Semi-online Task Partitioning and Communication between Local and Remote Processors

Jaya Prakash Champati and Ben Liang
Department of Electrical and Computer Engineering, University of Toronto, Canada

***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, we propose a Semi-online Partitioning and Communication (SPaC) algorithm, to jointly select the subset of tasks to be offloaded to the remote processor and their order of transmission, with an aim to minimize the overall makespan to process all tasks. Even though the offline version of this problem, with a priori known processing times, is NP-hard, we show that the proposed semi-online algorithm achieves a small competitive ratio when the communication times of tasks are smaller than their processing times at the remote processor. For general communication times, we use simulation to demonstrate that SPaC outperforms online list scheduling and performs comparably well with known offline heuristics.**

I. Introduction

Infrastructure-as-a-Service cloud computing enables parallel task processing with local and remote processors [1]. In the most common paradigm, a local machine (e.g., a mobile device) enlists the help of a remote server (e.g., an Amazon EC2 instance), by breaking down its jobs into multiple parallel tasks and offloading some of them to be processed remotely. The remote server often has higher computational capability and less stringent energy usage constraints than the local machine. However, offloading tasks necessarily incurs communication delay. For many delay sensitive applications, the makespan to process a given set of tasks is of paramount importance. Therefore, judicious designs are required to balance the remote processing gain and communication overhead in minimizing the makespan.

Although communication delay (or communication time) is a fundamental challenge in cloud computing, it is not a part of the general system model in the classical literature on parallel processing (e.g., [2]–[4]). Some recent studies on grid and cloud computing [5]–[8], and mobile cloud computing [9] in particular, have considered the effect of communication delay on remote processing. However, as detailed in Section II, these works either propose only heuristics or study highly simplified models.

The remote processor awaits the next task arrival when it is idle. Therefore, any communication delay in the next task arrival results in idle time on the remote processor. With sub-optimal scheduling, the wastage in idle time could be much more severe. 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 III, this joint partitioning and scheduling problem is NP-hard even if all processing times are known a priori, i.e., in the *offline* setting.

Furthermore, even though the communication time to transmit a task can be reasonably inferred from its data size and the transmission bandwidth, the processing time required for the task generally is unknown without first processing it [3]. In particular, 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 will have data size on the order of tens of bytes. With typical LTE or 3G 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 time. Therefore, a practical task offloading algorithm must progress in an *online* manner, without assuming knowledge of the processing time of each unprocessed task. This requirement for online decision making, coupled with the NP-hardness of the original offline problem, introduces substantial challenges in the design of an effective and computationally efficient solution.

In this work, we study the problem of scheduling independent tasks on parallel local and remote processors to minimize the makespan of processing these tasks, without a priori knowledge of the processing times. Our main contributions are as follows:

First, 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. It has computational complexity $O(n \log n)$, where n is the number of tasks.

Second, for $\eta_{max} \leq 1$, where η_{max} is defined as the maximum ratio, among all tasks, between the communication time and the processing time on the remote processor, we prove that SPaC has a competitive ratio $\theta \leq 1 + \min\left\{\frac{\max\{1,\rho\}}{\rho+1}, \frac{1}{\rho}, \rho\right\} \leq \frac{\sqrt{5}+1}{2}$, where ρ is the speed ratio between the remote and local processors. This suggests that SPaC performs well in scenarios where the communication time of each task is small in comparison with its processing time on the remote processor. Such scenarios are of practical importance, since tasks with heavy computation and small communication payload often are prime targets of cloud computing. 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 [9]. In addition, such scenarios also commonly arise when the local and remote processors are connected by a high-bandwidth wired network.

Third, for the special case of $\eta_{max} \leq 1$ 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. *Finally*, for general η_{max}, we compare the average maskespan of SPaC with classical online list scheduling and two of the best known offline heuristics to show that the proposed solution significantly outperforms the online alternative and performs close to the offline heuristics.

II. Related Work

The related works 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.

A. 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 N processors to minimize the overall 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 $(2 - \frac{1}{N})$-approximation ratio when the processors are identical. More complex algorithms for various processor settings were studied in [3], [10]. None of the above works accounts for communication delay. For a detailed review on scheduling on parallel processors without task communication delays, the readers are referred to [8].

B. Grid and Cloud Computing with Communication Delay

The problem of scheduling independent tasks with communication overhead on multiple processors was considered in [5] and [6] 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 [7], and they were assumed infinitely divisible with processing times proportional to their data size in [8, Ch. 7]. Furthermore, all of these works assume knowledge of the processing times and hence are offline.

More recently, in mobile cloud computing research, most studies focus on energy savings at the mobile device instead of makespan minimization [9], [11]–[14]. For example, in [9], the authors gave a general guideline that tasks should be offloaded only if the local computing time of the task is greater than its remote communication and computing time.

To the best of our knowledge, our work is the first to focus on analytical modelling and optimization of the makespan with consideration for communication delay, proposing a semi-online algorithm with provable competitive ratio when communication delay is small.

III. System Model

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.

