A 2-Approximation Algorithm for Scheduling Parallel and Time-Sensitive Applications to Maximize Total Accrued Utility Value

Shuhui Li*, Miao Song†, Peng-Jun Wan†, Shangping Ren†
* Department of Engineering Mechanics, Dalian University of Technology
† Department of Computer Science, Illinois Institute of Technology
Email:shuhuili@dlut.edu.cn, msong8@hawk.iit.edu, wan@iit.edu, ren@iit.edu

Abstract—For a time-sensitive application, the usefulness of its end results (also called the application's accrued utility value in the paper) depends on the time when the application is completed and its results are delivered. In this paper, we address the accrued utility value maximization problem for narrow parallel and time-sensitive applications. We first consider the problem in the context of a discrete time domain and present the Spatial-Temporal Interference Based (STIB) scheduling algorithm. We formally prove that the STIB algorithm is a 2-approximation algorithm. Second, we extend our work to a continuous time domain and present a heuristic scheduling algorithm, i.e., the Continuous Spatial-Temporal Interference Based (STIB-C) algorithm to maximize the system's total accrued utility value when the system operates in a continuous time domain. The extensive empirical evaluations reveal that: (1) in a discrete time domain, the systems' total accrued utility values obtained through the STIB algorithm are consistent with the theoretic bound, i.e., they never go below 50% of the optimal value. In fact, on average, the STIB algorithm can achieve over 92.5% of the optimal value; (2) compared to other scheduling policies listed in the literature, the developed STIB and STIB-C algorithms have clear advantages in terms of the system's total accrued utility value and the profitable application ratio. In particular, in terms of the system's total accrued utility value, both the STIB and the STIB-C algorithms achieve as much as 6 times for both the First Come First Come Serve(FCFS) with backfilling algorithm and the Gang Earliest Deadline First(EDF) algorithm, and 4.5 times for the 0-1 Knapsack based scheduling algorithm. In terms of the profitable application ratio, both the STIB and the STIB-C algorithms obtain as much as 4 times for both the FCFS with backfilling algorithm and the Gang EDF algorithm, and 2 times for the 0-1 Knapsack based scheduling algorithm.

Index Terms—Parallel, Time-Sensitive, Scheduling, Approximation Algorithm

1 Introduction

Many parallel applications are time-sensitive. Examples of these applications include threat detection applications in air defense systems [1] and in power system operations [2], anomaly detection application in collaborative information system [3], radar tracking applications [4], [5], target surveillance and tracking application [6] and weather forecasting applications [7], to name a few. These applications not only involve multiple concurrent tasks, but their completion times also have significant impact on the values of their end results [8], [9]. For example, for the threat detection application, the earlier a threat is detected, the more time a defender has to take action, hence the higher value the application provides [1].

For time-sensitive applications, a Time Utility Function (TUF) [10], [11] 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 the sensitivity to the completion time. 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 decreases faster with time than the TUF for 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 [12]. Hence, the execution of a parallel and time-sensitive application uses 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 once all the required processing units are available. Under limited resources, the contention in either dimension may delay the applications' completion time and result in their utility value decrease. In order to maximize the system's total accrued utility value for a given set of applications, scheduling decisions about applications' execution orders have to be judiciously made.

Scheduling problems on a single processor and multiple processors for a set of sequential applications have been studied for many years [13]–[16]. However, as summarized in [16], in the real-time scheduling community, each application is abstracted as a single task and the task is the smallest scheduling unit. As a result, scheduling decisions only need to resolve temporal conflicts among applications. However, for parallel and time-sensitive applications, the execution

The major work of this paper was done while Shuhui Li was a Ph.D student at Illinois Institute of Technology.

of one application can have both spatial and temporal interferences on the remaining applications. Furthermore, because each application's sensitivity to its completion time is different, their TUF functions may also be different. Hence, the application's execution order can significantly impact the system's total accrued utility value.

In this paper, we focus on scheduling narrow parallel and time-sensitive applications. By narrow parallel applications, we mean the applications' maximal number of parallelly executable branches is no more than one half the number of processing units provided by a system. Our goal is to maximize the system's total accrued utility value for a given set of narrow parallel and time-sensitive applications. To achieve this goal, we use a metric, which is first introduced in [17], to quantitatively measure the spatial-temporal interference among applications with respect to accrued utility values. Based on the metric, we present a 2-approximation algorithm for systems operating in a discrete time domain, i.e., the Spatial-Temporal Interference Based (STIB) scheduling algorithm, to maximize the system's total accrued utility value. The formal analysis of the STIB scheduling algorithm is added which was missing in [17]. We then extend the STIB algorithm to a continuous time domain and develop a heuristic scheduling algorithm, i.e., the Continuous Spatial-Temporal Interference Based (STIB-C) algorithm, to maximize continuous time system's total accrued utility value.

The rest of the paper is organized as follows. We discuss related work in Section 2. In Section 3, we define the parallel and time-sensitive application model and introduce terms used in the paper. Based on the model, we formulate the system total accrued utility value maximization problem. The calculation of the spatial-temporal interference among parallel and time-sensitive applications is given in Section 4. For systems operating in a discrete time domain, the Spatial-Temporal Interference Based (STIB) scheduling algorithm is presented and the approximation ratio is proved in Section 5. In Section 6, we consider a continuous time domain and introduce a heuristic scheduling algorithm, i.e., the Continuous Spatial-Temporal Interference Based (STIB-C) algorithm. Section 7 discusses empirical studies. We conclude the paper in Section 8.

2 Related Work

The TUF first introduced by Locke [10] is to describe that the utility value accrued by an application is a function of the activity's completion time. Jensen et al. have proposed to associate each task with a TUF to indicate the task's time-dependent aspect [11]. Since then, different types of TUFs have been proposed and studied. For instance, step time-utility functions are studied in [18]–[20] and non-step time-utility functions are discussed in [10], [21], [22]. Many researchers have used TUF and accrued utility value as a criteria in searching for an optimal schedule for a set of time-sensitive tasks [23], [24]. However, in the aforementioned work, applications are sequential, i.e., there is no concurrency within an application. How to maximize a system's total accrued utility value for parallel applications is still an open challenge.

Many researchers have looked into the problem of scheduling parallel applications, i.e., applications that need simultaneous use of multiple processors. The First Come First Serve (FCFS) with backfilling scheduling algorithm [25] is a commonly used approach to schedule parallel applications on multiprocessors when applications are not time-sensitive, while fairness and system utilization are the only concerns. The FCFS with backfilling scheduling is a First Come First Serve based scheduling algorithm. It advances small applications to fill in the fragments in the schedule only when such advancements do not delay the execution of the first application in the queue. The advantage of this approach is that each application is treated fairly and the application response time tends to be short and system utilization tends to be high.

Kato et al. [26] 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 (scheduling a group, or a gang, of tasks together) to explore the real-time deadline guarantee for parallel applications. Lakshmanan et al. [27] and Saifullah et al. [5] have studied the problem of scheduling parallel applications on multiprocessors. All of these studies have focused on the schedulability analysis of a given set of applications on a given set of resources based on the assumption that all of the applications' deadlines are hard deadlines and there is no incentive to complete the applications before their deadlines.

Kwon et al. later extended the EDF scheduling policy to maximize the utility value of parallel applications with given deadlines [28]. However, their work is based on the assumption that all applications have the same TUFs and are released at the same time. The developed scheduling policy fails to achieve the utility value maximization goal if applications do not satisfy these assumptions. Similarly, if applications' utility values are constants, the solution of the 0-1 Knapsack problem [29] can be applied to obtain the maximum system total accrued utility value, but the solution cannot be applied to applications whose utility values are functions of time.

It is worth noting that for task scheduling, though preemption provides advantages to high priority tasks, preemption itself can be prehabitively expensive in many practical situations [30]. It often introduces significant run-time overhead and causes high fluctuations in execution times [31]–[33]. Hence, many systems adapt a non-preemptive scheduling policy. The FCFS with backfilling scheduling, the Gang EDF scheduling and the 0-1 Knapsack based scheduling algorithms all can be used under a non-preemptive policy.

3 Problem Formulation

In this section, we first introduce the models and assumptions used in the paper. We then formulate the system's total accrued utility value maximization problem that the paper will 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 a single task. The execution of a task is non-preemptive.

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 is the application's release time, $r \in Q^+$ and Q^+ represents positive rational number;
- e is the application's execution time, $e \in Q^+$;
- m is the total number of non-preemptive tasks that must be executed concurrently on different processing units, $m \in I^+$ and I^+ represents positive integer;
- $\mathcal{G}(t)$ is the application completion time utility function.

We assume all parallel tasks within an application have the same execution time and share the same release time as the parent application. But each application itself can have a different completion time utility function and different release time.

It is worth pointing out that if a parallel and time-sensitive application starts its execution at time s, all of its m parallel tasks start at time s on m different processing units and they cannot be preempted during their execution. Furthermore, we assume all these m concurrently executed tasks release their processing units at the same time. In other words, they have the same execution time e. In this paper, we do not consider the $m > M$ situation where there are individual applications which need more processing units than what the system can provide. In addition, the time utility function $\mathcal{G}(t)$ for each application is a *non-increasing* function, representing the accrued utility value when the application completes at time t. For simplicity of discussion and illustration purposes, we let $\mathcal{G}(t)$ be a linear, non-increasing function given below:

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

where a and d are positive constants. As an application cannot finish before $r + e$ time point, it cannot earn any profit before $r + e$. For this reason, we define the domain of $\mathcal{G}(t)$ to be $[r + e, +\infty)$. However, it is worth pointing out that the work presented in this paper is not based on this assumption, rather, it is only based on the requirement that $\mathcal{G}(t)$ is non-increasing.

Non-Profit-Bearing Time Point (d): as $\mathcal{G}(t)$ is a non-increasing function, it intersects with the time-axis. The first time point t where $t = \mathcal{G}^{-1}(0)$ is called the *non-profit-bearing* time point. For the completion time utility function defined in Eq. (1), as $\mathcal{G}(d) = 0$, d is the non-profit-bearing time point. If an application completes after its non-profit-bearing point, it accrues zero value. Fig. 1 depicts $\mathcal{G}(t)$ defined in Eq. (1).

