2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)

Improving Multiprocessor Real-Time Systems with Bursty Inputs under Global EDF using Shapers

Yue Tang[1], Yuming Jiang[2], Xu Jiang[1], Nan Guan[1]

[1]The Hong Kong Polytechnic University, Hong Kong [2]Norwegian University of Science and Technology, Norway

***Abstract*—We propose an approach to calculate delay bound for multiprocessor real-time systems scheduled by GEDF. Different from most existing analysis techniques analyzing sporadic tasks, we consider bursty tasks which have more general arrival patterns. In detail, we use shapers to eliminate burst in original system inputs and generate sporadic job sequences, and then calculate the delay bound of each task. To further improve our approach, we design a heuristic algorithm to make as more tasks as possible to meet their deadlines by adjusting settings of shapers. Experiments show that the proposed algorithm can lead to improvement of acceptance ratio and the delay bound derived is much smaller than that by compared existing work.**

***Index Terms*—shaper, delay bound, global EDF**

I. Introduction

It is widely accepted that future real-time embedded systems will be deployed on multi-processors, to meet their rapidly increasing requirements of both high computational capacity and low power consumption. One of the crucial requirements that must be satisfied by a multiprocessor real-time system is that it can provide bounded delay, which describes the duration from each task activation to its execution completion. Deriving the delay bound of each task in a system during the design stage is important for both hard and soft real-time systems, since delay bound can not only be used to test the schedulability of hard real-time systems, but perform as an indicator for system performance of soft real-time systems. In our work, we consider the analysis of delay bound under GEDF for multiprocessor systems.

Global Earlist Deadline First (GEDF) scheduling is widely used and studied since it incurs less preemptions and migrations compared with optimal scheduling and the maximum delay is bounded when it is used to schedule tasks in soft real-time systems. Most of existing work for calculating delay bound on multiprocessors under GEDF analyzes sporadic tasks [3]–[5], [7], where any two consecutive jobs of task τ_i are assumed to be released with minimum separation time T_i. However in real applications burst exists, where a relatively large number of jobs are released over a short interval. Comparing sporadic and bursty inputs, the minimum separation time between two consecutive jobs in different intervals keeps more stable for the former, and it changes a lot for the latter. As a result, the sporadic task model denoting sporadic inputs is not a good choice for bursty inputs. When setting the period of a sporadic task as the minimum separation time during bursty interval in modeled bursty inputs, the over-estimated workload leads to pessimistic results, and setting that as the minimum separation time during more smooth intervals generates a wrong model due to under-estimated workload. Further, when jobs arrive simultaneously, the minimum separation time can not be specified. In this case, the period does not exist and bursty inputs can not be modeled as sporadic tasks.

As a result, most existing analysis techniques considering sporadic tasks can not directly be used to analyze systems with bursty inputs. What's more, it is difficult to extend the analysis techniques to bursty inputs since the complex job sequences make it hard to specify the worst case of execution. Compared with sporadic tasks, the work analyzing bursty tasks is much less. And for the problem under our setting, there is only one [8]. However, the delay bound derived by [8] is very large compared with the real requests of each input task. In summary, none of existing work provides a satisfactory solution for the analysis of bursty inputs .

Contributions. We propose a new approach to calculate delay bound for a multiprocessor system with bursty tasks under GEDF. In detail, we model bursty inputs as bursty tasks and deploy shapers for input bursty tasks. Once job sequences generated by bursty tasks enter the system, they first go through corresponding shapers, after which the behaviors of these jobs conform to sporadic patterns. Then these output jobs of shapers go into the scheduler and complete execution. With shapers, existing analysis techniques analyzing sporadic tasks can be applied (in this work we adopt the techniques in [7]), and the delay bound of each task is calculated. To further improve our approach, we design a heuristic algorithm which can increase the number of tasks meeting their deadlines in a task set by adjusting settings of shapers. Experiments show that the proposed algorithm can improve the acceptance ratio, and the derived delay bound is much smaller compared with the state-of-art.

II. Related Work

Shapers have been adopted to control traffic and transform arrival patterns since its analysis method was first proposed in [12]. In [9], shapers are used to control the traffic of higher-priority tasks so as to reduce interference to lower-priority ones under fixed-priority scheduling on uniprocessors, thus improving the system schedulability. Shapers are also used in EDF scheduling [6], where arrival patterns are changed to reduce peak temperature of chips. However, both of these two work

978-1-7281-0151-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ISORC.2019.00043

assumes uniprocessors and the analysis on multiprocessors is not considered.

There are other techniques which implement similar function as shapers. In [11], Richter et al. proposed event model interfaces (EMIFs) and event adaptation function (EAF) to transform event streams to a different event model, i.e., from periodic with jitter to simple periodic. However, the input types of these techniques are quite limited, which differ from shapers whose inputs are general arrival patterns.

Besides shaping system inputs, another important part of analysis is computing delay bound of multiprocessor real-time systems. For doing it under GEDF, there is already much work, and one of the representative techniques is [3], which proves that each task completes execution at most WCET plus an expression after its deadline, where the expression is the same for all tasks in a task set. The technique in [3] was improved in [4], where the above mentioned added expression is different for each task, thus reducing the pessimism. The analysis technique was further extended to arbitrary-deadline sporadic task systems in [5] and more general scheduling strategies in [7].

