Maximizing System's Total Accrued Utility Value for Parallel and Time-Sensitive Applications

Shuhui Li, Miao Song, Peng-jun Wan, Shangping Ren
Department of Computer Science
Illinois Institute of Technology
Email:{sli38,msong8}@hawk.iit.edu, {wan, ren}@iit.edu

***Abstract*—For a time-sensitive application, the usefulness or the quality of the application's end result depends on the time when the result is delivered, or when the application is completed. A Time Utility Function (TUF) is often used to represent the dependency between an application's accrued value and its completion time. For parallel and time-sensitive applications, each application has multiple tasks that must be executed *concurrently* in order to produce a result. Therefore, their execution occupies resources in two dimensions: spatial, i.e., the number of processing units needed to support concurrent tasks, and temporal, i.e., time duration needed to complete the application. Because of the parallelism and time-sensitive features of the applications, the execution interference among parallel and time-sensitive applications can be both in spatial and temporal domains. In this paper, we first introduce a metric to measure the spatial-temporal interference on applications' accrued values. Second, based on the metric, we develop a scheduling algorithm, i.e., the Discounting Spatial-Temporal Interference (DSTI) scheduling algorithm, to maximize system's total accrued utility value for a given set of parallel and time-sensitive applications. Our simulation results show that the proposed DSTI algorithm results in close to optimal solutions and also has clear advantage over existing approaches in the literature in terms of system total accrued utility values and profitable application ratio. It accrues up to 164%, 150%, and 97% more system value, and up to 21%, 35%, and 18% higher profitable application ratio than the Gang EDF, the FCFS with backfilling, and the 0-1 Knapsack based scheduling algorithms, respectively.**

I. Introduction

Many parallel applications are time-sensitive. Examples of these applications include threat detection applications in air defense systems [1], radar tracking applications [2], [3], and weather forecasting applications [4], to name a few. These applications not only involve multiple concurrent tasks, the usefulness or the quality of their end results also depends on the time when the results are delivered, or when the applications are completed [5]. Take the threat detection application as an example, clearly, the earlier the threat is detected, the higher value the application provides [1].

For time-sensitive applications, a Time Utility Function (TUF) [6], [7] is often used to represent the dependency between an application's accrued value and its completion time. Different applications may have different time utility functions to indicate their different time sensitivity. For example, a video surveillance application may be more sensitive to its completion time than a weather forecasting application. In this case, the TUF for the video surveillance application will have a higher value than the TUF of the weather forecasting application.

Another aspect of a parallel and time-sensitive application is that every concurrent task of the application exclusively occupies a processing unit [8]. Hence, the execution of a parallel and time-sensitive application occupies system resources in two dimensions: spatial, i.e., the number of processing units needed (which is the same as the number of concurrent tasks), and temporal, i.e., the time interval needed to complete the application's execution. Under limited resources, the competition among applications in either dimension may delay some applications' completion time and result in utility value decreases. In order to maximize the system's total accrued utility value for a given set of applications, scheduling decisions have to be made about applications' execution orders.

Scheduling problem on a single processor and multiple processors for a set of sequential applications has been studied for many years in real-time community [9]–[12]. However, as summarized in [12], in real-time scheduling community, each application is abstracted as a single task and task is the smallest scheduling unit, i.e., there is no parallelism within an application. As a result, scheduling decisions only resolve temporal conflict among applications. However, for parallel and time-sensitive applications, the execution of one application can have both spatial and temporal influence on the remaining applications. As different applications may have different number of concurrent tasks and have different execution time, their execution influence on other applications in the spatial and temporal domain may be different. Furthermore, as different application's sensitivity to their completion times, i.e., their TUF functions, may be different, the application execution order could significantly impact the system's total accrued utility value.

Many researchers have looked into the problem of scheduling parallel applications, i.e., simultaneous use of multiple processors for an individual application. For instance, the First Come First Serve with backfilling (bFCFS) scheduling algorithm [13] is a commonly used approach for scheduling parallel applications on multiprocessors when applications are not time-sensitive and scheduling fairness and system utilization are the only concerns.

Kato et al. [14] have introduced the Gang EDF scheduling algorithm to schedule parallel applications with deadline constraints. The Gang EDF scheduling algorithm applies the EDF policy to Gang scheduling to explore the real-time deadline guarantee of parallel application systems. Lakshmanan et al. [15] and Saifullah et al. [3] have studied the problem of scheduling parallel applications on multiprocessors. These studies have all focused on the schedulability analysis of a given set of applications on a given set of resources, rather than optimizing system accrued value under given resources.

Kwon et al. extended the EDF scheduling policy to maximize the utility value of parallel applications with given deadlines [16]. However, their work is based on the assumption that all applications have the same TUFs, and all applications have the same release time. These assumptions may be too restrictive for a real world system.

Although for given system resources and applications' utility values, the solution of the 0-1 Knapsack problem [17] can be applied to obtain the maximum system total accrued utility value at a time when a scheduling decision has to be made. However, the application value used in the 0-1 Knapsack problem is a constant and does not reflect possible change at later time. Hence, the scheduling decision made at its scheduling time points may not be the best choice for maximizing system's total accrued utility value.

In this paper, we focus on scheduling parallel and time-sensitive applications. Our goal is to maximize the system's total accrued utility value for a given application set. To achieve the goal, we first introduce a metric to measure the spatial-temporal interference among applications with respect to accrued values. Second, based on the metric, we develop a scheduling algorithm, i.e., the Discounting Spatial-Temporal Interference (DSTI) scheduling algorithm, to maximize system's total accrued utility value.

The rest of the paper is organized as follows. In Section II, we define the parallel and time-sensitive application model and introduce terms used in the paper. Based on the model, we formulate the system accrued utility value maximization problem. The calculation of spatial-temporal interference among parallel and time-sensitive applications is given in Section III. Section IV presents the Discounting Spatial-Temporal Interference (DSTI) scheduling algorithm. Experimental studies and result comparisons are discussed in Section V. We conclude the paper in Section VI.

II. Problem Formulation

In this section, we first introduce the models and assumptions used in the paper. We then formulate the system total accrued utility value maximization problem the paper to address.

Resource Model ($\mathcal{R}$): in the system, there is a set of M homogeneous and independent processing units, i.e., $\mathcal{R} = \{R_1, R_2, \cdots, R_M\}$. At any time, each processing unit can only process one task. The execution of tasks is non-preemptive. We also assume that the system operates in discrete time domain [18], [19].

