Delay and Cost Optimization in Computational Offloading Systems with Unknown Task Processing Times

Jaya Prakash Champati, *Member, IEEE,* and Ben Liang, *Fellow, IEEE*

Abstract—Computational offloading systems, where computational tasks can be processed locally or offloaded to a remote cloud, have become prevalent since the advent of cloud computing. The task scheduler in a computational offloading system decides both the selection of tasks to be offloaded to the remote cloud and the scheduling of tasks on the local processors. In this work, we consider the problem of minimizing a weighted sum of the makespan of the tasks and the offloading cost at the remote cloud. In contrast to prior works, we do not assume that the task processing times are known a priori. We show that the original problem can be solved by algorithms designed toward minimizing the maximum between the makespan and the weighted offloading cost, only with doubling of the competitive ratio. Furthermore, when the remote cloud is much faster than the local processors, the latter problem can be equivalently transformed into a makespan minimization problem with unrelated processors. For this case, we propose a Greedy-One-Restart (GOR) algorithm based on online estimation of the unknown processing times, and one-time cancellation and rescheduling of tasks that turn out to require long processing times. Given m local processors, we show that GOR has $O(\sqrt{m})$ competitive ratio, which is a substantial improvement over the best known algorithms in the literature. For the general case of arbitrary speed at the remote cloud, we extend GOR to a Greedy-Two-Restart (GTR) algorithm and show that it is $O(\sqrt{m})$-competitive. Furthermore, where tasks arrive dynamically with unknown arrival times, we extend GOR and GTR to Dynamic-GOR (DGOR) and Dynamic-GTR (DGTR), respectively, and find their competitive ratios. Finally, we discuss how GOR can be extended to accommodate multiple remote processors. In addition to performance bounding by competitive ratios, our simulation results demonstrate that the proposed algorithms are favorable also in terms of average performance, in comparison with the well-known list scheduling algorithm and other alternatives.

Index Terms—Computational offloading, edge computing, mobile cloud computing, hybrid cloud, offloading cost, semi-online algorithms

1 Introduction

Cloud computing has emerged as a vital technology used by many enterprises and individuals. The cloud infrastructure has paved way to computational offloading systems where computational tasks can be processed locally or offloaded to a remote cloud. Typical examples of computational offloading systems include hybrid cloud, Mobile Cloud Computing (MCC) systems, and Mobile Edge Computing (MEC) systems. In a hybrid cloud [2], tasks may be processed on local computing cluster owned by an enterprise or can be offloaded to reserved VMs in a public cloud. In an MCC system [3], tasks may be processed on the processor cores of a local device (e.g. smartphone, tablet etc.) or offloaded to remote servers. An MEC system [4], [5] is a special case of an MCC system where tasks are offloaded to MEC servers deployed by a cellular service provider.

Joint scheduling and offloading of tasks in a computational offloading system is an important and non-trivial problem. In this work, we consider the computational offloading system model presented in Figure 1. The m local processors are identical, and may model parallel CPU cores in a local device or processors in a local computing cluster of an enterprise. Tasks arrive at a scheduler residing in the local device or enterprise. They may be scheduled locally or offloaded to a more powerful remote cloud, which executes each task with a ρ fraction of the run time required by a local processor. However, each task offloaded to the remote cloud incurs a cost for offloading its data load, which may include multiple factors such as network bandwidth usage, transmission energy loss, etc. The scheduler at the local cluster has two important decisions to make: 1) which tasks are to be offloaded to the remote cloud, and 2) how to efficiently schedule the remaining tasks on the local processors.

We note that, in general, a scheduler may not have information about the processing time of a task until it is executed to completion [6]. Furthermore, it is known that task completion times in a public cloud are not known apirori due to random factors associated with machines and the network connecting these machines in a data center [7]. This motivates us to consider the online setting, where *the local scheduler in a hybrid cloud does not know the processing times of the tasks a priori*. We note that the algorithms developed under this setting can be used to benchmark the algorithms that assume the knowledge of the processing times of the tasks.

The problem of offloading tasks in specific computing

- *J. P. Champati is affiliated with the Division of Information Science and Engineering, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm, 11428, Sweden. E-mail: jpra@kth.se. B. Liang is affiliated with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, M5S 3G4, Canada. E-mail: liang@ece.utoronto.ca.*
- *This work was supported in part by a grant from the Natural Sciences and Engineering Research Council (NSERC) of Canada. Part of this work has been presented in [1].*

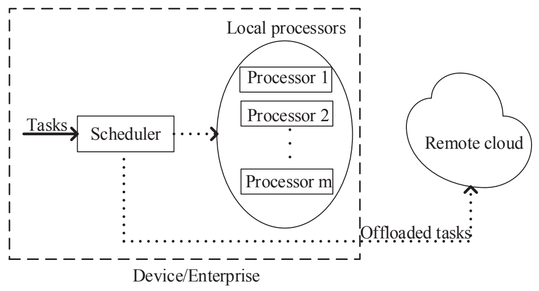

Fig. 1: Abstract model of computational offloading system

systems has received much attention in the recent literature. For example, the authors in [8], [9], [10], [11], [12] studied the problem in a hybrid cloud and the authors in [13], [14], [15] studied the problem in MCC. However, in Section 2 we will argue that these works either ignore the delay in processing tasks or ignore the costs in offloading the tasks. In this work, we jointly consider both the costs incurred in offloading tasks and the delay in processing tasks. We use makespan as the efficiency measure for delay in processing the tasks. A scheduler may reduce makespan by offloading a task to the remote cloud, but this incurs an offloading cost. Therefore, we study the joint optimization problem toward minimizing a weighted sum of the makespan and the offloading cost, which is denoted by $\mathcal{P}_{\text{sum}}$.

We note that $\mathcal{P}_{\text{sum}}$ is a challenging open problem. Even for the special case where the remote cloud has negligible processing time, i.e., $\rho = 0$, $\mathcal{P}_{\text{sum}}$ is NP-hard [16], [17]. Furthermore, to the best of our knowledge, all previous studies on this special case assume that both the processing times of the tasks and their offloading costs are known a priori [16], [17], [18], [19]. In contrast to the above works, we do not assume the knowledge of task processing times, so that solving $\mathcal{P}_{\text{sum}}$ requires *online* estimation of the processing times. We do assume that the cost for offloading the data load of a task is known a priori. This cost can be estimated given the size of the data load of a task.

Our general approach to solve this problem is as follows. We first consider a problem of minimizing the maximum of the completion times on the m local processors, completion time at the remote cloud, and the weighted offloading cost at the remote cloud, which is denoted by $\mathcal{P}_{\text{max}}$. We then show that any θ-competitive algorithm for $\mathcal{P}_{\text{max}}$ results in a 2θ-competitive algorithm for $\mathcal{P}_{\text{sum}}$. This key property motivates us to first focus on $\mathcal{P}_{\text{max}}$.

To this end we first solve the special case of $\mathcal{P}_{\text{max}}$ where $\rho = 0$. We observe that in this case, $\mathcal{P}_{\text{max}}$ is equivalent to a problem of scheduling independent tasks to minimize the makespan, on m local processors and a hypothetical processor, where the processing time of a task on the hypothetical processor is equal to its weighted offloading cost at the remote cloud. The makespan minimization problem is also known to be NP-hard even when all processing times are known a priori [20]. Further, we show that any algorithm with a pre-determined scheduling order to solve $\mathcal{P}_{\text{max}}$, when $\rho = 0$, has a competitive ratio of at least $\frac{n}{m} - 1$, where n is the number of tasks. This $\Omega(n)$ factor in the competitive ratio is due to the makespan penalty incurred by unknowingly scheduling a task with very large processing time on a local processor. Our approach to solve $\mathcal{P}_{\text{max}}$ is by using the notion of *task restart* [21], where we identify a task with large processing time during its run time, cancel it on the local processor, and offload it to the remote cloud, provided its offloading cost is not too high. This forms the basis of the proposed Greedy-One-Restart (GOR) algorithm. To identify a task that needs to be restarted during its run time, GOR uses an *estimation factor* η and the known offloading cost of each task. GOR restarts any task at most once, and we show that its competitive ratio is a convex function of η. For general $\rho > 0$, we further extend GOR and propose Greedy-Two-Restart (GTR), under which tasks may be cancelled and rescheduled twice.

The main contributions of our work are the following:

- When all tasks arrive at the scheduler at time zero, we propose the GOR algorithm. For the special case of $\rho = 0$, we prove that its competitive ratio for $\mathcal{P}_{\text{max}}$ is a convex function of the estimation factor η, given by $1 + f(\eta)$, where $f(\eta)$ is a convex function. Its competitive ratio for $\mathcal{P}_{\text{sum}}$ is then $2(1+f(\eta))$. We find the minimum of $f(\eta)$ and prove that GOR is $O(\sqrt{m})$-competitive for both $\mathcal{P}_{\text{max}}$ and $\mathcal{P}_{\text{sum}}$. Furthermore, for the case of $\rho = 0$ and $m = 1$, we show that GOR has tight 4 competitive ratio for $\mathcal{P}_{\text{max}}$. We further extend the above analysis and show that GOR is $O(\sqrt{m})$-competitive when ρ is either negligible or $\frac{1}{\rho} = O(\sqrt{m})$.
- Noting that GOR is not $O(\sqrt{m})$-competitive for general ρ, we extend GOR and propose GTR. We show that GTR has $O(\sqrt{m})$ competitive ratio for any $\rho \geq 0$. However, its average performance is generally below that of GOR.
- We further consider the weighted sum minimization problem where tasks arrive dynamically in time and their arrival times are unknown a priori, which is denoted by $\mathcal{P}^d_{\text{sum}}$. Adopting a general approach from [21], we extend GOR to Dynamic-GOR (DGOR) and GTR to Dynamic-GTR (DGTR) to accommodate this case. We show that DGOR has $O(\sqrt{m})$ competitive ratio for $\mathcal{P}^d_{\text{sum}}$ for the special cases where ρ is either negligible or $\frac{1}{\rho} = O(\sqrt{m})$. Similarly, DGTR has $O(\sqrt{m})$ competitive ratio for general ρ.
- Further simulation results suggest that, in terms of average performance, GOR provides $20 - 40\%$ improvement over the celebrated *list scheduling* algorithm [22], while DGOR provides $40 - 50\%$ improvement. However, despite its $O(\sqrt{m})$ competitive ratio for general ρ, GTR incurs higher average total cost than list scheduling. This can be attributed to the fact that under GTR some tasks may be restarted twice.
- Finally, we also discuss how to apply GOR in the case where the remote cloud is not abstracted as a single powerful server but as multiple remote processors. We propose a simple yet effective extension to GOR based on list scheduling over the unknown remote processors. We demonstrate through simulation that in this scenario GOR remains superior to existing alternatives.

The rest of this paper is organized as follows. In Section 2, we present the related work. The system model is given in Section 3. In Section 4, we provide some preliminary analysis essential to later derivations. In Section 5 we present GOR and its competitive ratio analysis for $\rho = 0$. In Section 6 we further discuss special cases of ρ for which GOR has provable competitive ratio and then propose GTR, deriving its competitive ratio for general ρ. In Section 7 we present DGOR and DGTR for dynamic task arrivals with unknown arrival times. We present simulation results in Section 8 and conclude in Section 9.

2 Related Work

In this section we first review the body of works where various computational offloading systems are studied. We then review other works that are closely related to $\mathcal{P}_{\text{sum}}$ and $\mathcal{P}_{\text{max}}$. Specifically, we focus on the special case of $\mathcal{P}_{\text{sum}}$ and $\mathcal{P}_{\text{max}}$ when $\rho = 0$, which is the only case to have been subject to rigorous analytical performance optimization in previous studies.

2.1 Computational Offloading Systems

In this subsection we review the works on hybrid cloud and MCC systems. Scheduling and offloading in a hybrid cloud has been studied under various system models [8], [9], [10], [11], [12]. The authors in [8] studied the problem of minimizing computational costs of the tasks scheduled locally and communication costs of the offloaded tasks. They formulated and solved the problem using a Markov Decision Process under the assumption that tasks have different communication costs but identical computational costs. We note that they do not consider task processing delays.

In contrast, the authors in [9], [10] studied the problem of scheduling tasks in a hybrid cloud with the objective of minimizing cost at the public cloud subject to deadline constraints of the tasks. They proposed heuristic algorithms to solve the formulated problem. The authors in [11] studied the problem of offloading tasks to the public cloud under a Lyapunov optimization framework. Their objective was to minimize the time average cost incurred at the public cloud, subject to average admission ratio being above a certain threshold, and provided a guarantee on the worst case completion time of the tasks. Similar formulation was studied in [12]. Even though [9], [10], [11], and [12] considered the delay in the processing of individual tasks, their formulations do not provide a means to optimize the makespan of the tasks. Also, in contrast to the above works we do not restrict the admission of tasks into the system, and we consider a joint optimization of cost incurred at the public cloud and the makespan of tasks.

In MCC systems [3], where a mobile device enlists the help of a remote processor in a remote cloud, most current research is focused on the task offloading problem with the objective of minimizing computational and transmission energy of the mobile device, e.g., [13], [14], [15], [23], [24], [25]. In addition, several empirical studies have been conducted on task offloading from a mobile device to remote servers [3], [26], [27]. None of these works explicitly considered the joint optimization of the cost incurred in offloading tasks and the makespan of the tasks. Therefore, none of the solutions provided in the above studies are applicable to our problem.