Narrow vs. **Wide Application**: if the number of parallel tasks in a given application is more than half of the total processing units in the system, i.e., $M/2 < m \le M$, the application is called a *wide* application. Otherwise, i.e., if $m \le M/2$, the application is called a *narrow* application [28].

Since each wide parallel and time-sensitive application requires more than half of the system's total processing units, no more than one wide application can be executed simultaneously by the system. Therefore, the system's total accrued utility value maximization problem for *wide* parallel and time-sensitive applications degenerates to a uniprocessor system utility maximization problem [28]. Existing scheduling algorithms, such as the Generic Utility Scheduling algorithm [22], the Profit and Penalty Aware scheduling algorithm [34], and the Prediction-based Highest Gain Density First scheduling algorithm [35], can be used to solve the problem. Therefore, we only focus on scheduling *narrow* parallel and time-sensitive applications. We define the problem the paper will address as follows:

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 $m_i \le M/2$, decide the starting time (s_i) for each application $\mathcal{A}_i \in \Gamma$, i.e., decide $(\mathcal{A}_i, s_i)$, such that

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

subject to

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

The constraint (i.e., Eq. (3)) indicates that at any time point the total number of processing units used by applications shall not exceed the total number of processing units, i.e., M. □

It should be noted that when $m_1 + m_2 + \cdots + m_N \le M$, it means that the system has sufficient processing units to execute all applications at the same time. However, when $m_1 + m_2 + \cdots + m_N > M$, it indicates there are resource contentions and not all applications can be executed at the time when they are released. Hence, decisions about the application execution order have to be made in order to maximize the system's total accrued utility value.

To solve the problem, we first study how the execution of a parallel and time-sensitive application may interfere with the execution of other applications in spatial and temporal domains. Second, based on the concept of spatial and temporal execution interferences, we develop scheduling algorithms that maximize the system total accrued utility value.

4 Spatial-Temporal Interference among Parallel and time-sensitive Applications

Before we formally define application spatial and temporal execution interference and present how we may calculate the interference impact on the system's total accrued utility value, we use an example to explain the rationales behind our strategies.

4.1 Motivating Example

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

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

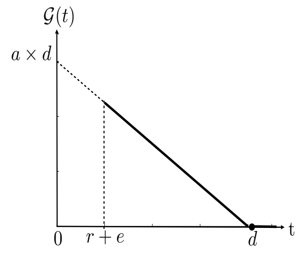


Fig. 1: Application completion time utility function

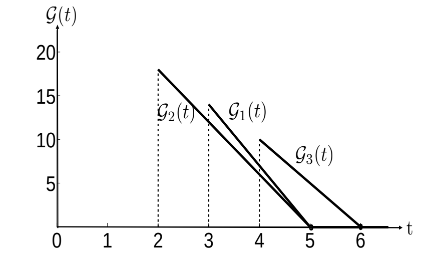


Fig. 2: Applications' completion time utility functions

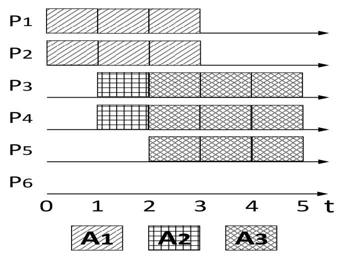


Fig. 3: $((\mathcal{A}_1, 0), (\mathcal{A}_2, 1), (\mathcal{A}_3, 2))$

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

For the given three applications, their completion time utility functions are shown in Fig. 2, and their non-profit-bearing time points are $d_1 = 5, d_2 = 5$, and $d_3 = 6$, respectively.

At time $t = 0$, since only application $\mathcal{A}_1$ is released, the system starts the execution of $\mathcal{A}_1$. Because $m_1 = 2$, four processing units remain available in the system for other applications after time 0.

At time $t = 1$, both $\mathcal{A}_2$ and $\mathcal{A}_3$ are released and they need two and three processing units, respectively. If we schedule $(\mathcal{A}_2, 1)$ (as shown in Fig. 3), $\mathcal{A}_3$'s starting time is delayed and it can only be scheduled at time 2, when application $\mathcal{A}_2$ completes. As a result, the completion time of $\mathcal{A}_3$ is delayed, causing its utility value contributed to the system to decrease.

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; the execution of either $\mathcal{A}_2$ or $\mathcal{A}_3$ may further interfere with $\mathcal{A}_3$ or $\mathcal{A}_2$, respectively. Whether such interference happens depends on (a) the total resources in the system, (b) the resources consumed by applications that have started but yet to be finished, and (c) the resources needed by applications to be executed.

At time 1, since $\mathcal{A}_3$ needs more resources than $\mathcal{A}_2$, it is more likely that $\mathcal{A}_1$ will interfere with $\mathcal{A}_3$'s execution than with $\mathcal{A}_2$'s execution. To quantify the possibility and intensity of potential interference, we can use the information about totally available, currently consumed, and currently needed resources. For instance, at time 1, since $\mathcal{A}_2$ needs two processing units and $\mathcal{A}_1$ uses two processing units, we can measure the risk of $\mathcal{A}_2$ being interfered by $(\mathcal{A}_1, 0)$ by $\frac{2}{6-2} = \frac{1}{2}$. The potential interference is related to not only the occupied resources ($\mathcal{A}_1$ uses two processing units), but also the total (six processing units) and needed resource ($\mathcal{A}_2$ needs two processing units). Similarly, the risk of $\mathcal{A}_3$ being interfered by $(\mathcal{A}_1, 0)$ can be measured as $\frac{2}{6-3} = \frac{2}{3}$. Moreover, for $\mathcal{A}_2$ and $\mathcal{A}_3$, if we schedule $\mathcal{A}_3$ before $\mathcal{A}_2$, there are less processing units left for the remaining application than if we schedule $\mathcal{A}_2$ before $\mathcal{A}_3$. Hence, it is more likely that $\mathcal{A}_3$ will interfere with $\mathcal{A}_2$'s execution than $\mathcal{A}_2$ will interfere with $\mathcal{A}_3$'s execution. We can calculate the risk of $\mathcal{A}_3$ interfering with $\mathcal{A}_2$ as $\frac{3}{6-2} = \frac{3}{4}$. Similarly, the risk of $\mathcal{A}_2$ interfering with $\mathcal{A}_3$ is $\frac{2}{6-3} = \frac{2}{3}$, which is less than $\frac{3}{4}$. However, the risk of interference 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 $\mathcal{A}_3$ since by the time application $\mathcal{A}_3$ is to start, $\mathcal{A}_2$ 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.

Furthermore, as different applications have different completion time utility functions, when considering application execution interference, we have to take into account not only the risk of 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 $(\mathcal{A}_1, 0), (\mathcal{A}_2, 1), (\mathcal{A}_3, 2)$. Since $(\mathcal{A}_3, 2)$ is the last one to start, it does not interfere with any other applications, therefore its utility value is also its completion time utility value, i.e., $\mathcal{G}_3(2+3) = 5$.

For $(\mathcal{A}_2, 1)$, although it starts before $(\mathcal{A}_3, 2)$, $\mathcal{A}_2$'s execution time interval is $[1, 2]$, which 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 $\mathcal{G}_2(1+1) = 18$.

However, for $(\mathcal{A}_1, 0)$, since its execution interval is $[0, 3]$, it may interfere with the executions of both $(\mathcal{A}_2, 1)$ and $(\mathcal{A}_3, 2)$. Hence, its accrued value has to be adjusted to reflect other applications' utility decreases caused by its interference. In particular, there is a $50\%(1/2)$ and $67\%(2/3)$ possibility that $\mathcal{A}_1$ may interfere with $\mathcal{A}_2$ and $\mathcal{A}_3$, respectively. Hence, we should deduct the possible reductions, i.e., $\frac{1}{2} \cdot \mathcal{G}_2(1+1) + \frac{2}{3} \cdot \mathcal{G}_3(2+3)$ from its initial utility, i.e., its adjusted accrued value is

$$\mathcal{G}_1(0+3) - \frac{1}{2} \cdot \mathcal{G}_2(1+1) - \frac{2}{3} \cdot \mathcal{G}_3(2+3) = 14 - \frac{1}{2} \cdot 18 - \frac{2}{3} \cdot 5 = 1.67$$

We observe that after taking into consideration of $\mathcal{A}_2$'s and $\mathcal{A}_3$'s potential utility reduction, $(\mathcal{A}_1, 0)$ can still bring a utility value of 1.67 to the system. Therefore, $(\mathcal{A}_1, 0), (\mathcal{A}_2, 1), (\mathcal{A}_3, 2)$ is a schedule in which all three applications can be profitable.

4.2 Calculating Spatial-Temporal Interference

From the motivating example, we observed that the interference of one application on other applications depends on the properties of both the interfering and interfered applications, i.e., their processing unit and execution time demands. We introduce the *potential interference factor* metric $\mathcal{C}\big((\mathcal{A}_i, s_j), (\mathcal{A}_k, s_l)\big)$ to measure the interference risk that execution of $\mathcal{A}_i$ at s_j, i.e., $(\mathcal{A}_i, s_j)$, has on $(\mathcal{A}_k, s_l)$. We consider two cases, i.e., interference between two different applications and interference of the same application at different starting 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, interference only exists during the execution interval of the interfering application, i.e., $\mathcal{A}_i$. The interference possibility between two different applications is

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

where $i \neq k \ \wedge \ s_j \leq s_l < s_j + e_i$. [1]

Case 2: interference of $(\mathcal{A}_i, s_j)$ on $(\mathcal{A}_i, s_l)$, where $s_j \leq s_l$. As the same application can only start at one time point, the interference possibility of the same application at a different starting time is 1, i.e.,

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

where $s_j \leq s_l$.

Combining these two cases, we have the following definition:

Potential 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:

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

We want to emphasize that $\mathcal{C}\big((\mathcal{A}_i, s_j), (\mathcal{A}_k, s_l)\big)$ is an auxiliary metric to quantitatively measure the spatial-temporal interference. It is not symmetric, i.e., $\mathcal{C}\big((\mathcal{A}_i, s_j), (\mathcal{A}_k, s_l)\big)$ may not necessarily be the same as $\mathcal{C}\big((\mathcal{A}_k, s_l), (\mathcal{A}_i, s_j)\big)$.

