Optimal Dataflow Scheduling on a Heterogeneous Multiprocessor With Reduced Response Time Bounds

Zheng Dong[1], Cong Liu[2], Alan Gatherer[3], Lee McFearin[4], Peter Yan[5], and James H. Anderson[6]

1 **The University of Texas at Dallas, Dallas, TX, USA**
zheng@utdallas.edu
2 **The University of Texas at Dallas, Dallas, TX, USA**
cong@utdallas.edu
3 **America Wireless Access Laboratory, Huawei Technologies Co. Ltd, USA**
alan.gatherer@huawei.com
4 **America Wireless Access Laboratory, Huawei Technologies Co. Ltd, USA**
lee.mcfearin@huawei.com
5 **America Wireless Access Laboratory, Huawei Technologies Co. Ltd, USA**
peter.yifey.yan@huawei.com
6 **University of North Carolina at Chapel Hill, Chapell Hill, NC, USA**
anderson@cs.unc.edu

Abstract

Heterogeneous computing platforms with multiple types of computing resources have been widely used in many industrial systems to process dataflow tasks with pre-defined affinity of tasks to subgroups of resources. For many dataflow workloads with soft real-time requirements, guaranteeing fast and bounded response times is often the objective. This paper presents a new set of analysis techniques showing that a classical real-time scheduler, namely earliest-deadline-first (EDF), is able to support dataflow tasks scheduled on such heterogeneous platforms with provably bounded response times while incurring no resource capacity loss, thus proving EDF to be an optimal solution for this scheduling problem. Experiments using synthetic workloads with widely varied parameters also demonstrate that the magnitude of the response time bounds yielded under the proposed analysis is reasonably small under all scenarios. Compared to the state-of-the-art soft real-time analysis techniques, our test yields a 68% reduction on response time bounds on average. This work demonstrates the potential of applying EDF into practical industrial systems containing dataflow-based workloads that desire guaranteed bounded response times.

1998 ACM Subject Classification C.3 Real-Time and Embedded Systems

Keywords and phrases real-time scheduling, schedulability, heterogeneous multiprocessor

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2017.15

1 Introduction

In many applications, such as traffic monitoring [12], trajectory tracking [14], and modem signal processing, dataflows periodically arrive in the form of data streams (e.g., continuous image frames). A stream processing system is required to handle such data streams often in a soft real-time fashion, e.g., results must be processed within a short and bounded time period. For example, in traffic monitoring systems and voice data processing systems in

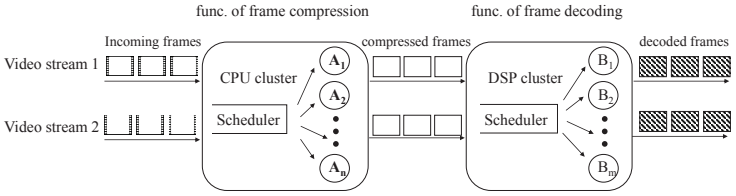


Figure 1 An example video stream processing system.

cellular networks, results become meaningless if they cannot be processed and returned to the end user within a bounded time window.

To handle continuously arrived dataflows with fast and bounded response times, heavily heterogeneous computing systems [2] containing various types of accelerators (e.g., digital signal processors–DSP) are used in many stream processing systems, such as Apache Storm and its successor Twitter Heron [13], StreamCloud [10] and Apache Spark Streaming [26]. In these systems, multiple types of resources are used to process different functionalities using dataflow tasks as the data source with pre-defined affinity of tasks to subgroups of resources (i.e., each functionality is designated to be processed on a specific type of computing resource, such as CPU, DSP, or one of the many types of hardware-accelerators). One significant challenge faced by such systems is the need to develop a multi-resource, real-time scheduler, which can correctly support a maximum load of dataflows that may fully utilize all system resources, along with the accompanying analytical schedulability test algorithms, which validate at design time whether a set of dataflows can feasibly run with bounded response times.