2.2 Minimizing Makespan Plus Offloading Cost in an Offline Setting

To the best of our knowledge $\mathcal{P}_{\text{sum}}$ has not been studied before, even for the case where task processing times are known a priori. However, for the special case where $\rho = 0$, $\mathcal{P}_{\text{sum}}$ is equivalent to minimizing the makespan plus a weighted penalty, which was of practical interest in operations research [16], mainly due to its applicability to highly loaded make-to-order production systems. In such a system, accepting all the tasks may result in an unacceptable delay to the completion time, so the production firm may reject some tasks at a penalty and aim to minimize a weighted sum of the completion time and the penalty.

The makespan-plus-weighted-penalty problem was first studied in [17], under the assumption that the processing times are known a priori. The authors considered two cases, all tasks available at time zero and tasks arriving dynamically in time. They proposed a $(2 - \frac{1}{m})$-approximation algorithm for the former case, and a Rejection-Total-Penalty algorithm that has $\frac{\sqrt{5}+3}{2}$ competitive ratio for the latter case. Further, they showed that this is the best competitive ratio any algorithm can achieve. The authors in [18] proposed a $\frac{1+\sqrt{3}}{2}$-competitive algorithm for the problem with the assumption that all the tasks have unit processing time. The authors in [19] studied the problem with $m = 2$. They proposed $\frac{3}{2}$-competitive algorithms for two variants of the problem.

We emphasize that, in addition to having solved $\mathcal{P}_{\text{sum}}$ for the special case $\rho = 0$, all of the above works require the assumption that the task processing times are known a priori. In our work, the processing time of a task is not known until the completion of its execution, and we also solve the problem for general ρ. Since the proposed algorithm GOR is $O(\sqrt{m})$-competitive for $\rho = 0$, a by-product result of our work is a first known algorithm that solves the makespan-plus-weighted-penalty problem for unknown task processing times with provable competitive ratio.

2.3 Minimizing Makespan on Parallel Processors

We note that, similar to $\mathcal{P}_{\text{sum}}$, $\mathcal{P}_{\text{max}}$ has not been studied before. However, we will show later that when $\rho = 0$, $\mathcal{P}_{\text{max}}$ is equivalent to a makespan-minimization problem. In the *offline* setting, where the task processing times are known a priori and all tasks are available at time zero, the problem of scheduling independent tasks on parallel processors to minimize the makespan has been well studied in the literature [20] [28] [29].

Works are sparse in the *online* setting, where the processing time of a task on a processor is not known until it is executed to completion. The celebrated *list scheduling* [22] is a greedy algorithm that selects a task from the given set in an arbitrary order and assigns it to whichever processor that becomes idle first. For m identical parallel processors, it has $(2 - \frac{1}{m})$ competitive ratio. For the case where the processors are unrelated, i.e., the processing times of a task on different processors are independent, an $O(\log n)$-competitive algorithm was proposed by Shmoys et. al. in [21]. When $\rho = 0$, $\mathcal{P}_{\text{max}}$ can be solved by either list scheduling or Shmoys' algorithm by ignoring the known processing times. However, we will show that the competitive ratio of list scheduling is at least $\frac{n}{m} - 1$ for $\mathcal{P}_{\text{max}}$. Shmoys' algorithm remains $O(\log n)$-competitive for $\mathcal{P}_{\text{max}}$. However, in Section 5.2 we will demonstrate that the average performance of Shmoys' algorithm is worse than list scheduling. This is due to the fact that it does task restarts based on conservative estimates of the task processing time, which may result in multiple restarts of a task. In contrast, in GOR and GTR any task experiences at most one restart and two restarts, respectively. We prove that GOR is $O(\sqrt{m})$-competitive for $\mathcal{P}_{\text{max}}$ for important special cases of ρ, including when $\rho = 0$, and GTR is $O(\sqrt{m})$-competitive for $\mathcal{P}_{\text{max}}$ for general ρ. This is a significant improvement over $O(\log n)$ noting the fact that, especially in the enterprise cloud environment, the number of tasks n is generally much larger than the number of processors m.

In [30], we also used the idea of task restart for solving a makespan minimization problem on a system of $m+1$ parallel processors, where task processing times on m processors are known and one processor are unknown. We note that the algorithm proposed in [30] cannot be used to solve our problem as the ratio of known and unknown processors in [27] is the exact opposite of that of P_{max}. Further, [30] uses the known processing times as estimates for the unknown processing times, but GOR and GTR use a design parameter η for estimating the unknown processing times.

3 System Model

We consider a hybrid cloud system model as illustrated in Figure 1. The device/enterprise has access to m identical local parallel processors or virtual machines, indexed by $i \in \mathcal{L} = \{1, \ldots, m\}$. It also has access to a remote cloud. We initially assume that the remote cloud is abstracted as a single powerful processor and refer to it by processor 0. Later, in Section 7 we show how the proposed algorithm can be extended to the case of multiple processors. We consider the remote processor as a reserved instance in the remote cloud, so that there is no further usage cost after the overhead cost of reservation.

Tasks that arrive at the scheduler may be scheduled on one of the local processors or offloaded to the remote cloud. Initially, we focus on the case where all tasks are available at time zero. In Section 7, we will extend this to the case of dynamic task arrivals with unknown arrival times.

3.1 Processing, Cloud Cost, and Scheduling

Consider n independent and non-preemptible tasks are available to the scheduler at time zero. Let $\mathcal{T} = \{1, \ldots, n\}$ be the set of task indices. The processing time of task $j \in \mathcal{T}$ on processor $i \in \mathcal{L}$ is given by u_j and is unknown. Its processing time on processor 0, when offloaded to the remote cloud, is given by ρu_j, where $\rho \in [0, \infty)$, and the factor $\frac{1}{\rho}$ denotes the relative speed of processing at the remote cloud when compared with local processing[1]. As the remote cloud is usually faster than the local processors we are primarily interested in the case $\rho \leq 1$. Nevertheless, we allow $\rho > 1$ and derive the results for the sake of completeness.

When a task j is offloaded to the remote cloud, a cost $\hat{a}_j$ is incurred for offloading its data load. It may be viewed as an aggregation of various penalties, e.g., transmission energy loss and network bandwidth usage. We assume that this cost is known for each task before it is processed, as it can be estimated using the size of its data load. We also assume that the offloading cost of a task is independent of its processing time. This assumption is appropriate in scenarios where the processing time of a task is independent of its data load size. For example, a task having a single for-loop may have data size on the order of tens of bytes. However, depending on the number of iterations in the for-loop, its processing can take a large amount of time. We further assume that the offloading time of a task is negligible compared with its processing time at the remote processor. We will see in Section 6.2 that the proposed algorithms do not depend on task data size and can still be used when the offloading times are non-negligible.

A natural objective for the device/enterprise is to minimize the makespan of the tasks scheduled on processors $\mathcal{L} \cup \{0\}$. The makespan can be reduced by offloading tasks to the remote cloud, but offloading a task incurs some cost. Hence, we consider the makespan and the offloading cost jointly, by combining them in a weighted sum.

Let $\mathbf{s}$ denote a schedule and $\mathcal{S}$ denote the set of all possible schedules. The schedule $\mathbf{s}$ decides whether to offload a task to the remote cloud or process it on one of the processors in $\mathcal{L}$. Let $\mathcal{T}_i(\mathbf{s})$ be the set of tasks scheduled on processor $i \in \mathcal{L} \cup \{0\}$ under schedule $\mathbf{s}$. Given the set of tasks at time 0, the makespan of a schedule $\mathbf{s}$ on processors from $\mathcal{L} \cup \{0\}$ is defined as the time when the processing of the last task from $\cup_{i \in \mathcal{L} \cup \{0\}} \mathcal{T}_i(\mathbf{s})$ is completed. It equals $\max_{i \in \mathcal{L} \cup \{0\}} \{C_i(\mathbf{s})\}$, where $C_i(\mathbf{s})$ is the completion time of the last task assigned to processor i and is given by

$$C_i(\mathbf{s}) = \sum_{j \in \mathcal{T}_i(\mathbf{s})} u_j, \forall i \in \mathcal{L},$$
$$C_0(\mathbf{s}) = \sum_{j \in \mathcal{T}_0(\mathbf{s})} \rho u_j.$$

Note that the schedule does not know $C_i(\mathbf{s})$ a priori, since u_j are unknown. The offloading cost of the tasks is given by $\Gamma(\mathbf{s}) = \sum_{j \in \mathcal{T}_0(\mathbf{s})} \hat{a}_j$. We define $\Upsilon(\mathbf{s}) \triangleq \max_{i \in \mathcal{L} \cup \{0\}} \{C_i(\mathbf{s})\} + w\Gamma(\mathbf{s})$ as the *total cost* of schedule $\mathbf{s}$, where the weight parameter w allows a system designer to tune the importance between makespan and offloading cost. We are interested in the following sum cost minimization problem $\mathcal{P}_{\text{sum}}$:

$$\underset{\mathbf{s} \in \mathcal{S}}{\text{minimize}} \quad \Upsilon(\mathbf{s}).$$

In the offline setting, all parameter values of the tasks are known at time 0. In this case, let $\bar{\mathbf{s}}^*$ denote an optimal

1. The speed-up ratio for different tasks between the public cloud and the local processor might be different for different tasks. To model this aspect we may consider that the processing time of task j on remote processor equals $\rho_j u_j$, where ρ_j is the speed-up ratio for task j. We note that the proposed algorithms can still be used for this case by setting $\rho = \min_j \rho_j$, and proofs for the competitive ratios for the algorithms also hold.

TABLE 1: List of symbols

m	Number of local processors
$\mathcal{L}$	Set of local processors
i	Processor index
ρ	Speed factor
n	Number of tasks
$\mathcal{T}$	Set of tasks
j	Task index
u_j	Local processing time
$\hat{a}_j$	Offloading cost
w	weight
a_j	Weighted offloading cost
$\mathbf{s}$	Schedule
$C_i(\mathbf{s})$	Completion time on processor i
$\Gamma(\mathbf{s})$	Total offloading cost
$\Upsilon(\mathbf{s})$	Total cost
η	Estimation factor (design parameter)
$C_{\max}$	Makespan
$C^*_{\max}$	Optimal makespan
$\mathbf{s}^*$	Optimal schedule
χ	Hypothetical processor with processing times a_j

schedule. $\mathcal{P}_{\text{sum}}$ is NP-hard even in the offline setting. This is because a special case of $\mathcal{P}_{\text{sum}}$, where $\rho = 0$, is known to be NP-hard [16], [17].

3.2 Semi-online Scheduling with Unknown Processing Times

In practice, the processing time required for the task generally is unknown without first processing it [21]. Therefore, we are interested in semi-online scheduling, where the processing times u_j, for all j, are not known a priori and their cloud costs $\hat{a}_j$, for all j, are known a priori.

The efficacy of an online algorithm is often measured by its competitive ratio in comparison with an optimal offline algorithm. We use the same measure for semi-online algorithm as well. Let P be a problem instance of $\mathcal{P}_{\text{sum}}$, $\mathbf{s}(P)$ be the schedule given by an online algorithm and $\bar{\mathbf{s}}^*(P)$ be the schedule given by an optimal offline algorithm. The online algorithm is said to have a competitive ratio θ if and only if

$$\max_{\forall P} \frac{\Upsilon(\mathbf{s}(P))}{\Upsilon(\bar{\mathbf{s}}^*(P))} \le \theta.$$

Furthermore, θ is said to be tight for the online algorithm if $\exists P$ such that $\Upsilon(\mathbf{s}(P)) = \theta\Upsilon(\bar{\mathbf{s}}^*(P))$.

For convenience of notation, we define $a_j \triangleq w\hat{a}_j, \forall j$. In Table 1 we summarize the notation used in this paper.

4 Preliminary Analysis

We first formally define the problem of minimizing the maximum of the makespan on the $m+1$ processors and the weighted offloading cost at the remote cloud, which is denoted by $\mathcal{P}_{\max}$:

$$\underset{\mathbf{s}\in\mathcal{S}}{\text{minimize}} \quad C_{\max}(\mathbf{s}),$$

where $C_{\max}(\mathbf{s}) \triangleq \max\{\max_{i\in\mathcal{L}\cup\{0\}}\{C_i(\mathbf{s})\}, w\Gamma(\mathbf{s})\}$. In the offline setting, let $\mathbf{s}^*$ denote the optimal schedule for $\mathcal{P}_{\max}$ and $C^*_{\max}$ denote the optimal objective value.

Note that, if $w = 0$, then $\mathcal{P}_{\max}$ is equivalent to minimizing makespan on $m+1$ parallel processors, which is an NP-hard problem [20]. Therefore, $\mathcal{P}_{\max}$ is NP-hard.

In the following proposition we establish a relation between the problems $\mathcal{P}_{\text{sum}}$ and $\mathcal{P}_{\max}$.