4.3 Adjusting Application Accrued Utility Value

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 potential decreases of other applications' accrued values caused by the execution of application $\mathcal{A}_i$.

An application which starts at any time instance within the interval $[r, d-e]$ will bring positive utility value to the system since it will complete before its *non-profit-bearing* point. Therefore, for a given application set $\Gamma = \{\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_N\}$, we can order the possible profitable candidates $(\mathcal{A}, s)$ based on the descending order of its starting time s, and obtain an ordered set. We use vector $\vec{S}$ to denote the ordered set and x to denote x-th element in $\vec{S}$, i.e., $\vec{S}[x] = (\mathcal{A}_k, s_i)$. We also use $\vec{S}[x].\mathcal{A}$ and $\vec{S}[x].T$ to represent the corresponding application and its starting time, respectively. If $x < y$, then $\vec{S}[x].T \geq \vec{S}[y].T$.

For instance, in the motivating example, the possible starting time candidates for application $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$ are $(\mathcal{A}_1, 0)$, $(\mathcal{A}_1, 1)$, $(\mathcal{A}_1, 2)$; $(\mathcal{A}_2, 1)$, $(\mathcal{A}_2, 2)$, $(\mathcal{A}_2, 3)$, $(\mathcal{A}_2, 4)$; and $(\mathcal{A}_3, 1)$, $(\mathcal{A}_3, 2)$, $(\mathcal{A}_3, 3)$, respectively. The ordered schedule set for the three applications is:

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

i.e.,

$$\begin{aligned}&\vec{S}[1] = (\mathcal{A}_2, 4), \text{ and } \vec{S}[1].\mathcal{A} = \mathcal{A}_2, \vec{S}[1].T = 4;\\ &\cdots\\ &\vec{S}[10] = (\mathcal{A}_1, 0), \text{ and } \vec{S}[10].\mathcal{A} = \mathcal{A}_1, \vec{S}[10].T = 0\end{aligned}$$

When the context is clear, we use s_i to denote $\vec{S}[x].T$.

We use a stack, denoted as $E^{(n-1)}$, to store the profitable candidates that still generate positive utility values after potential interference among applications are taken into consideration. In the notation of $E^{(n-1)}$, the superscript $n-1$ represents the state of the stack when the n-th element in $\vec{S}$ is being compared. In other words, if $\vec{S}[n].\mathcal{A}$'s adjusted accrued utility value is positive, the value of $\vec{S}[n]$ is pushed to the stack $E^{(n)}$, otherwise the value of $\vec{S}[n]$ is ignored and $E^{(n)} = E^{(n-1)}$.

Consider the applications given in the motivating example. We start with an empty stack $E^{(0)}$ as shown in Fig. 4(a). For $\vec{S}[1]$, i.e., $(\mathcal{A}_2, 4)$, it does not interfere with any other applications, therefore its adjusted utility value is its completion time utility value $\mathcal{G}_2(4+1) = 0$. As $\mathcal{G}_2(4+1) \not> 0$, the value of $\vec{S}[1]$, i.e., $(\mathcal{A}_2, 4)$, is ignored, hence $E^{(1)} = E^{(0)} = \emptyset$. For $\vec{S}[2]$, i.e.,$(\mathcal{A}_3, 3)$, since it does not interfere with any profitable candidates, its adjusted utility value is its completion time utility value $\mathcal{G}_3(3+3) = 0$. As $\mathcal{G}_3(3+3) \not> 0$, hence $E^{(2)} = E^{(1)} = \emptyset$.

However, for $\vec{S}[3]$, i.e., $(\mathcal{A}_2, 3)$, as $\mathcal{G}_2(3+1) = 6 > 0$, the value of $\vec{S}[3]$, i.e., $(\mathcal{A}_2, 3)$ is pushed to the stack, therefore we have $E^{(3)}[1] = \vec{S}[3] = (\mathcal{A}_2, 3)$ as shown in Fig. 4(b).

To generalize, we have the following definition:

Adjusted Accrued Utility Value ($\bar{\mathcal{G}}_{\vec{S}[n].\mathcal{A}}(s_n)$): given an application set $\Gamma = \{\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_N\}$, we obtain the ordered set $\vec{S}$ based on the descending order of the starting time s of all possible profitable candidates $(\mathcal{A}, s)$. The application $\vec{S}[n].\mathcal{A}$'s adjusted accrued utility value $\bar{\mathcal{G}}_{\vec{S}[n].\mathcal{A}}(t)$ is defined recursively as follows:

$$\begin{aligned}\bar{\mathcal{G}}_{\vec{S}[n].\mathcal{A}}(s_n) &= \mathcal{G}_{\vec{S}[n].\mathcal{A}}(s_n + e_{\vec{S}[n].\mathcal{A}})\\ &- \sum_{j=1}^{|E^{(n-1)}|} \mathcal{C}\big(\vec{S}[n], E^{(n-1)}[j]\big) \cdot \bar{\mathcal{G}}_{E^{(n-1)}[j].\mathcal{A}}(s_j)\end{aligned} \tag{7}$$

where $1 \leq n \leq L$ and $L = |\vec{S}|$, $E^{(n-1)}[j]$ is to get the j-th element in the stack $E^{(n-1)}$ from the bottom to the top, and

$$\begin{aligned}E^{(0)} &= \emptyset\\ E^{(n)} &= \begin{cases} \text{push}(E^{(n-1)}, \vec{S}[n]) & \text{if } \bar{\mathcal{G}}_{\vec{S}[n].\mathcal{A}}(s_n) > 0 \\ E^{(n-1)} & \text{otherwise} \end{cases}\end{aligned} \tag{8}$$

where push$(\vec{S}[n], E^{(n-1)})$ means $E^{(n)}[|E^{(n-1)}| + 1] = \vec{S}[n]$. Algorithm 1 illustrates the process of how the adjusted accrued utility value is to be calculated.

Continuing with the above example, for $\vec{S}[4]$, i.e.,$(\mathcal{A}_3, 2)$, its adjusted utility is

$$\bar{\mathcal{G}}_{\vec{S}[4].\mathcal{A}}(s_4) = \mathcal{G}_{\vec{S}[4].\mathcal{A}}(s_4 + e_{\vec{S}[4].\mathcal{A}}) - \mathcal{C}(\vec{S}[4], E^{(3)}[1]) \cdot \bar{\mathcal{G}}_{E^{(3)}[1].\mathcal{A}}(s_3)$$

1. It is worth pointing out that this formula is improved from the formula Eq.(4) in [17]. As shown in the motivating example, the current formula can also measure the risk between two applications $\mathcal{A}_2$ and $\mathcal{A}_3$ that are released at the same time. Application $\mathcal{A}_3$ has a higher risk of interfering with application $\mathcal{A}_2$, i.e., $\frac{3}{6-2} = \frac{3}{4}$, than the risk of $\mathcal{A}_2$ interfering with $\mathcal{A}_3$ which is $\frac{2}{6-3} = \frac{2}{3}$. This change not only reflects our intuition better, it also improves the experimental results.

Algorithm 1: CALCULATE $\bar{\mathcal{G}}_{\vec{S}[i].\mathcal{A}}(s_i)(\mathcal{R}, \vec{S})$

```
1: $\bar{\mathcal{G}}_{\vec{S}[i].\mathcal{A}}(s_i) = \mathcal{G}_{\vec{S}[i].\mathcal{A}}(s_i + e_{\vec{S}[i].\mathcal{A}})$
2: if $E^{(i-1)} \neq \emptyset$ then
3:     for $j = 1$ to $|E^{(i-1)}|$ do
4:         $\bar{\mathcal{G}}_{\vec{S}[i].\mathcal{A}}(s_i) =$
           $\bar{\mathcal{G}}_{\vec{S}[i].\mathcal{A}}(s_i) - \mathcal{C}(\vec{S}[i], E^{(i-1)}[j]) \cdot \bar{\mathcal{G}}_{E^{(i-1)}[j].\mathcal{A}}(s_j)$
5:     end for
6: end if
7: if $\bar{\mathcal{G}}_{\vec{S}[i].\mathcal{A}}(s_i) > 0$ then
8:     $E^{(i)} = \text{push}(E^{(i-1)}, \vec{S}[i])$
9: else
10:    $E^{(i)} = E^{(i-1)}$
11: end if
12: return $\bar{\mathcal{G}}_{\vec{S}[i].\mathcal{A}}(s_i)$
```

i.e.,

$$\bar{\mathcal{G}}_3(2) = \mathcal{G}_3(2+3) - \mathcal{C}\big((\mathcal{A}_3, 2), (\mathcal{A}_2, 3)\big) \cdot \bar{\mathcal{G}}_2(3) = 5 - \frac{3}{4} \times 6 > 0$$

therefore, as shown in Fig. 4(c), $E^{(4)}[2] = \vec{S}[4] = (\mathcal{A}_3, 2)$, where $E^{(4)}[2]$ indicates that the second element in the stack E from the bottom to the top.

After checking all elements in $\vec{S}$, we have six profitable candidates in $E^{(10)}$. As shown in Fig. 4(d), $(\mathcal{A}_1, 0)$, with $\bar{\mathcal{G}}_1(0) = 7.42$, is the earliest candidate. It is worth pointing out that the adjusted accrued utility value is calculated based on the order of the possible profitable candidates in set $\vec{S}$. If the order changes, the adjusted accrued utility value may change as well.