III. Preliminaries

A. System Model

Scheduling strategy and workload model. The system inputs are modeled as a set of n bursty tasks $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$, and they are considered to be scheduled under global EDF on $m \geq 2$ identical processors. In generally considered EDF scheduling, the relative deadline not only expresses the constraint for delay bound, but acts as a priority indicator. A job's execution priority is decided by its absolute deadline (release time plus the relative deadline of the task releasing it). The earlier the absolute deadline is, the higher priority a job has. In our work, we consider EDF of a more general type where the relative deadline only constrains a task's delay bound, and there exists another variable deciding the execution priority of each job. The relative deadline and the priority indicator are set as two independent values and they are not necessarily the same. Consistent with this general EDF scheduling, bursty task models are as follows:

Each bursty task τ_i defines an infinite sequence of jobs. Jobs released by one task are assumed to be sequential and at any time may execute on at most one processor. We also assume that more than one job can be released simultaneously by the same task. A task τ_i is characterized by $(C_i, D_i, \lambda_i, \alpha_i(\Delta))$, where C_i, D_i and λ_i are non-negative integers denoting worst-case execution time (WCET), relative deadline and priority level respectively. Since we consider a general-style EDF as introduced above, D_i and λ_i are independent. $\alpha_i(\Delta)$ is the arrival function of task τ_i modeling its job sequence . Specifically, $\alpha_i(\Delta)$ is the maximum number of jobs released by task τ_i in any (left-open or right-open) time interval of length Δ.

Generally, an arrival function can be any sub-additive curve, and it can model different types of job sequences (here we ignore the execution time of jobs and only consider their release time). The more complicated the expression of $\alpha_i(\Delta)$ is, the more complex job sequences it can model. For example, when $\alpha_i(\Delta) = \lceil \frac{\Delta}{p_i} \rceil$, it models job sequences generated by a sporadic task with period p_i. When $\alpha_i(\Delta) = \lceil \frac{\Delta + j_i}{p_i} \rceil$, it models sporadic job sequences with period p_i and jitter j_i, where the difference between supposed and actual release time of each job is at most j_i and at most $\lfloor \frac{j_i}{p_i} \rfloor + 1$ jobs can be released at the same time. In our work, we do not place any constraints on the expressions of arrival functions.

For each arrival function $\alpha_i(\Delta)$, there exists at least a pair of (η_i, B_i) which satisfies:

$$\alpha_i(\Delta) \leq \eta_i * \Delta + B_i \ for\ all\ \Delta \geq 0 \tag{1}$$

, where $\eta_i > 0$ and $B_i \geq 0$.

In the remaining part of paper, we set $\eta_i = lim_{\Delta \to +\infty} \frac{\alpha_i(\Delta)}{\Delta}$, which is the minimum value of η_i that satisfies formula (1).

An example. Figure 1 shows one job sequence of task τ_i which generates jobs with period 5 and jitter 10. The arrival function of task τ_i is $\alpha_i(\Delta) = \lceil \frac{\Delta + 10}{5} \rceil$. The jobs which should have been released at time 0 and 5 (shown with dashed up arrows) experience jitter of 10 and 5 respectively. As a result, three jobs are released simultaneously at 10, corresponding to $\alpha_i(0^+) = 3$. The smallest η_i satisfying formula (1) is $\frac{1}{5}$, and with $\eta_i = \frac{1}{5}$ the smallest B_i satisfying (1) is 3, that is, $y = \frac{\Delta}{5} + 3$ is the closest upper bound of $\alpha_i(\Delta)$, which is shown by the dashed black line.

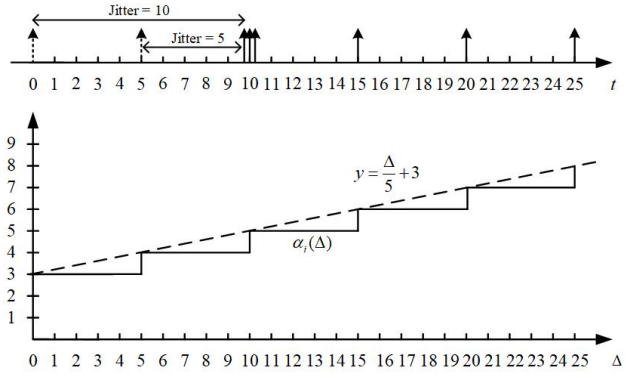


Fig. 1. A job sequence of τ_i and corresponding arrival function $\alpha_i(\Delta)$

For each task τ_i, the maximum amount of execution units it requires in any time interval of length Δ is $C_i * \alpha_i(\Delta)$. Its utilization is denoted by $U_i = C_i * \eta_i$, and the total utilization of task set τ is denoted by $U = \sum_{i=1}^{n} C_i * \eta_i$. Based on the analysis in [3], the following two conditions must be satisfied to guarantee that the system is not overloaded.

$$\eta_i * C_i \leq 1, \sum_{i=1}^{n} \eta_i * C_i \leq m$$

A task is schedulable if it meets its deadline, and a task set is schedualable when all tasks in it meet their deadlines.

Resource model. The amount of resource provided by processors is denoted by resource function $\beta(\Delta)$. The process of modeling resource as resource function is different depending on whether the resource provided to the task can be specified. When the amount of resource a task occupies can be specified, the resource function is derived by calculating the minimum number of execution units available over any time interval of length Δ. For example, if a task executes exclusively on an identical processor providing δ units of resource every u time units, the resource function is denoted as:

$$\beta(\Delta) = \frac{1}{u} * (\Delta - (u - \delta))$$

When it is hard to find out the exact amount of resource offered to each task such as in EDF scheduling, the resource function is derived based on the delay bound of each task when scheduled in the system.