Proposition 1. *Any θ-competitive algorithm for $\mathcal{P}_{max}$ is a 2θ-competitive algorithm for $\mathcal{P}_{sum}$.*

Proof. Let $\mathbf{s}'$ be the computed schedule of a θ-competitive algorithm for solving $\mathcal{P}_{\max}$. We have the following inequalities.

$$\begin{aligned}
\Upsilon(\mathbf{s}') &= \max_{i\in\mathcal{L}\cup\{0\}}\{C_i(\mathbf{s}')\} + w\Gamma(\mathbf{s}') \\
&\le 2\max\{\max_{i\in\mathcal{L}\cup\{0\}}\{C_i(\mathbf{s}')\}, w\Gamma(\mathbf{s}')\} \\
&\le 2\theta\max\{\max_{i\in\mathcal{L}\cup\{0\}}\{C_i(\mathbf{s}^*)\}, w\Gamma(\mathbf{s}^*)\} \\
&\le 2\theta\max\{\max_{i\in\mathcal{L}\cup\{0\}}\{C_i(\bar{\mathbf{s}}^*)\}, w\Gamma(\bar{\mathbf{s}}^*)\} \\
&\le 2\theta[\max_{i\in\mathcal{L}\cup\{0\}}\{C_i(\bar{\mathbf{s}}^*)\} + w\Gamma(\bar{\mathbf{s}}^*)] \\
&= 2\theta\Upsilon(\bar{\mathbf{s}}^*).
\end{aligned}$$

□

We therefore conclude that an effective solution to $\mathcal{P}_{\max}$ is suitable for $\mathcal{P}_{\text{sum}}$ as well. Hence, we next focus on designing algorithms for $\mathcal{P}_{\max}$.

In the following lemmas we establish a lower bound for $C^*_{\max}$ in terms of ρ, m, the processing times and the cloud costs. These results will be later used extensively for proving the main theorems in this paper.

Lemma 1.

$$\sum_{j=1}^{n} u_j \le \left(m + \frac{1}{\rho}\right) C^*_{max}.$$

Proof. Let C^*_i denote the completion times and $\mathcal{T}^*_i$ denote the set of tasks scheduled on processor i under an optimal schedule $\mathbf{s}^*$. We have,

$$C^*_i = \sum_{j\in\mathcal{T}^*_i} u_j, \forall i \in \mathcal{L}, \tag{1}$$

$$C^*_0 = \sum_{j\in\mathcal{T}^*_0} \rho u_j. \tag{2}$$

By substituting $C^*_{\max}$ in (1) and (2) we obtain

$$C^*_{\max} \ge \sum_{j\in\mathcal{T}^*_i} u_j, \forall i \in \mathcal{L}, \tag{3}$$

$$C^*_{\max} \ge \sum_{j\in\mathcal{T}^*_0} \rho u_j. \tag{4}$$

Summing the inequalities in (3) and (4), we obtain

$$\left(m + \frac{1}{\rho}\right) C^*_{\max} \ge \sum_{j=1}^{n} u_j.$$

□

Lemma 2. *For some $\eta \ge 1$, if $\mathcal{T}' \subseteq \mathcal{T}$ is a subset of tasks satisfying $u_j \ge \eta a_j$ for all $j \in \mathcal{T}'$, then*

$$\sum_{j\in\mathcal{T}'} a_j \le \left(1 + \frac{m}{\eta}\right) C^*_{max}.$$

Proof. Let $\mathbf{s}'$ denote the optimal schedule and $C'^*_{\max}$ denote the optimal objective value, with respect to $\mathcal{P}_{\max}$, for scheduling tasks from $\mathcal{T}'$ with the assumption that the processing time of a task $j \in \mathcal{T}'$ on processor $i \in \mathcal{L}$ is ηa_j. Also, let C'^*_i denote the corresponding schedule length on processor $i \in \mathcal{L} \cup \{0\}$. We have $u_j > \eta a_j$, for all $j \in \mathcal{T}'$ and $\mathcal{T}' \subseteq \mathcal{T}$. Therefore, $C'^*_{\max} \leq C^*_{\max}$. Let $\mathcal{T}'_0 \subseteq \mathcal{T}'$ denote the subset of tasks offloaded to cloud under schedule $\mathbf{s}'$. We have

$$\frac{1}{\eta}\sum_{i=1}^{m} C'^*_i + \sum_{j \in \mathcal{T}'_0} a_j = \sum_{j \in \mathcal{T}'} a_j. \tag{5}$$

Using $C'^*_{\max} \geq C'^*_i$, for all $i \in \mathcal{L} \cup \{0\}$ and $C'^*_{\max} \geq \sum_{j \in \mathcal{T}'_0} a_j$ in (5), we obtain

$$\left(1 + \frac{m}{\eta}\right) C'^*_{\max} \geq \sum_{j \in \mathcal{T}'} a_j$$
$$\Rightarrow \left(1 + \frac{m}{\eta}\right) C^*_{\max} \geq \sum_{j \in \mathcal{T}'} a_j.$$

Hence the result is proven. □

5 The Case of Negligible ρ

In this section we consider the special case of $\mathcal{P}_{\max}$ where $\rho = 0$. This case arises when the processing speed at the remote cloud is fast enough to be considered approximately infinite compared with the speed of local processors.

5.1 Equivalence to Makespan Minimization

We show that $\mathcal{P}_{\max}$ is equivalent to a makespan minimization problem when $\rho = 0$. For a given schedule $\mathbf{s}$, define

$$C_\chi(\mathbf{s}) = \max\{C_0(\mathbf{s}), w\Gamma(\mathbf{s})\}.$$

Note that for the case $\rho = 0$, $C_\chi(\mathbf{s}) = w\Gamma(\mathbf{s}) = \sum_{j \in \mathcal{T}_0(\mathbf{s})} a_j$. Therefore, for $\rho = 0$, $C_\chi(\mathbf{s})$ can be treated as the completion time of the offloaded tasks on a hypothetical processor χ on which the processing time of a task j is a_j. We emphasize here that the use of processor χ is purely for the purpose of problem transformation and should not be mistaken for processor 0, which is an actual physical processor on which processing time of task j is ρu_j. Now, we have $C_{\max}(\mathbf{s}) = \max\{\max_{i \in \mathcal{L}} C_i(\mathbf{s}), C_\chi(\mathbf{s})\}$ which can be viewed as the makespan of the tasks from $\mathcal{T}$ when they are scheduled on $m+1$ processors, where the processing time of task j on processor $i \in \mathcal{L}$ is u_j, and its processing time on processor χ is a_j.

In the following design of GOR, we use the observation that, for $\rho = 0$, the problem $\mathcal{P}_{\max}$ is a minimization of the above makespan. As explained in Section 2.3, this is a challenging problem when there are unknown processing times.

5.2 GOR Design Consideration

The classical list scheduling has $(2 - \frac{1}{m+1})$ competitive ratio, for solving makesapn minimization on $m+1$ *identical* parallel processors [22]. In $\mathcal{P}_{\max}$ we have $m+1$ parallel processors with m of them identical but the hypothetical processor χ is independent of the other processors. Therefore, the $(2 - \frac{1}{m+1})$ competitive ratio is not applicable when list scheduling is used to solve $\mathcal{P}_{\max}$ when $\rho = 0$. In fact, in the following theorem we show that list scheduling, or any other simple deterministic semi-online algorithm that performs list scheduling using some ordering on the tasks, has at least $\frac{n}{m} - 1$ competitive ratio.

Theorem 1. *For $\rho = 0$, the competitive ratio of any semi-online algorithm that performs list scheduling using a pre-determined ordering on tasks is at least $\frac{n}{m} - 1$ with respect to $\mathcal{P}_{\max}$.*

Proof. To prove the result it is sufficient to identify a problem instance where the algorithm gives a makespan whose ratio is $\frac{n}{m} - 1$. Consider the following family of problem instances: $u_j = 1$, for all $j \in \{1, \ldots, n-m\}$, $u_j = n^2$, for all $j \in \{n-m+1, \ldots, n\}$, and $a_j = \frac{n^2}{n-m}$, for all j. Since u_j are unknown, any algorithm using some pre-determined order to schedule the tasks can only use the knowledge of a_j. However, since all a_j are equal, the tasks cannot be differentiated by such an algorithm. Therefore, this may lead it to schedule the m tasks with processing time n^2 on processors 1 through m, with one task on each processor and all the other tasks on processor χ. This will result in a makespan of n^2. However, the optimal makespan is $\frac{mn^2}{n-m}$, which is achieved by executing tasks $\{n-m+1, \ldots, n\}$ on processor χ and performing simple list scheduling for the other tasks on processors 1 through m. This results in a makespan ratio of $\frac{n}{m} - 1$. □

In light of the result in Theorem 1, we consider more sophisticated algorithms that can be used to solve the problem. For $\rho = 0$, the problem $\mathcal{P}_{\max}$ is related to minimizing makespan on unrelated parallel processors where the processing times of tasks on *none* of the processors are known. For this fully online version of the problem, Shmoys et. al. in [21] have proposed an $O(\log n)$-competitive algorithm. It estimates the processing times of the tasks and then uses an offline algorithm to schedule them. Tasks that are not completed in the estimated time are cancelled and rescheduled using the same offline algorithm. The procedure is repeated until all tasks are executed to completion.

In Figure 2, we present the average makespan achieved by Shmoys' algorithm and list scheduling, to solve $\mathcal{P}_{\max}$ for the case where $\rho = 0$, $m = 1$, $n = 100$, and u_j and a_j are generated from an exponential distribution with mean 1500. The average is computed over 5000 problem instances. The $\frac{3}{2}$-approximation algorithm given by Potts in [31] is used as the offline component in Shmoys' algorithm.

We observe that, despite its $\Omega(n)$ competitive ratio as shown in Theorem 1, list scheduling gives a shorter average makespan than Shmoys' algorithm. We conjecture that this is due to Shmoys' algorithm using crude estimates of the unknown processing times, which results in multiple restarts of some tasks. Although restarting the tasks paves the way to obtain an $O(\log n)$ competitive ratio, it penalizes the makespan on average, as the time already spent in processing a cancelled task is wasted. This motivates us to combine the virtues of both algorithms in a new semi-online design.

Neither list scheduling nor Shmoys' algorithm utilizes the known processing times on the hypothetical processor

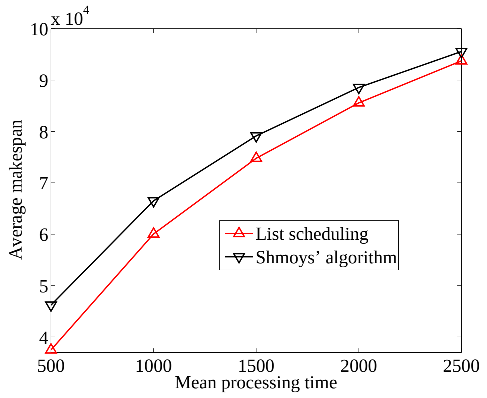

Fig. 2: Comparison of Shmoys' algorithm and list scheduling for varying mean processing time for solving $\mathcal{P}_{\max}$ for $\rho = 0$.

χ. In contrast, we design the GOR algorithm to judicially utilize the known processing times, while allowing at most *one restart* for any task. The idea behind one restart is that, the tasks that are scheduled on processors 1 to m and have large u_j compared with a_j are identified, cancelled, and rescheduled, so that they may be scheduled on processor χ in the new schedule. Cancelling a task with large u_j on some processor in $\mathcal{L}$ may allow some tasks that have smaller u_j values to be scheduled on that processor. At the same time, we avoid the wastage of time in cancelling a task more than once. A more detailed description of the GOR algorithm is given below.

5.3 GOR Algorithm Description

GOR forms a list of the tasks according to the ascending order of their known offloading costs a_j. Tasks from the start of the list are scheduled one by one on processor χ, i.e., they are offloaded to the remote processor. From the end of the list, they are scheduled on the processors in $\mathcal{L}$, i.e., the local processors, using list scheduling. GOR is not aware of the processing time of a task j that is scheduled on a processor in $\mathcal{L}$. Therefore, it uses ηa_j as as an estimate for the processing time of task j and cancels it if its run time exceeds this estimate and sets it aside. In this work we consider $\eta \geq 1$ so that a task is not cancelled until at least it is processed for a duration equivalent to its weighted offloading cost.

After going through all tasks in the above iteration, those that are cancelled are again sorted and a list is formed in the ascending order of a_j. In the next iteration, the list is scheduled using the same procedure as above, but this time we do not cancel any task, except the last one. The details of the algorithm are presented in Algorithm 1.

Note that, GOR can be readily implemented in practice as it only requires sorting of the tasks based on their offloading costs and η, which can be precomputed as will be shown shortly. GOR has two iterations and any task is restarted at most once. In each iteration, the sorting of the tasks dominates the runtime, and thus GOR has $O(n \log n)$ computational complexity. We use s^{GOR} to denote the resultant schedule.

Remark 1: The tasks scheduled on the hypothetical processor χ are the tasks offloaded to the remote cloud. In the above explanation of GOR, scheduling the next task on χ after duration a_j is equivalent to offloading the next task to the remote cloud after waiting for a duration equivalent to a_j. Also, in Algorithm 1 during the iterations where $j_0 <= j_1$, a task is simultaneously executed on processor χ and one of the processors from $\mathcal{L}$. This is because whenever processor χ becomes idle, GOR schedules the next unfinished task on it (cf. line 29 of Algorithm 1). In other words, when $j_0 <= j_1$ the task that is already being processed on some processor from $\mathcal{L}$ will be scheduled on the remote processor if it becomes idle.

