Semi-online Algorithms for Computational Task Offloading with Communication Delay

Jaya Prakash Champati, *Student Member, IEEE,* and Ben Liang, *Senior Member, IEEE*

Abstract—We study the scheduling of computational tasks on one local processor and one remote processor with communication delay. This problem has important application in cloud computing. Although the communication time to transmit a task can be inferred from the known data size of the task and the transmission bandwidth, the processing time of the task is generally unknown until it is processed to completion. Given a set of independent tasks with unknown processing times, our objective is to minimize makespan. We study the problem under two scenarios: 1) the communication times of the tasks to the remote processor are smaller than their corresponding processing times on the remote processor, and 2) the communication times of the tasks to the remote processor are larger than their corresponding processing times on the remote processor. For the first scenario we propose the Semi-online Partitioning and Communication (SPaC) algorithm, and for the second scenario we propose the SPaC-Restart (SPaC-R) algorithm. Even though the offline version of this problem, with a priori known processing times, is NP-hard, we show that the proposed semi-online algorithms achieve $O(1)$ competitive ratios for their intended scenarios. We also provide competitive ratios for both algorithms for more general communication times. We use simulation to demonstrate that SPaC and SPaC-R outperform online list scheduling and performs comparably well with the best known offline heuristics.

Index Terms—Computational offloading, mobile cloud computing, computation with communication, semi-online algorithms

1 Introduction

Consider a local processor and a remote processor. The local processor has some tasks and may enlist the help of the remote processor to process the tasks. However, offloading a task to the remote processor incurs communication delay. This system is of particular interest as it models the most common paradigm in Infrastructure-as-a-Service cloud computing [1], where a local machine (e.g., personal computer or mobile device) enlists the help of a remote server (e.g., Amazon EC2 instance), by breaking down its jobs into multiple parallel tasks and offloading some of them to be processed remotely.

We study the problem of minimizing the makespan of processing a set of independent tasks. In contrast to the classical literature on parallel processing [2], [3], [4], [5], [6], [7], where the communication delay is not considered, our study mainly focuses on the affects of the communication delay on the makepsan. Some recent studies on grid and cloud computing [8], [9], [10], [11], [12], and mobile cloud computing [13] in particular, have considered the effect of communication delay on makespan. However, as detailed in Section 2, these works either propose only heuristics or study highly simplified models.

To illustrate the effect of communication delay on the makespan of processing multiple tasks, we show in Figure 1 an example of the optimal scheduling of five tasks to minimize the makespan, obtained by brute-force search. We observe that there is substantial idle time on the remote processor while it waits for the tasks to be delivered. With sub-optimal scheduling, the wastage in idle time could be much more severe. Hence, we need to jointly optimize the partitioning of the set of tasks and the transmission schedule of tasks. Unfortunately, as we will show in Section 3, this joint partitioning and scheduling problem is NP-hard even if all processing times and communication times are known a priori, i.e., in the *offline* setting.

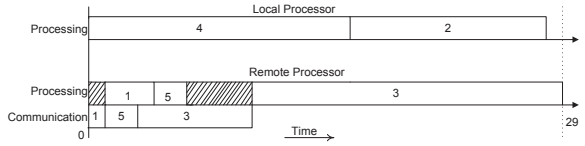


Fig. 1. Example of optimal offline task partitioning and scheduling. The local and remote processors are assumed identical. At time 0 we are given 5 tasks with processing times $\{3, 12, 18, 16, 2\}$ ms and communication times $\{1, 10, 8, 9, 2\}$ ms. The task numbers are as labeled on the tasks, and the shaded areas indicate idle time.

Furthermore, even though the communication time to transmit a task can be reasonably inferred from its data size and data transmission rate, the processing time required for the task generally is unknown without first processing it [3]. Therefore, in our work we consider the *semi-online* version of the problem where the task processing times are not known a priori but the communication times of the tasks are known a priori. We note that the processing time of a task is often independent of its communication delay. For example, a task having a single for-loop may have data size on the order of tens of bytes. With typical LTE uplink data rates, the task can be transmitted within milliseconds. However, depending on the number of iterations in the for-loop, its processing can take an indeterminate amount of

- *The authors are affiliated with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, M5S 3G4, Canada. E-mails:{champati,liang}@ece.utoronto.ca.*
- *This work has been funded in part by grants STPGP-447497 and RGPIN-2015-05506 from the Natural Sciences and Engineering Research Council of Canada.*

time. Similarly, tasks that require large input data may incur long communication delays compared with their processing times. We note that the unknown processing times of the tasks, processor idle times introduced by communication, and the NP-hardness of the original offline problem, all introduce substantial challenges in the analysis and design of an effective and computationally efficient solution.

In this work, we initially focus on two special scenarios: 1) the communication times of the tasks to the remote processor are smaller than their corresponding processing times on the remote processor, i.e., $\eta_{\max} \leq 1$, where $\eta_{\max}$ is the maximum ratio between a task's communication time and its processing time on the remote processor; and 2) the communication times of the tasks to the remote processor are larger than their corresponding processing times on the remote processor, i.e., $\eta_{\min} \geq 1$, where $\eta_{\min}$ is the minimum ratio between a task's communication time and its processing time on the remote processor. For the first case, we propose a Semi-Online Partitioning and Communication (SPaC) algorithm, which jointly selects the tasks to be processed locally or to be offloaded and determines the transmission schedule to the remote processor. For the second case, we modify SPaC and propose SPaC-Restart (SPaC-R).

The scenarios $\eta_{\max} \leq 1$ and $\eta_{\min} \geq 1$ are important in practice. In particular, $\eta_{\max} \leq 1$ is applicable to systems where tasks with heavy computation and small communication payload often are prime targets for offloading. This is particularly the case in mobile cloud computing, since tasks with a large communication payload would drain too much battery power to be transmitted wirelessly [13]. The case of $\eta_{\min} \geq 1$ is applicable to systems where the remote processor is powerful and the communication times of the tasks become the bottleneck.

Furthermore, we study the performance of SPaC and SPaC-R for the general scenario where the ratio of the communication time of a task to its remote processing time is arbitrary. We provide competitive ratios for both algorithms. Additionally, we illustrate, through simulation, their effectiveness compared with existing alternatives.

Our main contributions are summarized below:

- We propose the SPaC algorithm, which does not require a priori knowledge of the processing times. It has computational complexity $O(n \log n)$, where n is the number of tasks.
- For $\eta_{\max} \leq 1$ we show that SPaC has a competitive ratio θ_1, where $\theta_1 \leq 1 + \min\left\{\frac{\max\{1,\rho\}}{\rho+1}, \frac{1}{\rho}, \rho\right\} \leq \frac{\sqrt{5}+1}{2}$, and ρ is the speed ratio between the remote and local processors. Furthermore, for $\eta_{\max} \leq 1$ and $\rho = 1$, we show that SPaC is optimal, in the sense that it provides the minimum competitive ratio of $\frac{3}{2}$ among all deterministic semi-online algorithms.
- For $\eta_{\min} \geq 1$, we propose the SPaC-R algorithm and prove that it is θ_1'-competitive, where

$$\theta_1' = \begin{cases} 2 & \text{if } \rho \leq 1, \\ 4 + \frac{1}{\rho} & \text{if } 1 < \rho < 2, \\ \frac{9}{2} & \text{if } \rho \geq 2. \end{cases}$$

  Further, we show that for $\rho >> 1$, the competitive ratio $\frac{9}{2}$ is tight.
- For general $\eta_{\max}$ and $\eta_{\min}$ we prove $O(\eta_{\max})$ competitive ratio and $O(1/\eta_{\min})$ competitive ratio for SPaC and SPaC-R, respectively.
- Finally, using simulation we compare the average makespan of SPaC and SPaC-R with classical online list scheduling and two of the best known offline heuristics to show that the proposed solutions significantly outperforms the online alternative and performs close to the offline heuristics.

The rest of this paper is organized as follows. In Section 2 we discuss the related work. In Section 3, we detail the system model and the optimization problem. We present SPaC and SPaC-R in Section 4. Competitive ratio analysis of SPaC and SPaC-R for $\eta_{\max} \leq 1$ and $\eta_{\min} \geq 1$, respectively, is given in Sections 5 and 6. In Section 7 we discuss the competitive ratios for SPaC and SPaC-R for general $\eta_{\max}$ and $\eta_{\min}$. Simulation results are given in Section 8. We conclude in Section 9.

2 Related Work

The related work on computational task offloading and parallel processing may be categorized based on whether communication delay plays a role in the system model and task scheduling.

2.1 Parallel Processing without Communication Delay

In classical computer science literature, one of the most studied scheduling problems is the partitioning of a set of independent tasks for m processors to minimize the makespan. The celebrated *list scheduling* [2] 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. It does not require a priori knowledge of the processing times and has a $(2 - \frac{1}{m})$-approximation ratio when the processors are identical. More complex online algorithms for various processor settings were studied in [3]. They are based on a transformation framework that converts offline algorithms to online ones via dynamic cancellation and rescheduling of tasks. Neither [2] nor [3] accounts for communication delay.

In an offline setting, if the processing times are known and the list of tasks is pre-sorted in the longest-processing-time-first order, a $(\frac{4}{3} - \frac{1}{3N})$-approximation ratio is achieved by list scheduling on identical processors [2]. For processors with different speeds, this algorithm was shown to achieve an approximation ratio of $\frac{19}{12}$ in [4], which was later improved to 1.5773 in [5]. Furthermore, Polynomial Time Approximation Schemes (PTAS) and Fully Polynomial Approximation Schemes (FPTAS) exist [6]. More recently, the authors of [14] studied the performance of different offline heuristics for scheduling independent tasks on parallel processors under more general settings. For further reading in this line of work we refer the reader to [6] and [15].