Parallel and Time-Sensitive Application ($\mathcal{A}$): a parallel and time-sensitive application $\mathcal{A}$ is defined by a quadruple, i.e., $\mathcal{A} = (r, e, m, \mathcal{G}(t))$, where r and e are the application's release time and execution time, respectively; m is the total number of tasks that must be executed concurrently on different processing units; $\mathcal{G}(t)$ is the application completion time utility function. It is a *non-increasing* function. The time utility function represents the accrued value when the application finishes. As an application cannot finish before $r + e$, i.e., $t \geq r + e$, hence, $\mathcal{G}(t) \leq \mathcal{G}(r + e)$.

Though any non-increasing function can serve as time-sensitive application's completion time utility function, for simplification of discussion and illustration purposes, we assume that $\mathcal{G}(t)$ is a linear function. However, it is worth pointing out that the work presented in the paper is not based on this assumption, rather, it is only based on the requirement that $\mathcal{G}(t)$ is non-increasing.

In particular, we assume that

$$\mathcal{G}(t) = \begin{cases} -a(t - t_0) & r + e \leq t \leq t_0 \\ 0 & t > t_0 \end{cases} \tag{1}$$

As $\mathcal{G}(t)$ is non-increasing, it intersects with the x-axis. The first time point d where $d = \mathcal{G}^{-1}(0)$ is called *non-profit-bearing time point*. For the completion time utility function defined in (1) $\mathcal{G}(t_0) = 0$, the non-profit-bearing time point is $d = t_0$. Fig. 1 depicts $\mathcal{G}(t)$ defined in (1).

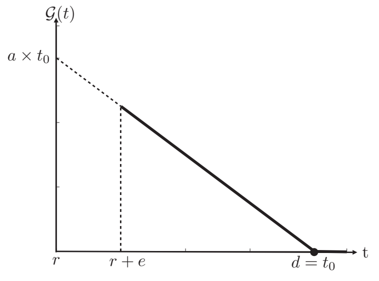


Fig. 1: Application completion time utility function

Depending on the number of concurrent tasks an application contains, parallel and time-sensitive applications can be categorized into two categories [16], i.e., *wide* applications when $m > M/2$ and *narrow* applications when $m \leq M/2$. It is worth highlighting that if a parallel application starts its execution at time s, it means that all its parallel tasks start at time s on m different processing units.

As each wide parallel and time-sensitive application requires more than half of the system processing units, hence, no more than one application can be executed simultaneously by the system. Therefore, the system total accrued utility value maximization problem for *wide* parallel and time-sensitive applications degenerates to a uniprocessor system utility maximization problem [16]. Existing scheduling algorithms, such as the Generic Utility Scheduling (GUS) algorithm introduced by Li et al. [20], the Profit and Penalty Aware (PP-aware) scheduling algorithm [21] and the Prediction-based Highest Gain Density First (PHGDF) scheduling algorithm [22] proposed by Li et al., can be applied to solve the problem. Therefore, the focus of the paper is on scheduling *narrow* parallel and time-sensitive applications. The problem to be addressed is defined below:

Problem 1: Given a set of M homogeneous, independent, and non-preemptive processing units $\mathcal{R} = \{R_1, R_2, \cdots, R_M\}$ and a set of parallel and time-sensitive applications $\Gamma = \{\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_N\}$ where $\forall \mathcal{A}_i \in \Gamma$, $\mathcal{A}_i = (r_i, e_i, m_i, \mathcal{G}_i(t))$, and $m_i \leq M/2$, develop a scheduling algorithm to decide $(\mathcal{A}_i, s_i)$, i.e., the start time (s_i) for each application $\mathcal{A}_i \in \Gamma$, such that

$$\max \sum_{\mathcal{A}_i \in \Gamma} \mathcal{G}_i(s_i + e_i) \tag{2}$$

subject to

$$\forall \mathcal{A}_i \in \Gamma,\ m_i + \sum_{\forall \mathcal{A}_k \in \Gamma \wedge s_k \leq s_i < s_k + e_k} m_k \leq M \tag{3}$$

□

We present a solution to the problem in the following sections.

III. Spatial-Temporal Interference among Parallel and time-sensitive Applications

Before we formally define application execution interference and calculate the interference impact on system's total accrued utility value, we use an example to explain the intuition behind these two concepts.

A. Example

Example 1: Assume a system has $M = 6$ homogeneous, independent, and non-preemptive processing units and three independent narrow parallel and time-sensitive applications, i.e., $\Gamma = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}$ where

- $\mathcal{A}_1 = (0, 3, 2, -7(t-5))$
- $\mathcal{A}_2 = (1, 1, 2, -6(t-5))$
- $\mathcal{A}_3 = (1, 3, 3, -5(t-6))$

For the given three applications, their non-profit-bearing time points are $d_1 = 5, d_2 = 5$, and $d_3 = 6$, respectively.

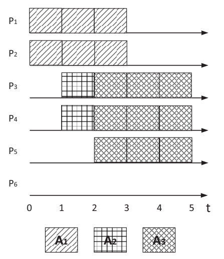


Fig. 2: $(\mathcal{A}_1, 0), (\mathcal{A}_2, 1), (\mathcal{A}_3, 2)$

At time $t = 0$, application $\mathcal{A}_1$ starts its execution. As $m_1 = 2$, only four processing units are available in the system for other applications after time 0.

At time $t = 1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ are released and they need two and three processing units, respectively. If we schedule $(\mathcal{A}_2, 1)$ (as show in Fig. 2), $\mathcal{A}_3$'s start time is delayed and it can only be scheduled at time 2. As a result its completion time increases and its utility value contributed to the system decreases.

Clearly, the execution of $\mathcal{A}_1$ at time 0 may interfere with the execution of $\mathcal{A}_2$ and $\mathcal{A}_3$ at time 1; and the execution of either $\mathcal{A}_2$ or $\mathcal{A}_3$ may interfere with the $\mathcal{A}_3$ or $\mathcal{A}_2$. The probability of whether such interference will happen depends on the available resource in the system, the resource consumed by applications that have started, and the resource needed by applications to be executed.