Corollary 1: [1] Suppose the delay bound experienced by a task when scheduled in a system is DLY, then the system provides a service function $\beta(\Delta)$ to the task where

$$\beta(\Delta) = \delta_{DLY}(\Delta) = \begin{cases} 0, & 0 \leq \Delta \leq DLY \\ +\infty, & \Delta > DLY \end{cases}$$

δ_{DLY} is called the 'impulse function', and it has the property that for any wide-sense increasing function $\theta(t)$ defined with $t \geq 0$,

$$(\theta \otimes \delta_{DLY})(t) = \begin{cases} \theta(t - DLY), & t \geq DLY \\ \theta(0), & otherwise \end{cases}$$

B. Analysis in Network Calculus

Execution of tasks in systems is modeled into abstract components in Network Calculus. Assume that a task τ_i with arrival function $\alpha_i(\Delta)$ and WCET C_i is executed in a system providing service function $\beta_i(\Delta)$, then C_i, $\alpha_i(\Delta)$ and $\beta_i(\Delta)$ make up inputs of the corresponding component, based on which the delay bound DLY_i of τ_i is calculated as:

$$DLY_i = Del(C_i, \alpha_i(\Delta), \beta_i(\Delta)) \tag{2}$$

where

$$Del(C, \alpha(\Delta), \beta(\Delta)) = \sup_{\lambda \geq 0}\{inf\{d \geq 0 : C * \alpha(\lambda) \leq \beta(\lambda + d)\}$$

If τ_i goes through an integrated system composed of a sequence of m subsystems providing service functions $\beta_1(\Delta), \beta_2(\Delta)...\beta_m(\Delta)$, then its execution on the integrated system is modeled into a sequence of m components , where the output arrival function of a component is the input arrival function of its subsequent component. There are two ways to calculate the delay bound of τ_i. The first one is to add up the segregated delay bound calculated with formula (2) at each component.

$$DLY^i_{add} = DLY^i_1 + DLY^i_2 + ... + DLY^i_m$$

, where $DLY^i_j = Del(C_i, \alpha^i_j, \beta_j)$, α^i_j is the input arrival function of $j-th$ component and $\alpha^i_0 = \alpha_i$.

The second way, which is more precise and widely used, is to concatenate the service function of m components first and get the service function provided by the integrated system

$$\beta_{cont}(\Delta) = \beta_1(\Delta) \otimes \beta_2(\Delta)... \otimes \beta_m(\Delta) \tag{3}$$

Then the delay bound of task τ_i through the integrated system is

$$DLY^i_{cont} = Del(C_i, \alpha_i, \beta_{cont}) \tag{4}$$

C. Greedy Shaper

A greedy shaper S processes job sequences and forces its output job sequences to conform to some time constraints which generally set the minimum separation time between two consecutive jobs. A shaper works as follows:

when a job arrives at a shaper, the shaper first checks whether outputting the job is consistent with the supposed time constraint. If so, the job will leave the shaper immediately. Otherwise, the shaper will buffer the job and output it as soon as outputting the job satisfies the constraint.

In Network Calculus, a greedy shaper is modeled as an abstract component. Each shaper S is designed with a shaping function $\sigma(\Delta)$, which uniquely decides the behavior of the shaper, and thus its output job sequence. For example, when the shaping function is designed as $\sigma(\Delta) = \lceil \frac{\Delta}{p} \rceil$, the output job sequence has minimum separation time p between any two consecutive jobs. In this special case of generating sporadic job sequences, we call p the period of the shaper.

Assume that the job sequence of a task with arrival function $\alpha(\Delta)$ passes a shaper with shaping function $\sigma(\Delta)$, then the arrival function of output job sequence is calculated by:

$$\alpha'(\Delta) = \alpha(\Delta) \otimes \sigma(\Delta) = min(\alpha(\Delta), \delta(\Delta)) \tag{5}$$

And the delay bound of the task experienced at the shaper is calculated by [1]

$$DLY_s = Del(1, \alpha(\Delta), \sigma(\Delta))$$

An example. Assume that the job sequence shown in upper part of Figure 2 passes a shaper, whose shaping function is shown by $\sigma'(\Delta)$ in Figure 3. The output job sequence of the shaper is shown in the lower part of Figure 2. Based on $\sigma'(\Delta)$, there can not be more than 1 output job in any time interval of length 3, so the three jobs released simultaneously at time 10 can not leave the shaper at the same time. While the first one of these simultaneously released 3 jobs is output exactly at 10, the other two jobs are delayed in the shaper until time 13 and 16, respectively. The delay of release time is shown by the dashed arrows. The arrival function of output job sequence, denoted by $\alpha'(\Delta)$, is same as $\sigma'(\Delta)$ according to formula (5). And the maximum delay experienced is 6, corresponding to the delay experienced by the third job released at time 10. If we change the shaping function to $\sigma''(\Delta)$, then the output arrival function is changed to $\alpha''(\Delta)$ correspondingly, and the maximum delay is 8. It can be seen that given the same input

[1] Since we only concern about the number of jobs that can be output during certain length of intervals, the WCET of the task can be viewed as 1.

function, the lower the shaping function is, the larger the delay bound is.

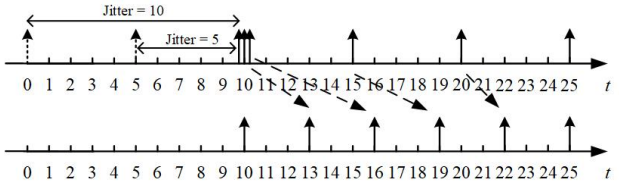


Fig. 2. The input job sequence and output job sequence of a shaper

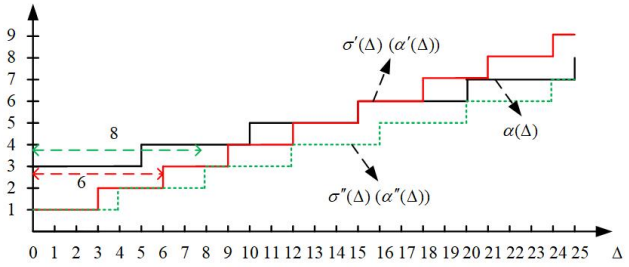


Fig. 3. Different shaping functions and corresponding output arrival functions

IV. Overview

Basic idea. In existing work about calculating the delay bound for tasks scheduled on multiprocessors under global EDF, jobs are assumed to directly enter the ready queue of the scheduler as soon as being released. Under this assumption, the more complex the released job sequence is, the harder it is to accurately analyze the worst-case amount of interference suffered by the analyzed task. As a result, it is difficult to analyze the delay bound of each task.