Fig. 1 shows a simplified example video stream application processed in a heterogeneous computing system. There are two types of computing resources: a cluster of CPUs and a cluster of DSPs [9]. Each video stream is independent and has two subtasks, where the first subtask is to compress incoming video frames and is dedicated to be executed on CPUs, while the second subtask is frame decoding and is dedicated to be executed on DSPs. In this example, each video stream has a period of $\frac{1}{24}s$, i.e., each stream periodically generates frames at a rate of 24 frame per second (FPS). For each frame, the output of the first subtask processed by CPUs is then fed to DSPs as data input to execute the second subtask. The multi-resource scheduler determines how to schedule and allocate resources to various subtasks belonging to different dataflow tasks.

Much recent work has been conducted on scheduling soft real-time (i.e., task response times must be provably bounded) tasks on a homogeneous multiprocessor [5, 7]. However, analysis is lacking for supporting soft real-time applications developed using the dataflow formalism in a heterogeneous computing system with pre-defined task affinity to resources. In this paper, we address this lack of support by presenting new schedulability analysis techniques for a classical real-time scheduler, namely earliest-deadline-first (EDF). The resulting scheulability test proves EDF to be an optimal solution for this heterogeneous dataflow scheduling problem in terms of system utilization. *That is, EDF can correctly support any set of dataflow tasks with probably bounded response times that may even require all types of computing resources to be fully utilized, thus incurring no capacity loss on any resource.*

Specifically, we present new schedulability analysis techniques showing that any dataflow task system is schedulable under EDF with bounded response times if $U^k_{sum} \leq M_k, 1 \leq k \leq m$, where m denotes the number of resource types, M_k denotes the number of processors of the k^{th} resource type, and U^k_{sum} denotes the total resource utilization required by all dataflow

tasks on processors of the k^{th} resource type. This schedulability test implies EDF's optimality due to no capacity loss on any type of resource. We have also conducted extensive experiments involving synthetic workloads with widely varied parameters, which demonstrate that the magnitude of the response time bound yielded under our schedulability test is reasonably small under all scenarios. Compared to the state-of-the-art soft real-time analysis techniques, our test yields a 68% reduction on response time bounds on average.

2 Related Work

Task scheduling has been a well-studied problem in many parallel and distributed systems (e.g., [21, 4, 24, 27]). A plethora of schedulers have been developed and used in practice, e.g., the Fair Scheduler, the Capacity Schedular, and delay scheduling. These scheduling strategies, however, cannot be applied to solve our problem of scheduling dataflow tasks with provably bounded response times on a heterogeneous computing platform, because they target at different application models and different performance objectives. On the other hand, the problem of scheduling task systems on multiprocessors with bounded response times [19, 3, 16] (e.g., several recent dissertations are produced focusing on this topic [6, 15]) or hard deadlines [7, 8, 23, 22, 11, 1] have received much recent attention. Such works mostly focus on supporting on sporadic task systems on homogeneous multiprocessors, where the traditional sporadic task model is a basic recurrent task model that is much simpler than the dataflow task model studied in this paper (both models will be specifically defined in later sections).

In recent works [25, 17], a release-enforcer technique has been proposed to schedule DAG-based tasks on both homogeneous [17] and heterogeneous multiprocessors [25], showing that no utilization loss can be achieved under GEDF scheduling. Specifically, the release-enforcer technique forces a DAG-based task to release jobs in a sporadic manner through arbitrarily delaying jobs' releases, thus eliminating the need of directly handling DAG-induced precedence constraints. A DAG-based task set can thus be transformed to an ordinary sporadic task set. Unfortunately, this technique forces any job's release to be delayed to the worst-case completion time of any of its predecessor jobs, thus causing a rather pessimisitic response time bound. Fundamentally differeant from this technique, this paper presents a new set of analysis techniques that *directly* analyze jobs' response time on its original GEDF schedule, without any artificial delay of jobs' releases. The resulting response time bounds are thus much improved (Sec. 7 shows the advantage of our proposed techniques over applying the existing technique [25] in terms of respones time bounds).