At time 1, as $\mathcal{A}_3$ needs more resources than $\mathcal{A}_2$, it is more likely that $\mathcal{A}_1$ will interfere with $\mathcal{A}_3$'s execution than interfere with $\mathcal{A}_2$. We can calculate the probability based on available, consumed, and needed resources. For instance, at time 1, as $\mathcal{A}_2$ needs 2 processing units, the probability that it is interfered by $(\mathcal{A}_1, 0)$ is $\frac{2}{6-2} = \frac{1}{2}$. Similarly, the probability of $\mathcal{A}_3$ being interfered by $(\mathcal{A}_1, 0)$ is $\frac{3}{6-2} = \frac{3}{4}$. However, the probability only measures the possibility of spatial interference among parallel applications.

The duration of possible interference is another concern. For instance, if we schedule $\mathcal{A}_2$ at time 1 and $\mathcal{A}_3$ at time 2, application $\mathcal{A}_2$ will not interfere with the execution of $\mathcal{A}_3$ as by the time application $\mathcal{A}_3$ is to start, the $\mathcal{A}_2$ has already finished. But if we schedule $\mathcal{A}_3$ at time 1, we cannot schedule $\mathcal{A}_2$ on $\mathcal{A}_3$'s processing units until time 4, i.e., the interference duration is 3 time units.

When considering application execution interference, as different applications have different completion time utility functions, we have to take into account not only the possibility of potential interference and the duration of the interference, but also the severity of the interference with respect to system total accrued utility values for all applications.

Assume the given three applications are scheduled as $(\mathcal{A}_1, 0), (\mathcal{A}_2, 1), (\mathcal{A}_3, 2)$. As $(\mathcal{A}_3, 2)$ is the last one to start, it does not interfere with any other applications. Therefore, its utility value is its original completion time utility value, i.e., $\mathcal{G}_3(2+3) = 5$.

However, for $(\mathcal{A}_2, 1)$, as it starts before $(\mathcal{A}_3, 2)$, it may interfere with $\mathcal{A}_3$. But, as mentioned before, since $\mathcal{A}_2$'s execution duration is $[1, 2]$, it does not interfere with $(\mathcal{A}_3, 2)$'s utility value contributed to the system. Hence, $(\mathcal{A}_2, 1)$'s utility value for the system is $\mathcal{G}_2(1+1) = 18$.

For $(\mathcal{A}_1, 0)$, as its execution interval is $[0, 3]$, it interferes with both $(\mathcal{A}_2, 1)$ and $(\mathcal{A}_3, 2)$'s utility value to the system. Hence, its accrued value has to be adjusted to reflect other applications' utility reductions caused by its interference:

$$\begin{aligned}
&\mathcal{G}_1(0+3) - \frac{1}{2} \times \mathcal{G}_2(1+1) - \frac{3}{4} \times \mathcal{G}_3(2+3) \\
&= 14 - \frac{1}{2} \times 18 - \frac{3}{4} \times 5 \\
&= 1.25
\end{aligned}$$

We observe that even taking into consideration of potential $\mathcal{A}_2$'s and $\mathcal{A}_3$'s utility reduction, $(\mathcal{A}_1, 0)$ can still bring utility value of 1.25 to the system. Therefore, $(\mathcal{A}_1, 0), (\mathcal{A}_2, 1), (\mathcal{A}_3, 2)$ is a schedule with three profitable applications.

□

B. Calculating Spatial-Temporal Interference

From Example 1, we have observed that the interference of one application on the other depends on both interfering and interfered applications processor demand and time demand. We introduce the *interference factor* $\mathcal{C}\big((\mathcal{A}_i, s_j), (\mathcal{A}_k, s_l)\big)$ metric to measure the potential that $(\mathcal{A}_i, s_j)$ has interference on $(\mathcal{A}_k, s_l)$. We consider two cases, i.e., interference between two different applications and interference within the same application but with various start times.

Case 1: interference between two different applications, i.e., $(\mathcal{A}_i, s_j)$ on $(\mathcal{A}_k, s_l)$, where $i \neq k$. In this case, the interference only exists within the execution interval of interfering application, i.e., $\mathcal{A}_i$.

$$\mathcal{C}\big((\mathcal{A}_i, s_j), (\mathcal{A}_k, s_l)\big) = \frac{m_k}{M - m_i} \tag{4}$$

where $i \neq k \ \wedge \ s_j \leq s_l < s_j + e_i$.

Case 2: interference of $(\mathcal{A}_i, s_j)$ on $(\mathcal{A}_i, s_l)$, where $s_j \leq s_l$.

$$\mathcal{C}\big((\mathcal{A}_i, s_j), (\mathcal{A}_i, s_l)\big) = 1 \tag{5}$$

where $s_j \leq s_l$.

Combining these two cases, we have the following definition:

Interference Factor: given $(\mathcal{A}_i, s_j)$ and $(\mathcal{A}_k, s_l)$, the potential interference factor of $(\mathcal{A}_i, s_j)$ on $(\mathcal{A}_k, s_l)$, i.e., $\mathcal{C}\big((\mathcal{A}_i, s_j), (\mathcal{A}_k, s_l)\big)$ is:

$$\mathcal{C}\big((\mathcal{A}_i, s_j), (\mathcal{A}_k, s_l)\big) = \begin{cases} \frac{m_k}{M - m_i} & i \neq k \ \wedge \ s_j \leq s_l < s_j + e_i \\ 1 & i = k \ \wedge s_j \leq s_l \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

It is worth pointing out that $\mathcal{C}\big((\mathcal{A}_i, s_j), (\mathcal{A}_k, s_l)\big)$ may not necessarily be the same as $\mathcal{C}\big((\mathcal{A}_k, s_l), (\mathcal{A}_i, s_j)\big)$.

C. Adjusting Application Accrued Utility Value by Discounting Potential Interference

The interference factor indicates the potential that the execution of an application $\mathcal{A}_i$ may postpone the execution of another application $\mathcal{A}_j$, and hence causes application $\mathcal{A}_j$'s accrued utility value to decrease. Therefore when we calculate application $\mathcal{A}_i$'s accrued value, we have to take into consideration of potential decreases of other applications' accrued values caused by the execution of application $\mathcal{A}_i$.