Algorithm 1: Greedy-One-Restart Algorithm (GOR)

```
1:  $l = 1, \mathcal{T}^{(l)} = \mathcal{T}$
2:  while $l \leq 2$ do
3:    $j_1 = 1, j_0 = |\mathcal{T}^{(l)}| + 1$
4:    Sort $\mathcal{T}^{(l)}$ in the ascending order of $a_j$. WLOG,
      re-index tasks such that $a_1 \leq a_2 \leq \ldots \leq a_{|\mathcal{T}^{(l)}|}$.
5:    Start processing task $j_1$ on processor $\chi$
6:    for $k = 1$ to $\min\{m, |\mathcal{T}^{(l)}|\}$ do
7:      $j_0 = j_0 - 1$
8:      Start processing task $j_0$ on processor $k$.
9:      if $l = 1$ then
10:       Cancel task $j_0$ if its execution time
          exceeds $\eta a_{j_0}$, and include it in $\mathcal{T}^{(2)}$
11:     end if
12:   end for
13:   while $\mathcal{T}^{(l)} \neq \emptyset$ do
14:     Wait until next event $E$ occurs
15:     if $E =$ a task $\hat{j}$ is cancelled or completed on
        processor $\hat{i} \in \mathcal{L}$ then
16:       Cancel task $\hat{j}$ if it is scheduled on processor $\chi$
17:       $\mathcal{T}^{(l)} = \mathcal{T}^{(l)} \backslash \{\hat{j}\}$
18:       $j_0 = j_0 - 1$
19:       if $j_0 > j_1$ then
20:         Schedule task $j_0$ on processor $\hat{i}$.
21:       end if
22:       if $l = 1$ then
23:         Cancel task $j_0$ if its execution time exceeds
            $\eta a_{j_0}$, and include it in $\mathcal{T}^{(2)}$
24:       end if
25:     else if $E =$ task $j_1$ is completed on processor $\chi$
        then
26:       Cancel task $j_1$ if it is scheduled on a processor
          from $\mathcal{L}$
27:       $\mathcal{T}^{(l)} = \mathcal{T}^{(l)} \backslash \{j_1\}$
28:       $j_1 = j_1 + 1$
29:       if Task $j_1$ is not completed or cancelled yet then
30:         Schedule task $j_1$ on processor $\chi$.
31:       end if
32:     end if
33:   end while
34:   $l = l + 1$
35: end while
```

5.4 Case Study

We demonstrate the working of GOR using the family of problem instances given in the proof of Theorem 1, i.e., $u_j =$

$1, \forall j \in \{1, \ldots, n-m\}$, $u_j = n^2, \forall j \in \{n-m+1, \ldots, n\}$ and $a_j = \frac{n^2}{n-m}, \forall j$. For now we simply use $\eta = 1$. Later, we will study in detail how to choose η. As a_j are the same for all tasks, GOR cannot differentiate the tasks. Therefore, in the first iteration, the schedule given by GOR is equivalent to list scheduling with the exception that the last task is scheduled both on processor χ and one of the processors in $\mathcal{L}$. Another exception is that, in the first iteration, the processing time of any task before cancellation on processors 1 through m is $\frac{n^2}{n-m}$ (since $\eta = 1$). Therefore, any task $j \in \{n-m+1, \ldots, n\}$ scheduled on a processor in $\mathcal{L}$ will be cancelled after being processed for duration $\frac{n^2}{n-m}$. Also, any task $j \in \{1, \ldots, n-m\}$ will be finished in the first iteration since it will not be cancelled if scheduled on a processor in $\mathcal{L}$, as $\frac{n^2}{n-m} > 1 = u_j, \forall j \in \{1, \ldots, n-m\}$. In the second iteration any cancelled task $j \in \{n-m+1, \ldots, n\}$ will be finished on processor χ.

Next, to illustrate how restarting tasks with large processing times improves the worst-case bound, we find a simple upper bound for the makespan achieved by GOR for the above family of problem instances. In the worst case, GOR may schedule tasks $n-m+1$ through n on some processor $\hat{i} \in \mathcal{L}$. Each of them will be cancelled in the first iteration after being processed for duration $\frac{n^2}{n-m}$. Therefore, $\frac{mn^2}{n-m}$ time is elapsed on processor $\hat{i}$. In this duration, m tasks will be finished on processor χ and the rest of the $n-2m$ tasks can be executed on processors $\mathcal{L}\backslash\{\hat{i}\}$ in at most $\frac{n-2m}{m-1} + 1$ duration. This is because the processing time of those tasks on processors 1 through m is 1. In the second iteration all the tasks $n-m+1$ through n will be scheduled on processor χ. Therefore, the duration of execution of these tasks will be $\frac{mn^2}{n-m}$.

From the above analysis, a simple upper bound for $C_{\max}(\mathbf{s})$ is

$$\begin{aligned} C_{\max}(\mathbf{s}^{\text{GOR}}) &\leq \frac{mn^2}{n-m} + \frac{n-2m}{m-1} + 1 + \frac{mn^2}{n-m} \\ \Rightarrow \frac{C_{\max}(\mathbf{s}^{\text{GOR}})}{C^*_{\max}} &\leq \frac{n-m}{mn^2}\left(\frac{2mn^2}{n-m} + \frac{n-m}{m-1}\right) \\ &= 2 + \frac{(1-m/n)^2}{m(m-1)}. \end{aligned}$$

In the second inequality above, we have used the optimal makespan value from the proof of Theorem 1. Therefore, the makespan ratio of GOR for this family of problem instances is $O(1)$, which is a huge improvement over the ratio $\frac{n}{m} - 1$ as shown in the proof of Theorem 1 for algorithms with pre-determined scheduling order.

5.5 GOR Competitive Ratio Analysis

In this subsection, we first derive a competitive ratio for GOR as a function of the estimation factor η. We then find η that minimizes the competitive ratio.

5.5.1 *Competitive Ratio for General η*

We refer to the time to process the set of tasks $\mathcal{T}^{(l)}$ in iteration l as the *schedule length* of this iteration, denoted by $C^{(l)}_{\max}(\mathbf{s}^{\text{GOR}})$. Let $\mathbf{s}_l$ denote the intermediate schedule obtained by breaking the loop in Line 13 of Algorithm 1 as soon as j_0 becomes equal to j_1. We note that $\mathbf{s}_l$ is a schedule over the set $\mathcal{T}^{(l)}$, and all the tasks from $\mathcal{T}^{(l)}$ will be scheduled on some machine under $\mathbf{s}_l$. To understand this, in the while loop from Line 13 of Algorithm 1, when $j_0 = j_1 - 1$, all the $|\mathcal{T}^{(l)}|$ tasks should have been scheduled on some processor. Now, any more iterations in the while loop will only result in scheduling a task that is already scheduled on some processor from $\mathcal{L}$ onto processor χ. Since under $\mathbf{s}_l$ the while loop breaks when $j_0 = j_1$, there will be only one task that is scheduled on both processor χ and some processor in $\mathcal{L}$. This will be the last task scheduled by $\mathbf{s}_l$ in iteration l, and we denote it by $q^{(l)} = j_0 = j_1$.

To differentiate the terms with respect to $\mathbf{s}^{\text{GOR}}$ and $\mathbf{s}_l$, we append onto them the labels of $(\mathbf{s}^{\text{GOR}})$ and $(\mathbf{s}_l)$, respectively. We note that in iteration l, the schedule produced by $\mathbf{s}^{\text{GOR}}$ improves on $\mathbf{s}_l$. To see this, observe that $\mathbf{s}_l$ stops scheduling when $j_0 = j_1$. The step $j_0 = j_1$ also occurs under $\mathbf{s}^{\text{GOR}}$ in both iterations. However, $\mathbf{s}^{\text{GOR}}$ may not stop at this step. If processor χ completes task $q^{(l)}$ first, then $\mathbf{s}^{\text{GOR}}$ cancels task $q^{(l)}$ on a processor from $\mathcal{L}$ on which it is scheduled. Further, it schedules task $q^{(l)} + 1$ on processor χ if it is not completed yet. The above procedure is repeated till all tasks are completed. This will result in a schedule length no longer than that given by $\mathbf{s}_l$, i.e.,

$$C^{(l)}_{\max}(\mathbf{s}^{\text{GOR}}) \leq C^{(l)}_{\max}(\mathbf{s}_l). \tag{6}$$

In the following lemma we give a bound for $C^{(1)}_{\max}(\mathbf{s}^{\text{GOR}})$.

Lemma 3. *For $\rho = 0$, $C^{(1)}_{max}(\mathbf{s}^{GOR}) \leq 2\eta C^*_{max}$.*

Proof. From (6) it is sufficient to prove the lemma for $C^{(1)}_{\max}(\mathbf{s}_1)$, which we do in the following. Let $\gamma_j = \min(u_j, \eta a_j)$. Noting that $\eta \geq 1$, we have

$$\begin{aligned} C^*_{\max} &\geq \frac{1}{m+1}\sum_{j=1}^{n} \min(u_j, a_j) \\ &= \frac{1}{\eta(m+1)}\sum_{j=1}^{n} \min(\eta u_j, \eta a_j) \\ &\geq \frac{1}{\eta(m+1)}\sum_{j=1}^{n} \gamma_j. \end{aligned} \tag{7}$$

Note that $C^*_{\max}$ cannot be less than minimum processing time any task incurs in the system. Therefore,

$$C^*_{\max} \geq \min(u_j, \max(a_j, \rho u_j)), \forall j. \tag{8}$$

From (8) we obtain

$$C^*_{\max} \geq \frac{1}{\eta}\gamma_j,\ \forall j. \tag{9}$$

We also note that, in the first iteration of GOR, a task j scheduled on processor $i \in \mathcal{L}$ is processed for γ_j duration. Let $\mathcal{T}^{(1)}_i(\mathbf{s}_1) \subseteq \mathcal{T}^{(1)}$ denote the set of tasks scheduled on processor i, when $\mathbf{s}_1$ is used to schedule tasks from $\mathcal{T}^{(1)}$. We now consider the following cases for schedule $\mathbf{s}_1$.

Case 1: $C^{(1)}_{\max}(\mathbf{s}_1) = C^{(1)}_{\chi}(\mathbf{s}_1)$. For this case, task $q^{(1)}$ should have been scheduled on some processor $\hat{i} \in \mathcal{L}$, but its processing was completed first on processor χ. In other words, processing $q^{(1)}$ on processor $\hat{i}$ to completion would have increased the schedule length. Therefore, we have

$$C^{(1)}_{\max}(\mathbf{s}_1) \leq \sum_{j \in \mathcal{T}^{(1)}_{\hat{i}}(\mathbf{s}_1)} \gamma_j + \gamma_{q^{(1)}}. \tag{10}$$

Also, at time $C^{(1)}_{\max}(\mathbf{s}_1) - \gamma_{q^{(1)}}$, all the processors $i \in \mathcal{L}\backslash\{\hat{i}\}$ should have been busy executing some task, since otherwise task $q^{(1)}$ would have been scheduled on processor $i \in \mathcal{L}\backslash\{\hat{i}\}$ which is idle at that time. Therefore,

$$C^{(1)}_{\max}(\mathbf{s}_1) - \gamma_{q^{(1)}} \leq \sum_{j \in \mathcal{T}^{(1)}_{i}(\mathbf{s}_1)} \gamma_j, \ \forall i \in \mathcal{L}\backslash\{\hat{i}\}. \tag{11}$$

Note that task $q^{(1)}$ is finished on processor χ. Therefore, $q^{(1)} \notin \cup_{i\in\mathcal{L}}\mathcal{T}^{(1)}_i(\mathbf{s}_1)$. Now, from (10) and (11) we have

$$\begin{aligned} & C^{(1)}_{\max}(\mathbf{s}_1) - \gamma_{q^{(1)}} \leq \sum_{j \in \mathcal{T}^{(1)}_{i}(\mathbf{s}_1)} \gamma_j, \ \forall i \in \mathcal{L} \\ & \Rightarrow mC^{(1)}_{\max}(\mathbf{s}_1) - (m-1)\gamma_{q^{(1)}} \leq \sum_{j \in \cup_{i\in\mathcal{L}}\mathcal{T}^{(1)}_{i}(\mathbf{s}_1)} \gamma_j + \gamma_{q^{(1)}} \\ & \Rightarrow mC^{(1)}_{\max}(\mathbf{s}_1) \leq \sum_{j=1}^{n} \gamma_j + (m-1)\gamma_{q^{(1)}} \\ & \Rightarrow mC^{(1)}_{\max}(\mathbf{s}_1) \leq \sum_{j=1}^{n} \gamma_j + (m-1)\gamma_{\max}, \end{aligned} \tag{12}$$

where $\gamma_{\max} = \max_j \gamma_j$. In the third inequality above, we have used $q^{(1)} \notin \cup_{i\in\mathcal{L}}\mathcal{T}^{(1)}_i(\mathbf{s}_1)$ and $\cup_{i\in\mathcal{L}}\mathcal{T}^{(1)}_i(\mathbf{s}_1) \cup \{q^{(1)}\} \subseteq \mathcal{T}$. Now, using (7) and (9) in (12), we obtain

$$\begin{aligned} C^{(1)}_{\max}(\mathbf{s}_1) &\leq \frac{\eta(m+1)}{m} C^*_{\max} + \frac{\eta(m-1)}{m} C^*_{\max} \\ &= 2\eta C^*_{\max}. \end{aligned}$$