Semi-online scheduling on parallel processors is a relatively new paradigm. All previous works define semi-online as the case where the individual processing time of each task is unknown, but the total processing time of all tasks is known [16], [17], [18], [19], [20]. Under such an assumption, the authors of [16] studied the problem of scheduling independent tasks on two identical parallel

processors. They provided an algorithm with competitive ratio $\frac{4}{3}$. It is further known that, for the case of identical processors, the lower bound of the competitive ratio is 1.565 [18], which is achieved by the algorithm proposed in [19]. In addition, for two processors with different speeds, the authors of [20] proposed two algorithms and found their competitive ratios as functions of the ratio of speeds between the processors.

We note that none of the above works consider communication delay incurred for offloading a task to a processor on which it is scheduled for processing, and there is no straightforward method to accommodate communication delays into the proposed schemes/algorithms. More importantly, in the semi-online setting considered in this work, we assume that neither the task processing times nor the total processing time is known a priori, which is a natural model for offloading with communication delay.

2.2 Grid and Cloud Computing with Communication Delay

The problem of scheduling independent tasks with communication overhead on multiple processors was considered in [8], [9], [10] for the grid/cloud computing environment. The authors proposed a set of heuristics without performance bounds. Simplifying assumptions were made in other works to improve analytical tractability. For example, the tasks were assumed identical in [11], and they were assumed infinitely divisible with processing times proportional to their data size in [12, Ch. 7]. Furthermore, all of these works assume knowledge of the processing times and hence are offline.

More recently, task offloading and scheduling were considered in the mobile cloud computing environment. In [13], the authors proposed a heuristic guideline that tasks should be offloaded only if the local computing time of the task is greater than its remote communication and computing time. In [21], the authors focused on the partitioning of an application at the software engineering level to improve its execution time, instead of explicitly considering processor speeds or communication overhead. Most other studies focused on energy savings at the mobile device instead of makespan minimization [22], [23], [24], [25].

Offloading computational tasks to remote servers in a public cloud was studied in the hybrid cloud paradigm [26], [27], [28], [29]. However, none of these works consider communication delay to offload tasks in their system model. To the best of our knowledge, our work is the first to focus on analytical modeling and optimization of the makespan with consideration for communication delay, proposing semi-online algorithms with provable competitive ratios.

3 System Model

In this section, we describe the task processing model and detail the makespan minimization problem in the semi-online setting.

3.1 Processing, Communication, and Scheduling

We index the processors by $i \in \mathcal{Q} = \{0, 1\}$, where 0 and 1 represent the local and remote processors, respectively. Let n denote the number of tasks to be processed and $j \in \mathcal{T} = \{1, \ldots, n\}$ be the task index. We assume that the tasks are independent and no preemption is allowed.

Let α_j denote the time required to process task j on the remote processor. We assume that the time required to process the same task on the local processor is given by $\rho\alpha_j$, where $\rho > 0$ represents the speed ratio between the processors. We consider that ρ is also part of the input problem instance. In other words, ρ is also not known a priori. Let $\alpha_{\max} = \max_{j\in\mathcal{T}} \alpha_j$.

The remote processor can process a task only after all the data load of the task is received. The data size of task j is denoted by β_j in bits. The time taken to transmit the task j to the remote processor may be given by $b\beta_j$ where b is the inverse of data rate to the remote processor. For simplicity of presentation, without loss of generality, we suppress writing b by merging it into β_j. Let $\eta_j = \frac{\beta_j}{\alpha_j}$, $\eta_{\max} = \max_{j\in\mathcal{T}} \eta_j$ and $\eta_{\min} = \min_{j\in\mathcal{T}} \eta_j$. We further assume that after each task is processed on the remote processor, a short acknowledgement is returned with negligible delay.

A schedule $\mathbf{s}$ consists of a pair of functions $(\pi, \mathbf{g})$, where $\pi : \mathcal{T} \to \mathcal{Q}$ partitions the set $\mathcal{T}$ and maps the partitions to processors. Let $\mathcal{T}_i(\mathbf{s}) \subseteq \mathcal{T}$ denote the partition assigned to processor i. We use function $\mathbf{g}$ to specify the sequence in which the tasks assigned to remote processor are transmitted, i.e., $\mathbf{g} : \{1, \ldots, |\mathcal{T}_1(\mathbf{s})|\} \to \mathcal{T}_1(\mathbf{s})$, where task $\mathbf{g}(k)$ is transmitted kth in the sequence. Let $\mathcal{S}$ denote the set of all possible schedules.

Throughout this paper, we consider only non-wasteful scheduling. That is, given a schedule $\mathbf{s}$, the tasks assigned to the local processor are processed one-by-one without gaps starting at time 0, and the tasks assigned to the remote processor are transmitted one-by-one without gaps starting at time 0. It is clear that this is without loss of generality with respect to makespan minimization.

We make a note that, there is a significant amount of work in the literature on scheduling with communication delays (see [12, Ch. 6]), where the communication delays are due to inter-task communication that occur when two dependent tasks are scheduled on different processors. In contrast, since we focus on *independent* tasks, the communication delay we consider in this paper is solely due to the communication overhead incurred in transmitting a task to the remote processor.

3.2 Optimization Problem

Given the set of tasks $\mathcal{T}$ at time 0, the *makespan* is defined as the time when the processing of the last task in $\mathcal{T}$ is complete. Let $C_{\max}(\mathbf{s})$ represent the makespan under schedule $\mathbf{s}$. It equals $\max\{C_0(\mathbf{s}), C_1(\mathbf{s})\}$, where $C_i(\mathbf{s})$ represents the time when processor i finishes processing the tasks assigned to it. It is clear that $C_0(\mathbf{s}) = \rho \sum_{j\in\mathcal{T}_0(\mathbf{s})} \alpha_j$. In the following we establish a closed-form expression for $C_1(\mathbf{s})$.

Let $I(\mathbf{s})$ denote the idle time of the remote processor under schedule $\mathbf{s}$. Any task scheduled on this processor should go through a communication stage and a processing stage. We note that this is equivalent to a two-machine flow shop model [30], and hence we have

$$I(\mathbf{s}) = \max_{1\le u\le|\mathcal{T}_1(\mathbf{s})|} \left\{ \sum_{j=\mathbf{g}(1)}^{\mathbf{g}(u)} \beta_j - \sum_{j=\mathbf{g}(1)}^{\mathbf{g}(u-1)} \alpha_j \right\}. \quad (1)$$

Since $C_1(\mathbf{s})$ equals the total idle time plus the total processing time of the tasks, we have

$$C_1(\mathbf{s}) = I(\mathbf{s}) + \sum_{j \in \mathcal{T}_1(\mathbf{s})} \alpha_j = \max_{1 \le u \le |\mathcal{T}_1(\mathbf{s})|} \left\{ \sum_{j=\mathbf{g}(1)}^{\mathbf{g}(u)} \beta_j + \sum_{j=\mathbf{g}(u)}^{\mathbf{g}(|\mathcal{T}_1(\mathbf{s})|)} \alpha_j \right\}. \quad (2)$$

We are interested in the following makespan minimization problem $\mathcal{P}$:

$$\underset{(\pi,\mathbf{g})=\mathbf{s}\in\mathcal{S}}{\text{minimize}} \ \max\{C_0(\mathbf{s}), C_1(\mathbf{s})\}.$$

In the offline setting, all parameter values of the tasks are known at time 0. In this case, let $\mathbf{s}^*$ denote an optimal offline schedule and $C^*_{\max}$ denote the minimum makespan. We note that, when $b = 0$ and $\rho = 1$, $\mathcal{P}$ is equivalent to the problem of scheduling independent tasks on two identical processors to minimize makespan which is NP-hard [12]. Therefore, $\mathcal{P}$ is NP-hard.

3.3 Semi-online Scheduling with Unknown Processing Times

Even though the communication time to transmit a task can be reasonably inferred from its data size, the processing time required for the task generally is unknown without first processing it [3]. Therefore, we are interested in semi-online scheduling, where α_j, for all j, are not known a priori and β_j, for all j, are known a priori.

The efficacy of an online algorithm is often measured by its competitive ratio in comparison with the optimal offline algorithm. We use the same measure for semi-online algorithms as well. Let P be a problem instance of $\mathcal{P}$, $s(P)$ be the schedule given by an online algorithm and $s^*(P)$ be the schedule given by an optimal offline algorithm. The online algorithm is said to have a competitive ratio θ if and only if for all P, $C_{\max}(s(P)) \le \theta C_{\max}(s^*(P))$. Furthermore, θ is said to be tight for the online algorithm if there exists P such that $C_{\max}(s(P)) = \theta C_{\max}(s^*(P))$.

4 The Semi-online Partitioning and Communication Algorithm

In this section we first present SPaC. We will show later in Section 5 that SPaC has $O(1)$ competitive ratio for $\eta_{\max} \le 1$. However, when the communication times are larger relative to the remote processing times, SPaC does not provide a strong competitive ratio. Therefore, we modify SPaC and propose SPaC-R, which will be shown in Section 6 to have $O(1)$ competitive ratio for $\eta_{\min} \ge 1$.

4.1 SPaC

The following observations motivate the design of SPaC. We *first* note that the tasks to be scheduled on the remote processor should be transmitted without any gaps, since there is no advantage in adding artificial communication delay. *Second*, from (1), it is desirable to assign tasks with smaller communication times to the remote processor, in order to reduce the idle time. *Third*, given the set of tasks chosen to be offloaded to the remote processor, the optimal order in which they are to be transmitted is known and given by Johnson's rule [30]. Unfortunately, Johnson's rule requires the knowledge of processing times. Therefore, we resort to ordering the tasks based on their communication times alone, while maintaining observation of the task processing progress. Interestingly, as will be shown in Section V, in the case of $\eta_{\max} \le 1$, the proposed procedure gives a schedule that satisfies Johnson's rule.