An application which starts within time interval $[r, d - e]$ will bring utility value to the system as it will complete before its *non-profit-bearing* point. Therefore, each application $(\mathcal{A})$ may have several possible profitable start time candidates $(\mathcal{A}, s)$, where $r \leq s \leq d - e$. For a given application set $\Gamma = \{\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_N\}$, we can order their possible profitable start time candidates $(\mathcal{A}, s)$ based on the descending order of s, and obtain an ordered set $S = \{(\mathcal{A}'_1, s'_1), \cdots (\mathcal{A}'_i, s'_i), (\mathcal{A}'_j, s'_j), \cdots, (\mathcal{A}'_k, s'_k) \cdots, (\mathcal{A}'_L, s'_L)\}$, where $s'_i \geq s'_j$ if $i < j$. If $\mathcal{A}'_i = \mathcal{A}'_k$, then $(\mathcal{A}'_i, s'_i)$ and $(\mathcal{A}'_k, s'_k)$ belong to same application but with different start times.

Let f be a mapping between $\mathcal{A}'$ in the ordered set S and the index in the application set Γ. Let E_{n-1} denote an ordered subset of S which may be interfered by $(\mathcal{A}'_n, s'_n)$. We can adjust the application's accrued utility value recursively as following for $1 < n \leq L$:

$$\begin{aligned} \bar{\mathcal{G}}_{f(\mathcal{A}'_1)}(s'_1) =& \mathcal{G}_{f(\mathcal{A}'_1)}(s'_1 + e_{f(\mathcal{A}'_1)}) \\ \bar{\mathcal{G}}_{f(\mathcal{A}'_n)}(s'_n) =& \mathcal{G}_{f(\mathcal{A}'_n)}(s'_n + e_{f(\mathcal{A}'_n)}) \\ & - \sum_{\forall (\mathcal{A}'_t, s'_t) \in E_{n-1}} \mathcal{C}\big((\mathcal{A}'_n, s'_n), (\mathcal{A}'_t, s'_t)\big) \cdot \bar{\mathcal{G}}_{f(\mathcal{A}'_t)}(s'_t) \end{aligned} \tag{7}$$

and

$$\begin{aligned} & E_0 = \emptyset \\ & E_{n-1} = \begin{cases} \{(\mathcal{A}'_{n-1}, s'_{n-1})\} \cup E_{n-2} & \text{if } \bar{\mathcal{G}}_{f(\mathcal{A}'_{n-1})}(s'_{n-1}) > 0 \\ E_{n-2} & \text{otherwise} \end{cases} \end{aligned} \tag{8}$$

After iterating the ordered set S, the ordered set E contains only the applications with start time points at which executing the applications will contribute utility value to the system.

We use an example to explain the process for adjusting application's accrued utility value.

Example 2: Consider the three applications as given in Example 1.

The possible start time interval for $\mathcal{A}_1$ is $[0, 2]$, $\mathcal{A}_2$ is $[1, 4]$ and $\mathcal{A}_3$ is $[1, 3]$. For each possible start time s, we calculate adjusted utility, i.e., $\bar{\mathcal{G}}(s)$. Only if $\bar{\mathcal{G}}(s) > 0$, we consider $(\mathcal{A}, s)$ as a profitable candidate.

In order to calculate $\bar{\mathcal{G}}(s)$ for application $\mathcal{A}$, we have to have the order set S and form E_{n-1} as defined in Eq. (8). In this example, we have possible $(3 + 4 + 3 = 10)$ different start time for the three applications, i.e., $(\mathcal{A}_1, 0)$, $(\mathcal{A}_1, 1)$, $(\mathcal{A}_1, 2)$, $(\mathcal{A}_2, 1)$, $(\mathcal{A}_2, 2)$, $(\mathcal{A}_2, 3)$, $(\mathcal{A}_2, 4)$, $(\mathcal{A}_3, 1)$, $(\mathcal{A}_3, 2)$, and $(\mathcal{A}_3, 3)$. As S is ordered set based on application start times, we have

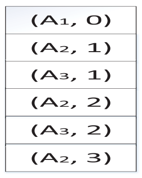
$$\begin{aligned} S = \{&(\mathcal{A}_2, 4), (\mathcal{A}_3, 3), (\mathcal{A}_2, 3), (\mathcal{A}_3, 2), (\mathcal{A}_2, 2), \\ &(\mathcal{A}_1, 2), (\mathcal{A}_3, 1), (\mathcal{A}_2, 1), (\mathcal{A}_1, 1), (\mathcal{A}_1, 0)\} \end{aligned}$$

Based on recursive processing of adjusted utility value (7) and (8), we have $\bar{\mathcal{G}}_2(4) = \mathcal{G}_2(4+1) = 0$, and $E_1 = E_0 = \emptyset$; therefore $\bar{\mathcal{G}}_3(3) = \mathcal{G}_3(3+3) = 0$ and hence $E_2 = E_1 = \emptyset$. As $\bar{\mathcal{G}}_2(3) = \mathcal{G}_2(3+1) = 6 > 0$, we have $E_3 = \{(\mathcal{A}_2, 3)\} \cup E_2 = \{(\mathcal{A}_2, 3)\}$. For $(\mathcal{A}_3, 2)$, its adjusted utility is

$$\begin{aligned} \bar{\mathcal{G}}_3(2) &= \mathcal{G}_3(2+3) - \mathcal{C}\big((\mathcal{A}_3, 2), (\mathcal{A}_2, 3)\big) \cdot \bar{\mathcal{G}}_2(3) \\ &= 5 - \frac{2}{3} \times 6 \\ &= 1 > 0 \end{aligned}$$

therefore, $E_4 = \{(\mathcal{A}_3, 2)\} \cup E_3 = \{(\mathcal{A}_3, 2), (\mathcal{A}_2, 3)\}$.

Continue the process, we have six profitable candidates as shown in Fig. 3 and $(\mathcal{A}_1, 0)$ with $\bar{\mathcal{G}}_1(0) = 6.6875$ is the earliest application. $\square$



$(\mathcal{A}_1, 0)$
$(\mathcal{A}_2, 1)$
$(\mathcal{A}_3, 1)$
$(\mathcal{A}_2, 2)$
$(\mathcal{A}_3, 2)$
$(\mathcal{A}_2, 3)$

Fig. 3: Profitable candidates

Once we have the profitable candidates, our next step is to decide the start time for each application from the profitable candidates.