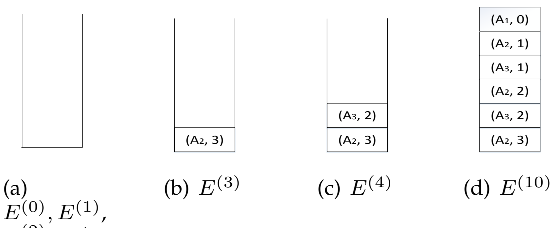


Fig. 4: Profitable candidate set

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

5 Spatial-Temporal Interference Based Scheduling Algorithm for Maximizing System Total Accrued Utility Value in Discrete Time Domain

5.1 The Spatial-Temporal Interference Based Algorithm

From the previous section, we obtain a set of profitable candidates stored in stack E. We use a greedy approach to schedule these applications based on the profitable candidates to fully utilize the processing units. Continuing the above example, as shown in Fig. 4(d), at time 0, $\mathcal{A}_1$ is the only candidate, hence we schedule $\mathcal{A}_1$ at time 0, i.e., we have $(\mathcal{A}_1, 0)$. The candidate $(\mathcal{A}_1, 0)$ is then removed from stack E. The next element on the top of the stack is $(\mathcal{A}_2, 1)$. Since there are still sufficient processing units to support application $\mathcal{A}_2$ at time 1, $\mathcal{A}_2$ is scheduled at time 1, i.e., we have $(\mathcal{A}_2, 1)$, and $(\mathcal{A}_2, 1)$ is removed from the stack. Once $\mathcal{A}_2$ starts at time 1, as both $\mathcal{A}_1$ and $\mathcal{A}_2$ are running, the system does not have sufficient processing units to start $\mathcal{A}_3$ at time 1, therefore $(\mathcal{A}_3, 1)$ is removed from stack E. Since we have decided to start $\mathcal{A}_2$ at time 1, $(\mathcal{A}_2, 2)$ is also removed from the stack due to each application only having one starting time. Then, $(\mathcal{A}_3, 2)$ is on the top of the stack and there are sufficient processing units for $\mathcal{A}_3$ to start at time 2, hence $(\mathcal{A}_3, 2)$ is added to the schedule. Once all three applications are scheduled, then the remaining elements in the stack are removed. We obtain an application execution schedule $\big((\mathcal{A}_1, 0), (\mathcal{A}_2, 1), (\mathcal{A}_3, 2)\big)$.

From the example, we can see that if the system operates in a discrete time domain, we can construct a schedule for a given set of parallel and time-sensitive applications based on their execution spatial-temporal interferences. The construction steps are:

Step 1: Form a possible profitable candidate set without considering potential interferences ($\vec{S}$). For a given set of applications, the size of $\vec{S}$ is $|\vec{S}| = \sum_{i=1}^{N}(d_i - e_i - r_i + 1)$, where d_i, e_i, and r_i are application($\mathcal{A}_i$)'s non-profit-bearing time point, execution time, and release time, respectively, and N is the number of applications to be scheduled.

Step 2: Decide a profitable candidate set after interferences are taken into consideration (E). Since finding E is an iterative process, until the process completes, we do not know the size of E. To accommodate the dynamic growth of E, we use a 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 2 gives the details of the STIB algorithm. where $\mathcal{R}$ and Γ are given system resource and application set.

Algorithm 2: STIB SCHEDULING($\mathcal{R}, \Gamma$)

```
1: set $E = \Pi = \emptyset$;
2: set $\vec{S} = \{(\mathcal{A}_i, k) | \mathcal{A}_i \in \Gamma, k \in Q^+ \wedge r_i \le k \le d_i - e_i\}$;
3: sort $\vec{S}$ in descending order of starting time $s$;
4: $\Pi$ = STIB-Exe($\mathcal{R}, \vec{S}$);
5: return $\Pi$
```

Line 2 to line 3 in Algorithm 2 implement the first step of obtaining the ordered set $\vec{S}$ with all possible profitable candidates without considering potential interference. Line 3 takes $O(|\vec{S}| \log |\vec{S}|)$ time, where $|\vec{S}| = \sum_{i=1}^{N}(d_i - e_i - r_i + 1)$, i.e., the size of possible profitable candidates set without considering potential interferences, and N is the number of applications to be scheduled. Line 4 implements the second and the third steps of the STIB algorithm, i.e., STIB-Exe($\mathcal{R}, \vec{S}$), which is defined as Algorithm 3 given below:

The *for* loop from line 1 to line 3 in Algorithm 3 implements the second step, i.e., finding profitable candidates after taking into consideration of potential interferences among applications. It calculates the adjusted application's accrued utility and pushes the profitable $(\mathcal{A}_i, s_j)$ into stack E. Line 4 through line 9 implement the third step which checks whether application $\mathcal{A}_i$ is already scheduled or whether the system's capacity is enough for $(\mathcal{A}_i, s_j)$. If the

Algorithm 3: STIB-EXE($\mathcal{R}, \vec{S}$)

```
1: for $j = 1$ to $|\vec{S}|$ do
2:     Calculate $\bar{\mathcal{G}}_i(s_j)$
3: end for
4: while !empty($E$) do
5:     $(\mathcal{A}_i, s_j) = \text{pop}(E)$;
6:     if $(\mathcal{A}_i \notin \Pi) \wedge (m_i + \sum_{\forall(\mathcal{A}_k, s_l) \in \Pi \wedge s_l \le s_j < s_l + e_k} m_k \le M)$ then
7:         add $(\mathcal{A}_i, s_j)$ to $\Pi$
8:     end if
9: end while
10: return $\Pi$
```

application is not scheduled and there is enough capacity in the system, then it is added into schedule Π. Otherwise, the candidate is removed. The time complexity of STIB-Exe($\mathcal{R}, \vec{S}$) is $O(|\vec{S}|^2)$. Hence, the complexity of the STIB algorithm is $O(|\vec{S}|^2 + |\vec{S}| \log |\vec{S}|)$, i.e., $O(|\vec{S}|^2)$.

5.2 Formal Analysis of the Spatial-Temporal Interference Based Algorithm

In this subsection, we prove that the STIB algorithm is a 2-approximation algorithm.

First, we show that the application execution order Π generated by the STIB algorithm results in at least the same amount of accrued utility value as all the profitable candidates in stack E would generate when the applications' execution interferences are taken into consideration. Second, we show that the amount of adjusted utility value of all profitable candidates in stack E is at least half of the accrued utility value of the optimal solution.

Theorem 1. Given a parallel and time-sensitive application set $\Gamma = \{\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_N\}$ where $\mathcal{A}_i = (r_i, e_i, m_i, \mathcal{G}_i(t))$, assume $\Pi = \{(\mathcal{A}_i, s_j) | \mathcal{A}_i \in \Gamma\}$ is an execution schedule generated by the STIB algorithm based on the profitable candidate stack E, after taking into consideration potential interferences, then

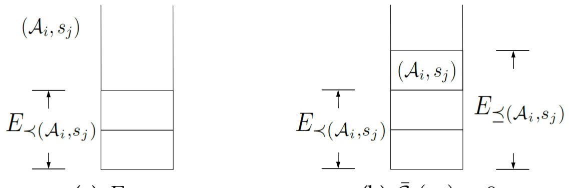
$$\mathcal{G}(\Pi) \ge \bar{\mathcal{G}}(E)$$

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

Proof: We prove the theorem in the following two steps: first, from the process of calculating the adjusted utility value, we have that the total utility value obtained by following schedule Π can be represented by a function of the adjusted utility value of candidates in stack E; second, the greedy selection of Π from stack E guarantees that the function has a larger value than the summation of the adjusted utility value of all candidates in stack E.

First, since Π is selected from E, i.e., $\Pi \subseteq E$, therefore $\forall(\mathcal{A}_i, s_j) \in \Pi$, we have $(\mathcal{A}_i, s_j) \in E$.

Assume, at the time when we need to decide if $(\mathcal{A}_i, s_j)$ shall be pushed into the stack of E, the elements that are already in the stack are denoted as $E_{\prec(\mathcal{A}_i, s_j)}$ as shown in Fig. 5(a). According to the procedure of forming E, i.e., Eq. (8), if $\bar{\mathcal{G}}_i(s_j) > 0$, then $(\mathcal{A}_i, s_j)$ is pushed into the stack as shown in Fig. 5(b). Clearly, once $(\mathcal{A}_i, s_j)$ is pushed to the top of the stack, application $\mathcal{A}_i$ can potentially execute before all other elements in the stack, i.e., $(\mathcal{A}_m, s_n) \in E_{\preceq(\mathcal{A}_i, s_j)}$, hence it may interfere with the execution of $(\mathcal{A}_m, s_n)$.


$(\mathcal{A}_i, s_j)$
$E_{\prec(\mathcal{A}_i, s_j)}$
$(\mathcal{A}_i, s_j)$
$E_{\prec(\mathcal{A}_i, s_j)}$
$E_{\preceq(\mathcal{A}_i, s_j)}$

(a) $E_{\prec(\mathcal{A}_i, s_j)}$ (b) $\bar{\mathcal{G}}_i(s_j) > 0$

Fig. 5: Forming profitable candidate set E

Assume at the end of the STIB algorithm, the state of E is shown in Fig. 6. Let $E_{\succeq(\mathcal{A}_k, s_l)}$ denote the subset which includes elements from the top to $(\mathcal{A}_k, s_l)$ in E, i.e., the elements in the subset that can potentially execute before $(\mathcal{A}_k, s_l)$.

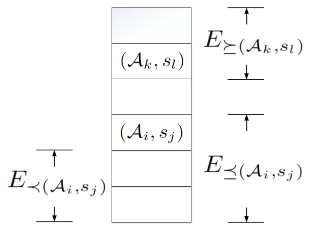

$(\mathcal{A}_k, s_l)$
$E_{\succeq(\mathcal{A}_k, s_l)}$
$(\mathcal{A}_i, s_j)$
$E_{\prec(\mathcal{A}_i, s_j)}$
$E_{\preceq(\mathcal{A}_i, s_j)}$

Fig. 6: Profitable candidates set E

By the definition of $\mathcal{G}(\Pi)$, we have

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

From Eq. (7) and Eq. (8), we have

$$\mathcal{G}_{\vec{S}[n].\mathcal{A}}(s_n + e_{\vec{S}[n].\mathcal{A}}) = \bar{\mathcal{G}}_{\vec{S}[n].\mathcal{A}}(s_n) + \sum_{j=1}^{|E^{(n-1)}|} \mathcal{C}(\vec{S}[n], E^{(n-1)}[j]) \cdot \bar{\mathcal{G}}_{E^{(n-1)}[j].\mathcal{A}}(s_j)$$

where $1 \le n \le L$ and $L = |\vec{S}|$. Combine the right two parts together, we have

$$\mathcal{G}_{\vec{S}[n].\mathcal{A}}(s_n + e_{\vec{S}[n].\mathcal{A}}) = \sum_{j=1}^{|E^{(n)}|} \mathcal{C}(\vec{S}[n], E^{(n)}[j]) \cdot \bar{\mathcal{G}}_{E^{(n)}[j].\mathcal{A}}(s_j)$$

where $1 \le n \le L$ and $L = |\vec{S}|$. Convert to the same representation, we have

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

Expand the right-hand side of Eq. (9), we have

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

Eq. (10) examines every element $(\mathcal{A}_i, s_j)$ in Π 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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2474360, IEEE Transactions on Parallel and Distributed Systems

also iterate through every element $(\mathcal{A}_k, s_l)$ in E and find all elements in Π which may interfere with $(\mathcal{A}_k, s_l)$, i.e., $\forall(\mathcal{A}_i, s_j) \in \Pi \cap E_{\succeq(\mathcal{A}_k,s_l)}$. Therefore, the value of $\mathcal{G}(\Pi)$ can also be calculated as follows:

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

As $\Pi \subseteq E$, $\forall(\mathcal{A}_k, s_l) \in E$, either $(\mathcal{A}_k, s_l) \in \Pi$ or $(\mathcal{A}_k, s_l) \notin \Pi$.