In SPaC, we list the tasks in the increasing order of their communication times. We process the tasks one by one from the *end* of the list on the local processor, and transmit the tasks, that are not yet processed to completion, one by one from the *start* of the list to the remote processor. At the remote processor the tasks received are processed in the same order. The details of SPaC are given in Algorithm 1, where E_1 in line 6 denotes the event that the processing of a task is complete on either processor and E_2 in line 13 denotes the event that the transmission of a task to the remote processor is complete. Note that the last remaining task may be processed on both processors at the same time. In such a case, when the task is processed to completion on one processor, we terminate its processing on the other processor. To achieve this, we assume that a short message indicating task index can be exchanged between the processors whenever the task is processed to completion.

Algorithm 1: SPaC

1: Sort $\mathcal{T}$ in ascending order of communication times. WLOG, consider $\beta_1 \le \beta_2 \le \ldots \le \beta_n$.
2: $j_0 = n$, $j_1 = 1$ and $k = 1$.
3: Start processing task j_0 on processor 0; Start transmitting task j_1 to processor 1.
4: **while** $j_0 \neq j_1$ **do**
5: Wait until next event E occurs
6: **if** $E = E_1$ **then**
7: **if** E_1 is due to processor 0 **then**
8: $j_0 = j_0 - 1$
9: Start processing task j_0 on processor 0.
10: **else if** E_1 is due to processor 1 **then**
11: $j_1 = j_1 + 1$
12: **end if**
13: **else if** $E = E_2$ **then**
14: $k = k + 1$
15: Start transmitting task k to processor 1.
16: **end if**
17: **end while**
18: $q = j_0 = j_1$
19: Task q is scheduled both on processor 0 and processor 1. If task q is finished processing on processor 0 first, cancel its execution on processor 1 and vice-versa.

The computational complexity of SPaC is $O(n \log n)$, since line 1 of Algorithm 1 requires sorting the communication times of all tasks, while the rest of the algorithm has no more than linear complexity. Throughout this paper, we use $\mathbf{s}^S$ to denote the schedule given by SPaC.

4.2 SPaC-R

The guiding principle behind the design of SPaC-R is the following. Since $\eta_{\min} \ge 1$ or $\beta_j \ge \alpha_j$, for all j, the

completion time on the remote processor is dominated by the communication times of the tasks scheduled on it. Intuitively, in this case, to lower the makespan, a task with higher $\rho\alpha_j$ value compared with β_j should be scheduled on processor 1 and vice-versa. Therefore, in SPaC-R the tasks that are scheduled on processor 0 and have $\rho\alpha_j$ greater than β_j are identified, cancelled, and rescheduled, so that they may be scheduled on processor 1 in the new schedule. We note that cancelling a task with large $\rho\alpha_j$ on processor 0 may allow some tasks that have smaller $\rho\alpha_j$ values to be scheduled on that processor. The notion of cancelling and rescheduling a task is called *restart*.

Task restart was originally used by Shmoys et. al. [3] to schedule independent tasks on parallel processors in online setting. Recently we used this idea to propose an algorithm for the problem of scheduling and offloading in a hybrid cloud [29]. We note that in both of the above works communication delay is not considered in the system model, and hence the solutions proposed there cannot be applied to the problem at hand.

SPaC-R runs SPaC for two iterations. In the first iteration, the schedule given by SPaC is implemented with an exception that any task j scheduled on processor 0 and processed longer than β_j duration is cancelled. In the second iteration, all the tasks that are cancelled in the first iteration are rescheduled using SPaC. We note that, even though cancelling a task and rescheduling it penalizes the makespan, it paves a way for obtaining constant competitive ratio for the case $\eta_{\min} \geq 1$. The details of SPaC-R are presented in Algorithm 2, where E_1 and E_2 have same meaning as defined before. We note that SPaC-R similarly runs in $O(n \log n)$ time. We use $\mathbf{s}^{\text{SR}}$ to denote the resultant schedule.

Algorithm 2: SPaC-R

1: $l = 1, \mathcal{T}^{(l)} = \mathcal{T}$
2: **while** $l \leq 2$ **do**
3: Sort $\mathcal{T}^{(l)}$ in ascending order of β_j. WLOG, re-index tasks such that $\beta_1 \leq \beta_2 \leq \ldots \leq \beta_{|\mathcal{T}^{(l)}|}$.
4: $j_0 = |\mathcal{T}^{(l)}|$, $j_1 = 1$ and $k = 1$.
5: Start processing task j_0 on processor 0;
Start transmitting task j_1 to processor 1.
6: **if** $l = 1$ **then**
7: Cancel task j_0 if its execution time exceeds β_{j_0} and include it in $\mathcal{T}^{(l+1)}$
8: **end if**
9: **while** $j_0 \neq j_1$ **do**
10: Wait until next event E occurs
11: **if** $E = E_1$ **then**
12: **if** E_1 is due to processor 0 **then**
13: $j_0 = j_0 - 1$
14: Start processing task j_0 on processor 0.
15: **if** $l = 1$ **then**
16: Cancel task j_0 if its execution time exceeds β_{j_0} and include it in $\mathcal{T}^{(l+1)}$
17: **end if**
18: **else if** E_1 is due to processor 1 **then**
19: $j_1 = j_1 + 1$
20: **end if**
21: **else if** $E = E_2$ **then**
22: $k = k + 1$
23: Start transmitting task k to processor 1
24: **end if**
25: **end while**
26: $q = j_0 = j_1$
27: Task q is scheduled both on processor 0 and transmitted to processor 1. If task q is finished or cancelled on processor 0 first, cancel its transmission to processor 1. If processing of task q on processor 1 finishes first, cancel its processing on processor 0.
28: $l = l + 1$
29: **end while**

5 SPaC Competitive Ratio for Small Communication Times

In this section we derive the competitive ratio of SPaC for $\eta_{\max} \leq 1$, i.e., $\alpha_j \geq \beta_j$, for all j. We first present several preliminary results that will be used extensively in the remaining analysis. Lemma 1 below provides a simple upper bound on the makespan of $\mathbf{s}^S$.

Lemma 1. $C_{max}(\mathbf{s}^S) \leq (1+\rho)C^*_{max}$.

Proof. By the definition of $C_0(\mathbf{s})$, $C_1(\mathbf{s})$ and $I(\mathbf{s})$ given in Section 3, under any schedule $\mathbf{s}$, we have

$$\frac{1}{\rho}C_0(\mathbf{s}) + [C_1(\mathbf{s}) - I(\mathbf{s})] = \sum_{j=1}^{n} \alpha_j.$$

We use $C_{\max}(\mathbf{s}) \geq C_0(\mathbf{s})$ and $C_{\max}(\mathbf{s}) \geq C_1(\mathbf{s})$ to obtain

$$C_{\max}(\mathbf{s}) \geq \frac{\rho}{\rho+1}\left(\sum_{j=1}^{n}\alpha_j + I(\mathbf{s})\right), \ \forall \mathbf{s}. \tag{3}$$

Now, if the communication time plus the processing time of the task at the start of the list formed by SPaC exceeds $\rho\sum_{j=1}^{n}\alpha_j$, in $\mathbf{s}^S$ all tasks will be processed on processor 0. In all other cases it can be easily argued that the makespan

under $\mathbf{s}^S$ will be smaller than that of the schedule where all the tasks are assigned to processor 0. Therefore,

$$C_{\max}(\mathbf{s}^S) \leq \rho \sum_{j=1}^{n} \alpha_j \leq (1+\rho) C^*_{\max},$$

where the second inequality is due to (3) when $\mathbf{s} = \mathbf{s}^*$. □

Furthermore, from (3) in the proof above, we have

$$\frac{\rho \alpha_j}{\rho + 1} \leq \frac{\alpha_{\max}}{\sum_{j=1}^{n} \alpha_j} C^*_{\max}, \ \forall j. \tag{4}$$

Finally, the optimal makespan cannot be smaller than the processing time of any task on the fastest processor. Therefore,

$$\begin{aligned} C^*_{\max} &\geq \min\{\rho\alpha_j, \beta_j + \alpha_j\}, \forall j \\ &\geq \min\{\rho, 1\}\alpha_j, \forall j. \end{aligned} \tag{5}$$

In the following we focus on how SPaC minimizes $C_1(\mathbf{s})$ given $\mathcal{T}_1(\mathbf{s})$. As stated in Section 3.2, scheduling tasks on processor 1, via the communication and processing stages, is equivalent to the two-machine flow shop problem. In Corollary 1 we state that, for $\eta_{\max} \leq 1$, the optimal Johnson's rule for the two-machine flow shop problem degrades to simply ordering the tasks in the increasing order of their communication times.

Corollary 1. *In the two-machine flow shop problem, under the condition $\eta_{max} \leq 1$, tasks should be scheduled in the increasing order of their communication times to minimize the makespan.*

Proof. Let tasks j_1 and j_2 be any two tasks in the two-machine flow shop problem. From Johnson's rule, in the optimal schedule, task j_1 precedes j_2 if and only if

$$\min\{\beta_{j_1}, \alpha_{j_2}\} \leq \min\{\beta_{j_2}, \alpha_{j_1}\}.$$

Since $\beta_{j_1} \leq \alpha_{j_1}$ and $\beta_{j_2} \leq \alpha_{j_2}$, a sufficient condition for the above inequality to hold is $\beta_{j_1} \leq \beta_{j_2}$. Hence the result. □

From Corollary 1 we conclude that, when $\eta_{\max} \leq 1$, the SPaC scheduling policy to order the tasks in $\mathcal{T}_1(\mathbf{s})$ in the increasing order of communication times serves to minimize $C_1(\mathbf{s})$.

Let $I^*(\mathcal{B})$ denote the idle time when tasks belonging to some set $\mathcal{B}$ are scheduled on processor 1 in the sequence of increasing order of their communication times. Lemma 2 below helps provide insight into the idle time on processor 1 under SPaC.