A. Processing, Communication, and Scheduling

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. Let $\alpha_{max} = \max_{j\in\mathcal{T}} \alpha_j$ and $V = \sum_{j=1}^{n} \alpha_j$.

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 $\eta_j = \frac{\beta_j}{\alpha_j}$ and $\eta_{max} = \max_{j\in\mathcal{T}} \eta_j$. 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 k-th in the sequence. Let $\mathcal{S}$ denote the set of all possible schedules.

B. Optimization Problem

Given $\mathcal{T}$ 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 [15], and hence we have

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

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

$$\begin{aligned} C_1(\mathbf{s}) &= I(\mathbf{s}) + \sum_{j\in\mathcal{T}_1(\mathbf{s})} \alpha_j \\ &= \max_{1\leq u\leq|\mathcal{T}_1(\mathbf{s})|} \left\{ \sum_{j=\mathbf{g}(1)}^{\mathbf{g}(u)} \beta_j + \sum_{j=\mathbf{g}(u)}^{\mathbf{g}(|\mathcal{T}_1(\mathbf{s})|)} \alpha_j \right\}. \end{aligned} \tag{2}$$

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 schedule and C^*_{max} denote the minimum makespan. We note that, when $b = 0$, $\mathcal{P}$ is equivalent to the problem of scheduling independent tasks on two identical processors to minimize makespan which is NP-hard [8]. Therefore, $\mathcal{P}$ is NP-hard.

C. Semi-online Scheduling with Unknown Processing Times

Even though the communication time to transmit a task can be reasonably inferred from its data size and the transmission bandwidth, 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, $\forall j$, are not known a priori and β_j, $\forall j$, 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 $\forall P,\ C_{max}(s(P)) \leq \theta C_{max}(s^*(P))$.

IV. The Semi-Online Partitioning and Communication Algorithm

Despite the lack of a priori knowledge of the task processing times, we demonstrate that a simple algorithm based on the communication times and online observation of the task processing progress can achieve small makespans.

The following observations motivate the design of the SPaC algorithm. 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 [15]. 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-B, in the case of $\eta_{max} <= 1$, the proposed procedure is equivalent to 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 denotes the event that the processing of a task is complete on either processor and E_2 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 the ascending order of communication times. WLOG, consider $\beta_1 \leq \beta_2 \leq \ldots \leq \beta_n$.
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 Step 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.

V. SPaC Competitive Ratio Analysis

In this section, we derive a competitive ratio for SPaC when the communication time of a task is always less than its remote processing time, i.e., $\eta_{max} \leq 1$.

A. Preliminary Analysis

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. Under any schedule $\mathbf{s}$, we have

$$\frac{1}{\rho}C_0(\mathbf{s}) + [C_1(\mathbf{s}) - I(\mathbf{s})] = V.$$

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

$$C_{max}(\mathbf{s}) \geq \frac{\rho}{\rho+1}\left(V + I(\mathbf{s})\right). \tag{3}$$

Now, if the communication time plus the processing time of the task at the start of the list formed by SPaC exceeds ρV, in $\mathbf{s}^S$ all tasks will be processed on the local processor. 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 the local processor. Therefore,

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

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

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

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

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

$$C^*_{max} \geq \min\{1,\rho\}\alpha_{max}. \tag{5}$$

B. SPaC Competitive Ratio for $\eta_{max} \leq 1$

In the following we focus on how SPaC minimizes $C_1(\mathbf{s})$ given $\mathcal{T}_1(\mathbf{s})$. As stated in Section III-B, scheduling tasks on the remote processor, via the communication and processing stages, is equivalent to the two-machine flow shop problem. In Lemma 2 we observe 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.

Lemma 2. *In the two-machine flow shop problem, under the condition $\eta_{max} \leq 1$, scheduling tasks in the increasing order of their communication times minimizes 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 Lemma 2 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 the remote processor in the sequence of increasing order of their communication times. Lemma 3 below provides insight into the idle time of SPaC. Due to page limitation, its proof is omitted.

Lemma 3. *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 $I^*(\mathcal{B}_1) \leq I^*(\mathcal{B}_2)$.*

The implication of Lemma 3 is the following. When $\eta_{max} \leq 1$, given a set of tasks, the idle time of a schedule that places all those tasks on the remote processor in the increasing order of their communication times cannot be greater than that of any other schedule applied to any subset of those tasks, unless that subset does not have the task with maximum communication time. Later, in the proof of Theorem 1 we use Lemma 3 to argue that the idle time on the remote processor 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 $\frac{C_{max}(\mathbf{s}^S)}{C^*_{max}} \leq \theta$, where*

$$\theta = \min\{\theta_1, \theta_2\}$$
$$\theta_1 = 1 + \frac{\alpha_{max}}{V}$$
$$\theta_2 = 1 + \min\left\{\frac{\max\{1,\rho\}}{\rho+1}, \frac{1}{\rho}, \rho\right\}$$

Proof. The proof is given in Appendix A. □

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

1) Simple Upper Bound: A simple upper bound for θ can be obtained by solving for ρ that maximizes $\min\{\frac{\max\{1,\rho\}}{\rho+1}, \frac{1}{\rho}, \rho\}$. The solution is $\rho = \frac{\sqrt{5}+1}{2}$, and hence the upper bound is $\frac{\sqrt{5}+1}{2} \approx 1.618$.