The difficulty brought by complex job sequences to scheduling analysis can be sidestepped when deriving the delay bound, which is implemented in our work by the adoption of shapers. We assume that jobs first go through shapers then enter the ready queue of scheduler, rather than directly being ready for scheduling after being released as in existing work. Using the traffic control function of shapers, complex job sequences are shaped to conform to sporadic patterns, where the separation time between two consecutive jobs is no smaller than a constant, and then existing analysis techniques targeting sporadic tasks are applied.

A system with shapers behaves as follows: after a job is released by a task, it first goes through a shaper. The shaper checks the separation between release time of current job and the output time of its predecessor job. If it is no shorter than the supposed constant specified by the shaping function, the shaper outputs the job instantaneously and then the job enters the ready queue of the scheduler. Otherwise, the shaper buffers the job and outputs it as soon as the minimum separation time is satisfied. In total, the delay a job experiences is from its release to leaving the shaper to completing execution at the scheduler, rather than the single part at the scheduler. And the delay bound of a task is the maximum delay among that of all its jobs.

System architecture. A system with shapers eliminating burst in inputs is shown in Figure 4. For each bursty task τ_i, a greedy shaper S_i is deployed to shape the job sequence released by it. Each shaper has a buffer, which is used to store jobs that can not enter the ready queue of the scheduler due to violation of minimum separation time constraint. We assume that the buffer size of each shaper equals 0 when the first job is released and is large enough so that there is no job loss. Then the input of the scheduler is transformed from bursty tasks (when without shapers) to sporadic tasks and the delay bound becomes analyzable.

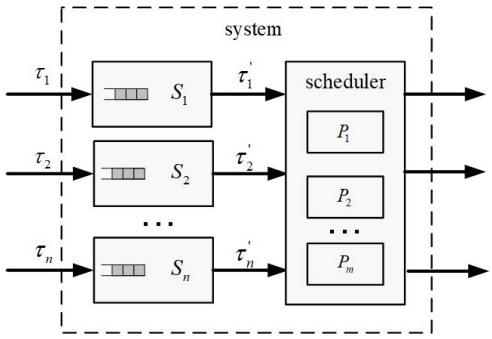


Fig. 4. The system architecture with shapers

V. Analyzing the System with Shapers

This section introduces the approach to derive the delay bound for a multiprocessor real-time system with bursty inputs under global EDF by the adoption of shapers. We first model the scheduler as an abstract component. Then the system with shapers can be modeled as sequential components where the outputs of shapers constitute one of the inputs of the scheduler component. After that, we show how to compute the delay bound and adjust shapers to increase the number of schedulable tasks in a task set.

A. Modeling the Scheduler

In this subsection we first model the input job sequence of the scheduler, then model EDF scheduler as an abstract component, and specify how to calculate its inputs and outputs. Note that original input tasks and shapers are assumed to be given.

In a system with shapers, the burst in original input job sequences is eliminated and the inputs to the scheduler conform to a sporadic pattern. Based on this, we define *virtual sporadic task*:

Definition 1 (Virtual sporadic task): Suppose that the job sequence released by task τ_i passes a greedy shaper S_i with shaping function $\theta_i(\Delta) = \lceil \frac{\Delta}{T_s^i} \rceil$. Then the output job sequence of S_i has minimum separation time T_s^i, and is modeled by an implicit-deadline sporadic task τ_i'. τ_i' is called the virtual

sporadic task of τ_i, whose period equals T_s^i and WCET equals that of τ_i. [2]

Next we model an EDF scheduler as an EDF component.

Definition 2 (EDF component): A scheduler that schedules a task set $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ under global EDF is modeled as an EDF component, which is characterized by $(\overrightarrow{C}, \overrightarrow{\alpha}, \overrightarrow{\beta}, \overrightarrow{\alpha'})$, where $\overrightarrow{C} = \{C_1, C_2, ..., C_n\}$ is the WCET of each task, $\overrightarrow{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ is the arrival function, $\overrightarrow{\beta} = \{\beta_1, \beta_2, ..., \beta_n\}$) denotes the service curve provided for each task, and $\overrightarrow{\alpha'} = \{\alpha'_1, \alpha'_2, ..., \alpha'_n\}$ is the arrival function of processed jobs.

In the framework shown in Figure 4, the inputs to EDF scheduler are virtual sporadic tasks. So for an EDF component, each element in $\overrightarrow{C}$ equals the WCET of corresponding input bursty task. Next we will show how to derive $\overrightarrow{\alpha}, \overrightarrow{\beta}, \overrightarrow{\alpha'}$ for an EDF component.