Lemma 2. *Consider $\beta_1 \leq \beta_2 \leq \ldots \leq \beta_n$ and $\eta_{max} \leq 1$. Let $\mathcal{B}_1 = \{1, 2, \ldots, l\}, l \leq n$, and $\mathcal{B}_2 \subseteq \mathcal{T}$, $\mathcal{B}_2 \nsubseteq \{1, 2, \ldots, l-1\}$. Then $I^*(\mathcal{B}_1) \leq I^*(\mathcal{B}_2)$.*

Proof. Consider any set $\mathcal{B} \subseteq \mathcal{T}$. We claim that for any $q \leq \arg\max_{j \in \mathcal{B}} \beta_j$ we have $I^*(\mathcal{B} \cup \{q\}) \leq I^*(\mathcal{B})$. To prove the claim we proceed as follows. Let $\mathcal{B} = \{j_1, j_2, \ldots, j_{|\mathcal{B}|}\}$, where $j_1 \leq j_2 \leq \ldots \leq j_{|\mathcal{B}|}$. Also, for some $1 \leq k < |\mathcal{B}| - 1$ let $\beta_{j_k} \leq \beta_q \leq \beta_{j_{k+1}}$. Noting that $I^*(\mathcal{B})$ is the idle time when the tasks from $\mathcal{B}$ are scheduled on processor 1 in the increasing order of their communication times, using (1) we have

$$\begin{aligned} I^*(\mathcal{B}) &= \max_{1 \leq u \leq |\mathcal{B}|} \left\{ \sum_{j=j_1}^{j_u} \beta_j - \sum_{j=j_1}^{j_{u-1}} \alpha_j \right\} \\ &= \max\{I^*_1(\mathcal{B}), I^*_2(\mathcal{B})\}, \end{aligned}$$

where

$$I^*_1(\mathcal{B}) = \max_{1 \leq u \leq k} \{\beta_{j_u} + \sum_{j=j_1}^{j_{u-1}} (\beta_j - \alpha_j)\}$$

$$I^*_2(\mathcal{B}) = \max_{k+1 \leq u \leq |\mathcal{B}|} \{\beta_{j_u} + \sum_{j=j_1}^{j_{u-1}} (\beta_j - \alpha_j)\}.$$

Furthermore, we observe that

$$\begin{aligned} I^*(\mathcal{B} \cup \{q\}) = \max\{&I^*_1(\mathcal{B}), \beta_q + \sum_{j=j_1}^{j_k} (\beta_j - \alpha_j), \\ &I^*_2(\mathcal{B}) + (\beta_q - \alpha_q)\}. \end{aligned}$$

Since $\beta_q - \alpha_q \leq 0$ and $\beta_q \leq \beta_{j_{k+1}}$, we obtain

$$\begin{aligned} I^*_2(\mathcal{B}) &\geq \max\{\beta_{j_{k+1}} + \sum_{j=j_1}^{j_k} (\beta_j - \alpha_j), I^*_2(\mathcal{B}) + (\beta_q - \alpha_q)\} \\ &\geq \max\{\beta_q + \sum_{j=j_1}^{j_k} (\beta_j - \alpha_j), I^*_2(\mathcal{B}) + (\beta_q - \alpha_q)\}. \end{aligned}$$

From the above analysis we conclude that $I^*(\mathcal{B} \cup \{q\}) \leq \max\{I^*_1(\mathcal{B}), I^*_2(\mathcal{B})\}$. Therefore, the claim holds.

Now, let $r = \arg\max_{j \in \mathcal{B}_2} \beta_j$ and $\bar{\mathcal{B}}_2 = \{1, 2, \ldots, r\}$. By using the claim we can show that $I^*(\bar{\mathcal{B}}_2) \leq I^*(\mathcal{B}_2)$. Also, from (1) we infer that $I^*(\mathcal{B}_1 \cup \{u\}) \geq I^*(\mathcal{B}_1)$ for any $u \geq l$. Therefore, noting that $r \geq l$ (since $\mathcal{B}_2 \nsubseteq \{1, \ldots, l-1\}$), we have $I^*(\mathcal{B}_1) \leq I^*(\bar{\mathcal{B}}_2)$. Hence the result. □

The implication of Lemma 2 combined with Corollary 1 is the following. When $\eta_{\max} \leq 1$, suppose given a set of tasks and a subset $\mathcal{B}$ containing tasks with least communication times, i.e., communication time of a task in $\mathcal{B}$ is less than that of any other task that does not belong to $\mathcal{B}$. The idle time of a schedule that places all tasks from $\mathcal{B}$ on processor 1 in the increasing order of their communication times cannot be greater than that of any other schedule applied to any subset of the original set, unless that subset is a proper subset of $\mathcal{B}$. Later, in the proof of Theorem 1 we use Lemma 2 to argue that the idle time on processor 1 under SPaC is no greater than the idle time under an optimal offline schedule.

We now present one of our key results in Theorem 1.

Theorem 1. *If $\eta_{max} \leq 1$, then SPaC is θ_1-competitive for $\mathcal{P}$, where*

$$\begin{aligned} \theta_1 &= \min\{\theta_{11}, \theta_{12}\} \\ \theta_{11} &= 1 + \frac{\alpha_{max}}{\sum_{j=1}^{n} \alpha_j} \\ \theta_{12} &= 1 + \min\left\{ \frac{\max\{1, \rho\}}{\rho + 1}, \frac{1}{\rho}, \rho \right\}. \end{aligned}$$

Proof. The schedule produced by SPaC is such that $\mathcal{T}_1(\mathbf{s}^S) = \{1, \ldots, k-1\}$ and $\mathcal{T}_0(\mathbf{s}^S) = \{k, \ldots, n\}$, where $1 \leq k \leq n+1$. Note that, if $k = 1$, $\mathcal{T}_1(\mathbf{s}^S) = \emptyset$, and if $k = n+1$, $\mathcal{T}_0(\mathbf{s}^S) = \emptyset$. We now consider the following cases.

Case 1: $C_0(\mathbf{s}^S) \leq C_1(\mathbf{s}^S)$, i.e., $C_{\max}(\mathbf{s}^S) = C_1(\mathbf{s}^S)$. For this case we have $k > 1$. We claim that $\mathcal{T}_1(\mathbf{s}^*) \nsubseteq \{1, \ldots, k-2\}$. Otherwise, scheduling any task $j \in \mathcal{T}_1(\mathbf{s}^S) \backslash \mathcal{T}_1(\mathbf{s}^*)$ on processor 0 by SPaC would have reduced the makespan over $C_{\max}(\mathbf{s}^S)$. In particular, scheduling task $k-1$ on processor 0 by SPaC would have reduced the makespan. However, this could not be true since, by the definition of k,

SPaC has already checked for this condition and scheduled task $k-1$ on processor 1. Hence, by contradiction the claim is true. Therefore, from Corollary 1 and Lemma 2 we have $I(\mathbf{s}^*) \geq I(\mathbf{s}^S)$.

Now we know that $C_{\max}(\mathbf{s}^S) - \rho\alpha_{k-1} \leq C_0(\mathbf{s}^S)$, since otherwise, the processing of task $k-1$ would have completed on processor 0 first, and under SPaC we would have $k-1 \in \mathcal{T}_0(\mathbf{s}^S)$. Therefore, we obtain

$$\begin{aligned}
&\frac{1}{\rho}C_0(\mathbf{s}^S) + C_1(\mathbf{s}^S) - I(\mathbf{s}^S) = \sum_{j=1}^{n}\alpha_j \\
\Rightarrow &\frac{1}{\rho}[C_{\max}(\mathbf{s}^S) - \rho\alpha_{k-1}] + C_{\max}(\mathbf{s}^S) - I(\mathbf{s}^S) \leq \sum_{j=1}^{n}\alpha_j \\
\Rightarrow &C_{\max}(\mathbf{s}^S) \leq \frac{\rho}{\rho+1}\left[\sum_{j=1}^{n}\alpha_j + \alpha_{k-1} + I(\mathbf{s}^S)\right] \\
\Rightarrow &C_{\max}(\mathbf{s}^S) \leq \frac{\rho}{\rho+1}\left[\sum_{j=1}^{n}\alpha_j + \alpha_{k-1} + I(\mathbf{s}^*)\right] \\
\Rightarrow &C_{\max}(\mathbf{s}^S) \leq C^*_{\max} + \frac{\rho\alpha_{k-1}}{\rho+1}. \qquad (6)
\end{aligned}$$

The last inequality above is due to (3).

We substitute (4) into (6) to obtain $\frac{C_{\max}(\mathbf{s}^S)}{C^*_{\max}} \leq \theta_{11}$ and substitute (5) into (6) to obtain

$$\frac{C_{\max}(\mathbf{s}^S)}{C^*_{\max}} \leq 1 + \frac{\rho}{\min\{1,\rho\}(\rho+1)} = 1 + \frac{\max\{1,\rho\}}{(\rho+1)}.$$

Furthermore, we have the relation $C_{\max}(\mathbf{s}^S) \leq \sum_{j=1}^{n}\alpha_j + I(\mathbf{s}^S)$. We use $I(\mathbf{s}^*) \geq I(\mathbf{s}^S)$ and (3) to obtain $\frac{C_{\max}(\mathbf{s}^S)}{C^*_{\max}} \leq 1 + \frac{1}{\rho}$.

Case 2: $C_0(\mathbf{s}^S) \geq C_1(\mathbf{s}^S)$, i.e., $C_{\max}(\mathbf{s}^S) = C_0(\mathbf{s}^S)$. For this case we have $k \leq n$. We claim that $\mathcal{T}_1(\mathbf{s}^*) \nsubseteq \{1,\ldots,k-1\}$. This claim can be proved using a similar argument as in **Case 1**. Therefore, from Corollary 1 and Lemma 2 we have $I(\mathbf{s}^*) \geq I(\mathbf{s}') \geq I(\mathbf{s}^S)$, where $\mathbf{s}'$ is a schedule under which $\mathcal{T}_1(\mathbf{s}') = \mathcal{T}_1(\mathbf{s}^S) \cup \{k\}$, and the tasks of set $\mathcal{T}_1(\mathbf{s}')$ are transmitted in the increasing order of their communication times.