For the second step of the proof, we consider the following two cases:

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

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

Case 2: $(\mathcal{A}_k, s_l) \notin \Pi$. For this case, we discuss the following two scenarios:

Case 2.1: $(\mathcal{A}_k, s_m) \in \Pi \wedge s_m < s_l$. Based on the definition of the potential interference factor given in Eq. (6), we have $\mathcal{C}\big((\mathcal{A}_k, s_m),(\mathcal{A}_k, s_l)\big) = 1$. Therefore, the following inequality holds:

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

Case 2.2: $\nexists\ ((\mathcal{A}_k, s_m) \in \Pi \wedge s_m < s_l)$. According to the selection of Π, the reason that $(\mathcal{A}_k, s_l) \notin \Pi$ is that the available processing units are 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.,

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

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

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

□

Theorem 2. The STIB algorithm given in Algorithm 2 is a 2-approximation algorithm for Problem 1.

Proof: We take the following steps: we first prove that for each possible profitable schedule candidate $\vec{S}[i]$ $(i = 1, \cdots, L)$, where $L = |\vec{S}|$, their accrued utility value can be represented by the adjusted accrued utility value of candidates in the stack of $E^{(i-1)}$, i.e., $\forall\vec{S}[i] \in \vec{S}$,

$$\mathcal{G}_{\vec{S}[i].\mathcal{A}}(s_i + e_{\vec{S}[i].\mathcal{A}}) \leq \sum_{\forall(\mathcal{A}_k,s_l)\in E^{(i-1)}} \mathcal{C}\big(\vec{S}[i],(\mathcal{A}_k,s_l)\big)\cdot\bar{\mathcal{G}}_k(s_l)$$

Convert to another representation with the same meaning, we first want to prove

$$\mathcal{G}_i(s_j + e_i) \leq \sum_{\forall(\mathcal{A}_k,s_l)\in E_{\preceq(\mathcal{A}_i,s_j)}} \mathcal{C}\big((\mathcal{A}_i,s_j),(\mathcal{A}_k,s_l)\big)\cdot\bar{\mathcal{G}}_k(s_l) \tag{12}$$

Second, we prove that if the accrued utility value of the optimal solution O is $\mathcal{G}(O)$, then the total adjusted accrued utility value of element $(\mathcal{A}_k, s_l)$ in $E^{(L)}$ is at least $\frac{1}{2}\mathcal{G}(O)$, i.e., $\sum_{\forall(\mathcal{A}_k,s_l)\in E^{(L)}}\bar{\mathcal{G}}_k(s_j) \geq \frac{1}{2}\mathcal{G}(O)$, where $L = |\vec{S}|$. Then, by Theorem 1, the approximation ratio of Algorithm 2 is at most 2.

Let $\vec{S}$ be the set of all possible profitable candidates without considering potential interferences for the schedule, E be the profitable candidate set after potential interferences are taken into consideration, Π be the schedule generated by Algorithm 2, and O be an optimal solution.

As $E \subseteq \vec{S}$, hence, for each element $(\mathcal{A}_i, s_j)$ in $\vec{S}$, either $(\mathcal{A}_i, s_j) \in E$ or $(\mathcal{A}_i, s_j) \notin E$. For the first step of proof, we divide all elements in $\vec{S}$ into two parts: (a) $(\mathcal{A}_i, s_j) \in E$ and (b) $(\mathcal{A}_i, s_j) \notin E$.

For each $(\mathcal{A}_i, s_j) \in E$, based on Eq. (7) and Eq. (8), we have,

$$\mathcal{G}_i(s_j + e_i) = \sum_{\forall(\mathcal{A}_k,s_l)\in E_{\preceq(\mathcal{A}_i,s_j)}} \mathcal{C}\big((\mathcal{A}_i,s_j),(\mathcal{A}_k,s_l)\big)\cdot\bar{\mathcal{G}}_k(s_l) \tag{13}$$

For each $(\mathcal{A}_i, s_j) \notin E$, we have,

$$\begin{aligned}\mathcal{G}_i(s_j + e_i) &= \bar{\mathcal{G}}_i(s_j)\\ &\quad+ \sum_{\forall(\mathcal{A}_k,s_l)\in E_{\prec(\mathcal{A}_i,s_j)}} \mathcal{C}\big((\mathcal{A}_i,s_j),(\mathcal{A}_k,s_l)\big)\cdot\bar{\mathcal{G}}_k(s_l)\end{aligned} \tag{14}$$

where $E_{\prec(\mathcal{A}_i,s_j)}$ denotes the subset of E. It contains the elements in E when deciding whether $(\mathcal{A}_i, s_j)$ should be added into E in the process of adjusting accrued utility value. Based on Eq. (8), for $(\mathcal{A}_i, s_j) \notin E$, we have $\bar{\mathcal{G}}_i(s_j) \leq 0$ and $E_{\preceq(\mathcal{A}_i,s_j)} = E_{\prec(\mathcal{A}_i,s_j)}$. Therefore, for Eq. (14), we have

$$\mathcal{G}_i(s_j + e_i) \leq \sum_{\forall(\mathcal{A}_k,s_l)\in E_{\preceq(\mathcal{A}_i,s_j)}} \mathcal{C}\big((\mathcal{A}_i,s_j),(\mathcal{A}_k,s_l)\big)\cdot\bar{\mathcal{G}}_k(s_l) \tag{15}$$

From Eq. (13) and Eq. (15), we have $\forall(\mathcal{A}_i, s_j) \in \vec{S}$,

$$\mathcal{G}_i(s_j + e_i) \leq \sum_{\forall(\mathcal{A}_k,s_l)\in E_{\preceq(\mathcal{A}_i,s_j)}} \mathcal{C}\big((\mathcal{A}_i,s_j),(\mathcal{A}_k,s_l)\big)\cdot\bar{\mathcal{G}}_k(s_l)$$

Clearly, each element $(\mathcal{A}_i, s_j)$ in an optimal solution O, must be in $\vec{S}$, i.e., $\forall(\mathcal{A}_i, s_j) \in O$, $(\mathcal{A}_i, s_j) \in \vec{S}$. Hence, we have:

$$\begin{aligned}\mathcal{G}(O) &= \sum_{\forall(\mathcal{A}_i,s_j)\in O}\mathcal{G}_i(s_j + e_i)\\ &\leq \sum_{\forall(\mathcal{A}_i,s_j)\in O}\ \sum_{\forall(\mathcal{A}_k,s_l)\in E_{\preceq(\mathcal{A}_i,s_j)}} \mathcal{C}\big((\mathcal{A}_i,s_j),(\mathcal{A}_k,s_l)\big)\cdot\bar{\mathcal{G}}_k(s_l)\end{aligned} \tag{16}$$

Similar to the transformation from Eq. (10) to Eq. (11) in Theorem 1, Eq. (16) can also be transformed to:

$$\begin{aligned}\mathcal{G}(O) &\leq \sum_{\forall(\mathcal{A}_k,s_l)\in E}\ \sum_{\forall(\mathcal{A}_i,s_j)\in\ O\cap E_{\succeq(\mathcal{A}_k,s_l)}} \mathcal{C}\big((\mathcal{A}_i,s_j),(\mathcal{A}_k,s_l)\big)\cdot\bar{\mathcal{G}}_k(s_l)\\ &= \sum_{\forall(\mathcal{A}_k,s_l)\in E}\bar{\mathcal{G}}_k(s_l) \sum_{\forall(\mathcal{A}_i,s_j)\in\ O\cap E_{\succeq(\mathcal{A}_k,s_l)}} \mathcal{C}\big((\mathcal{A}_i,s_j),(\mathcal{A}_k,s_l)\big)\end{aligned} \tag{17}$$

Each element $(\mathcal{A}_k, s_l) \in E$ may or may not belong to O, i.e., $\forall(\mathcal{A}_k, s_l) \in E$, either $(\mathcal{A}_k, s_l) \in E \wedge (\mathcal{A}_k, s_l) \in O$ or $(\mathcal{A}_k, s_l) \in E \wedge (\mathcal{A}_k, s_l) \notin O$. To prove $\bar{\mathcal{G}}(E)$ has the lower bound of $\frac{1}{2}\mathcal{G}(O)$, we consider the following two cases:

Case 1: $(\mathcal{A}_k, s_l) \in E \wedge (\mathcal{A}_k, s_l) \in O$. Based on the definition of the potential interference factor from Eq. (6), we have

$\mathcal{C}\big((\mathcal{A}_k, s_l), (\mathcal{A}_k, s_l)\big) = 1$. Since $(\mathcal{A}_k, s_l) \in O$, other applications in the optimal solution, i.e., $\forall(\mathcal{A}_i, s_j) \in O \setminus \{(\mathcal{A}_k, s_l)\}$, occupy at most $M - m_k$ processing units at time s_l, i.e.,

$$\sum_{\forall(\mathcal{A}_i,s_j)\in(O\setminus\{(\mathcal{A}_k,s_l)\})\cap E_{\succeq(\mathcal{A}_k,s_l)}} \mathcal{C}\big((\mathcal{A}_i, s_j), (\mathcal{A}_k, s_l)\big) \le 1$$

otherwise O is not schedulable. Adding them together, we have,

$$\sum_{\forall(\mathcal{A}_i,s_j)\in\ O\cap E_{\succeq(\mathcal{A}_k,s_l)}} \mathcal{C}\big((\mathcal{A}_i, s_j), (\mathcal{A}_k, s_l)\big) \le 2$$