Case 2: $C^{(1)}_{\max}(\mathbf{s}_1) \neq C^{(1)}_{\chi}(\mathbf{s}_1)$. Let $C^{(1)}_{\max}(\mathbf{s}_1) = C^{(1)}_{\hat{i}}(\mathbf{s}_1)$ for some $\hat{i} \in \mathcal{L}$. Also, let task $\hat{j}$ be the last task completed processing on processor $\hat{i}$. Again, at time $C^{(1)}_{\max} - \gamma_{\hat{j}}$ any processor $i \in \mathcal{L}\backslash\{\hat{i}\}$ should be busy executing some task. Otherwise, scheduling task $\hat{j}$ on some processor $i \in \mathcal{L}\backslash\{\hat{i}\}$ that is idle by that time would result in a smaller schedule length, and $\mathbf{s}_1$ would have done so. Therefore,

$$\begin{aligned} C^{(1)}_{\max}(\mathbf{s}_1) &= \sum_{j \in \mathcal{T}^{(1)}_{\hat{i}}(\mathbf{s}_1)} \gamma_j, \ \forall i \in \mathcal{L}, \\ C^{(1)}_{\max}(\mathbf{s}_1) - \gamma_{\hat{j}} &\leq \sum_{j \in \mathcal{T}^{(1)}_{i}(\mathbf{s}_1)} \gamma_j, \ \forall i \in \mathcal{L}\backslash\{\hat{i}\}. \end{aligned}$$

Again, we obtain (12) by summing the above inequalities, and the result is derived using the same manipulation as in **Case 1**. ☐

We note that, a task j scheduled in the second iteration of GOR should have been scheduled on some processor $i \in \mathcal{L}$ in the first iteration and have been cancelled as $u_j \geq \eta a_j$. Therefore, for task j scheduled in the second iteration we know some information about its processing time u_j. This insight forms the basis for deriving a bound for $C^{(2)}_{\max}(\mathbf{s}^{\text{GOR}})$, which is stated in the following lemma.

Lemma 4. *For $\rho = 0$,*

$$C^{(2)}_{max}(\mathbf{s}^{GOR}) \leq \left(1 + \frac{m}{\eta}\right) C^*_{max}.$$

Proof. Note that $C^{(2)}_{\max}(\mathbf{s}^{\text{GOR}})$ cannot be greater than the schedule length of tasks from $\mathcal{T}^{(2)}$ when all of them are offloaded to the cloud. This implies

$$\begin{aligned} C^{(2)}_{\max}(\mathbf{s}^{\text{GOR}}) &\leq \max\left\{ \sum_{j\in\mathcal{T}^{(2)}} a_j, \sum_{j\in\mathcal{T}^{(2)}} \rho u_j \right\} \\ &\leq \sum_{j\in\mathcal{T}^{(2)}} a_j. \end{aligned} \tag{13}$$

In the second inequality above we have used $\rho = 0$. Since $u_j \geq \eta a_j$, for all $j \in \mathcal{T}^{(2)}$, the lemma follows from (13) and Lemma 2. ☐

Noting that $C_{\max}(\mathbf{s}^{\text{GOR}}) = C^{(1)}_{\max}(\mathbf{s}^{\text{GOR}}) + C^{(2)}_{\max}(\mathbf{s}^{\text{GOR}})$, the following theorem is a direct consequence of Lemmas 3 and 4.

Theorem 2. *For $\rho = 0$,*

$$\frac{C_{max}(\mathbf{s}^{GOR})}{C^*_{max}} \leq 1 + f(\eta),$$

where

$$f(\eta) = 2\eta + \frac{m}{\eta}.$$

5.5.2 *Minimizing the Competitive Ratio*

One interesting feature of GOR is that the competitive ratio of the algorithm can be tuned by choosing an appropriate value for η. By using a large η, in the first iteration, we allow the tasks to run for a longer duration before cancellation and the worst-case bound for $C^{(1)}_{\max}$ increases. On the other hand, using a small η value results in aggressive cancellation of the tasks in the first iteration and the worst-case bound for $C^{(2)}_{\max}$ increases.

Hence, we consider the following optimization problem to minimize the upper bound of the competitive ratio:

$$\underset{\eta \geq 1}{\text{minimize}} \quad f(\eta). \tag{14}$$

The solution to (14) is given by

$$\eta = \begin{cases} 1 & m = 1, \\ \sqrt{\frac{m}{2}} & m \geq 2. \end{cases} \tag{15}$$

Substituting η in Theorem 2, we get

$$\frac{C_{\max}(\mathbf{s}^{GOR})}{C^*_{\max}} \leq \begin{cases} 4 & m = 1, \\ 1 + 2\sqrt{2m} & m \geq 2. \end{cases}$$

Therefore, GOR is $O(\sqrt{m})$-competitive for $\mathcal{P}_{\max}$. It has the same competitive order for $\mathcal{P}_{\text{sum}}$ by Proposition 1. We state this in the following theorem.

Theorem 3. *For $\rho = 0$, GOR is $O(\sqrt{m})$-competitive for $\mathcal{P}_{max}$ and $\mathcal{P}_{sum}$.*

Remark 2: We note that the proofs of Lemmas 3 and 4 do not require the tasks to be ordered in the ascending order of their costs. Therefore, GOR without the sorting step will still have $O(\sqrt{m})$ competitive ratio. However, we have observed that for problem instances generated randomly from typical distributions, sorting reduces the total cost on average.

5.6 Tight Competitive Ratio for $m = 1$

For $m = 1$, the competitive ratio of GOR is 4. In the following theorem we state that this competitive ratio is tight.

Theorem 4. *For $\rho = 0$, $m = 1$ and choosing $\eta = 1$, GOR has tight 4 competitive ratio for $\mathcal{P}_{max}$.*

Proof. To show that the competitive ratio is tight, we provide the following problem instance for which the competitive ratio is achieved by GOR. Consider $n = 8$ and the task processing times are as follows:

$$a_j = \begin{cases} 10 - \delta & j = 1, 2, 3, 4 \\ 10 & j = 5, 6, 7, 8, \end{cases}$$

$$u_j = \begin{cases} \delta & j = 1, 2, 3, 4 \\ 40 & j = 5, \\ 10 + \delta & j = 6, 7, 8, \end{cases}$$

where δ is a positive real number close to 0. GOR lists the tasks in the ascending order of a_j. Tasks 1 through 4 are scheduled on processor χ and tasks 5 through 8 are scheduled on processor 1 in the first iteration. Since $u_j > a_j, \forall j \in \{5, 6, 7, 8\}$, all the tasks scheduled on processor 1 will be cancelled. Therefore, the schedule length in the first iteration is $40 - 4\delta$. Tasks 5 through 8 have the same a_j. Therefore, in the second iteration, GOR cannot differentiate the tasks and may schedule task 5 on processor 1 and tasks 6 through 8 on processor 2. In this case the schedule length in the second iteration is 40. This results in a makespan $C_{\max}(\mathbf{s}^{\text{GOR}}) = 80 - 4\delta$.

The optimal schedule $\mathbf{s}^*$ is the following. Schedule tasks $1, 2, 3, 4, 6, 7$ on processor 1 and tasks $5, 8$ on processor χ. The optimal maskespan is $C^*_{\max} = 20 + 6\delta$. Since δ can be chosen arbitrarily close to 0, the competitive ratio 4 is achieved. □

The significance of Theorem 4 is the following. When $\rho = 0$ and $m = 1$, $\mathcal{P}_{\max}$ is a semi-online version of the problem of minimizing the makespan on two unrelated parallel processors. For the offline version of the problem, where the processing times of the tasks on both the processors are known, the authors in [31] gave a $\frac{3}{2}$-approximation algorithm. Further, $\frac{3}{2}$ is the lower bound on the approximation ratio that any polynomial-time algorithm can achieve, unless P=NP [32]. On the other extreme, in the fully online version of the problem, where the processing times of the tasks on neither processors are known, an $O(\log n)$-competitive algorithm was given in [21]. GOR has constant-competitive ratio for solving the semi-online version of the problem, where the processing times of the tasks on one processor are known and those on the other processor are unknown. We observe substantial improvement in the achievable competitive ratio when the processing times on one processor become available. Further, the competitive ratio 4 compares well with the lower bound $\frac{3}{2}$ in the offline case.

6 The case of General ρ

In this section, we first study some additional special cases of ρ values where GOR has a provable competitive ratio. We then extend GOR and propose Greedy-Two-Restart (GTR). We provide competitive ratio analysis for GTR showing that it has $O(\sqrt{m})$ competitive ratio for $\mathcal{P}_{\max}$ and $\mathcal{P}_{\text{sum}}$ for general ρ.

6.1 Competitive Ratio of GOR for Small and Large ρ Values

In the following lemmas we give new upper bounds for the schedule lengths of GOR for $\rho > 0$.

Lemma 5. *For $\rho > 0$,*

$$C^{(1)}_{max}(\mathbf{s}^{GOR}) \leq \min(2\eta, \zeta) C^*_{max},$$

where

$$\zeta = \left(1 + \frac{1}{m\rho} + \frac{(m-1)}{m} \max(1, 1/\rho)\right).$$

Proof. We note that the proof of Lemma 3 does not depend on the value of ρ. Therefore, we have $C^{(1)}_{\max}(\mathbf{s}^{\text{GOR}}) \leq 2\eta C^*_{\max}$. To prove the other part of the bound, in the following we reuse the inequalities from the proof of Lemma 3.

From (8) we obtain

$$C^*_{\max} \geq \min(1, \rho) u_j \geq \min(1, \rho) \gamma_j, \ \forall j. \tag{16}$$

Now, substituting in (12) the results from Lemma 1, and (16), we obtain

$$C^{(1)}_{\max}(\mathbf{s}_1) \leq \left(1 + \frac{1}{m\rho} + \frac{(m-1)}{m} \max(1/\rho, 1)\right) C^*_{\max},$$

where $\mathbf{s}_1$ is defined in the proof of Lemma 3. The result follows by noting that $C^{(1)}_{\max}(\mathbf{s}^{\text{GOR}}) \leq C^{(1)}_{\max}(\mathbf{s}_1)$. □

Lemma 6. *If $\frac{1}{\rho} \geq \max_j \frac{u_j}{a_j}$, then*

$$C^{(2)}_{max}(\mathbf{s}^{GOR}) \leq \left(1 + \frac{m}{\eta}\right) C^*_{max}$$

Proof. The proof is similar to the proof of Lemma 4 and is omitted. □

We now consider the scenario of small ρ values, where $\frac{1}{\rho} \geq \max_j \frac{u_j}{a_j}$. For this case we obtain the following competitive ratio for GOR in terms of η.

Lemma 7. *If $\frac{1}{\rho} \geq \max_j (\frac{u_j}{a_j})$, then*

$$\frac{C_{max}(\mathbf{s}^{GOR})}{C^*_{max}} \leq 1 + f(\eta),$$

where

$$f(\eta) = 2\eta + \frac{m}{\eta}.$$

Proof. Noting that $C_{\max}(\mathbf{s}^{\text{GOR}}) = C^{(1)}_{\max}(\mathbf{s}^{\text{GOR}}) + C^{(2)}_{\max}(\mathbf{s}^{\text{GOR}})$, the theorem is a direct consequence of Lemmas 5 and 6. □

From Lemma 7 we note that the competitive ratio achieved by GOR for $\frac{1}{\rho} \geq \max_j (\frac{u_j}{a_j})$ is the same as that of the case of $\rho = 0$ in Theorem 2. Therefore, using same analysis that followed Theorem 2 we can prove that GOR is $O(\sqrt{m})$ competitive for $\frac{1}{\rho} \geq \max_j (\frac{u_j}{a_j})$.

Next, we consider the scenario of large ρ values, where $\frac{1}{\rho} = O(\sqrt{m})$. Using this in ζ, we obtain $\zeta = O(\sqrt{m})$. Therefore, from Lemma 5 we have

$$\frac{C^{(1)}_{\max}(\mathbf{s}^{GOR})}{C^{*}_{\max}} = O(\sqrt{m}), \text{ if } \frac{1}{\rho} = O(\sqrt{m}). \tag{17}$$

We note that the above upper bound for $C^{(1)}_{\max}(\mathbf{s}^{\text{GOR}})$ is independent of η. Therefore, for this scenario we choose $\eta = \infty$, i.e., for $\frac{1}{\rho} = O(\sqrt{m})$, GOR does not do any restarts of the tasks. Since in this case $C_{\max}(\mathbf{s}^{GOR}) = C^{(1)}_{\max}(\mathbf{s}^{GOR})$, from (17) we conclude that GOR is $O(\sqrt{m})$-competitive. The above analysis leads to our main theorem below.