IV. Discounting Spatial-Temporal Interference Scheduling Algorithm for Maximizing System Total Accrued Utility Value

From the previous section, we are able to obtain the profitable candidates for starting given applications. To fully utilize the processing units, we greedily schedule from the earliest start time. Continue Example 2, based on profitable candidates shown in Fig. 3, $(\mathcal{A}_1, 0)$ is chosen for the schedule; so is $(\mathcal{A}_2, 1)$. As system capacity is not enough for $(\mathcal{A}_3, 1)$, the candidate of $(\mathcal{A}_3, 1)$ is removed. After checking all profitable candidates in ascending start time order, we have schedule $\big((\mathcal{A}_1, 0), (\mathcal{A}_2, 1), (\mathcal{A}_3, 2)\big)$.

From the example, we can construct a schedule for a given set of parallel and time-sensitive applications based on their execution spatial-temporal interference in the following three steps:
Step 1: form the original candidate set S. For a given set of applications, the size of S is known a prior.
Step 2: decide profitable candidate set E. As finding E is an iterative process, hence, until the process completes, we do not know the size of the E. To accommodate the dynamic growth of E, we can use stack structure to store the elements of E.
Step 3: decide a schedule for a given application set based on the profitable candidate stack E.

Algorithm 1 gives the details of the DSTI algorithm.

Algorithm 1: DSTI SCHEDULING$(\mathcal{R}, \Gamma)$

```
1: set $S = E = \Gamma' = \emptyset$;
2: for $\mathcal{A}_i \in \Gamma$ do
3:     $S \leftarrow S \cup \{(\mathcal{A}_i, r_i), \cdots, (\mathcal{A}_i, d_i - e_i)\}$;
4: end for
5: sort S in descending order of start time $s$;
6: for $j = 1$ to $|S|$ do
7:     calculate $\bar{\mathcal{G}}_i(s_j)$
8:     if $\bar{\mathcal{G}}_i(s_j) > 0$ then
9:         push(E, $(\mathcal{A}_i, s_j)$);
10:    end if
11: end for
12: while !empty(E) do
13:    $(\mathcal{A}_i, s_j) =$ pop(E);
14:    if $(\mathcal{A}_i \notin \Gamma') \wedge (m_i + \sum_{\forall(\mathcal{A}_k, s_l) \in \Gamma' \wedge s_l \le s_j < s_l + e_k} m_k \le M)$
       then
15:        add $(\mathcal{A}_i, s_j)$ to $\Gamma'$
16:    end if
17: end while
18: return $\Gamma'$
```

where line 2 to line 5 in Algorithm 1 implements the first step, i.e., finding the original candidate set S and ordering the elements in descending order of the start times. The *for* loop from line 6 to line 11 implements the second step, i.e., finding the profitable candidate. It calculates the adjusted application accrued utility, and pushes the profitable $(\mathcal{A}_i, s_j)$ into the stack E. Line 12 to line 17 implements the third step which checks whether application $\mathcal{A}_i$ is already in the schedule and whether the system's capacity is enough for $(\mathcal{A}_i, s_j)$. If the application is not in the schedule and there is enough capacity in the system, then it is added into the schedule Γ'. Otherwise, the candidate is ignored.

The time complexity of step 2 is $O(|S|)$, where $|S| = \sum_{i=1}^{N}(d_i - e_i - r_i + 1)$, i.e., the size of the original candidate set for the schedule, and N is the number of applications to be scheduled. Line 5 takes $O(|S| \lg |S|)$ time. Hence, the complexity of the algorithm is $O(|S| \lg |S|)$.

For algorithm 1, we have the following property:

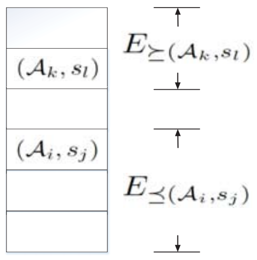
Theorem 1: The schedule Γ' generated by Algorithm 1 satisfies that the system utility value of the schedule is no less than the summation of adjusted utility value of the candidates in profitable candidate set E, i.e,

$$\mathcal{G}(\Gamma') \ge \bar{\mathcal{G}}(E)$$

where $\mathcal{G}(\Gamma') = \sum_{\forall(\mathcal{A}_i, s_j) \in \Gamma'} \mathcal{G}_i(s_j + e_i)$, and $\bar{\mathcal{G}}(E) = \sum_{\forall(\mathcal{A}_k, s_l) \in E} \bar{\mathcal{G}}_k(s_l)$.

Proof: As Γ' is selected from E, i.e., $\Gamma' \subseteq E$, therefore $\forall(\mathcal{A}_i, s_j) \in \Gamma'$, we have $(\mathcal{A}_i, s_j) \in E$.

Let $E_{\preceq(\mathcal{A}_i, s_j)}$ denote the set which is right after $(\mathcal{A}_i, s_j)$ is pushed into E, and $E_{\succeq(\mathcal{A}_k, s_l)}$ denote the set which is from top to bottom until $(\mathcal{A}_k, s_l)$ in E as shown in Fig. 4. Depending on $(\mathcal{A}_i, s_j)$ and $(\mathcal{A}_k, s_l)$, it is possible that $E_{\preceq(\mathcal{A}_i, s_j)} \cap E_{\succeq(\mathcal{A}_k, s_l)} \neq \emptyset$. If $(\mathcal{A}_m, s_n) \in E_{\preceq(\mathcal{A}_i, s_j)}$, $(\mathcal{A}_m, s_n)$ may be interfered by $(\mathcal{A}_i, s_j)$. If $(\mathcal{A}_m, s_n) \in E_{\succeq(\mathcal{A}_k, s_l)}$, $(\mathcal{A}_m, s_n)$ may interfere with $(\mathcal{A}_k, s_l)$.