Case 2: $(\mathcal{A}_k, s_l) \in E \wedge (\mathcal{A}_k, s_l) \notin O$. Because $(\mathcal{A}_k, s_l) \notin O$, at time s_l, $\forall(\mathcal{A}_i, s_j) \in O$ occupy at most M processing units, i.e.,

$$\sum_{\forall(\mathcal{A}_i,s_j)\in\ O\cap E_{\succeq(\mathcal{A}_k,s_l)}} \mathcal{C}\big((\mathcal{A}_i, s_j), (\mathcal{A}_k, s_l)\big) \le \frac{M}{M - m_k}$$

otherwise O would not be schedulable. Since we only consider narrow applications in this paper, i.e., $m_k \le M/2$, we have

$$\sum_{\forall(\mathcal{A}_i,s_j)\in\ O\cap E_{\succeq(\mathcal{A}_k,s_l)}} \mathcal{C}\big((\mathcal{A}_i, s_j), (\mathcal{A}_k, s_l)\big) \le 2$$

Therefore, combining these two cases in Eq. (17), we have

$$\mathcal{G}(O) \le 2 \cdot \sum_{\forall(\mathcal{A}_k,s_l)\in E} \bar{\mathcal{G}}_k(s_l) = 2 \cdot \bar{\mathcal{G}}(E) \qquad (18)$$

where $\bar{\mathcal{G}}(E) = \sum_{\forall(\mathcal{A}_k,s_l)\in E} \bar{\mathcal{G}}_k(s_l)$.

Finally, by Theorem 1, we have

$$\mathcal{G}(\Pi) \ge \bar{\mathcal{G}}(E)$$

where Π is the schedule generated by Algorithm 2 and $\mathcal{G}(\Pi) = \sum_{\forall(\mathcal{A}_i,s_j)\in\Pi} \mathcal{G}_i(s_j + e_i)$.

So,

$$\mathcal{G}(O) \le 2 \cdot \bar{\mathcal{G}}(E) \le 2 \cdot \mathcal{G}(\Pi) \qquad (19)$$

Therefore, Algorithm 2 is a 2-approximation algorithm for Problem 1. $\square$

With the STIB algorithm, every possible starting point is checked, and the size of $\vec{S}$ is $\sum_{i=1}^{N}(d_i - e_i - r_i + 1)$. In the next section, we will empirically investigate whether we can reduce the number of points being checked in finding the best starting time point for each application.

5.3 Impact of Reducing Potential Starting Time Points

Before discussing the empirical investigation, we first introduce the terms which will be used in the experiments.

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 $\delta_{\max}$ of the application set is defined as

$$\delta_{\max} = \max_{\mathcal{A}_i\in\Gamma}\{\frac{e_i}{d_i - r_i}\} \qquad (20)$$

System Workload (ω): system workload ω is defined as the product of the application arrival rate λ and the maximum application demand density $\delta_{\max}$ of the application set, i.e., $\omega = \lambda \cdot \delta_{\max}$. The workload ω indicates statistical application resource demand under a worst case scenario. For instance, $\omega = 2$ indicates that the total application resource demand under the worst case is twice the resource provided by the system.

We take an empirical approach to study the impact of reducing the number of needed checks for potential starting time points. In this experiment, rather than selecting every point within an application's valid starting time interval $[r, d - e]$, we choose (a) only two end time points, i.e., r and $d - e$; and (b) only the release time r, of each application as potential starting time points. We only check these points for each application's adjusted accrued utility value and compare the system total accrued utility value with the value when all time points within the interval $[r, d - e]$ are selected as potential starting time points.

For the experiment, we assume that there are 40 processing units in the system, i.e., $M = 40$. We randomly generate application sets with 100 applications and repeat this 100 times. The average values of the results of the 100 times test are used in the evaluation.

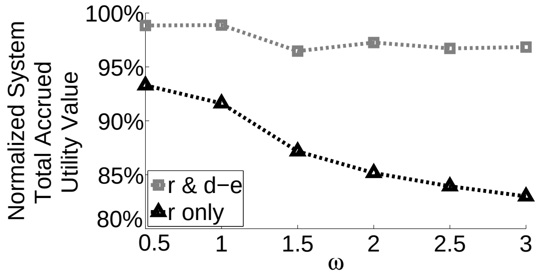


Fig. 7: Impact of reducing potential starting time point set on the performance of the STIB

Fig. 7 shows the result of the system's total accrued utility value normalized to the system's total accrued utility value obtained when all time points within $[r, d - e]$ are evaluated. Fig. 7 reveals the results of two conditions: (a) only the application's release time and non-profit-bearing starting time point are used as the application's potential starting points; (b) only the release time r is used. For the first condition, the STIB can still achieve almost the same amount of accrued utility value (97% to be specific) as when all integer points within the interval $[r, d - e]$ are used as potential starting points. However, if only the release time r is used, the accrued utility value has significant decreases, especially when the system is highly overloaded.

Further investigation reveals that although only two end time points, i.e., r and $d - e$, are selected from each application for considering their adjusted accrued utility values, when the system is overloaded, i.e., applications' execution time intervals overlap with each other, it is possible that the actual points selected for checking within an application are larger than two. We use an example to explain this scenario.

Given three applications $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$, their r and $d-e$ time points are shown in Table 1.

If we only check the two end points for each application, then time instants 0, 1, 2, 3, 4, and 7 are checked, where 1, 2, and 3 fall in application $\mathcal{A}_2$'s valid execution interval.

TABLE 1: Applications' important time points

Application	r	d-e
$\mathcal{A}_1$	0	7
$\mathcal{A}_2$	1	3
$\mathcal{A}_3$	2	4

Similarly, for application $\mathcal{A}_3$, 2, 3, and 4 are all checked. For application $\mathcal{A}_1$, if only r and $d-e$ for each application are selected as the potential starting time points, we would lose time points 5 and 6, which could be a better solution for application $\mathcal{A}_1$. Even in such case, most of the time points within the interval of $\mathcal{A}_1$, i.e., $[0,7]$, are compared. The observation provides the support that we can reduce the number of potential starting time points and still accrue a good system total accrued utility value.

However, if only application's release time is selected, we do lose a relatively large number of potential starting time points and hence, the results become inferior.

From the experiment, we observe that the larger the potential starting time point sets are selected, the higher the system's total accrued utility value obtained by the STIB algorithm. Based on the observation, we present an approach that extends the STIB strategy to systems that operate in a continuous time domain.

6 Apply Spatial-Temporal Interference Based Scheduling Strategy in Continuous Time Domain

From the previous Section 5.3, we observed that we can still achieve good performance even if we do not explore all possible starting time point candidates. This observation reveals a possible solution for maximizing the system's total accrued utility value in a continuous time domain. That is, although there are infinitely many possible starting time points for a given application within its profitable interval $[r, d-e]$ in a continuous time domain, we can judiciously select a finite subset of starting time points and apply the STIB strategy to the finite set. Based on this idea, we propose a scheduling algorithm, i.e. the Continuous Spatial-Temporal Interference Based (STIB-C) algorithm, to solve Problem 1 defined in Section 3 for a continuous time domain.

6.1 Deciding Finite Starting Time Points

The basic idea of handling infinite starting time points in a continuous time domain is to judiciously select a finite number of starting time points within the infinite point set.

For a given application, if it starts within the time interval $[r, d-e]$, it will generate a positive utility value since it will complete before its *non-profit-bearing* point. Therefore, r and $d-e$ are two important time points for an application. For a given application set $\Gamma = \{\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_N\}$, if we only use these important time points as potential starting time points, there are a total of at most $2N$ such crucial time points that the applications can start with. It is worth pointing out that if applications have dense overlapping execution intervals, it is possible that all $2N$ points fall in a single application's execution interval, as explained in the example in Section 5.3.

Clearly, we can also select more intermediate points within each application's profitable execution interval to increase the number of potential comparisons. We let n control the number of intermediate points we select. Once a set of discrete points is decided for applications operating in a continuous time domain, we can apply the STIB-C algorithm. The next subsection gives the details of the STIB-C scheduling algorithm for a continuous time domain.

6.2 The Continuous Spatial-Temporal Interference Based Algorithm

The Continuous STIB algorithm involves two steps, i.e., (a) deciding on a finite set of potential starting time points within each application's potential profitable interval $[r, d-e]$, and (b) applying the STIB scheduling strategy to the finite time point set obtained in step (a). The details of the STIB-C algorithm are given in Algorithm 4, where parameter n in the algorithm decides how many intermediate points are to be selected within each application's potential profitable interval $[r, d-e]$.

Algorithm 4: STIB-C SCHEDULING($\mathcal{R}, \Gamma, n$)

```
1:  set $E = \Pi = S' = \emptyset$;
2:  for $\mathcal{A}_i \in \Gamma$ do
3:      $S' \leftarrow S' \cup \{r_i, r_i + \frac{1}{n} \cdot (d_i - e_i - r_i), \cdots,$
        $r_i + \frac{n-1}{n} \cdot (d_i - e_i - r_i), d_i - e_i\}$;
4:  end for
5:  for $j = 1$ to $|S'|$ do
6:      $\vec{S} \leftarrow \vec{S} \cup \{(\mathcal{A}_i, s_j) | \mathcal{A}_i \in \Gamma \wedge s_j \in [r_i, d_i - e_i]\}$
7:  end for
8:  sort $\vec{S}$ in descending order of starting time $s$;
9:  $\Pi$ =STIB-Exe($\mathcal{R}, \vec{S}$);
10: return $\Pi$
```

Line 2 to line 8 of Algorithm 4 decides the finite profitable candidates without considering potential interferences for the application set Γ and sorts them in descending order. The time complexity is $O(|\vec{S}|^2)$. The STIB-Exe($\mathcal{R}, \vec{S}$) in line 9 is given in Section 5.1 and its time complexity is $O(|\vec{S}|^2)$. Hence, the time complexity of STIB-C is $O(|\vec{S}|^2)$.