1) Deriving $\overrightarrow{\alpha}$

Suppose the periods of virtual sporadic tasks which are inputs to an EDF scheduler are: $\overrightarrow{T_s} = \{T_s^1, T_s^2, ..., T_s^n\}$, then the arrival function of each input task of EDF component is denoted by:

$$\alpha_i(\Delta) = \lceil \frac{\Delta}{T_s^i} \rceil$$

Then we have

$$\overrightarrow{\alpha} = \{\lceil \frac{\Delta}{T_s^1} \rceil, \lceil \frac{\Delta}{T_s^2} \rceil, ..., \lceil \frac{\Delta}{T_s^i} \rceil\}$$

2) Deriving $\overrightarrow{\beta}$ and $\overrightarrow{\alpha'}$

Suppose the delay bound experienced by each input task at EDF scheduler is known as $DLY_{schd} = \{DLY_{schd}^1, DLY_{schd}^2, ..., DLY_{schd}^n\}$, then $\overrightarrow{\beta}$ and $\overrightarrow{\alpha'}$ can be derived. Next we will first introduce the derivation of delay bound of each task at EDF scheduler part, and then show how to derive $\overrightarrow{\beta}$ and $\overrightarrow{\alpha'}$ with DLY_{schd}.

Since each input task of EDF scheduler is a virtual sporadic task in our system, the delay bound experienced can be calculated by existing analysis techniques analyzing sporadic tasks scheduled under global EDF. We first model the analysis process of deriving the delay bound at the scheduler:

Definition 3 (Calculation of delay bound): Assume that the input of a global EDF scheduler is a task set of n sporadic tasks. The task set is characterized by $\overrightarrow{C^*} = \{C_1^*, C_2^*, ..., C_n^*\}$, $\overrightarrow{T^*} = \{T_1^*, T_2^*, ..., T_n^*\}$, and $\overrightarrow{D^*} = \{D_1^*, D_2^*, ..., D_n^*\}$, which denote each sporadic task's WCET, period and relative deadline respectively. Suppose the analysis technique ξ is adopted to derive the delay bound $DLY_{schd} = \{DLY_{schd}^1, DLY_{schd}^2, ..., DLY_{schd}^n\}$ at the scheduler for each task in the task set, then the analysis process can be modeled as a function $Caldelay$ with

$$DLY_{schd} = Caldelay(\xi, \overrightarrow{C^*}, \overrightarrow{T^*}, \overrightarrow{D^*})$$

[2]Note that virtual task τ_i' does not exist in real systems and is defined only for analysis. Also, the deadline of a virtual sporadic task is more used as an indicator for scheduling priority than a constraint for completion time.

In our work, we adopt analysis techniques in [7] (denoted by ξ_G) and the delay bound at the scheduler can be calculated as

$$DLY_{schd} = Caldelay(\xi_G, \overrightarrow{C}, \overrightarrow{T_s}, \overrightarrow{T_s})$$

, where $\overrightarrow{C}$ equals the WCET and $\overrightarrow{T_s} = \{T_s^1, T_s^2, ..., T_s^n\}$ equals the period of n virtual sporadic tasks.

Then $\overrightarrow{\beta}$ and $\overrightarrow{\alpha'}$ can be derived based on DLY_{schd}.

The service function β_i provided for each virtual task τ_i' can be derived based on DLY_{schd} and Corollary 1:

$$\beta_i(\Delta) = \delta_{DLY_{schd}^i}(\Delta) \begin{cases} 0, & 0 \le \Delta \le DLY_{schd}^i \\ +\infty, & \Delta > DLY_{schd}^i \end{cases}$$

Each output arrival function is calculated as $\alpha_i'(\Delta) = \alpha_i(\Delta - DLY_{schd}^i)$, so we have

$$\overrightarrow{\alpha'} = \{\alpha_1(\Delta - DLY_{schd}^1), \alpha_2(\Delta - DLY_{schd}^2), ..., \alpha_i(\Delta - DLY_{schd}^i)\}$$

B. Calculating the Delay Bound

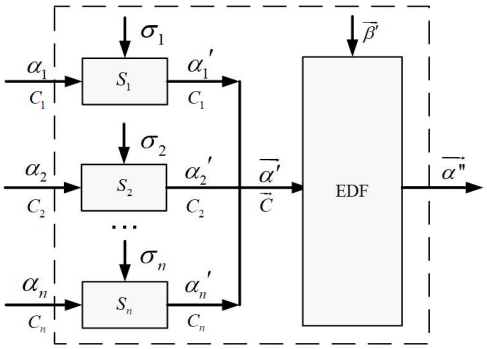


Fig. 5. Modeling the system in Figure 4 into a set of abstract components

After modeling the EDF scheduler as EDF component, the system in Figure 4 is modeled into a network composed of n greedy shapers and one EDF component in Figure 5. Based on formula (3) and (4), to derive the delay bound of each task traversing the system with shapers, the service function provided by each shaper component and EDF component must be known. In this subsection, we assume that $\sigma_i(\Delta)$ is given, based on which we can derive the period of each virtual sporadic task, the delay bound at the scheduler DLY_{schd}, and $\overrightarrow{\beta}$. Next we explore the remaining unknown parameter, that is, the service function provided by each shaper.

Lemma 1: Assume a job sequence generated by task τ_i with WCET C_i passes a shaper with shaping function σ_i, then the shaper provides to the task a service function $\beta_s(\Delta)$ denoted by $\beta_s(\Delta) = C_i * \sigma_i$.

Proof 1: The lemma can be easily proved with Corollary 1.5.1 in [1].

Now we can calculate the delay bound DLY_i for each task τ_i.

Lemma 2: The delay bound of each task τ_i is calculated as $DLY_i = DLY_s^i + DLY_{schd}^i$.

Proof 2: Based on the concatenation property, the delay bound is computed as

$$DLY_i = Del(C_i, \alpha_i, (C_i * \sigma_i(\Delta)) \otimes \beta_i)$$

, where $\beta_i = \delta_{DLY^i_{schd}}$.