$(\mathcal{A}_k, s_l)$
$E_{\succeq(\mathcal{A}_k, s_l)}$
$(\mathcal{A}_i, s_j)$
$E_{\preceq(\mathcal{A}_i, s_j)}$

Fig. 4: Profitable candidates set E

By definition of $\mathcal{G}(\Gamma')$, we have

$$\mathcal{G}(\Gamma') = \sum_{\forall(\mathcal{A}_i, s_j) \in \Gamma'} \mathcal{G}_i(s_j + e_i) \tag{9}$$

Based on Eq. (7) and Eq. (8), we expand the right-hand side of Eq. (9) and have

$$\mathcal{G}(\Gamma') = \sum_{\forall(\mathcal{A}_i, s_j) \in \Gamma'} \sum_{\forall(\mathcal{A}_k, s_l) \in E_{\preceq(\mathcal{A}_i, s_j)}} \mathcal{C}\big((\mathcal{A}_i, s_j), (\mathcal{A}_k, s_l)\big) \cdot \bar{\mathcal{G}}_k(s_l) \tag{10}$$

Eq. (10) exams every element $(\mathcal{A}_i, s_j)$ in Γ' and finds all elements in E that may be interfered by $(\mathcal{A}_i, s_j)$, i.e., $\forall(\mathcal{A}_k, s_l) \in E_{\preceq(\mathcal{A}_i,s_j)}$. To have the same result, we can also iterates through every element $(\mathcal{A}_k, s_l)$ in E and find all elements in Γ' which may interfere with $(\mathcal{A}_k, s_l)$, i.e., $\forall(\mathcal{A}_i, s_j) \in \Gamma' \cap E_{\succeq(\mathcal{A}_k,s_l)}$. Therefore, the value of $\mathcal{G}(\Gamma')$ can also be calculated as below:

$$\begin{aligned}\mathcal{G}(\Gamma') &= \sum_{\forall(\mathcal{A}_k,s_l)\in E} \sum_{\forall(\mathcal{A}_i,s_j)\in\Gamma'\cap E_{\succeq(\mathcal{A}_k,s_l)}} \mathcal{C}\big((\mathcal{A}_i,s_j),(\mathcal{A}_k,s_l)\big)\cdot\bar{\mathcal{G}}_k(s_l) \\ &= \sum_{\forall(\mathcal{A}_k,s_l)\in E} \bar{\mathcal{G}}_k(s_l) \sum_{\forall(\mathcal{A}_i,s_j)\in\Gamma'\cap E_{\succeq(\mathcal{A}_k,s_l)}} \mathcal{C}\big((\mathcal{A}_i,s_j),(\mathcal{A}_k,s_l)\big)\end{aligned} \tag{11}$$

As $\Gamma' \subseteq E$, hence, for each element $(\mathcal{A}_k, s_l)$ in E, it may or may not belong to Γ', i.e., $\forall(\mathcal{A}_k, s_l) \in E$, either $(\mathcal{A}_k, s_l) \in \Gamma'$ or $(\mathcal{A}_k, s_l) \notin \Gamma'$.

Case 1: $(\mathcal{A}_k, s_l) \in \Gamma'$. Based on the definition of interference factor Eq. (6), we have. $\mathcal{C}\big((\mathcal{A}_k, s_l), (\mathcal{A}_k, s_l)\big) = 1$. Therefore, the following inequality holds:

$$\sum_{\forall(\mathcal{A}_i,s_j)\in\Gamma'\cap E_{\succeq(\mathcal{A}_k,s_l)}} \mathcal{C}\big((\mathcal{A}_i,s_j),(\mathcal{A}_k,s_l)\big) \geq 1$$

Case 2: $(\mathcal{A}_k, s_l) \notin \Gamma'$. According to the selection of Γ', the reason of $(\mathcal{A}_k, s_l) \notin \Gamma'$ is that the available processing units is not enough for $(\mathcal{A}_k, s_l)$, i.e., the interference from the applications in the schedule is no less than 1, i.e.,

$$\sum_{\forall(\mathcal{A}_i,s_j)\in\Gamma'\cap E_{\succeq(\mathcal{A}_k,s_l)}} \mathcal{C}\big((\mathcal{A}_i,s_j),(\mathcal{A}_k,s_l)\big) \geq 1$$

Therefore, combining these two cases in Eq. (11), we have

$$\mathcal{G}(\Gamma') \geq \sum_{\forall(\mathcal{A}_k,s_l)\in E} \bar{\mathcal{G}}_k(s_l) \qquad = \bar{\mathcal{G}}(E)$$

□

It is worth pointing out that in the algorithm, when we sort S in descending order based on the start time s, the ties are broken randomly. We have experimentally studied if different tie breaker will impact the DSTI's performance. In particular, we have used three different tie-break rules, i.e., (1) break the ties arbitrarily; (2) candidate with smaller gradient of $\mathcal{G}(t)$, i.e., a, is ordered first; (3) candidate with larger gradient of $\mathcal{G}(t)$ is ordered first. Our experimental results demonstrate that the difference of the three rules is less than 1%. Due to page limited, the experiment is omitted.

V. Experimental Results

In this section, we empirically evaluate the proposed DSTI algorithm by comparing it with the optimal solution obtained with brute-force search for small application sets, and comparing it with three existing approaches in the literature for large application sets, i.e., the Gang EDF scheduling [14], the FCFS with backfilling scheduling [13], and the 0-1 Knapsack based scheduling [17] approaches. The comparisons are from two perspectives, namely, the system total accrued utility value and the profitable application ratio.

Before giving the experiment settings, we first introduce terms which will be used in the experiments:

Profitable Application Ratio (γ): the total number of applications being successfully completed with positive utility value versus the total number of applications submitted to the system.

Maximum Application Demand Density ($\delta_{\max}$): given a parallel and time-sensitive application set $\Gamma = \{\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_N\}$, where $\mathcal{A}_i = (r_i, e_i, m_i, \mathcal{G}_i(t))$, the maximum application demand density of the application set $\delta_{\max}$ is defined as

$$\delta_{\max} = \max_{\mathcal{A}_i\in\Gamma}\{\frac{e_i}{d_i - r_i}\} \tag{12}$$

Average System Load (ω): average system load ω is defined as the product of the application arrival rate λ and the maximum application demand density of the application set $\delta_{\max}$, i.e., $\omega = \lambda \times \delta_{\max}$.