It is worth pointing out that when $n = 0$, as each application's release time and its non-profit-bearing time points are selected by default, the size of potential starting time point set S' can be as large as $2N$, where N is the number of applications to be scheduled. Hence, the number of possible profitable candidates without considering potential interferences, i.e., the size of $\vec{S}$, can reach $2N^2$.

7 Empirical Evaluation

In this section, we empirically evaluate the proposed STIB and STIB-C algorithms. For a discrete time domain, we compare the STIB algorithm with (a) the optimal solutions obtained by brute-force search for small application sets; and (b) three existing approaches in the literature for large application sets, i.e., the FCFS with backfilling scheduling [25], the Gang EDF scheduling [26], and the 0-1 Knapsack based scheduling [29]. It should be noted that the technique of reducing potential starting time points is not considered for a discrete time domain. For a continuous time domain, we (a) study the impact of the number of intermediate points on the performance of the STIB-C algorithm; (b) compare the STIB-C algorithm with the FCFS with backfilling scheduling, the Gang EDF scheduling, and the 0-1 Knapsack based scheduling for large application sets. The comparison criteria are the system's total accrued utility value and profitable application ratio which is defined below:

Profitable Application Ratio (γ) is the ratio between the total number of applications being successfully completed with positive accrued utility value and the total number of applications submitted to the system, i.e.,

$$\gamma = \frac{|\Gamma_{\text{positive}}|}{|\Gamma_{\text{total}}|} \tag{21}$$

where $|\Gamma_{\text{positive}}|$ and $|\Gamma_{\text{total}}|$ are the total number of applications completed with positive accrued utility value and total number of applications submitted, respectively.

7.1 Experiment Settings

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 follows:

- Total number of processing units M is set to be 12 for small systems and 40 for large systems;
- Number of tasks m is randomly generated based on a uniform distribution in the range of $[1, M/2]$ for a given M;
- Release time r is randomly generated based on the Poisson distribution with a given λ which is a varying parameter in our evaluation;
- Execution time e is randomly generated based on a uniform distribution within $[1, \delta_{\max} \cdot (d - r)]$ for a given maximum application demand density $\delta_{\max}$ 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 a uniform distribution within $[10, 30]$;
- The gradient of $\mathcal{G}$, i.e., a, is randomly generated based on a uniform distribution in $[4, 10]$.

7.2 Performance of the Spatial-Temporal Interference Based Algorithm

7.2.1 Comparison between the STIB and 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 in a discrete time domain and use it for comparison. We then apply the STIB algorithm to the same application sets and obtain the corresponding system total accrued 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 results of the 100 times test are used in the evaluation.

We use three different ways to vary the system load. In the first set of experiments, we set application arrival rate $\lambda = 3$ and let maximum application demand density $\delta_{\max}$ change from $1/6$ to 1 with a step size of $1/6$. Fig. 8(a) shows the system total accrued utility value obtained by the STIB algorithm normalized to the optimal solution.

For the second set of experiments, we set the maximum application demand density $\delta_{\max} = 0.5$ and let application arrival rate λ change from 1 to 6 with a step size of 1. Fig. 8(b) shows the system total accrued utility value obtained by the STIB algorithm normalized to the optimal solution.

For the third set of experiment, we increase the 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 maximum application demand density $\delta_{\max}$ within $(0, 1]$ and set application arrival rate $\lambda = \omega/\delta_{\max}$. Fig. 8(c) shows the system total accrued utility value normalized to the optimal solution.

From the three experiments, we have the following observations: (a) when the system load is low, the system total accrued utility value obtained by the STIB algorithm is close to the optimal; (b) although the deviation between the STIB algorithm and the optimal brute-force solution increases when the system load increases, it always achieves above 50% of the optimal solution, which is consistent with our theoretic conclusion, i.e., the STIB algorithm is a 2-approximation algorithm; (c) although the STIB algorithm is a 2-approximation algorithm, on average, the STIB algorithm can obtain over 92.5% of the optimal value.

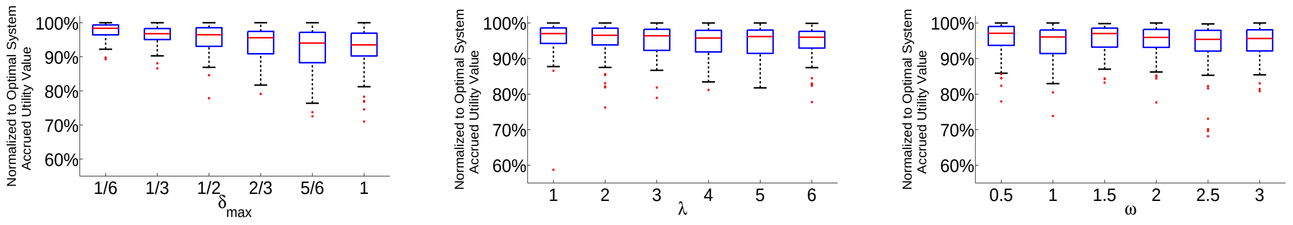
7.2.2 Performance Comparison with the FCFS with backfilling, the Gang-EDF, and the 0-1 Knapsack Based Approaches for Discrete Time Systems

In this experiment, we set $M = 40$ and randomly generate application sets with 500 applications and repeat for 100 times. Due to time complexity, for this set of experiments, we do not compare the results with the optimal solution found by through brute-force search, instead, we compare the STIB algorithm with three approaches commonly used in the literature, i.e., the FCFS with backfilling, the Gang EDF, and the 0-1 Knapsack based scheduling algorithms. The average values of the results of the 100 tests are used in plotting the figures.

System Utility Value Comparison

We vary the value of system workload ω as we did in the third experiment in Section 7.2.1, and obtain system total accrued utility values with three different approaches. The results are normalized to the system's total accrued utility value obtained by the FCFS with backfilling approach. They are depicted in Fig. 9.

As shown in Fig. 9, the STIB algorithm has a clear advantage over the other three approaches with respect to the system's total accrued utility value. More specifically, it obtains as much as over 6 times accrued utility value obtained by the FCFS with backfilling approach when the system workload is overloaded, i.e., the workload is above one. Fig. 9 also indicates that the performances of the Gang EDF and the FCFS with backfilling are similar. The system total accrued utility value obtained by the STIB algorithm is over 6 times and about 4.5 times as much as the values obtained by the Gang EDF algorithm and the 0-1 Knapsack based algorithm, respectively. When the system workload



(a) Under different $\delta_{\max}$ (b) Under different λ (c) Under different ω

Fig. 8: STIB's comparison with the optimal solution

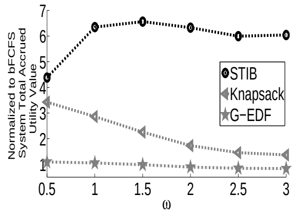


Fig. 9: System accrued utility value comparison

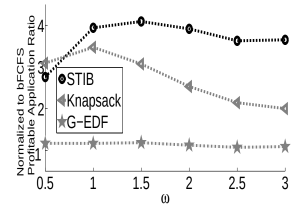


Fig. 10: Profitable application ratio comparison

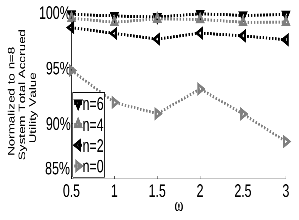


Fig. 11: Impact of smaller n on the system total accrued utility value

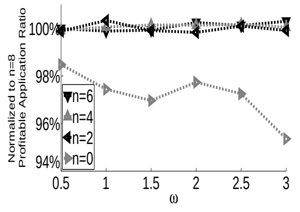


Fig. 12: Impact of smaller n on the profitable application ratio

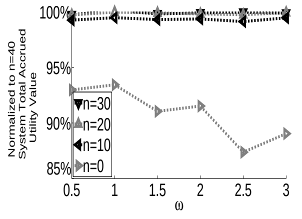


Fig. 13: Impact of larger n on the system total accrued utility value

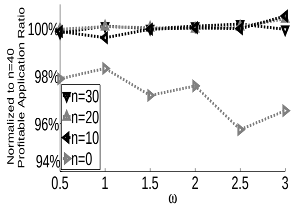


Fig. 14: Impact of larger n on the profitable application ratio

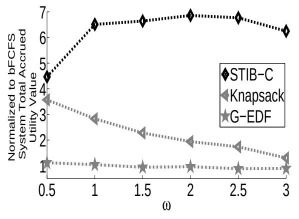


Fig. 15: System accrued utility value comparison

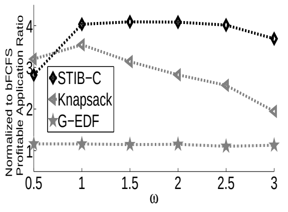


Fig. 16: Profitable application ratio comparison

is 0.5, the STIB algorithm can achieve the system's total accrued utility value 4.5 times as much for both the FCFS with backfilling and the Gang EDF algorithms, and 1.2 times as much for the 0-1 Knapsack based scheduling algorithm.

Profitable Application Ratio

Fig. 10 shows the profitable application ratio, which is normalized to the FCFS with backfilling approach, under different algorithms with different system loads ω.

The profitable application ratio is related to the system accrued utility value. The more applications that finish before their non-profit-bearing point, the higher the profitable application ratio. The STIB and 0-1 Knapsack based algorithms try to maximize the application utility value so both algorithms have higher application profitable ratios than the FCFS with backfilling and the Gang EDF scheduling algorithms.

In Fig. 10, when the 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 STIB algorithm (about 10% higher). As the system load increases, the potential interference among applications increases. The STIB algorithm, which takes into consideration possible interference, once again outperforms the other three scheduling approaches. The profitable application ratio obtained by the STIB algorithm is about four times as much as what can be achieved by the FCFS with backfilling algorithm when the system load is larger than one. Furthermore, the performance of the Gang EDF and the FCFS with backfilling with respect to profitable application ratio is similar, and the STIB algorithm achieves over four and close to two times as much as the profitable application ratios achieved by the Gang EDF and the 0-1 Knapsack based algorithms, respectively.

7.3 Performance of the Continuous Spatial-Temporal Interference Based Algorithm

7.3.1 Impact of the Number of Intermediate Time Points

In this experiment, we examine the impact of the number of intermediate points on the performance of the STIB-C algorithm. In the experiment, we set $M = 40$. We randomly generate application sets with 100 applications and repeat that 100 times. The average values of the results of the 100 times test are used in the evaluation.