Based on the property of β_i,

$$(C_i * \sigma_i(\Delta)) \otimes \beta_i = C_i * \sigma_i(\Delta - DLY^i_{schd})$$

Then we have

$$\begin{aligned} DLY_i &= Del(C_i, \alpha_i, C_i * \sigma_i(\Delta - DLY^i_{schd})) \\ &= Del(C_i, \alpha_i, C_i * \sigma_i(\Delta)) + DLY^i_{schd} \\ &= Del(1, \alpha_i, \sigma_i(\Delta)) + DLY^i_{schd} \\ &= DLY^i_s + DLY^i_{schd} \end{aligned}$$

Then the lemma is proved.

From Lemma 2, the delay bound of each task equals the sum of that at the shaper and at the scheduler. Also, given fixed inputs, processors and analysis techniques for sporadic tasks, the shaping function of each shaper uniquely decides the delay bound of each task.

C. The Design of Shapers

Different from above two subsections where shapers are known, in this subsection they are not given and become the target of our analysis. We will discuss how to implement a shaper with sporadic output and specify settings of shapers to increase the number of schedulable tasks.

1) The implementation of greedy shaper

At the first sight, a token bucket seems to be one choice for generating sporadic outputs. However, the following example shows that the output job sequence of a token bucket can not satisfy the minimum separation time constraint in all cases.

An example. Suppose that a token bucket with token generation rate $1/T_s$ is used to generate a job sequence with minimum separation time T_s. Consider that two jobs arrive at the same time at $a(1)$ and the bucket is full with one token generated at $e(1)$ with $e(1) \leq a(1)$. At $a(1)$, one of the two jobs can pass the shaper and the other is buffered. Then at $e(1)+T_s$, a new token is generated and the second job can be emitted. However, the separation time between it and the first job is $e(1) + T_s - a(1) \leq T_s$. The example shows that only when $e(1) = a(1)$, the minimum separation time constraint can be guaranteed by the token bucket. However, $e(1) = a(1)$ is only one special case and the behavior of token bucket when $e(1) = a(1)$ can not represent general cases.

Based on this observation, we define an interval-guaranteeing shaper whose output always satisfies the minimum separation time constraint.

Definition 4 (Interval-guaranteeing shaper [3]): [2] An interval-guaranteeing shaper outputs jobs at $e(i)$ satisfying

$$e(1) = a(1); e(i) = max\{a(i), e(i-1) + T_s\}$$

[3]An interval-guaranteeing shaper is a type of greedy shaper.

where $a(i)$ denotes the arrival time of job i to the shaper, and T_s is the minimum separation time in the output job sequence.

Lemma 3: The shaping function $\sigma(\Delta)$ of an interval-guaranteeing shaper satisfies $\sigma(\Delta) = \lceil \frac{\Delta}{T_s} \rceil$, where T_s is the minimum separation time in its output.

Proof 3: Can be obtained based on the behavior of an interval-guaranteeing shaper.

2) Specify the setting of shapers

Now we have already designed a shaper with shaping function $\sigma(\Delta) = \lceil \frac{\Delta}{T_s} \rceil$ to implement minimum separation time T_s in its output job sequence. From last two subsections, for fixed input tasks executed on a multiprocessor platform, the periods of shapers uniquely decide the delay bound. Next we focus on how to adjust the value of period T^i_s of each shaper S_i to make more tasks schedulable.

To guarantee the system is not overloaded and jobs buffered at the shaper are limited, the following two conditions must be satisfied:

$$\sum_{i=1}^{n} \frac{C_i}{T^i_s} \leq m \tag{6}$$

$$0 \leq T^i_s \leq \frac{1}{\eta_i} \tag{7}$$

A simple choice to set $T^i_s = \frac{1}{\eta_i}$ with η_i defined in Section III. However, this can not guarantee the task meets its deadline. Actually, the shaper that makes a task meet its deadline can not be derived with simple observation and the reason is two-fold. First, the total delay bound is the sum of that at the shaper and that at the scheduler. Although larger period leads to lower utilizations and intuitively smaller delay bound in the part of EDF scheduler, it causes larger delay bound at the shaper. So it is hard to decide whether to increase or decrease the shaper's period without real computation. Second, changing the period of shaper corresponding to one virtual sporadic task will influence the execution of other virtual sporadic tasks. So the choice of period can not be independent.

To make more tasks schedulable, it is necessary to enumerate all possible values of T^i_s satisfying formula (6) and (7). The process can be time-consuming and we propose a simple heuristic to increase the number of schedulable tasks.

In Algorithm 1, τ is the input task set. A shaper is put into set *sched* if its corresponding bursty task meets its deadline, else is put into *unsched*. We first decrease the period of each shaper in *unsched* on the condition that the number of shapers in *sched* does not decrease and the corresponding bursty task's delay bound is smaller than before. After adjusting the periods of all shapers in *unsched*, we increase the period of each shaper in *sched* on the condition that its corresponding bursty task does not miss its deadline. For each iteration, we check whether the number of shapers in *sched* increases. If not, the iteration stops.

An example. We consider a task set composed of 5 tasks $\tau = \{\tau_1, \tau_2, ..., \tau_5\}$. The arrival function α_i of each task τ_i is characterized by $\alpha_i(\Delta) = \lceil \frac{\Delta + j_i}{p_i} \rceil$. The parameters and calculated delay bounds are shown in Table I. DLY is the calculated delay bound when setting $T^i_s = p_i$. T' is randomly

Algorithm 1 Decide the period of each shaper

for each shaper S_i, $T_s^i = 1/\eta_i$
calculate $DLY_i = DLY_s^i + DLY_{sched}^i$
calculate *sched* and *unsched*
$num_{sched} = size(sched)$
sort *unsched* with descending $DLY - D$
sort *sched* with descending $D - DLY$
$num = size(\tau)$
while $unsched(\tau)$ and $num_{sched} < num$ **do**
  $num_{sched} = size(sched)$
  while notempty(*unsched*) **do**
    current $\longleftarrow$ pop the first from *unsched*
    while $unsched(\tau)$ and $candecrease(T_s^{current})$ **do**
      $T_s^{current} = T_s^{current} - 1$
    end while
  end while
  calculate *sched* and *unsched*
  while notempty(*sched*) **do**
    current $\longleftarrow$ pop the first from *sched*
    while $unsched(\tau)$ and $canincrease(T_s^{current})$ **do**
      $T_s^{current} = T_s^{current} + 1$
    end while
  end while
  calculate *sched* and *unsched*
  $num = size(sched)$
end while

generated period for shapers on the condition that formula (6) and (7) are satisfied, and DLY' is corresponding delay bound. T'' is period derived with Algorithm 1, and DLY'' is the corresponding delay bound.