Now, we also have

$$\max\{C_1(\mathbf{s}^S), \sum_{j=1}^{k}\beta_j\} + \alpha_k \geq C_{\max}(\mathbf{s}^S) \qquad (7)$$

since otherwise, the processing of task k would have completed on processor 1 first and under SPaC we would have $k \in \mathcal{T}_1(\mathbf{s}^S)$. If $C_1(\mathbf{s}^S) \geq \sum_{j=1}^{k}\beta_j$, then $C_{\max}(\mathbf{s}^S) - \alpha_k \leq C_1(\mathbf{s}^S)$ and similar steps as in (6) can be followed to prove the bounds θ_{11} and θ_{12}. If $C_1(\mathbf{s}^S) < \sum_{j=1}^{k}\beta_j$, then we proceed as follows. For this case we have the following inequalities.

$$\begin{aligned}
&\sum_{j=1}^{k}\beta_j \geq C_{\max}(\mathbf{s}^S) - \alpha_k \\
\Rightarrow &C_1(\mathbf{s}^S) - I(\mathbf{s}^S) + \sum_{j=1}^{k}\beta_j - \sum_{j=1}^{k-1}\alpha_j \\
&\geq C_{\max}(\mathbf{s}^S) - \alpha_k \\
\Rightarrow &C_1(\mathbf{s}^S) - I(\mathbf{s}^S) \\
&\geq C_{\max}(\mathbf{s}^S) - \alpha_k - \left(\sum_{j=1}^{k}\beta_j - \sum_{j=1}^{k-1}\alpha_j\right) \\
&\geq C_{\max}(\mathbf{s}^S) - \alpha_k - I(\mathbf{s}^*).
\end{aligned}$$

In the second step above we have used $C_1(\mathbf{s}^S) = I(\mathbf{s}^S) + \sum_{j=1}^{k-1}\alpha_j$, and in the last inequality, we have used $I(\mathbf{s}^*) \geq I(\mathbf{s}') \geq \sum_{j=1}^{k}\beta_j - \sum_{j=1}^{k-1}\alpha_j$. Therefore,

$$\begin{aligned}
&\frac{1}{\rho}C_0(\mathbf{s}^S) + C_1(\mathbf{s}^S) - I(\mathbf{s}^S) = \sum_{j=1}^{n}\alpha_j \\
\Rightarrow &\frac{1}{\rho}C_{\max}(\mathbf{s}^S) + C_{\max}(\mathbf{s}^S) - \alpha_k - I(\mathbf{s}^*) \leq \sum_{j=1}^{n}\alpha_j \\
\Rightarrow &C_{\max}(\mathbf{s}^S) \leq \frac{\rho}{\rho+1}\left[\sum_{j=1}^{n}\alpha_j + \alpha_k + I(\mathbf{s}^*)\right] \\
\Rightarrow &C_{\max}(\mathbf{s}^S) \leq C^*_{\max} + \frac{\rho\alpha_k}{\rho+1}. \qquad (8)
\end{aligned}$$

We substitute (4) into (8) to obtain $\frac{C_{\max}(\mathbf{s}^S)}{C^*_{\max}} \leq \theta_{11}$ and substitute (5) into (8) to obtain

$$\frac{C_{\max}(\mathbf{s}^S)}{C^*_{\max}} \leq 1 + \frac{\max\{1,\rho\}}{\rho+1}.$$

Furthermore, from (7) we have $C_{\max}(\mathbf{s}^S) \leq C_{\max}(\mathbf{s}')$. Therefore,

$$C_{\max}(\mathbf{s}^S) \leq \sum_{j=1}^{k}\alpha_j + I(\mathbf{s}') \leq \sum_{j=1}^{n}\alpha_j + I(\mathbf{s}^*)$$

We use (3) in the above inequality to obtain $\frac{C_{\max}(\mathbf{s}^S)}{C^*_{\max}} \leq 1 + \frac{1}{\rho}$.

Finally, from Lemma 1 we have $\frac{C_{\max}(\mathbf{s}^S)}{C^*_{\max}} \leq 1 + \rho$. Hence the result. □

The main implication of Theorem 1 is that when $\eta_{\max} \leq 1$, SPaC has $O(1)$ competitive ratio. In the following we present several additional observations on θ_1.

Remark 1: A simple upper bound for θ_1 can be obtained by solving for ρ that maximizes θ_{12}. The solution is $\rho = \frac{\sqrt{5}+1}{2}$ and the upper bound is $\frac{\sqrt{5}+1}{2} \approx 1.618$.

Remark 2: For $\rho < \frac{\sqrt{5}+1}{2}$ we have $\theta_1 \leq 1 + \min\{\frac{\max\{1,\rho\}}{1+\rho}, \rho\}$, and for $\rho \geq \frac{\sqrt{5}+1}{2}$ we have $\theta_1 \leq 1 + \frac{1}{\rho}$. Note that for $\rho >> 1$, θ_1 approaches 1. This is intuitive because in this case any competent algorithm will schedule all the tasks on processor 1 and the problem reduces to the two-machine flow shop problem, for which SPaC gives an optimal schedule when $\eta_{\max} \leq 1$ (Corollary 1).

Furthermore, for $\rho = 1$ and $\eta_{\max} \leq 1$, SPaC has competitive ratio $\frac{3}{2}$. In Proposition 1 below, we observe that no deterministic semi-online algorithm can have a competitive ratio better than $\frac{3}{2}$. This suggests that in this case SPaC is the most competitive among all deterministic semi-online algorithms.

Proposition 1. *For $\eta_{max} \leq 1$ and $\rho = 1$, the competitive ratio of any semi-online algorithm with predetermined scheduling order is at least $\frac{3}{2}$.*

Proof. The proof is presented in Section 10.1. □

Remark 3: We observe that SPaC is asymptotically optimal with respect to n under the following regularity condition [31]:

$$\lim_{n\to\infty} \frac{\alpha_{\max}}{\sum_{j=1}^{n} \alpha_j} = 0.$$

This condition describes problem instances where no task itself is dominating in terms of processing time. By applying it in the result of Theorem 1, we see that if $\eta_{\max} \leq 1$, $\lim_{n\to\infty} \frac{C_{\max}(\mathbf{s}^S)}{C^*_{\max}} = 1$. This asymptotic optimality property of SPaC makes it highly desirable for large n, where the offline computation of an optimal schedule quickly becomes intractable.

6 SPaC-R Competitive Ratio for Large Communication Times

In this section we analyse the competitive ratio of SPaC-R when $\eta_{\min} \geq 1$, i.e., $\alpha_j \leq \beta_j$, for all j. We first establish the following lemma that finds a closed form expression for completion time on processor 1 under SPaC-R.

Lemma 3. *Given a schedule* $\mathbf{s}$ *where the tasks assigned to processor* 1 *are transmitted in the increasing order of their communication times, if* $\eta_{min} \geq 1$*, then* $C_1(\mathbf{s}) = \sum_{j\in\mathcal{T}_1(\mathbf{s})} \beta_j + \alpha_l$*, where task* $l = \arg\max_{j\in\mathcal{T}_1(\mathbf{s})} \beta_j$*.*

Proof. We have $\beta_j \geq \alpha_j$, for all j. Also, under $\mathbf{s}$ we have $\beta_{g(k)} \leq \beta_{g(k+1)}$. This implies $\beta_{g(k+1)} \geq \alpha_{g(k)}$. Using these in (2) we get

$$C_1(\mathbf{s}) = \sum_{j\in\mathcal{T}_1(\mathbf{s})} \beta_j + \alpha_{g(|\mathcal{T}_1(\mathbf{s})|)}.$$

Furthermore, under $\mathbf{s}$ we have $g(|\mathcal{T}_1(\mathbf{s})|) = l$. Hence the result. □

Since SPaC-R schedules the tasks assigned to processor 1 in the increasing order of their communication times, from Lemma 3 we conclude that the completion time on processor 1 is dominated by the communication times of the tasks. Therefore, the makespan in this case is determined by the processing times of tasks on processor 0 and their communication times. As stated in Section 4.2, this fact forms the guiding principle in the design of SPaC-R. In the following theorem we present the competitive ratio of SPaC-R for $\eta_{\min} \geq 1$.

Theorem 2. *If* $\eta_{min} \geq 1$*, then SPaC-R is* θ_1'*-competitive for* $\mathcal{P}$*, where*

$$\theta_1' = \begin{cases} 2 & \text{if } \rho \leq 1, \\ 4 + \frac{1}{\rho} & \text{if } 1 < \rho < 2, \\ \frac{9}{2} & \text{if } \rho \geq 2. \end{cases}$$

Proof. We refer to the time to process the set of tasks $\mathcal{T}^{(l)}$ in iteration l of SPaC-R as the *schedule length* of this iteration. Let $C^{(l)}_{\max}$ denote the schedule length in iteration l. Let $C^{(l)}_i$ and $\mathcal{T}^{(l)}_i$ denote the schedule length and the set of tasks scheduled, respectively, on processor i in iteration l. Let $\gamma_j = \min\{\rho\alpha_j, \beta_j\}$.

We note that in the first iteration of SPaC-R, the processing time of any task j scheduled on processor 0 is γ_j because in this case a task j scheduled on processor 0 is cancelled if $\rho\alpha_j$ exceeds β_j. Therefore,

$$C^{(1)}_{\max} \leq \sum_{j\in\mathcal{T}} \gamma_j. \tag{9}$$

In the following we derive a lower bound for $C^*_{\max}$.