Theorem 5. *If $\frac{1}{\rho} \geq \max_j(\frac{u_j}{a_j})$, then choose η given in* (15)*. If $\frac{1}{\rho} = O(\sqrt{m})$, then choose $\eta = \infty$. For both these scenarios GOR is $O(\sqrt{m})$-competitive for $\mathcal{P}_{max}$ and $\mathcal{P}_{sum}$.*

Discussion

Note that Theorem 5 does not provide any performance guarantee for GOR for $\frac{1}{\rho} < \max_j(\frac{u_j}{a_j})$. For this scenario we choose the value of η given in (15). Therefore, in our implementation of GOR this is the default value of η unless $\frac{1}{\rho} = O(\sqrt{m})$, in which case we set η to a large value such that no task is restarted. Since GOR is not $O(\sqrt{m})$-competitive for $\mathcal{P}_{\text{sum}}$ for general ρ, in the next section we propose the GTR algorithm, which has $O(\sqrt{m})$ competitive ratio for general ρ.

6.2 The GTR Algorithm

To improve on the competitive performance of GOR for general ρ, we propose GTR and show that it has $O(\sqrt{m})$ competitive ratio for $\mathcal{P}_{\max}$ and $\mathcal{P}_{\text{sum}}$. The GTR algorithm is an extension of GOR as follows. GTR has the same first iteration as GOR, where a list is formed by sorting the tasks in the ascending order of a_j. Tasks from the start of the list are offloaded to the remote processor, and tasks from the end of the list are scheduled locally. If the task processing time of task j on a local processor exceeds ηa_j it is cancelled and is kept aside. In the second iteration of GTR, the cancelled tasks from the first iteration are scheduled in the same manner as in the first iteration. However, in contrast to GOR which does not cancel tasks in its second iteration, GTR cancels a task if its processing on a processor from $\mathcal{L}$ exceeds $\frac{a_j}{\rho}$ or its processing on processor χ exceeds a_j. We will see later that this step will aid in deriving the $O(\sqrt{m})$ competitive ratio for GTR. The cancelled tasks from the second iteration are scheduled using the same procedure as in the first and second iterations, but this time they are not cancelled.

The details of GTR are presented in Algorithm 2. Note that GTR has three iterations and a task may be restarted at most twice. In each iteration, the algorithm performs sorting of the tasks which takes $O(n \log n)$ time, and a fixed number of operations to schedule each task which takes $O(n)$ time. Since the dominating operation in GTR is sorting, its runtime complexity is $O(n \log n)$. We use $\mathbf{s}^{\text{GTR}}$ to denote the resultant schedule.

Algorithm 2: Greedy-Two-Restart Algorithm (GTR)

```
1:  $l = 1, \bar{\mathcal{T}}^{(l)} = \mathcal{T}$
2:  while $l \leq 3$ do
3:      $j_1 = 1, j_0 = |\bar{\mathcal{T}}^{(l)}| + 1$
4:      Sort $\bar{\mathcal{T}}^{(l)}$ in the ascending order of $a_j$. WLOG,
        re-index tasks such that $a_1 \leq a_2 \leq \ldots \leq a_{|\bar{\mathcal{T}}^{(l)}|}$.
5:      Start processing task $j_1$ on processor $\chi$
6:      if $l = 2$ then
7:          Cancel task $j_1$ if its execution time exceeds $a_{j_0}$,
            and include it in $\bar{\mathcal{T}}^{(3)}$
8:      end if
9:      for $k = 1$ to $\min\{m, |\bar{\mathcal{T}}^{(l)}|\}$ do
10:         $j_0 = j_0 - 1$
11:         Start processing task $j_0$ on processor $k$.
12:         if $l = 1$ then
13:             Cancel task $j_0$ if its execution time
                exceeds $\eta a_{j_0}$, and include it in $\bar{\mathcal{T}}^{(2)}$
14:         end if
15:         if $l = 2$ then
16:             Cancel task $j_0$ if its execution time
                exceeds $\frac{a_{j_0}}{\rho}$, and include it in $\bar{\mathcal{T}}^{(3)}$
17:         end if
18:     end for
19:     while $\bar{\mathcal{T}}^{(l)} \neq \emptyset$ do
20:         Wait until next event $E$ occurs
21:         if $E =$ a task $\hat{j}$ is cancelled or completed on
            processor $\hat{i} \in \mathcal{L}$ then
22:             Cancel task $\hat{j}$ if it is scheduled on processor $\chi$
23:             $\bar{\mathcal{T}}^{(l)} = \bar{\mathcal{T}}^{(l)} \backslash \{\hat{j}\}$
24:             $j_0 = j_0 - 1$
25:             If $j_0 > j_1$, schedule it on processor $\hat{i}$.
26:             if $l = 1$ then
27:                 Cancel task $j_0$ if its execution time exceeds
                    $\eta a_{j_0}$, and include it in $\bar{\mathcal{T}}^{(2)}$
28:             end if
29:             if $l = 2$ then
30:                 Cancel task $j_0$ if its execution time exceeds
                    $\frac{a_{j_0}}{\rho}$, and include it in $\bar{\mathcal{T}}^{(3)}$
31:             end if
32:         else if $E =$ task $j_1$ is completed on processor $\chi$
            then
33:             Cancel task $j_1$ if it is scheduled on a processor
                from $\mathcal{L}$
34:             $\bar{\mathcal{T}}^{(l)} = \bar{\mathcal{T}}^{(l)} \backslash \{j_1\}$
35:             $j_1 = j_1 + 1$
36:             If task $j_1$ is not completed or cancelled yet,
                schedule it on processor $\chi$
37:             if $l = 2$ then
38:                 Cancel task $j_1$ if its execution time exceeds $a_{j_0}$,
                    and include it in $\bar{\mathcal{T}}^{(3)}$
39:             end if
40:         end if
41:     end while
42:     $l = l + 1$
43: end while
```

Competitive Ratio Analysis

We use $\bar{\mathcal{T}}^{(l)}$ to denote the set of tasks in iteration l and $C^{(l)}_{\max}(\mathbf{s}^{\text{GTR}})$ to denote the corresponding schedule length. Since GTR is an extension of GOR, the upper bounds of the schedule lengths of GTR are similar to those of GOR. In the following lemmas we present these upper bounds.

Lemma 8.

$$C^{(1)}_{max}(\mathbf{s}^{GTR}) \le \min(2\eta, \zeta) C^*_{max}$$

Proof. Noting that the first iteration of GTR is the same as in GOR, the result is a direct consequence of Lemma 3. □

Again note that in the first iteration of GTR, any task j that is cancelled should satisfy $u_j > \eta a_j$. Therefore,

$$u_j > \eta a_j, \ \forall j \in \bar{\mathcal{T}}^{(2)}. \tag{18}$$

Lemma 9.

$$C^{(2)}_{max}(\mathbf{s}^{GTR}) \le \left(1 + \frac{m}{\eta}\right) C^*_{max}.$$

Proof. Noting the property of $\bar{\mathcal{T}}^{(2)}$ from (18), the result is proved by using the same steps of the proof of Lemma 4. □

Lemma 10.

$$C^{(3)}_{max}(\mathbf{s}^{GTR}) \le \left(\frac{\sqrt{1+4m}+1}{2}\right) C^*_{max}.$$

Proof. Since $\bar{\mathcal{T}}^{(3)} \subseteq \mathcal{T}$, from Lemma 1, we have

$$\left(m + \frac{1}{\rho}\right) C^*_{\max} \ge \sum_{j\in\mathcal{T}} u_j \ge \sum_{j\in\bar{\mathcal{T}}^{(3)}} u_j. \tag{19}$$

Note that in the second iteration of GTR, a task is cancelled if its processing on a processor from $\mathcal{L}$ exceeds $\frac{a_j}{\rho}$ or its processing on processor χ exceeds a_j. This implies

$$\rho u_j \ge a_j, \forall j \in \bar{\mathcal{T}}^{(3)}. \tag{20}$$

Therefore, we have $C^{(3)}_{\max}(\mathbf{s}^{\text{GTR}}) = \max_{i\in\mathcal{L}\cup\{0\}}\{C^{(3)}_i(\mathbf{s}^{\text{GTR}})\}$. Next, we claim that the following inequality holds.

$$\left(m + \frac{1}{\rho}\right) C^{(3)}_{\max}(\mathbf{s}^{\text{GTR}}) \le \sum_{j\in\bar{\mathcal{T}}^{(3)}} u_j + m u_{q^{(3)}}, \tag{21}$$

where $q^{(3)}$ is the last task scheduled under the intermediate schedule $\mathbf{s}_3$, which is defined in Section 5. From (6) we conclude that, to justify our claim it is sufficient to prove (21) is true when $\mathbf{s}^{\text{GTR}}$ is replaced by $\mathbf{s}_3$. To this end we consider the following cases.

Case 1: $C^{(3)}_{\max}(\mathbf{s}_3) = C^{(3)}_0(\mathbf{s}_3)$. We note that, in this case the last task $q^{(3)}$ is completed on processor 0. Scheduling task $q^{(3)}$ on some processor $i \in \mathcal{L}$ and completing it on that processor would increase the schedule length $C^{(3)}_i(\mathbf{s}_3)$ beyond $C^{(3)}_{\max}(\mathbf{s}_3)$. This implies

$$C^{(3)}_{\max}(\mathbf{s}_3) - u_{q^{(3)}} \le C^{(3)}_i(\mathbf{s}_3), \forall i \in \mathcal{L}. \tag{22}$$

Using (22) along with

$$C^{(3)}_{\max}(\mathbf{s}_3) = \rho \sum_{j\in\bar{\mathcal{T}}^{(3)}_0} u_j,$$

$$C^{(3)}_i(\mathbf{s}_3) = \sum_{j\in\bar{\mathcal{T}}^{(3)}_i} u_j, \forall i \in \mathcal{L},$$

and $\cup_i \bar{\mathcal{T}}^{(3)}_i = \bar{\mathcal{T}}^{(3)}$, we obtain

$$\sum_{i=1}^{m}[C^{(3)}_{\max}(\mathbf{s}_3) - u_{q^{(3)}}] + \frac{1}{\rho} C^{(3)}_{\max}(\mathbf{s}_3) \le \sum_{j\in\bar{\mathcal{T}}^{(3)}} u_j$$

$$\Rightarrow \left(m + \frac{1}{\rho}\right) C^{(3)}_{\max}(\mathbf{s}_3) \le \sum_{j\in\mathcal{T}^{(3)}} u_j + m u_{q^{(3)}}.$$

Case 2: $C^{(3)}_{\max}(\mathbf{s}_3) = C^{(3)}_i(\mathbf{s}_3)$ for some $i \in \mathcal{L}$. The proof of this case follows similar arguments as in **Case 1** and is omitted.

Now, using (6), (19), and $C^*_{\max} \ge \min(1,\rho) u_{q^{(3)}}$ in (21), we obtain

$$C^{(3)}_{\max}(\mathbf{s}^{\text{GTR}}) \le \left(1 + \frac{\min(1,\rho)m}{\rho m + 1}\right) C^*_{\max}. \tag{23}$$

Note that, if $\rho \ge 1$, from (23) the ratio $\frac{C^{(3)}_{\max}(\mathbf{s}^{\text{GTR}})}{C^*_{\max}} = O(1)$. Therefore, the lemma is true for $\rho \ge 1$. Hence, in the following we consider $\rho < 1$.

We note that $C^{(3)}_{\max}(\mathbf{s}^{\text{GTR}})$ cannot be greater than the schedule length that is a result of scheduling all tasks from $\bar{\mathcal{T}}^{(3)}$ on processor 0. Therefore,

$$\begin{aligned} C^{(3)}_{\max}(\mathbf{s}^{\text{GTR}}) &\le \rho \sum_{j\in\bar{\mathcal{T}}^{(3)}} u_j \\ &\le (1 + \rho m) C^*_{\max}. \end{aligned} \tag{24}$$

In the last inequality above we have used (19). From (23) and (24), we get

$$C^{(3)}_{\max}(\mathbf{s}^{\text{GTR}}) \le \left[1 + \min\left(\rho m, \frac{m}{\rho m + 1}\right)\right] C^*_{\max}. \tag{25}$$

Note that in (25) the term ρm increases with ρ and the term $\frac{m}{\rho m+1}$ decreases with ρ. Therefore, a simple upper bound for $\min\left(\rho m, \frac{m}{\rho m+1}\right)$ is obtained by solving

$$\rho m = \frac{m}{\rho m + 1}.$$

The solution to the above equation is given by

$$\rho = \frac{\sqrt{1+4m}-1}{2m}. \tag{26}$$

Substituting (26) in (25) gives the desired result. □

Remark 3: We note that the proofs of Lemmas 8 , 9 and 10 do not require the tasks to be ordered in the ascending order of their costs. Therefore, similar to GOR, GTR without the sorting step will still have $O(\sqrt{m})$ competitive ratio. However, we have observed that for problem instances generated randomly from typical distributions, sorting helps in reducing the total cost on average.

We now state our second main theorem below.

