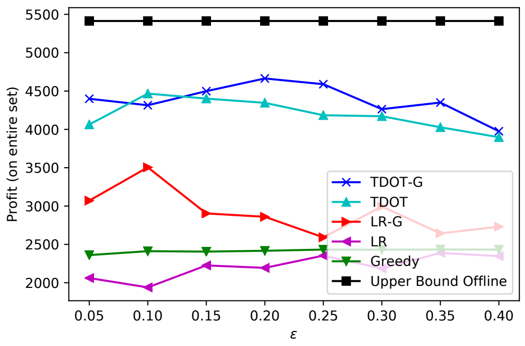

Fig. 2: Effect of ϵ on overall profit

C. Profit on Non-training Set of Tasks

In this section, we analyze the profit obtained on the non-training set of tasks, $\mathcal{A}^c$, through simulation. Figure 1 plots the profit for different values of maximum load L. We see that the profit increases as the load constraint is relaxed, as expected. TDOT and TDOT-G exhibit near-optimal performance, indicating the effectiveness of training. It outperforms LR, LR-G, and greedy for the entire range of load considered.

D. Overall Profit and ϵ Values

Although we use the set of first $\lfloor \epsilon \lambda T \rfloor$ tasks for training purposes, in practice some profit can be made on these tasks. Towards this end, we schedule each arriving task j during the training phase to the processor $(r', k') = \text{argmax}_{r,k}\, p_{rk} t_{jrk}$, as long as the total load on the processor does not violate ϵL. Adding this profit obtained on the training set to the profit obtained on the non-training set, from Section VI-C, gives us the overall profit. Thus, for fair comparison, we add this training set profit to TDOT-G, LR, and LR-G. Both greedy and upper-bound can obtain profit on the entire set of tasks, by definition.

For Figure 2, we plot the overall profit on the entire set of tasks versus ϵ for $L = 3500$. We see that the upper-bound offline and greedy solutions have performance that is independent of ϵ as expected. However, a value of $\epsilon = 0.2$ produces the best profit performance on average while using TDOT-G. This suggests that using around 20% of the expected number of tasks for training in this setting allows the algorithm to both have enough tasks to learn well but also have enough tasks to exploit the benefit of training. We note that TDOT and TDOT-G still outperform the other online alternatives, regardless of the value of ϵ chosen.

VII. Conclusion

We study the online scheduling of tasks to multiple cloud servers with an objective to maximize profit subject to load constraints. The processors in our model are heterogeneous and unary-capacity, and the tasks arrive dynamically, resulting in a challenging problem. We have proposed a polynomial-time TDOT algorithm that consists of a training phase and an exploitation phase to obtain effective scheduling solutions. We provided a performance bound for TDOT under the assumption that profit can also be obtained on partially-completed tasks if the load is already met. We also proposed a modified algorithm, TDOT-G, for implementations where profit can only be obtained on fully-completed tasks. Through trace-driven simulation, we saw that TDOT and TDOT-G consistently outperforms the comparison targets and can be tuned to exhibit near-optimal performance.

References

[1] Y. Fang, F. Wang, and J. Ge, "A task scheduling algorithm based on load balancing in cloud computing," in *Proc. International Conference on Web Information Systems and Mining*, 2010.

[2] J. P. Champati and B. Liang, "One-restart algorithm for scheduling and offloading in a hybrid cloud," in *Proc. IEEE International Symposium on Quality of Service (IWQoS)*, 2015.

[3] J. P. Champati and B. Liang, "Semi-online algorithms for computational task offloading with communication delay," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1189–1201, 2017.

[4] Z. Peng, D. Cui, J. Zuo, Q. Li, B. Xu, and W. Lin, "Random task scheduling scheme based on reinforcement learning in cloud computing," *Cluster computing*, vol. 18, no. 4, pp. 1595–1607, 2015.

[5] J. P. Champati and B. Liang, "Single restart with time stamps for computational offloading in a semi-online setting," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, 2017.

[6] 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.

[7] M. Lin, Z. Liu, A. Wierman, and L. L. Andrew, "Online algorithms for geographical load balancing," in *Proc. IEEE International Conference on Green Computing (IGCC)*, 2012.

[8] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1378–1391, 2013.

[9] F. Nwanganga, M. Saebi, G. Madey, and N. Chawla, "A minimum-cost flow model for workload optimization on cloud infrastructure," in *Proc. IEEE International Conference on Cloud Computing (CLOUD)*, 2017.

[10] H. Goudarzi and M. Pedram, "Maximizing profit in cloud computing system via resource allocation," in *Proc. IEEE Distributed Computing Systems Workshops (ICDCSW)*, 2011.

[11] L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, and F. C. Lau, "Moving big data to the cloud: An online cost-minimizing approach," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 2710–2721, 2013.

[12] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, "Online optimization for scheduling preemptable tasks on iaas cloud systems," *Journal of Parallel and Distributed Computing*, vol. 72, no. 5, pp. 666–677, 2012.

[13] J. Wilkes, "More google cluster data," *Available at https://github.com/google/cluster-data*, Nov. 2011.

[14] G. Aggarwal, G. Goel, C. Karande, and A. Mehta, "Online vertex-weighted bipartite matching and single-bid budgeted allocations," in *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 2011.

[15] N. R. Devanur and T. P. Hayes, "The adwords problem: online keyword matching with budgeted bidders under random permutations," in *Proc. ACM Conference on Electronic Commerce*, 2009.

[16] M. Chrobak, W. Jawor, J. Sgall, and T. Tichỳ, "Online scheduling of equal-length jobs: Randomization and restarts help," *SIAM Journal on Computing*, vol. 36, no. 6, pp. 1709–1728, 2007.

[17] B. Kalyanasundaram and K. Pruhs, "Maximizing job completions online," in *Proc. European Symposium on Algorithms*, 1998.

[18] E. L. Grab and I. R. Savage, "Tables of the expected value of 1/x for positive bernoulli and poisson variables," *Journal of the American Statistical Association*, vol. 49, no. 265, pp. 169–177, 1954.

[19] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proc. ACM Symposium on Theory of Computing*, 1984.

[20] B. Krishnapuram, L. Carin, M. A. Figueiredo, and A. J. Hartemink, "Sparse multinomial logistic regression: Fast algorithms and generalization bounds," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 6, pp. 957–968, 2005.