Let C^*_0 and C^*_1 denote the completion times, and $\mathcal{T}^*_0$ and $\mathcal{T}^*_1$ the set of tasks scheduled, on processor 0 and processor 1, respectively, under the optimal offline schedule. We have

$$C^*_0 = \sum_{j\in\mathcal{T}^*_0} \rho\alpha \geq \sum_{j\in\mathcal{T}^*_0} \min\{\rho\alpha_j, \beta_j\} = \sum_{j\in\mathcal{T}^*_0} \gamma_j.$$

Also, note that the completion time on processor 1 should be at least the total communication times of the tasks from $\mathcal{T}^*_1$. Therefore,

$$C^*_1 \geq \sum_{j\in\mathcal{T}^*_1} \beta_j \geq \sum_{j\in\mathcal{T}^*_1} \gamma_j.$$

Hence, we have

$$C^*_0 + C^*_1 \geq \sum_{j\in\mathcal{T}^*_0} \gamma_j + \sum_{j\in\mathcal{T}^*_1} \gamma_j.$$

Furthermore, since $C^*_{\max} = \max\{C^*_0, C^*_1\}$ and $\mathcal{T} = \mathcal{T}^*_0 \cup \mathcal{T}^*_1$, we have

$$C^*_{\max} \geq \frac{1}{2}(C^*_0 + C^*_1) \geq \frac{1}{2}\sum_{j\in\mathcal{T}} \gamma_j. \tag{10}$$

From (9) and (10) we obtain $C^{(1)}_{\max} \leq 2C^*_{\max}$. Now, note that if $\rho \leq 1$, no task scheduled on processor 0 will be cancelled as $\beta_j \geq \alpha_j \geq \rho\alpha_j$, or $\gamma_j = \rho\alpha_j$, for all j. All the tasks will be finished in the first iteration. Therefore, SPaC-R is 2-competitive for $\rho \leq 1$.

If $\rho > 1$, then in the second iteration, in the worst case, SPaC-R may schedule all the tasks on processor 1. The schedule length of SPaC-R when all the tasks from $\mathcal{T}^{(2)}$ are scheduled on processor 1 can be deduced from Lemma 3 and is given on the right-hand side of the following inequality:

$$C^{(2)}_{\max} \leq \sum_{j\in\mathcal{T}^{(2)}} \beta_j + \alpha_l, \tag{11}$$

where $l = \arg\max_{j\in\mathcal{T}^{(2)}} \beta_j$. Using $\beta_l \geq \alpha_l$ in (5), we get

$$\begin{aligned} C^*_{\max} &\geq \min\{\rho\alpha_l, \beta_l + \alpha_l\} \\ &\geq \min\{\rho, 2\}\alpha_l. \end{aligned} \tag{12}$$

Note that for all j in $\mathcal{T}^{(2)}, \rho\alpha_j > \beta_j$, because each of those tasks are cancelled in the first iteration only when $\rho\alpha_j$ exceeds β_j. Now, using (12) and $\rho\alpha_j > \beta_j$, for all j in $\mathcal{T}^{(2)}$, in (11) we get

$$\begin{aligned} C^{(2)}_{\max} &\leq \sum_{j\in\mathcal{T}^{(2)}} \min\{\rho\alpha_j, \beta_j\} + C^*_{\max}/\min\{\rho, 2\} \\ &\leq \sum_{j=1}^{n} \gamma_j + C^*_{\max}/\min\{\rho, 2\} \\ &\leq (2 + 1/\min\{\rho, 2\})\, C^*_{\max}. \end{aligned}$$

In the last inequality we have used (10). The result follows by noting that $C_{\max}(\mathbf{s}^{SR}) = C^{(1)}_{\max} + C^{(2)}_{\max}$. □

From Theorem 2 we conclude that the competitive ratio of SPaC-R is upper bounded by 5.

Remark 4: An interesting feature of SPaC-R is that, if processor 1 is much faster than processor 0, i.e., $\rho >> 1$, then the competitive ratio $\frac{9}{2}$ is tight; see Section 10.2 for a proof.

7 Competitive ratio for general η_j

In the previous sections we analysed the performance of SPaC and SPaC-R for the cases $\eta_{\max} \geq 1$ and $\eta_{\min} \leq 1$, respectively. By leveraging the results from Section 5 and Section 6, we provide competitive ratios for SPaC and SPaC-R for general η_j, i.e., no restriction on the values taken by either $\eta_{\max}$ or $\eta_{\min}$, in Theorems 3 and 4, respectively. We then remark on the performance of SPaC when $\eta_{\min} \geq 1$ and the performance of SPaC-R for $\eta_{\max} \leq 1$.

Theorem 3. *SPaC is θ-competitive for $\mathcal{P}$, where*

$$\theta = \min\{\max\{1, \eta_{max}\}\theta_1, 1+\rho\}.$$

Proof. From Lemma 1 we have $C_{\max}(\mathbf{s}^S) \leq (1+\rho)C^*_{\max}$. For $\eta_{\max} \leq 1$, $C_{\max}(\mathbf{s}^S) \leq \theta_1 C^*_{\max}$ follows from Theorem 1. For $\eta_{\max} > 1$ we proceed as follows.

Let $\alpha_j(P)$ and $\beta_j(P)$ denote the remote processing time and communication time of task j in a problem instance P. Let $\eta_{\max}(P) = \max_j \frac{\beta_j(P)}{\alpha_j(P)} > 1$. Suppose P' is another problem instance related to P as follows. In P', the remote processing time of each task j is $\eta_{\max}(P)\alpha_j(P)$ and the communication time of the task is $\beta_j(P)$. Note that for problem instance P', the condition $\eta_{\max}(P') \leq 1$ is satisfied.

We use $\mathbf{s}(P)$ to denote a particular schedule $\mathbf{s}$ used to solve problem instance P. To distinguish between the makespans of P and P', we use $C_{\max}(\mathbf{s}(P))$ and $C'_{\max}(\mathbf{s}(P'))$, respectively. We have the following inequalities:

$$\begin{aligned}
\eta_{\max}(P)C_{\max}(\mathbf{s}^*(P)) &\geq C'_{\max}(\mathbf{s}^*(P)) \\
&\geq C'_{\max}(\mathbf{s}^*(P')) \\
&\geq \frac{C'_{\max}(\mathbf{s}^S(P'))}{\theta_1} \\
&\geq \frac{C_{\max}(\mathbf{s}^S(P))}{\theta_1}. \qquad (13)
\end{aligned}$$

In the above, we note that $C'_{\max}(\mathbf{s}^*(P))$ is the makespan evaluated for problem instance P' under an optimal schedule given for problem instance P. The third inequality is due to Theorem 1 applied to problem instance P'. The final inequality is due to SPaC being applied to both problems P and P'. Since the communication times of the tasks are the same in both problem instances, the list order of the tasks for both problem instances is the same. This implies that the completion times on the processors under P' is at least as late as that of P. Finally, since (13) holds for any problem instance P with $\eta_{\max}(P) > 1$, the result follows. □

The result in Theorem 3 shows that SPaC has a small competitive ratio as long as the communication time of any task is moderate when compared to its processing time at the remote processing.

Theorem 4. *SPaC-R is θ'-competitive for $\mathcal{P}$, where*

$$\theta' = \max\{1, 1/\eta_{min}\}\theta'_1.$$

Proof. The proof uses a similar approach as the proof of Theorem 3 and is presented in Section 10.3. □

Theorem 4 asserts that SPaC-R is a desirable choice only when the communication times dominate the processing times at processor 1.

Remark 5: In the following we comment on the performance of SPaC for the case $\eta_{\min} \geq 1$. Consider the following family of problem instances. $\alpha_j = 1$, for all j in $\{1, 2, \ldots, n-1\}$, $\alpha_n = (n-1)^2$, $\beta_j = n(n-1)$, for all j, and $\rho = n$. Note that $\eta_{\min} \geq 1$ for these problem instances. Since all β_j are equal in this case, SPaC cannot differentiate the tasks and may schedule task n on processor 0. Then the rest of the tasks will end up being processed on processor 1. In this case, $C_0(\mathbf{s}^S) = n(n-1)^2$ and $C_1(\mathbf{s}^S) = n(n-1)^2+1$, and the makespan $C_{\max}(\mathbf{s}^S) = n(n-1)^2+1$.

The optimal schedule is obtained by scheduling task n on processor 1 and processing all the other tasks on processor 0. The optimal makespan is $C^*_{\max} = n(n-1)+1$. Therefore,

$$\frac{C_{\max}(\mathbf{s}^S)}{C^*_{\max}} \geq \frac{n(n-1)^2+1}{n(n-1)+1}.$$

From the above analysis we conclude that SPaC has competitive ratio of at least $\frac{n(n-1)^2+1}{n(n-1)+1}$ for $\eta_{\min} \geq 1$. Therefore, SPaC-R is better than SPaC in terms of competitive ratio for this case.

Remark 6: In the following we give a simple example to show that SPaC-R does not have any better competitive ratio when compared with SPaC for the scenario $\eta_{\max} \leq 1$. Consider the following problem instance: $n = 3, \rho = 1, \alpha_j = 10$, for all j, and $\beta_j = 10 - \delta$, for all j, where δ is a positive real number close to zero. Clearly, for this problem instance $\eta_{\max} < 1$. We claim that for this problem instance the makespan achieved by SPaC-R is $40 - 2\delta$.

Since the communication times of the tasks are the same in this case, they cannot be differentiated by the algorithm. Therefore, in the first iteration, the algorithm may schedule task 1 on processor 0 and start transmitting task 3 to processor 1. Task 1 will be cancelled since $\rho\alpha_1 > \beta_1$. Task 2 will end up being processed on processor 0 and transmitted to processor 1 simultaneously. However, it will be cancelled on processor 0 before its processing is completed on processor 1. Thus tasks $\{1, 2\}$ will be scheduled in the second iteration. The schedule length in the first iteration will be $20 - \delta$.

In the second iteration of the algorithm, the schedule length is again $20 - \delta$. This is achieved by processing one task on processor 0 and the other task on processor 1. This results in a makespan of $40 - 2\delta$. The optimal makespan is 20, which is achieved by scheduling two tasks on processor 0 and the other task on processor 1. Thus, the competitive ratio of SPaC-R for this problem instance is 2, since δ can be chosen close to zero. From the above analysis we conclude that, for $\eta_{\max} \leq 1$, the competitive ratio of SPaC-R is at least 2, whereas the competitive ratio of SPaC is upper bounded by $\frac{\sqrt{5}+1}{2} \approx 1.68$.