Small Number of Intermediate Points

We increase the system load ω and the number of intermediate points n, and obtain the system's total accrued utility value and the profitable application ratio. Fig. 11 shows the result of the system's total accrued utility value normalized to the value obtained with $n = 8$, i.e, the largest number of intermediate points in the experiment. Fig. 12 depicts the impact of the number of intermediate points on the profitable application ratio, which is normalized to the profitable application ratio obtained with $n = 8$.

From Fig. 11, we observe that the larger the number of intermediate points (n), the higher the system total accrued utility value. The system's total accrued utility value difference between $n = 0$ and $n = 2$ is up to 9%. However, when n increases from 2 to 8, the utility value increase is only about 2.4%.

This result is consistent with the observation in Section 5.3 for discrete time domain, i.e., the larger the potential starting time point set, the higher the system total accrued utility value. However the difference among them is limited. That is because for the same application set, the time interval $[r, d - e]$ for each application is fixed. As the number of intermediate points becomes larger, the time difference between every two successive potential starting time points becomes smaller. Hence, the accrued utility value differences between two consecutive potential starting time points become smaller. Therefore, the variation of the system total accrued utility value with a different number of intermediate points selected becomes smaller.

From Fig. 12, it can be seen that the profitable application ratio is not sensitive to the number of intermediate points either, except for the case when $n = 0$. When n increases from 2 to 8, the variation of the profitable application ratio is less than 0.5%.

Large Number of Intermediate Points

We further extend the range of the number of intermediate points n and set $n = 0, 10, 20, 30$ and 40. The normalized system total accrued utility value and the normalized profitable application ratio are depicted in Fig. 13 and Fig. 14, respectively. Both figures are normalized to the results obtained with $n = 40$.

Fig. 13 shows that, as the number of intermediate points n increases from 10 to 40, the system's total accrued utility value change is less than 1%. Similar property can be observed on the profitable application ratio as shown in Fig. 14. Hence, in our next experiment study, we set the number of intermediate points n of the STIB-C algorithm to 2, i.e., $n = 2$.

7.3.2 Performance Comparison with the FCFS with backfilling, the Gang-EDF, and the 0-1 Knapsack Based Approaches for Continuous Time Systems

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

System Utility Value Comparison

Fig. 15 depicts the result of the system total accrued utility value of different approaches, which are normalized to the system total accrued utility value obtained by the FCFS with backfilling approach.

From Fig. 15, it can be seen that, on average, the STIB-C algorithm outperforms the other three algorithms. It obtains as much as over 6 times system utility than the FCFS with backfilling when the work load is above one. Fig. 15 also shows that the performances of the Gang EDF and the FCFS with backfilling are similar, and the system total accrued utility value obtained by the STIB-C algorithm is over 6 times and over 4.5 times as much as the values obtained by the Gang EDF algorithm and the 0-1 Knapsack based algorithm, respectively. When the system load is 0.5, the STIB-C algorithm can achieve the system's total accrued utility value as much as 4.5 times for both the FCFS with backfilling and the Gang EDF algorithms, and 1.2 times as much for the 0-1 Knapsack based scheduling algorithm. These results are similar to the result of the STIB algorithm for a discrete time system.

Profitable Application Ratio Comparison

Fig. 16 shows the profitable application ratio under different algorithms with different system loads ω.

It is consistent with the result for a discrete time domain, both the STIB-C and the 0-1 Knapsack based algorithms have higher application profitable ratios than the FCFS with backfilling and the Gang EDF scheduling algorithms. Under overload situations, the STIB-C algorithm outperforms the other three scheduling approaches.

The STIB-C algorithm can obtain about four times as much as the profitable application ratio of the FCFS with backfilling algorithm when the system load is larger than one. Moreover, the performance of the Gang EDF and the FCFS with backfilling in terms of the profitable application ratio is similar, and the STIB-C algorithm achieves about four and close to two times as much as the profitable application ratios achieved by the Gang EDF and the 0-1 Knapsack based algorithms, respectively.

8 Conclusion

This paper presents two Spatial-Temporal Interference Based scheduling algorithms for parallel and time-sensitive applications, i.e., the STIB and the STIB-C algorithms, with the optimization goal to maximize the system's total accrued utility value when the system operates in a discrete or continuous time domain, respectively. We have formally proved that the STIB algorithm is a 2-approximation algorithm.

Our simulation results have not only confirmed that the STIB algorithm is indeed a 2-approximation algorithm, but also shows that the accrued utility values obtained through the STIB algorithm is very close to the optimal accrued utility value. In fact, on average, the value obtained by the STIB is over 92.5% of the optimal value. In addition, compared with other approaches in the literature, on average, both the STIB and the STIB-C algorithms have clear advantages in terms of the system's total accrued utility values and the profitable application ratio. However, as the time complexity of the STIB algorithm is in terms of valid time points within a given application set which may be larger than the number of applications, hence, it is possible that the STIB algorithm has a higher time complexity than the Gang EDF and the FCFS with backfilling algorithms. One of our future works is to reduce the number of checking points needed for the STIB algorithm.

It is worth pointing out that in this work the accrued utility value is associated with a parallel application, not with an individual task or a subset of parallel tasks within a parallel application. We believe such association is more general as if we need to optimize accrued utility value based on individual task(s) within a parallel application, we can treat such task(s) themselves as parallel applications and schedule them with the STIB or the STIB-C algorithms depending on if the time domain is discrete or continuous.

However, this current work is based on the assumption that all applications are narrow applications, i.e., the number of concurrent tasks within an application is no more than half the number of processing units existed in the system. For our future work, we will study the case when the assumption is relaxed, i.e., when there are both narrow and wide applications in the application set. For a continuous time system, we currently decide on finite starting time points with uniform distribution. How to decide a finite set of potential starting time points to improve the system's total accrued utility value is another study in our future work. Currently, we do not know the sufficient condition for the proposed methodology and we will study it in our future work. We will also extend our experimental study on the robustness of both the STIB and the STIB-C algorithms, such as the application execution time diversion.

Acknowledgment

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

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] J. Valenzuela, J. Wang, and N. Bissinger, "Real-time intrusion detection in power system operations," *Power Systems, IEEE Transactions on*, vol. 28, no. 2, pp. 1052–1062, 2013.

[3] Y. Chen, S. Nyemba, and B. Malin, "Detecting anomalous insiders in collaborative information systems," *Dependable and Secure Computing, IEEE Transactions on*, vol. 9, no. 3, pp. 332–344, 2012.

[4] 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.

[5] 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.

[6] L. Su, Q. Li, S. Hu, S. Wang, J. Gao, H. Liu, T. Abdelzaher, J. Han, X. Liu, Y. Gao, and L. Kaplan, "Generalized decision aggregation in distributed sensing systems," in *Real-time Systems Symposium. RTSS 2014.*, 2014, pp. 1–10.

[7] 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.

[8] W. L. D. Leung, R. Vanijjirattikhan, Z. Li, L. Xu, T. Richards, B. Ayhan, and M. Y. Chow, "Intelligent space with time sensitive applications," in *Advanced Intelligent Mechatronics. Proceedings, 2005 IEEE/ASME International Conference on*, 2005, pp. 1413–1418.

[9] T. Henderson and S. Bhatti, "Networked games: a qos-sensitive application for qos-insensitive users," *Proceedings of the Acm Sigcomm Revisiting Ip Qos Workshop*, pp. 141–147, 2003.

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

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

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

[13] 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.

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

[15] 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.

[16] 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.

[17] S. Li, M. Song, P. jun Wan, and S. Ren, "Maximizing system's total accrued utility value for parallel and time-sensitive applications," in *Performance Computing and Communications Conference (IPCCC), 2014 IEEE International*, Dec 2014, pp. 1–8.

[18] R. K. Clark, "Scheduling dependent real-time activities," Ph.D. dissertation, Carnegie Mellon University, 1990.

[19] G. Koren and D. Shasha, "Dover: an optimal on-line scheduling algorithm for overloaded real-time systems," in *Real-time Systems Symposium. RTSS 1992. 13th IEEE International*, 1992, pp. 290–299.

[20] D. Mosse, M. Pollack, and Y. Ronen, "Value-density algorithms to handle transient overloads in scheduling," in *Real-Time Systems, Proceedings of the 11th Euromicro Conference on*, 1999, pp. 278–286.

[21] K. Chen and P. Muhlethaler, "A scheduling algorithm for tasks described by time value function," *Real-Time Systems*, vol. 10, no. 3, pp. 293–312, 1996.

[22] 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.

[23] P. Li, "Utility accrual real-time scheduling: Models and algorithms," Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2004.

[24] A. Burns, D. Prasad, A. Bondavalli, F. Di Giandomenico, K. Ramamritham, J. Stankovic, and L. Strigini, "The meaning and role of value in scheduling flexible real-time systems," *Journal of Systems Architecture*, vol. 46, no. 4, pp. 305–325, 2000.

[25] 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.

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

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

[28] O.-H. Kwon and K.-Y. Chwa, "Scheduling parallel tasks with individual deadlines," *Theoretical Computer Science*, vol. 215, no. 1, pp. 209–223, 1999.

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

[30] M. Grenier and N. Navet, "Fine-tuning mac-level protocols for optimized real-time qos," *Industrial Informatics, IEEE Transactions on*, vol. 4, no. 1, pp. 6–15, 2008.

[31] C.-G. Lee, J. Hahn, Y.-M. Seo, S.-L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C.-S. Kim, "Analysis of cache-related preemption delay in fixed-priority preemptive scheduling," *Computers, IEEE Transactions on*, vol. 47, no. 6, pp. 700–713, 1998.

[32] H. Ramaprasad and F. Mueller, "Tightening the bounds on feasible preemptions," *ACM Trans. Embed. Comput. Syst.*, vol. 10, no. 2, pp. 27:1–27:34, Jan. 2011.

[33] G. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems a survey," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 3–15, Feb. 2012.

[34] 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.

[35] S. Li, M. Song, Z. Li, S. Ren, and G. Quan, "Maximizing online service profit for time-dependent applications," in *Proceedings of the 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, 2013, pp. 342–345.