From the table, in the first two settings of shapers, some task will miss its deadline. After adjusting the period with the heuristic, all tasks meet their deadlines and the original task set becomes schedulable.

	τ_1	τ_2	τ_3	τ_4	τ_5
C	4	4	12	8	4
p	24	8	16	12	28
j	24	8	16	12	28
λ	16	8	16	12	24
D	36	36	56	52	40
T'	12	7	14	11	22
T''	10	8	16	12	12
DLY	62	30	54	42	70
DLY'	39	29	51	41	59
DLY''	34	30	54	42	38

TABLE I
A TASK SET AND ITS DELAY BOUND CALCULATED WITH DIFFERENT SETTINGS OF SHAPERS

VI. EVALUATION

We implement our proposed approach in RTC Toolbox [10] and conduct experiments to evaluate the performance.

Task generation. We first generate its arrival function based on the PJD workload model in RTC Toolbox [10] characterized by (p, j, d), where p denotes the period ($1/p$ is the long-time slope of arrival function, so $p = 1/\eta$, where η is defined in task model), j is the jitter, and d is the minimum separation time between two consecutive jobs:

$$\alpha^u(\Delta) = min\{\lceil\frac{\Delta + j}{p}\rceil, \lceil\frac{\Delta}{d}\rceil\}$$

The range of p, j, d will be specified in following content and j/p is set to be an integer in all experiment settings. After generating p, j, d, other parameters are randomly chosen in following ranges: $C \in [1, p]$, $\lambda \in [C, p]$, and $D \in [p * \frac{j}{p} + C + p, p * \frac{j}{p} + C + p + 500]$.

For each task set, we generate 5 tasks. Then the total utilization U of the task set is calculated as $U = \sum_{i=1}^{5} \frac{C_i}{p_i}$, and the number of processors m is set as $m = ceil(U)$.

In this section, we compare the performance of 4 approaches:

(1) the approach in [8], denoted by $'EXT'$.

(2) the approach proposed in our work with setting $T_s^i = p_i$, denoted by $'T = p'$.

(3) the approach proposed in our work with randomly generated T_s^i satisfying formula (6) and (7), denoted by $'T = random'$.

(4) the approach proposed in our work with the period setting derived with Algorithm 1, denoted by $'T = heuristic'$.

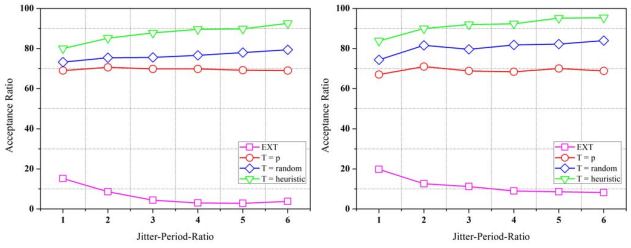
Note that in (2) (3) (4) we adopt the techniques in [7] at EDF scheduler part, T_s^i is the period of shaper corresponding to task τ_i and p_i is defined in the generation of task τ_i.

Experimental results. The above 4 approaches are compared in two aspects: acceptance ratio and normalized delay bound.

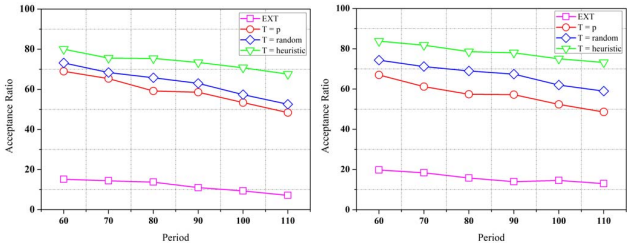
1) Acceptance ratio

Figure 6-(a) and (b) show the comparison among 4 approaches under different jitter-period-ratio (X-axis). We generate 500 task sets for each different jitter-period-ratio. In Figure 6-(a), p is randomly generated in $[1, 60]$ and $d = 0$. $d = 0$ means that jobs can be released simultaneously. Figure 6-(b) has the same parameter setting as Figure 6-(a) except that d is randomly generated in $[1, p]$, meaning that no two jobs are released at the same time. The Y-axis shows the percentage of schedulable task sets among all task sets generated for each jitter-period-ratio, denotedy by *Acceptance Ratio*. Based on the experiment results, our approach performs better than $'EXT'$ regardless of the jitter-period-ratio and settings of shapers. In detail, with the increasing of jitter-period-ratio, the acceptance ratio of $'EXT'$ decreases, since jobs released simultaneously can potentially lead to unfair allocation of processor resources, causing large incremental in calculated delay bound. What's more, the improvement by the heuristic increases with jitter-period-ratio, since same decrease of shaper period can potentially lead to more delay reduction at the shaper. The performance under $d \neq 0$ is slightly better than $d = 0$, since when $d \neq 0$ no two jobs are released simultaneously and the delay experienced by each task at the shaper is comparatively smaller.