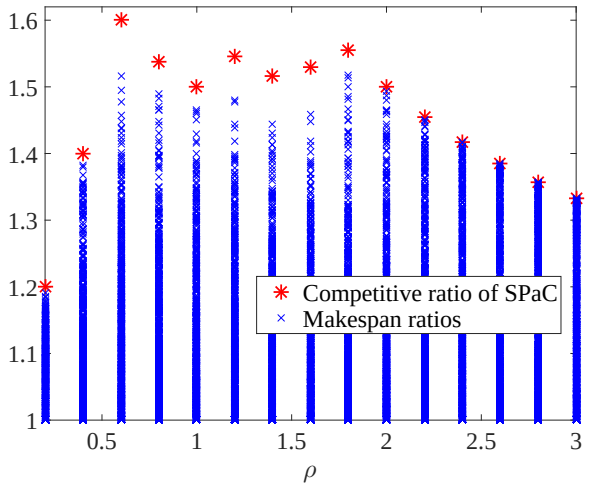


Fig. 2. Tightness of competitive ratio of SPaC, for $n = 5$ and $\eta_{\max} \leq 1$.

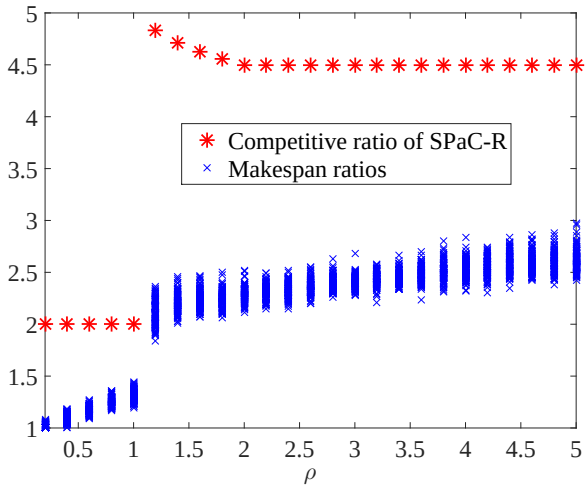


Fig. 3. Tightness of competitive ratio of SPaC-R, for $n = 5$ and $\eta_{\min} \geq 1$.

8 Simulation Results

In this section, using simulation we first study the tightness of the derived *competitive ratios* for SPaC and SPaC-R for $\eta_{\max} \leq 1$ and $\eta_{\min} \geq 1$, respectively. We then compare the *average performance* of SPaC and SPaC-R against alternatives for general η values.

8.1 Tightness of Competitive Ratios θ_1 and θ_1'

Recall that a competitive ratio is tight if there exits at least one problem instance P for which the makespan ratio, i.e., $\frac{C_{\max}(\mathbf{s}(P))}{C^*_{\max}(\mathbf{s}^*(P))}$, is equal to the competitive ratio. To evaluate the tightness of the derived competitive ratios, we aim at estimating the maximum of the makespan ratios achieved by SPaC and SPaC-R using simulation, and compare them with θ_1 and θ_1', respectively. For each of the algorithms, under different ρ values, we randomly generate 5000 problem instances and observe the makespan ratios. To find $C^*_{\max}(\mathbf{s}^*(P))$, we use exhaustive search over all possible bi-partitions of the set of tasks. Given any bi-partition, Johnson's rule [30] is used to obtain the minimum completion time on processor 1. Due to the extremely high computational complexity of this approach, we limit the number of tasks n to 5. For large n values we study the average makespan performance of SPaC and SPaC-R in Section 8.2.

In Figure 2, we compare the competitive ratio θ_1 with the makespan ratios achieved by SPaC, for varying ρ when $\eta_{\max} \leq 1$. We observe that θ is tight for ρ greater than 2.2, and it is nearly tight for other values of ρ. In Figure 3, we compare the competitive ratio θ_1' with the makespan ratios achieved by SPaC-R, for varying ρ when $\eta_{\min} \geq 1$. We infer that θ_1' may not be tight. Recall from Remark 4 that θ_1' is tight for $\rho >> 1$. Therefore, it remains an open question if there exists problem instances for which θ_1' is achieved for ρ values comparable to 1. If θ_1' is not tight for ρ values comparable to 1, then another interesting and challenging question is if a better competitive ratio can be proved for SPaC-R. We emphasize here that θ' being not tight simply means that SPaC-R performs in simulation even better than what we can analytically predict through the derivation of θ'.

8.2 Average Performance Comparison

We simulate the two-processor task offloading system in MATLAB. The default value of ρ is set to 5. The following parameter values are chosen based on the well-known experimental results from MAUI [22]. The data size of a task is chosen from an exponential distribution with default mean 562.5 kB (typical data size of tasks are of the order of hundreds of kB). Typical uplink data rate in today's cellular network (e.g. LTE) is 3 Mbps. Thus, β_j, for all j, are determined by the above parameters and its mean value is 1500 ms. We assume that the processing times α_j, for all j, are exponentially distributed with mean 1500 ms. This choice was made based on the typical execution time required for one run of face recognition application on current smartphones [22]. Note that in the above parameter settings we do not restrict the values of $\eta_{\max}$ or $\eta_{\min}$. In other words, we conduct the simulation for general and random ratios between communication time and remote processing time. We note that similar results have been observed when other distributions are used, but such results are omitted to avoid redundancy.

Since we are not aware of any semi-online algorithm that accounts for communication delay, we first compare the average makespan performance of SPaC and SPaC-R with the online list scheduling algorithm [2]. We simply ignore communication delay in applying the list scheduling decisions to schedule tasks. We further note that comparison with the online algorithmic framework of [3] is not viable, since there is no clear means to accommodate communication delay for the tasks that are cancelled and rescheduled.

To further demonstrate the strength of SPaC and SPaC-R, we also compare them with offline scheduling algorithms that account for communication delay. Since the offline problem is NP-hard, we resort to two heuristics reported in [9]. For convenience of exposition, we name the two heuristics offlineMin and offlineMax. Algorithm 3 describes offlineMin. offlineMax only differs from offlineMin in line 8 of Algorithm 3 where $\arg\min$ is replaced by $\arg\max$. We note that these heuristics *require a priori knowledge of processing times of tasks* and have a time complexity of $O(n^2)$.

Algorithm 3: offlineMin heuristic

Initialize $\mathcal{B} \leftarrow \mathcal{T}$
while $\mathcal{B} \neq \emptyset$ **do**
  for each Processor i **do**
    for each task $j \in \mathcal{B}$ **do**
      Evaluate $f(j,i)$ = Minimum completion time of task j if mapped to processor i.
    end for
  end for
  $k = \arg\min_{j \in \mathcal{B}} \min_{i \in \{0,1\}} f(j,i)$
  Schedule task k on processor i immediately
  $\mathcal{B} \leftarrow \mathcal{B} \backslash \{k\}$
end while

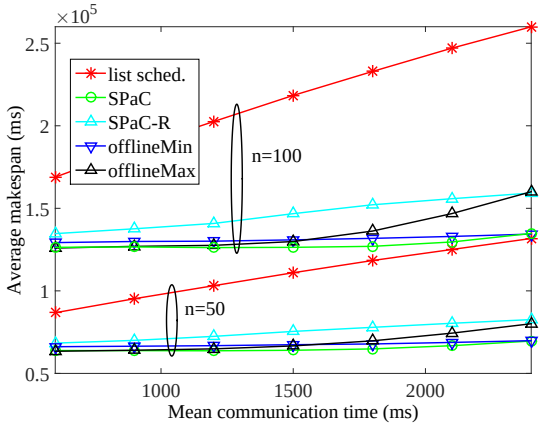


Fig. 4. Comparison of average makespan for different algorithms. $\rho = 5$.

For each parameter setting, we generate 5000 problem instances to evaluate the average makespan for each algorithm. Figure 4 and Figure 5 compare the algorithms, with varying mean communication time and ρ, respectively. We observe that both SPaC and SPaC-R perform significantly better than list scheduling. Therefore, we conclude that scheduling tasks without taking communication times into consideration will have considerably detrimental effect on the makespan. Furthermore, we observe that the performance of SPaC is comparable to or even better than the offline heuristics. Finally, we note that, although SPaC-R provides a smaller competitive ratio when $\eta_{\min} \geq 1$, on average SPaC outperforms SPaC-R in general. This is because restarting tasks in SPaC-R penalizes the makespan, which is not the case for SPaC.

9 Conclusions

We have studied the problem of computational task offloading with communication delay. Without assuming a priori knowledge of task processing times, we have proposed the SPaC algorithm and variant SPaC-R, proving $O(1)$ competitive ratio for $\eta_{\max} \leq 1$ and $\eta_{\min} \geq 1$, respectively. We have also derived competitive ratios for general η_j. Simulation results further suggest that SPaC provides average makespans that can be as small as those obtained from the best known offline heuristics.

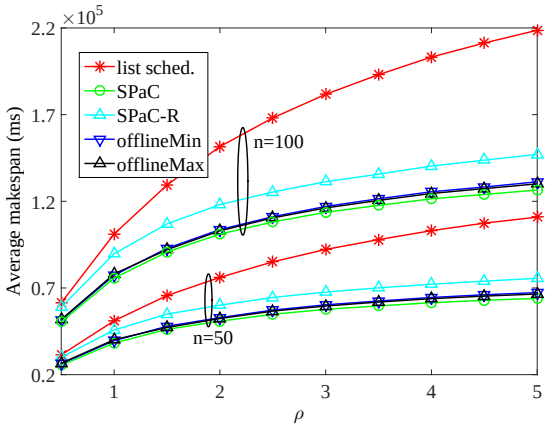


Fig. 5. Comparison of average makespan for different algorithms. Mean communication time 1500.