A. Experiment Setting

The experiments are conducted on a simulator we have developed. In our experiments, the parallel and time-sensitive applications, i.e., $\mathcal{A} = (r, e, m, \mathcal{G}(t))$, are generated as following:

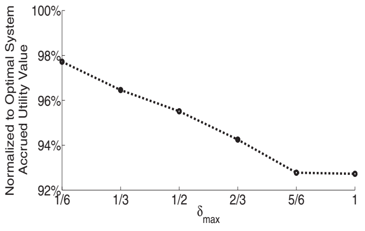
- Number of tasks m is randomly generated based on uniform distribution in the range of $[1, M/2]$ for a given M, which is set as 12 for small application sets and 40 for large application sets;
- Release time r is randomly generated based on Poisson distribution with a given λ which is a varying parameter in our evaluation;
- Execution time e is randomly generated based on uniform distribution within $[1, \delta_{\max} \times (d - r)]$ for a given $\delta_{\max}$ which is a varying parameter in our evaluation;
- Non-profit-bearing time point of $\mathcal{G}$, i.e., d, is set as $d = r + D$, where D is randomly generated based on uniform distribution within $[10, 30]$;
- The gradient of $\mathcal{G}$, i.e., a, is randomly generated based on uniform distribution in the range of $[4, 10]$;

B. Performance Comparison with the Optimal Solutions

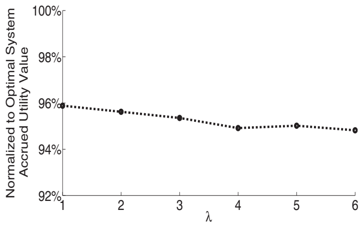
In this set of experiments, we use brute-force search to find the optimal schedule that results in the maximal system total accrued utility value and use it as a comparison base. We then apply the DSTI algorithm to the same application sets and obtain the corresponding system utility value. In these experiments, we assume that there are 12 processing units in the system, i.e., $M = 12$. We randomly generate application sets with 10 applications and repeat for 100 times. The average values are used in the evaluation.

In the first experiment, we set $\lambda = 3$ and let $\delta_{\max}$ change from $1/6$ to 1 with a step size of $1/6$. Fig. 5(a) shows the system total accrued utility value obtained by the DSTI algorithm normalized to the optimal solution.

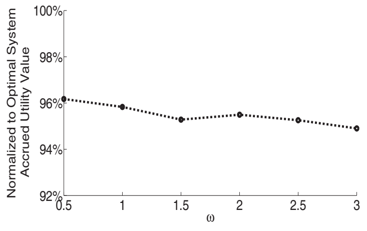
For the second experiment, we set $\delta_{\max} = 0.5$ and let λ change from 1 to 6 with a step size of 1. Fig. 5(b) shows



(a) Under different δ_{max}



(b) Under different λ



(c) Under different ω

Fig. 5: Comparison with the optimal solution for small application sets

the system total accrued utility value obtained by the DSTI algorithm normalized to the optimal solution.

For the third experiment, we increase average system load, i.e., ω, from light load ($\omega = 0.5$) to overload ($\omega = 3$) with a step size of 0.5. To do so, we randomly generate δ_{max} within $(0, 1]$ and set $\lambda = \omega/\delta_{\text{max}}$. Fig. 5(c) shows the system total accrued utility value normalized to the optimal solution.

These three experiments use different ways to vary average system load and the results clearly show that when the average system load is low, system total accrued utility value obtained by the DSTI algorithm is close to the optimal. Although the deviation between the DSTI algorithm and the optimal brute-force solution increases when average system load increases slightly, the difference is less than 7.5% in the worst case.

C. Performance Comparison with Gang-EDF, FCFS with backfilling, and 0-1 Knapsack Based Approaches

This set of experiments is to compare the DSTI with three existing scheduling approaches for parallel applications, i.e., the Gang EDF, the FCFS with backfilling, and the 0-1 Knapsack based approach. In these experiments, we set $M = 40$. We randomly generate 100 applications for an application set and repeat for 500 times. The average values are used in plotting the figures.

1) System Utility Value Comparison: We vary the value of δ_{max}, λ, and ω, and obtain system accrued utility value, the results are depicted in Fig. 6(a), Fig. 6(b), and Fig. 6(c), respectively.

From Fig. 6(a), Fig. 6(b), and Fig. 6(c), it can be seen that the DSTI algorithm always outperforms the other three algorithms. It can obtain up to 164%, 150%, and 97% more system utility value than the Gang EDF, the FCFS with backfilling, and the 0-1 Knapsack based approaches, respectively.

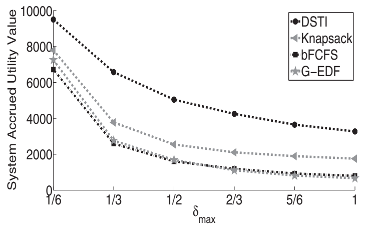
2) Profitable Application Ratio: If an application finishes before its non-profit-bearing point, it is profitable. Fig. 7(a), Fig. 7(b), and Fig. 7(c) show the profitable application ratio under different algorithms with different settings.

In Fig. 7(a), when $\delta_{\text{max}} = 1/6$, where the maximum application demand density is below 17%, and most applications can finish before their non-profit-bearing time point. In this case, the Gang EDF performs better than the DSTI algorithm. However, when δ_{max} is above 17%, the DSTI outperforms the Gang EDF scheduling algorithm. The results also show that the performances of the FCFS with backfilling algorithms always below the DSTI algorithms.

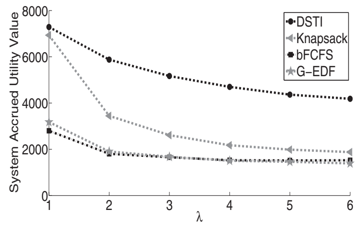
Profitable application ratio is related to system accrued utility value. The more applications that are finished before their non-profit-bearing point, the higher the profitable application ratio. Both the DSTI and the 0-1 Knapsack based algorithms try to maximize application utility value, so both algorithms have a higher application profitable ratio than the Gang EDF and the FCFS with backfilling. When the average system load is low, i.e., the resource competition among applications is low, the 0-1 Knapsack based algorithm has a higher profitable application ratio than the DSTI algorithm (about 10 percent). As average system load increases, the potential interference among applications increases, the DSTI algorithm, which takes into consideration of the interference, once again outperforms the other three scheduling approaches. It can obtain up to 21%, 35%, and 18% higher profitable application ratio than the Gang EDF, the FCFS with backfilling, and the 0-1 Knapsack based approaches, respectively.