Theorem 6. *GTR is $O(\sqrt{m})$-competitive for $\mathcal{P}_{max}$ and $\mathcal{P}_{sum}$.*

Proof. We have $C_{\max}(\mathbf{s}^{\text{GTR}}) = C^{(1)}_{\max}(\mathbf{s}^{\text{GTR}}) + C^{(2)}_{\max}(\mathbf{s}^{\text{GTR}}) + C^{(3)}_{\max}(\mathbf{s}^{\text{GTR}})$. Therefore, from Lemmas 8, 9 and 10, we obtain

$$\frac{C_{\max}(\mathbf{s}^{\text{GTR}})}{C^*_{\max}} \le 1 + f(\eta) + \frac{\sqrt{1+4m}+1}{2}.$$

Using η given in (15), which minimizes $f(\eta)$, we obtain

$$\frac{C_{\max}(\mathbf{s}^{\text{GTR}})}{C^*_{\max}} \le \begin{cases} \frac{9+\sqrt{5}}{2} & m=1, \\ 1+2\sqrt{2m}+\frac{\sqrt{1+4m}+1}{2} & m\ge 2. \end{cases}$$

Therefore, the result is true for $\mathcal{P}_{\text{max}}$, and from Proposition 1 it is true for $\mathcal{P}_{\text{sum}}$. □

Remark 4: Note that, in the third iteration of GTR we have $\rho u_j > a_j, \forall j \in \bar{\mathcal{T}}^{(3)}$. Therefore, we have $C^{(3)}_{\max}(\mathbf{s}^{\text{GTR}}) = \max_{i\in\mathcal{L}\cup\{0\}}\{C^{(3)}_i(\mathbf{s}^{\text{GTR}})\}$, since the offloading cost is less than the completion time on processor 0. Therefore, the problem of scheduling tasks from $\bar{\mathcal{T}}^{(3)}$ to minimize the makespan is equivalent to the problem of minimizing makespan on $m+1$ uniform parallel processors, where m are identical processors and the processing times of the tasks are not known. For this case, using list scheduling on $\bar{\mathcal{T}}^{(3)}$ until at most m tasks are left unfinished, and then using Shmoys' algorithm for scheduling those m tasks with LPT as the offline component algorithm, we get $8\log m + 17$ competitive ratio [21], [33]. However, as noted in Section 5.2, using Shmoys' algorithm results in poor average performance.

Remark 5: We note that in the second iteration of GTR, a task offloaded to the cloud may be restarted. Even though this step facilitates our proof of the $O(\sqrt{m})$ competitive ratio for GTR, it penalizes the sum cost objective. For example, if a task j is cancelled at the cloud in the second iteration, then the cloud cost a_j paid for the task is wasted. If in the third iteration, the task j is again scheduled at the cloud, then it incurs an additional cost a_j. We also note that cancelling tasks in the first and second iterations of GTR on processors from $\mathcal{L}$ also penalizes the makespan part in the objective. From this we infer that the benefit of restarting a task multiple times will be mitigated by the penalty it incurs to makspan, and results in a poor average performance. Therefore, we do not investigate algorithms which allow more than two restarts.

Remark 6: For the case where the offloading times of the tasks are non-negligible, the completion time on the remote processor takes a complex form given in [34]. Nevertheless, both GOR and GTR can be used in this case as they do not assume any knowledge about the completion time of a task on the remote processor a priori.

7 Extensions

In this section we present two extensions to our system model. First, we consider dynamic task arrivals and show how GOR and GTR can be extended for this case. We then describe an extension to GOR for the case of multiple remote processors.

7.1 Dynamic Task Arrivals

So far we have assumed that all n tasks are available to be scheduled at time zero. In practice, the tasks may arrive dynamically in time, and their arrival times may not be known a priori. We consider $\mathcal{P}_{\text{sum}}$ and $\mathcal{P}_{\text{max}}$ under such dynamic task arrivals, and relabel them as $\mathcal{P}^d_{\text{sum}}$ and $\mathcal{P}^d_{\text{max}}$, respectively.

Given a θ-competitive algorithm for $\mathcal{P}_{\text{max}}$, we may adopt the general approach proposed in [21] to extend the algorithm to one that has a 2θ competitive ratio for $\mathcal{P}^d_{\text{max}}$. We use this general approach to extend GOR, termed Dynamic-GOR (DGOR), which is described as follows. Without loss of generality, suppose there is at least one task available at time 0. Let $\mathcal{T}(0)$ be the set of tasks available at time 0. Schedule $\mathcal{T}(0)$ using GOR. Accumulate the tasks that arrive while waiting for the time when all the tasks scheduled from $\mathcal{T}(0)$ are finished. Then schedule the accumulated tasks using GOR. Again, accumulate tasks and repeat the above procedure until no more tasks are available to be scheduled. Using the same approach above we extend GTR, termed Dynamic-GTR (DGTR).

We note that Proposition 1 holds for problems $\mathcal{P}^d_{\text{sum}}$ and $\mathcal{P}^d_{\text{max}}$ as well. Therefore, an extended algorithm for $\mathcal{P}^d_{\text{max}}$ will have 4θ competitive ratio for $\mathcal{P}^d_{\text{sum}}$. Using this result along with Theorem 5 we conclude that DGOR is $O(\sqrt{m})$-competitive for $\mathcal{P}^d_{\text{max}}$ and $\mathcal{P}^d_{\text{sum}}$ when $\frac{1}{\rho} \ge \max_j\left(\frac{u_j}{a_j}\right)$ or $\frac{1}{\rho} = O(\sqrt{m})$. Similarly, DGTR is $O(\sqrt{m})$-competitive for both $\mathcal{P}^d_{\text{max}}$ and $\mathcal{P}^d_{\text{sum}}$ for general ρ.

7.2 Multiple Remote Processors

For the case where the remote cloud is not abstracted as a single powerful server but as multiple remote processors, we propose a simple extension to GOR. In this case, in addition to offloading a task, GOR has to assign it to one of the remote processors. Since the processing times of the tasks are unknown on the remote processors, we have to choose an online algorithm, namely, list scheduling or Shmoys' algorithm, to do this assignment. We use list scheduling in our implementation as Shmoys' algorithm was shown to have poor average performance (Section 5.2).

8 Average Performance Comparison

In addition to the proven worst-case bounds via competitive ratios presented in the previous sections, we next study the average performance of the proposed algorithms over randomly generated problem instances, for general ρ, using simulation in MATLAB. We first present simulation results for the case of all tasks available at time zero and then present the results for the case of dynamic task arrivals.

8.1 All Tasks Available at Time Zero

For the purpose of comparison we consider the online list scheduling algorithm [22], and two semi-online algorithms, namely, Semi-Online Highest-Cost-First (SO-HCF) and Semi-Online Least-Cost-First (SO-LCF). Further, for a base-line comparison, we also propose an offline algorithm called Offline Processing time to weighted-offloading-Cost Ratio (Offline-PCR), that uses the task processing times u_j and the weighted offloading costs a_j intelligently.

1) **SO-HCF and SO-LCF:** In SO-HCF we list the tasks in descending order of their cloud costs and then perform list scheduling on the processors. SO-LCF is the same as SO-HCF except that the tasks are listed in ascending order.
2) **Offline-PCR:** Under Offline-PCR, we assume that u_j are known apriori. A list is formed by sorting the tasks in the ascending order of their processing time to weighted-offloading cost ratio, i.e., $\frac{u_j}{a_j}$.

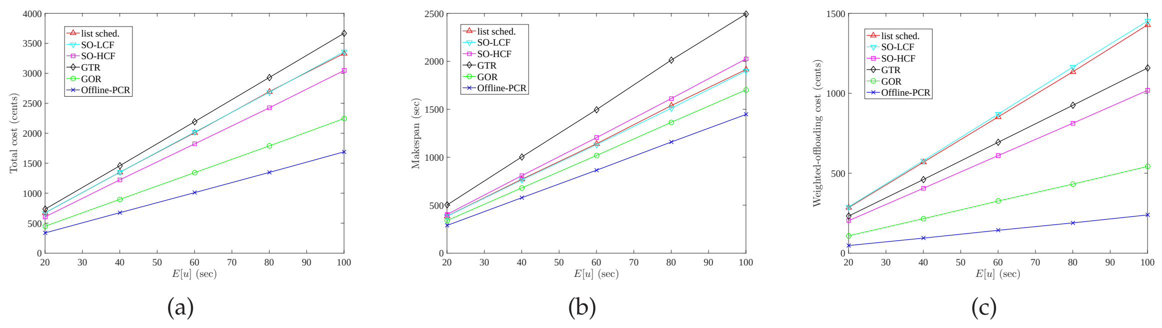

Fig. 3: Effect of varying mean processing time, with $\rho = \frac{1}{m}$. All tasks at time zero.

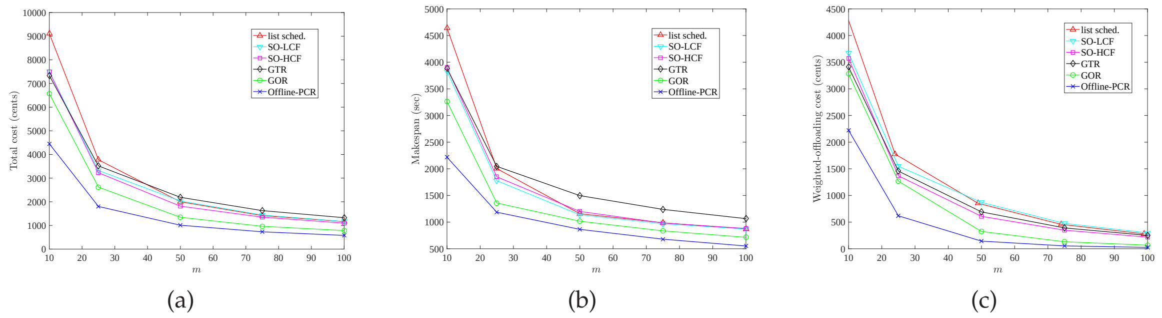

Fig. 4: Effect of varying number of local processors, with $\rho = \frac{1}{m}$. All tasks at time zero.

Since it is desirable to offload the tasks with long processing time u_j and low offloading cost a_j to the cloud and process tasks with small processing times and high costs locally, Offline-PCR schedules the next task from the start of the list whenever a local processor becomes idle, and offloads the next task from the end of the list whenever the remote processor finishes a task.

In the following simulation results, both the processing times u_j and offloading cost a_j are chosen from an exponential distribution, with default means 60 sec and $\frac{60}{m}$ cents/sec, respectively. Our choice of the offloading cost a_j is inspired by the pricing of the EC2 on-demand instances which range from 0.03 to 13 cents/sec [35]. In order to compute the total cost in cents, at the local cloud processors we consider a cost of 1 cent/sec of the makespan. Note that, even though we have chosen parameter values from an exponential distribution, similar results are observed when other distributions are used. The other default parameters are as follows: speed factor $\rho = 1/m$, number of tasks $n = 1500$, number of processors $m = 50$, and weight factor $w = 1$. We generate over 5000 problem instances for each data point and compare the average total cost achieved by different algorithms.

Figures 3(a), 4(a), 5, 6, and 7 present the average total cost with varying mean processing time $\mathbb{E}[u_j]$, number of processors m, speed factor ρ, number of tasks n, and weight factor w, respectively. We observe that, in general, GOR achieves lower average total cost compared with the online and semi-online alternatives. Furthermore, GOR provides a reduction of $20-40\%$ in the average total cost when compared with online list scheduling. We observe further that, despite the fact that GOR does not know u_j apriori, its total cost is only $10-20\%$ higher than that of the Offline-PCR (which assumes u_j are known) for most parameter settings, except for very low ρ or high w values. Thus, in addition to providing lower total cost when compared with semi-online and online alternatives, GOR can be used to benchmark algorithms that assume additional knowledge on the statistics of u_j.

For varying mean processing time and varying m we have presented the performance of the algorithms with respect to makespan and the weighted-offloading costs in Figures 3(b) and 4(b), and Figures 3(c) and 4(c), respectively. We again observe that GOR provides $20-30\%$ reduction in makespan, and $30-40\%$ reduction in offloading costs when compared with list scheduling.

Despite of its $O(\sqrt{m})$ competitive ratio for general ρ, GTR achieves higher average total cost compared with the alternatives. As explained in ***Remark 3***, this can be attributed to the additional penalty incurred to makespan due to the tasks restarting twice. This is demonstrated in Figures 3(b) and 4(b), where the maksepan achieved by GTR is higher than the other algorithms in contrast to its relatively lower weighted-offloading cost as seen in Figures 3(c) and 4(c).

Since GOR and GTR has the same competitive ratio for small ρ and large ρ, GOR is clearly superior given its better average performance. These results further suggest the following choice between GOR and GTR for the case of moderate ρ values: use GOR when the average performance is more desirable, and use GTR when worst-case performance guarantee is more important. From Figure 5, we observe that GOR and GTR have the same average performance for small ρ values. This is because the first iteration of GOR and GTR is identical, and they differ in the second iteration only through the additional condition used for cancelling a task in GTR. Now, for any task j that

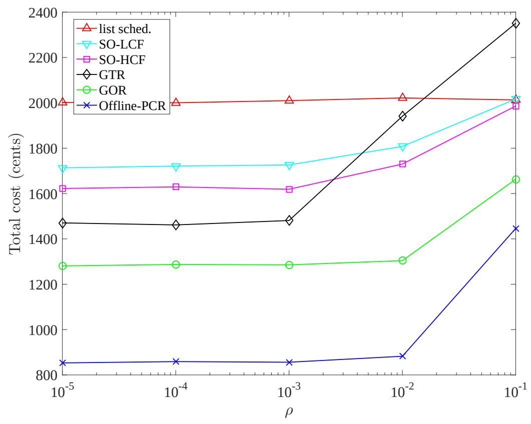

Fig. 5: Effect of varying speed factor. All tasks at time zero.