In this work we have assumed that all the tasks are available at time 0. However, using the approach in [3] we may extend SPaC to schedule tasks that arrive randomly in time. In such a scenario, a 2θ competitive ratio can be derived from Theorem 3 and Theorem 2.1 in [3].

10 Additional Proofs

10.1 Proof of Proposition 1

We argue that an adversary can construct problem instances for any given semi-online algorithm with pre-determined scheduling order for which the ratio of makespan achieved by the algorithm to the optimal makespan approaches $\frac{3}{2}$. Consider three tasks with equal communication times, $\beta_j = \beta \leq 1$, for all j. Let $\alpha_1 = 1, \alpha_2 = 1$ and $\alpha_3 = 2$. We note that the only known information about the tasks is the communication times which are equal in this problem instance and hence the tasks are indistinguishable. Now, given the pre-determined scheduling order of the tasks by the semi-online algorithm, the adversary can present the tasks in such a way that the algorithm schedules task 1 and task 2 first, and then schedule task 3, which results in a makespan of at least 3. The optimal makespan $2 + \beta$ is obtained by scheduling tasks 1 and 2 on processor 0 and scheduling task 3 on processor 1. Therefore, the competitive ratio that can be achieved by the semi-online algorithm is at least $\frac{3}{2+\beta}$ for the given problem instance. The result follows by noting that β can be chosen arbitrarily small.

10.2 Proof of Remark 4

To show that the competitive ratio $\frac{9}{2}$ is tight for $\rho >> 1$, we provide the following problem instance, where $n = 8$,

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

$$\alpha_j = \begin{cases} \delta/\rho & j = 1,2,3,4 \\ (10+\delta)/\rho & j = 5,6,7 \\ 10 + \delta & j = 8, \end{cases}$$

where δ is a positive real number close to zero. Now, we claim that in the worst case SPaC-R may achieve a makespan of 90 for the above problem instance. To see this

we summarize a possible execution sequence of the tasks of the problem instance in both iterations of the algorithm.

In the first iteration, tasks $\{5,6,7,8\}$ are scheduled on processor 0 and tasks $\{1,2,3,4\}$ are scheduled on processor 1. Since $\beta_j < \rho\alpha_j$, for all j in $\{5,6,7,8\}$, the tasks will be cancelled in the first iteration. The schedule length in the first iteration is $\max\{40, 40-4\delta+\delta/\rho\} = 40$, since δ is close to zero and $\rho >> 1$.

In the second iteration, the communication times of the tasks $\{5,6,7,8\}$ are equal and hence cannot be differentiated by the algorithm. This may lead the algorithm to start processing task 8 on processor 0 and transmit task 5 to processor 1. Since $\rho >> 1$, task 8 is processed for a long duration on processor 0 and eventually the algorithm will transmit task 8 to processor 1 after transmitting tasks $\{5,6,7\}$. Therefore, all of the tasks $\{5,6,7,8\}$ will be completed on processor 1. This results in a schedule length of 50 in the second iteration. Therefore, in the worst case SPaC-R may achieve a makespan of 90 for the above problem instance.

Now, the optimal makespan is obtained by scheduling tasks $\{1,2,3,4,5,6\}$ on processor 0 and tasks $\{7,8\}$ on processor 1 with task 8 transmitted first. This will result in an optimal makespan of $\max\{20+6\delta, 20+\delta+(10+\delta)/\rho\}$. Choosing δ close to zero and $\rho >> 1$, we can obtain an optimal makespan arbitrarily close to 20. Therefore, the makespan ratio of SPaC-R for this problem instance can be arbitrarily close to $\frac{9}{2}$. Hence the result.

10.3 Proof of Theorem 4

For $\eta_{\min} \geq 1$, $C_{\max}(\mathbf{s}^{SR}) \leq \theta_1' C^*_{\max}$ follows from Theorem 2. For $\eta_{\min} < 1$ we use a similar approach as in the proof of Theorem 3.

Let $\alpha_j(P)$ and $\beta_j(P)$ denote the remote processing time and communication time of task j in a problem instance P. Let $\eta_{\min}(P) = \min_j \frac{\beta_j(P)}{\alpha_j(P)} < 1$. Suppose P' is another problem instance related to P as follows. In P', the communication time of each task j is $\beta_j(P)/\eta_{\min}(P)$ and processing time of the task on processor 1 is $\alpha_j(P)$. Note that for problem instance P', the condition $\eta_{\min}(P') \geq 1$ is satisfied.

We use $\mathbf{s}(P)$ to denote a particular schedule $\mathbf{s}$ used to solve problem instance P. To distinguish between the makespans of P and P', we use $C_{\max}(\mathbf{s}(P))$ and $C'_{\max}(\mathbf{s}(P'))$, respectively. We have the following inequalities:

$$\begin{aligned} C_{\max}(\mathbf{s}^*(P))/\eta_{\min}(P) &\geq C'_{\max}(\mathbf{s}^*(P)) \\ &\geq C'_{\max}(\mathbf{s}^*(P')) \\ &\geq \frac{C'_{\max}(\mathbf{s}^{SR}(P'))}{\theta_1'} \\ &\geq \frac{C_{\max}(\mathbf{s}^{SR}(P))}{\theta_1'}. \end{aligned} \tag{14}$$

In the above, we note that $C'_{\max}(\mathbf{s}^*(P))$ is the makespan evaluated for problem instance P' under an optimal schedule given for problem instance P. The third inequality is due to Theorem 2 applied to problem instance P'. The final inequality is due to SPaC-R being applied to both problems P and P'. Since the communication times of the tasks in P are scaled by the same factor $1/\eta_{\min} > 1$, the list order of the tasks for both problem instances is the same. This implies that the completion times on the processors under P' is at least as late as that of P. Finally, since (14) holds for any problem instance P with $\eta_{\min}(P) < 1$, the result follows.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[2] R. L. Graham, "Bounds for certain multiprocessing anomalies," *Bell System Technical Journal*, vol. 45, pp. 1563–1541, 1966.

[3] 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.

[4] G. Fries, "Scheduling independent tasks on uniform processors," *SIAM J. Computing*, vol. 13, no. 1, pp. 705–716, 1984.

[5] A. Kovcs, "New approximation bounds for lpt scheduling." *Algorithmica*, vol. 57, no. 2, pp. 413–433, 2010.

[6] 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.

[7] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*, 1st ed. Cambridge University Press, 2011.

[8] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for scheduling parameter sweep applications in grid environments," in *Heterogeneous Computing Workshop*, 2000, pp. 349–363.

[9] A. Giersch, Y. Robert, and F. Vivien, "Scheduling tasks sharing files on heterogeneous master-slave platforms," *Journal of Systems Architecture*, vol. 52, no. 2, pp. 88–104, 2006.

[10] K. Kaya and C. Aykanat, "Iterative-improvement-based heuristics for adaptive scheduling of tasks sharing files on heterogeneous master-slave environments." *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 8, pp. 883–896, 2006.

[11] O. Beaumont, A. Legrand, and Y. Robert, "The master-slave paradigm with heterogeneous processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 9, pp. 897–908, 2003.

[12] M. Drozdowski, *Scheduling for Parallel Processing*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[13] 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.

[14] E. K. Tabak, B. B. Cambazoglu, and C. Aykanat, "Improving the performance of independent task assignment heuristics minmin, maxmin and sufferage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 5, pp. 1244–1256, May 2014.

[15] V. V. Vazirani, *Approximation Algorithms*. New York, NY, USA: Springer-Verlag New York, Inc., 2001.

[16] H. Kellerer, V. Kotov, M. G. Speranza, and Z. Tuza, "Semi on-line algorithms for the partition problem," *Operations Research Letters*, vol. 21, no. 5, pp. 235 – 242, 1997.

[17] T. E. Cheng, H. Kellerer, and V. Kotov, "Semi-on-line multiprocessor scheduling with given total processing time," *Theoretical Computer Science*, vol. 337, no. 13, pp. 134 – 146, 2005.

[18] S. Albers and M. Hellwig, "Semi-online scheduling revisited," *Theoretical Computer Science*, vol. 443, pp. 1 – 9, 2012.

[19] H. Kellerer, V. Kotov, and M. Gabay, "An efficient algorithm for semi-online multiprocessor scheduling with given total processing time," *Journal of Scheduling*, vol. 18, no. 6, pp. 623–630, 2015.

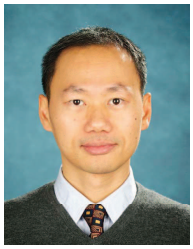
[20] C. Ng, Z. Tan, Y. He, and T. Cheng, "Two semi-online scheduling problems on two uniform machines," *Theoretical Computer Science*, vol. 410, no. 810, pp. 776 – 792, 2009.

[21] R. K. Balan, M. Satyanarayanan, S. Y. Park, and T. Okoshi, "Tactics-based remote execution for mobile computing," in *Proc. International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2003, pp. 273–286.

[22] 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. International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2010, pp. 49–62.

[23] 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.
[24] 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.
[25] J. Champati and B. Liang, "Energy compensated cloud assistance in mobile cloud computing," in *Proc. IEEE INFOCOM Workshop on Mobile Cloud Computing*, April 2014.
[26] M. Rahman, X. Li, and H. N. Palit, "Hybrid heuristic for scheduling data analytics workflow applications in hybrid cloud environment," in *Proc. IEEE IPDPS Workshops*, 2011, pp. 966–974.
[27] 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.
[28] 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.
[29] J. P. Champati and B. Liang, "One-restart algorithm for scheduling and offloading in a hybrid cloud," in *Proc. IEEE/ACM International Symposium on Quality of Service (IWQoS)*, Jun. 2015.
[30] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included," *Naval Research Logistics Quarterly*, vol. 1, no. 1, pp. 61–68, 1954.
[31] K. R. Baker and D. Trietsch, *Principles of Sequencing and Scheduling*. Wiley Publishing, 2009.

Jaya Prakash Champati is a PhD student in 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 optimization 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 with 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 as an editor for 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 senior member of IEEE and a member of ACM and Tau Beta Pi.