VI. Conclusion

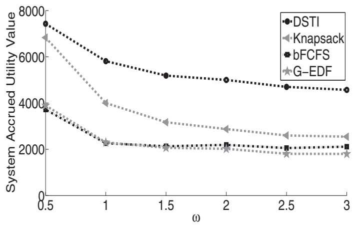
For parallel and time-sensitive applications, each application has multiple tasks that must be executed *concurrently* in order to produce a result. Therefore, their execution occupies resources in two dimensions: spatial, i.e., the number of processing units needed to support concurrent tasks, and temporal, i.e., time duration needed to complete the application. Because of the parallelism and time-sensitive features of the applications, the execution interference among parallel and time-sensitive applications can be both in spatial and temporal domains. In this paper, we have presented a scheduling approach aiming to maximize system's total accrued utility value. The scheduling algorithm, i.e., the DSTI algorithm, takes into consideration of spatial-temporal interference among parallel and time-sensitive applications, and start applications at the time when it can still bring values to the system even after their potential interference to other applications is discounted. Our simulation results show that the proposed DSTI algorithm results in close to optimal solutions and also has clear advantage over existing approaches in the literature in terms of system total accrued utility values and profitable application ratio. It accrues up to 164%, 150%, and 97% more system value, and up to 21%, 35%, and 18% higher profitable application ratio than the



(a) Under different $\delta_{\max}$

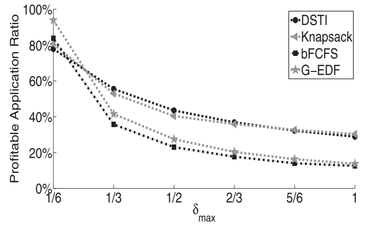


(b) Under different λ

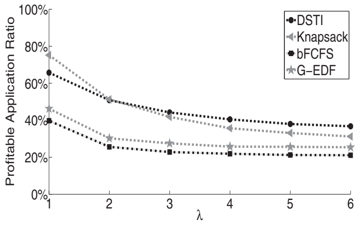


(c) Under different ω

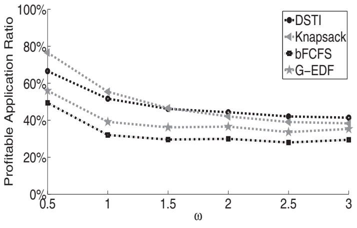
Fig. 6: System accrued utility value comparison



(a) Under different $\delta_{\max}$



(b) Under different λ



(c) Under different ω

Fig. 7: Profitable application ratio comparison

Gang EDF, the FCFS with backfilling, and the 0-1 Knapsack based scheduling algorithms, respectively.

However, our current solution is based on the assumption that all applications in the given application set are either *narrow* applications, or *wide* applications. Our future work is to study how to handle the case where there are both narrow and wide applications in the application set.

ACKNOWLEDGMENT

This work is supported in part by NSF under awards CNS-1018731, CNS-0746643(CAREER), and CNS-1035894.

REFERENCES

[1] L. R. Welch and S. Brandt, "Toward a realization of the value of benefit in real-time systems," in *Parallel and Distributed Processing Symposium., Proceedings 15th International*, April 2001, pp. 962–969.

[2] D. Ferry, J. Li, M. Mahadevan, K. Agrawal, C. Gill, and C. Lu, "A real-time scheduling service for parallel tasks," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, April 2013, pp. 261–272.

[3] A. Saifullah, J. Li, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," *Real-Time Systems*, vol. 49, no. 4, pp. 404–435, 2013.

[4] C. F. Mass and Y.-H. Kuo, "Regional real-time numerical weather prediction: Current status and future potential," *Bulletin of the American Meteorological Society*, vol. 79, no. 2, pp. 253–263, 1998.

[5] Applications performance equals response time, not resource utilization. [Online]. Available: http://www.virtualizationpractice.com/applications-performance-equals-response-time-not-resource-utilization-9916/

[6] C. D. Locke, "Best-effort decision making for real-time scheduling," Ph.D. dissertation, Carnegie-Mellon University, 1987.

[7] E. D. Jensen, C. D. Locke, and H. Tokuda, "A time-driven scheduling model for real-time operating systems," in *Real-time Systems Symposium, 1985. RTSS 1985. 6th IEEE International*, 1985, pp. 112–122.

[8] G. C. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo, *Soft real-time systems: Predictability vs. Efficiency*. Springer, 2006.

[9] M. Bertogna, M. Cirinei, and G. Lipari, "Improved schedulability analysis of edf on multiprocessor platforms," in *Real-Time Systems, 2005. (ECRTS 2005). Proceedings. 17th Euromicro Conference on*, July 2005, pp. 209–218.

[10] S. Baruah, "Techniques for multiprocessor global schedulability analysis," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, Dec 2007, pp. 119–128.

[11] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller, "Improved multiprocessor global schedulability analysis," *Real-Time Systems*, vol. 46, no. 1, pp. 3–24, 2010.

[12] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys*, vol. 43, no. 4, pp. 1–44, Oct 2011.

[13] D. G. Feitelson and A. M. Weil, "Utilization and predictability in scheduling the ibm sp2 with backfilling," in *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998*. IEEE, Mar 1998, pp. 542–546.

[14] S. Kato and Y. Ishikawa, "Gang edf scheduling of parallel task systems," in *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, Dec 2009, pp. 459–468.

[15] K. Lakshmanan, S. Kato, and R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *Real-Time Systems Symposium, 2010, RTSS 2010. 31st IEEE*, Nov 2010, pp. 259–268.

[16] K. Oh-Heum and C. Kyung-Yong, "Scheduling parallel tasks with individual deadlines," *Theoretical Computer Science*, vol. 215, no. 1, pp. 209–223, 1999.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein *et al.*, *Introduction to algorithms*. MIT press Cambridge, 2001, vol. 2.

[18] H. Kwakernaak and R. Sivan, *Linear optimal control systems*. Wiley-interscience New York, 1972, vol. 1.

[19] X. Hua, Z. Li, H. Wu, and S. Ren, "Scheduling periodic tasks on multiple periodic resources," in *Proceedings of the 4th International Conference on Advanced Communications and Computation*, 2014, pp. 35–40.

[20] P. Li, H. Wu, B. Ravindran, and E. Jensen, "A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints," *Computers, IEEE Transactions on*, vol. 55, no. 4, pp. 454–469, April 2006.

[21] S. Li, S. Ren, Y. Yu, X. Wang, L. Wang, and G. Quan, "Profit and penalty aware scheduling for real-time online services," *Industrial Informatics, IEEE Transactions on*, vol. 8, no. 1, pp. 78–89, Feb. 2012.

[22] S. Li, M. Song, Z. Li, S. Ren, and G. Quan, "Maximizing online service profit for time-dependent applications," in *Proceedings of RTCSA 2013: International Conference on Embedded and Real-Time Computing Systems and Applications*, 2013, pp. 342–345.