Figure 6-(c) and (d) show the comparison among 4 approaches under different range of p (X-axis). For each value i

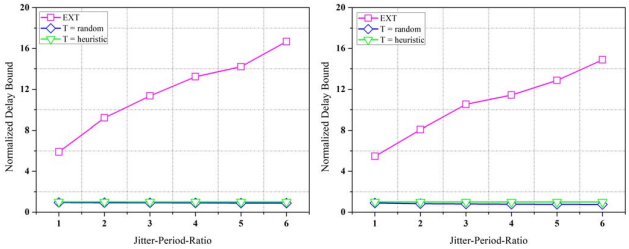


(a) Different jitter/period, $d = 0$ (b) Different jitter/period, $d \neq 0$

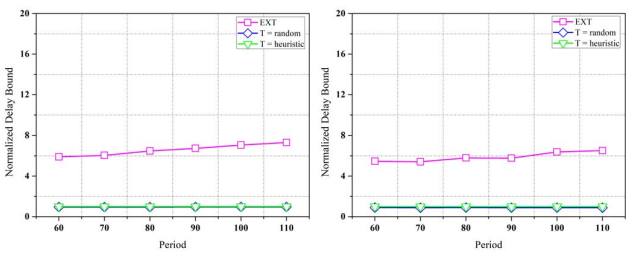


(c) Different period, $d = 0$ (d) Different period, $d \neq 0$

Fig. 6. The comparison of acceptance ratio



(a) Different jitter/period, $d = 0$ (b) Different jitter/period, $d \neq 0$



(c) Different period, $d = 0$ (d) Different period, $d \neq 0$

Fig. 7. The comparison of normalized delay bound

in X-axis, we generate 500 task sets with $p \in [1, i]$. In Figure 6-(c), $j/p = 1$ and $d = 0$. Figure 6-(d) has the same parameter setting as 6-(c) except that $d \in [1, p]$. It can be seen that $'T = heuristic'$ still has the best performance.

2) Normalized delay bound

Figure 7-(a) and (b) show the comparison among 4 approaches under different jitter-period-ratio (X-axis). The parameter settings and corresponding number of task sets are same as Figure 6-(a) and (b) respectively. We choose the delay bound calculated with $'T = p'$ as the standard, and the Y-axis shows the ratio between the delay bounds derived with other three different approaches and $'T = p'$, namely *Normalized delay bound*, calculated as $\frac{\sum_{i=1}^{5} DLY_{comp}^{i}/DLY_{p}^{i}}{5}$, where DLY_{p}^{i} is the delay bound of task τ_i calculated with $'T = p'$, and DLY_{comp}^{i} is that calculated with one of other three approaches. Each result for X-axis value i is the average of the normalized delay bound of task sets with jitter-period-ratio equal to i.

Figure 7-(c) and (d) show the comparison among 4 approaches under different range of p (X-axis). The parameter settings and corresponding number of task sets are same as Figure 6-(c) and (d) respectively.

Experiments show that the delay bound calculated by our methods is much smaller than that by $'EXT'$ under both different jitter-period-ratio and range of p.

VII. Conclusion

We propose an approach to derive the delay bound of real-time multiprocessor systems with bursty inputs scheduled by GEDF. In detail, shapers are inserted before the scheduler to transform bursty tasks into sporadic ones. Then we calculate the delay bound of each task based on scheduling analysis and Network Calculus. What's more, we propose a heuristic algorithm to improve the acceptance ratio of task sets by adjusting the periods of shapers.

In the future, we aim to combine shapers with other different types of workload and scheduling strategies.

VIII. Acknowledgment

This work is supported by the Research Grants Council of Hong Kong (GRF 15204917 and 15213818), National Natrual Science Foundation of China under grant 61532007 and 61672140 and the Ministry of Education Joint Foundation for Equipment Pre-Research under grant 6141A020333, and in part by the Fundamental Research Funds for the Central Universities under Grant N172304025.

References

[1] J. L. Boudec and P. Thiran. Network calculus - a theory of deterministic queuing systems for the internet. Springer Verlag, 2012.
[2] Cheng-Shang Chang. *Performance guarantees in communication networks*. 2000.
[3] U. Devi and J. Anderson. Tardiness bounds for global edf scheduling on a multiprocessor. *Real-Time Systems*, 38(2):133–189, 2008.
[4] J.P. Erickson, U. Devi, and S.K. Baruah. Improved tardiness bounds for global edf. In *ECRTS*, pages 14–23, 2010.
[5] J.P. Erickson, N. Guan, and S.K. Baruah. Tardiness bounds for global edf with deadlines different from periods. In *OPODIS.*, pages 286–301, 2010.
[6] Pratyush Kumar and Lothar Thiele. Cool shapers: shaping real-time tasks for improved thermal guarantees. In *DAC*, pages 468–473, 2011.
[7] H. Leontyev and J. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. *Real-Time Systems*, 44(1-3):26–71, 2010.
[8] H. Leontyev, S. Chakraborty, and J. Anderson. Multiprocessor extensions to real-time calculus. *Real-Time Systems*, pages 47–562, 2011.
[9] L. Phan and I. Lee. Improving schedulability of fixed-priority real-time systems using shapers. In *RTAS*, pages 217–226, 2013.
[10] K. Richter. Compositional scheduling analysis using standard event models. In *Ph.D. Thesis, Technical University Carolo-Wilhelmina of Braunschweig*. Springer Verlag, 2005.
[11] K. Richter, M. Jersak, and R. Ernst. A formal approach to mpsoc performance verification. *Computer*, 36(4):60–67, 2003.
[12] E. Wandeler, A. Maxiaguine, and L. Thiele. Performance analysis of greedy shapers in real-time systems. In *DATE*, pages 444–449, 2006.