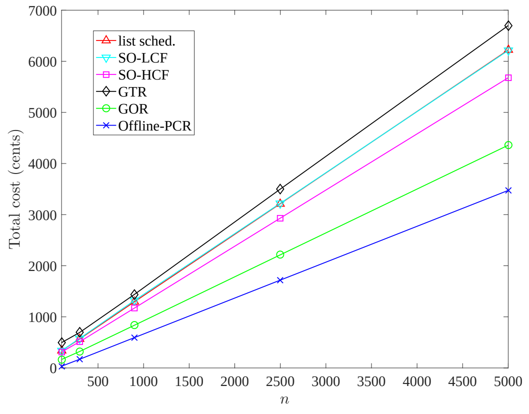

Fig. 6: Effect of varying number of tasks, with $\rho = \frac{1}{m}$. All tasks at time zero.

is to be cancelled in the second iteration of GTR, it should satisfy $u_j \geq \frac{a_j}{\rho}$. Since ρ is small, tasks getting cancelled in the second iteration of GTR is less probable.

Finally, from Figure 6 we observe that, as the number of tasks increase, the ratio between the total cost achieved by list scheduling and that of GOR is increasing. This suggests that GOR will perform far better than list scheduling in an enterprise where the number of tasks is large.

8.2 Dynamic Task Arrivals

In this subsection we simulate dynamic task arrivals and observe the average performance of DGOR and DGTR. We compare it with list scheduling. Since the tasks arrive in time and their arrival times are not known, SO-LCF and SO-HCF cannot be used for this case. We generate task arrivals with inter arrival times chosen from an exponential distribution with mean 100 ms. All the other parameter values are the same as described in the previous subsection. We simulate for over 10^5 task arrivals for each data point.

In Figures 8 and 9, we present the average total cost of the algorithms with varying mean processing time $\mathbb{E}[u_j]$ and weight factor w, respectively. We observe that DGOR provides a reduction of $40-50\%$ in the total cost compared with list scheduling. We note that the simulation results using other parameters follow similar trends as for the case of all tasks available at time zero and hence are omitted.

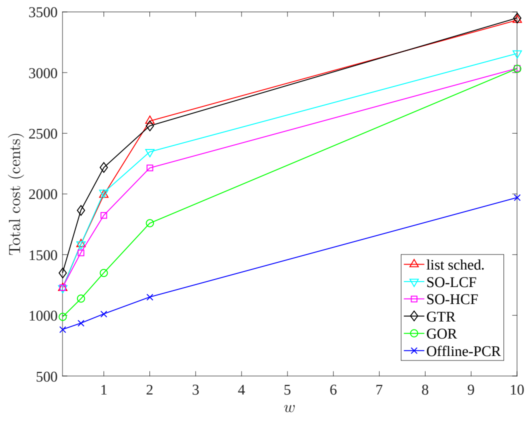

Fig. 7: Effect of varying weight factor, with $\rho = \frac{1}{m}$. All tasks at time zero.

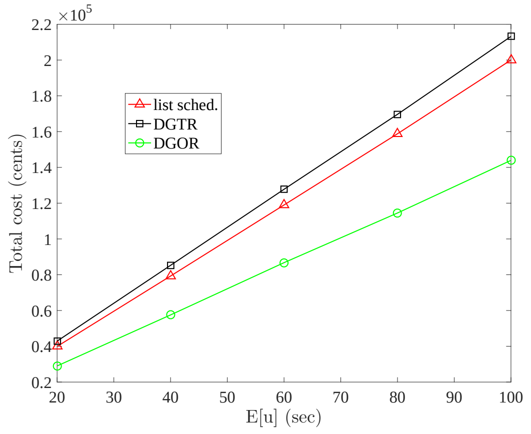

Fig. 8: Effect of varying mean processing time. Dynamic task arrival.

8.3 Multiple Remote Processors

In Figure 10, we present the performance of different algorithms by varying the number of remote processors. We observe that GOR significantly outperforms the existing algorithms. More importantly its total cost reduces as the number of remote processors increase. We observe similar trends for other parameter setting as well and the figures are not presented to avoid redundancy. From these results we conclude that, for the problem at hand, the design principles developed by modelling the public cloud as a single powerful processor are equally applicable for the case of multiple remote processors and the proposed extensions have similar performance gains.

9 Conclusion

We have studied joint scheduling and offloading in a computational offloading system. We have formulated an optimization problem to minimize the weighted sum of the makespan and the offloading cost at the remote cloud. We have proposed the GOR algorithm to solve this problem under the challenging yet practical semi-online setting where the task processing times are not known a priori. Even though GOR is $O(\sqrt{m})$-competitive only for some special cases of ρ, simulation results suggest that, on average, it

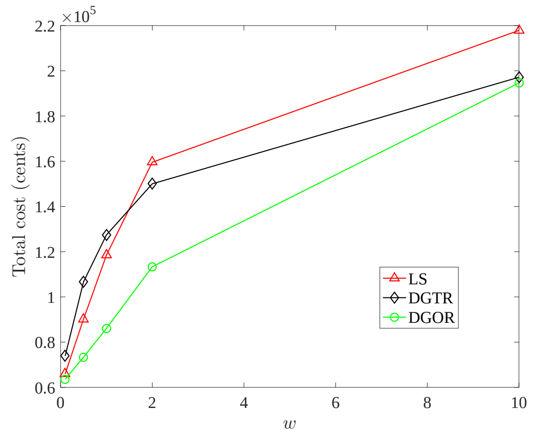

Fig. 9: Effect of varying weight factor. Dynamic task arrival.

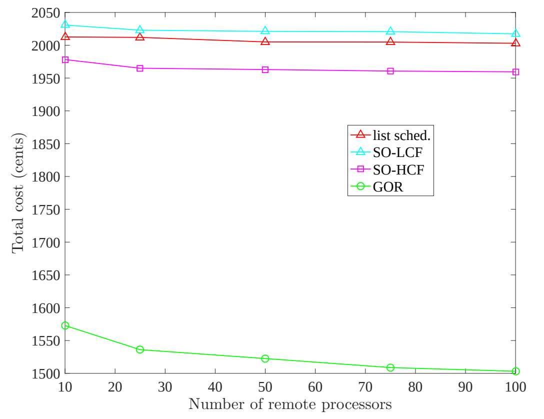

Fig. 10: Effect of varying number of remote processors. $\rho = 0.1$. All tasks at time zero.

provides significant improvement over previously known algorithms. On the other hand, GTR is $O(\sqrt{m})$-competitive for general ρ, but often performs worse than simple list scheduling, on average, because of the penalties incurred in restarting some tasks twice.

References

[1] J. P. Champati and B. Liang, "One-restart algorithm for scheduling and offloading in a hybrid cloud," in *Proc. IEEE/ACM IWQoS*, Jun. 2015.

[2] T. Guo, U. Sharma, P. Shenoy, T. Wood, and S. Sahu, "Cost-aware cloud bursting for enterprise applications," *ACM Trans. Internet Tech.*, vol. 13, no. 3, pp. 10:1–10:24, May 2014.

[3] 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.

[4] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing - a key technology towards 5g," European Telecommunications Standards Institute (ETSI) White Paper, 2015.

[5] 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.

[6] A. Fiat and G. Woeginger, Eds., *Online Algorithms: the State of the Art*, ser. Lecture notes in computer science. New York: Springer, 1998.

[7] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri," in *Proc. of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2010, pp. 265–278.

[8] M. Shifrin, R. Atar, and I. Cidon, "Optimal scheduling in the hybrid-cloud," in *Proc. IFIP/IEEE International Symposium on Integrated Network Management*, 2013, pp. 51–59.

[9] R. Van Den Bossche, K. Vanmechelen, and J. Broeckhove, "Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds," *Future Gener. Comput. Syst.*, vol. 29, no. 4, pp. 973–985, Jun. 2013.

[10] A. Pasdar, K. Almi'ani, and Y. C. Lee, "Data-aware scheduling of scientific workflows in hybrid clouds," in *Proc. International Conference on Computational Science*, 2018, pp. 708–714.

[11] X. Qiu, W. L. Yeow, C. Wu, and F. C. M. Lau, "Cost-minimizing preemptive scheduling of mapreduce workloads on hybrid clouds," in *Proc. IEEE IWQoS*, 2013, pp. 213–313.

[12] S. Li, Y. Zhou, L. Jiao, X. Yan, X. Wang, and M. R. Lyu, "Delay-aware cost optimization for dynamic resource provisioning in hybrid clouds," in *Proc. IEEE International Conference on Web Services, ICWS*, 2014, pp. 169–176.

[13] 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, pp. 49–62.

[14] 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 INFOCOM*, 2012, pp. 945–953.

[15] J. Champati and B. Liang, "Energy compensated cloud assistance in mobile cloud computing," in *Proc. IEEE INFOCOM Workshop on Mobile Cloud Computing*, April 2014.

[16] D. Shabtay, N. Gaspar, and M. Kaspi, "A survey on offline scheduling with rejection," *J. Scheduling*, vol. 16, no. 1, pp. 3–28, 2013.

[17] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, and L. Stougie, "Multiprocessor scheduling with rejection," *SIAM J. Discrete Math.*, vol. 13, no. 1, pp. 64–78, 2000.

[18] C. Miao and Y. Zhang, "On-line scheduling with rejection on identical parallel machines," *J. Systems Science & Complexity*, vol. 19, no. 3, pp. 431–435, 2006.

[19] X. Min, Y. Wang, J. Liu, and M. Jiang, "Semi-online scheduling on two identical machines with rejection," *J. Comb. Optim.*, vol. 26, no. 3, pp. 472–479, 2013.

[20] M. Drozdowski, *Scheduling for Parallel Processing*. Springer Publishing Company, 2009.

[21] D. B. Shmoys, J. Wein, and D. P. Williamson, "Scheduling parallel machines on-line," *SIAM J. Comput.*, vol. 24, no. 6, pp. 1313–1331, Dec. 1995.

[22] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal on Applied Mathematics*, vol. 17, pp. 416–429, 1969.

[23] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones." in *Proc. IEEE INFOCOM*, 2012, pp. 2716–2720.

[24] P. Shu, F. Liu, H. Jin, M. Chen, F. Wen, Y. Qu, and B. Li, "eTime: Energy-efficient transmission between cloud and mobile devices." in *Proc. IEEE INFOCOM*, 2013, pp. 195–199.

[25] Y. Cui, J. Song, K. Ren, M. Li, Z. Li, Q. Ren, and Y. Zhang, "Software defined cooperative offloading for mobile cloudlets," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1746–1760, June 2017.

[26] 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.

[27] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mob. Netw. Appl.*, vol. 18, no. 1, pp. 129–140, Feb. 2013. [Online]. Available: http://dx.doi.org/10.1007/s11036-012-0368-0

[28] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of discrete mathematics*, vol. 5, no. 2, pp. 287–326, 1979.

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

[30] J. P. Champati and B. Liang, "Single restart with time stamps for computational offloading in a semi-online setting," in *Proc. IEEE INFOCOM*, 2017, pp. 945–953.

[31] C. N. Potts, "Analysis of a linear programming heuristic for scheduling unrelated parallel machines," *Discrete Applied Mathematics*, vol. 10, no. 2, pp. 155–164, 1985.

[32] J. K. Lenstra, D. B. Shmoys, and E. Tardos, “Approximation algorithms for scheduling unrelated parallel machines,” *Math. Program.*, vol. 46, no. 3, pp. 259–271, Feb. 1990.

[33] T. Gonzalez, O. H. Ibarra, and S. Sahni, “Bounds for LPT Schedules on Uniform Processors,” *SIAM Journal on Computing*, vol. 6, no. 1, pp. 155–166, 1977.

[34] 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, April 2017.

[35] “Amazon EC2 on-demand pricing,” https://aws.amazon.com/ec2/pricing/on-demand/.

Jaya Prakash Champati is a post-doctoral researcher at Information Science and Engineering Department, KTH Royal Institute of Technology. He obtained his PhD degree from Electrical and Computer Engineering Department at University of Toronto. He obtained his bachelor of technology degree from National Institute of Technology Warangal, India, and master of technology degree from Indian Institute of Technology (IIT) Bombay, India. His general research interest is in the design and analysis of algorithms for scheduling problems that arise in networking and information systems. His favorite tools are combinatorial optimization, approximations algorithms, network calculus, and queueing theory. Prior to joining PhD he worked at Broadcom Communications, where he was part of protocol stack development team for an upcoming LTE chipset project. He was a recipient of the best paper award at National Conference on Communications 2011, IISc, Bangalore, India.

Ben Liang received honors-simultaneous B.Sc. (valedictorian) and M.Sc. degrees in Electrical Engineering from Polytechnic University in Brooklyn, New York, in 1997 and the Ph.D. degree in Electrical Engineering with a minor in Computer Science from Cornell University in Ithaca, New York, in 2001. In the 2001 - 2002 academic year, he was a visiting lecturer and post-doctoral research associate at Cornell University. He joined the Department of Electrical and Computer Engineering at the University of Toronto in 2002, where he is now a Professor. His current research interests are in networked systems and mobile communications. He has served on the editorial boards of the IEEE Transactions on Mobile Computing since 2017 and the IEEE Transactions on Communications since 2014, and he was an editor for the IEEE Transactions on Wireless Communications from 2008 to 2013 and an associate editor for Wiley Security and Communication Networks from 2007 to 2016. 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.