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

3) Equally Powerful Processors: For $\rho = 1$ we have $\theta = \frac{3}{2}$. Furthermore, in Theorem 2 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.

Theorem 2. *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 given in Appendix B. □

VI. Numerical Analysis

In addition to deriving the competitive ratio of SPaC, we use simulation to compare the average performance of the proposed solution against alternatives. 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 experimental results from task offloading system MAUI [11]. We use an uplink data rate of 3 Mbps, which typical in today's wireless networks (e.g., LTE). The default data size of a task is chosen from an exponential distribution with mean 562.5 kB. Thus, $\beta_j, \forall j$, are determined by the above parameters, with a default mean value of 1500

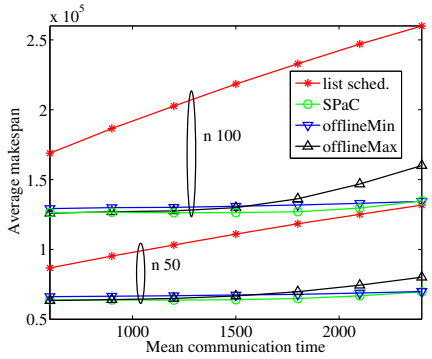

Fig. 1. Comparison of average makespan for different algorithms. $\rho = 5$.

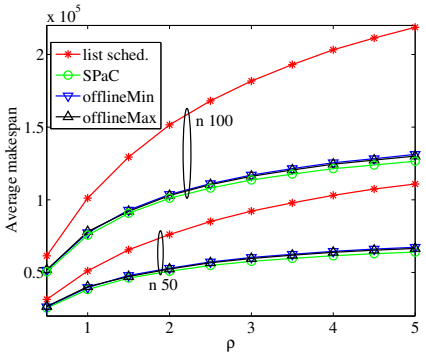

Fig. 2. Comparison of average makespan for different algorithms. Mean communication time 1500 ms.

ms. We assume that the processing times $\alpha_j, \forall j$, are exponentially distributed with mean 1500 ms. Similar results have been observed when other distributions are used, but such results are omitted to avoid redundancy. Note that in the above parameter settings we do not restrict the values of η_{max}, i.e., we simulate for general η_{max}.

Since we are not aware of any online algorithm that accounts for communication delay, we first compare the average makespan performance of SPaC with the online list scheduling algorithm [2]. We simply ignore communication delay in applying the list scheduling decisions to schedule tasks. To further demonstrate the strength of SPaC, we also compare it with offline scheduling that accounts for communication delay. Since the offline problem is NP-hard, we resort to two heuristics reported in [6]. For convenience of exposition, we name the two heuristics offlineMin and offlineMax. OfflineMin selects the next task such that, when that task is assigned to its best processor (i.e., one that completes the task in the shortest time), it results in the minimum completion time among all tasks when they are assigned to their corresponding best processors next. In contrast, offlineMax selects the next task based on the maximum completion time. We note that these heuristics require a priori knowledge of the task processing times and have computational complexity $O(n^2)$.

For each parameter setting, we generate 5000 problem instances to evaluate the average makespan for each algorithm. Figs. 1 and 2 compare the algorithms, with varying mean communication time and ρ, respectively. We observe that SPaC performs significantly better than list scheduling. Therefore, we conclude that scheduling tasks while taking communication times into consideration will have considerable benefit on reducing the makespan. Furthermore, we observe that the performance of SPaC is comparable to or sometimes even better than the offline heuristics.

VII. 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 proving $O(1)$ competitive ratio for $\eta_{max} \leq 1$. For general η_{max}, simulation results suggest that SPaC outperforms online list scheduling and provides average makespans that can be as small as those obtained from known offline heuristics.

VIII. Appendix

A. Proof of Theorem 1

Suppose 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 Lemma 3 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

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

The last inequality above is due to (3).

We substitute (4) into (6) to obtain $\frac{C_{max}(\mathbf{s}^S)}{C^*_{max}} \leq \theta_1$ 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 $C_{max}(\mathbf{s}^S) \leq V + I(\mathbf{s}^S)$. 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 Lemma 3 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

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

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_1 and θ_2. 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,

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

We substitute (4) into (8) to obtain $\frac{C_{max}(\mathbf{s}^S)}{C^*_{max}} \leq \theta_1$ 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 $C_{max}(\mathbf{s}^S) \leq C_{max}(s')$. Therefore, $C_{max}(\mathbf{s}^S) \leq \sum_{j=1}^{k} \alpha_j + I(\mathbf{s}') \leq V + I(\mathbf{s}^*)$. We use (3) 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.

B. Proof of Theorem 2

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, $\beta_j = \beta \leq 1, \forall j$. 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.

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, 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] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included," *Navel Research Logistics Quaterly*, vol. 1, no. 1, pp. 61–68, 1954.