3 The Dataflow Task Model

In this section, we introduce the studied dataflow task system model, which is directly motivated by the workloads seen in a couple of industrial cellular network systems.

We consider the problem of scheduling n dataflow tasks on a heterogeneous computing platform consisting of $m > 1$ types of processors, where each type of processor is designated to execute a specific functionality. Let M^k denote the number of processors of type k ($1 \leq k \leq m$). Each dataflow task is specified as a 2-tuple (G_i, P_i), where G_i is a chain of m subtasks, and P_i is a positive real number. We discuss these parameters below.

- Since a dataflow task may continuously generate data streams that need to be processed (i.e., "jobs"), the period P_i denotes the minimum amount of time that must elapse between the releases of successive jobs of τ_i. That is, if a job is released at time t, then the next

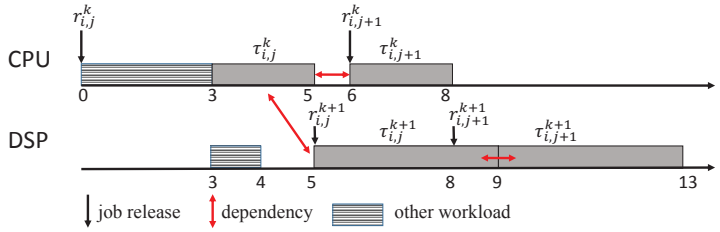


Figure 2 Two kinds of dependencies.

job of τ_i may not be released prior to the time instance $t + P_i$. Let $\tau_{i,j}$ denote the j^{th} job released by τ_i and $r_{i,j}$ denote the release time of $\tau_{i,j}$.

- The chain of subtasks G_i is specified as $\tau_i^1, ..., \tau_i^m$, where τ_i^k denotes the k^{th} subtask of dataflow task τ_i. Subtask τ_i^k is designated to be executed on processors of type k. Any j^{th} job released by τ_i is thus composed by a chain of m subjobs, each of which belongs to the corresponding subtask. Any subjob $\tau_{i,j}^k$ is characterized by its corresponding subtask τ_i^k's worst-case execution time (WCET) e_i^k [20]. The edge connecting any two subtasks represents dependencies between the released subjobs. That is, any subjob $\tau_{i,j}^k$ (i.e., the j^{th} subjob belonging to subtask τ_i^k) must complete execution before subjob $\tau_{i,j}^{k+1}$ can begin execution. Moreover, any subjob $\tau_{i,j}^k$ must complete execution before subjob $\tau_{i,j+1}^k$ can begin execution, which is the next released subjob belonging to the same subtask. These two types of dependencies among subjobs are illustrated in Fig. 2 and by the following example,

▶ **Example 1.** The above dataflow task model studied in this paper is mainly motivated by many industrial applications seen in practice. Consider the video stream application introduced in Fig. 1 as an example to illustrate the Dataflow Task Model. A video stream task τ_i periodically generates frames at a rate of 24 FPS, thus $P_i = \frac{1}{24}s$. Any j^{th} job (i.e., j^{th} frame in this example) $\tau_{i,j}$ needs to be processed using two functionalities, frame compression and frame decoding, which corresponds to the two subtasks of τ_i that are designated to be processed on two types of computing resources, CPUs and DSPs respectively. Thus $G_i = (\tau_i^1, \tau_i^2)$. As illustrated in Fig. 2, there are two types of dependencies among subjobs: (i) There is dependency between any two subjobs $\tau_{i,j}^k$ and $\tau_{i,j}^{k+1}$. This is because for each frame, the functionality of frame decoding can only happen after the same frame has been compressed. (ii) There is dependency between any two subjobs $\tau_{i,j}^k$ and $\tau_{i,j+1}^k$ (or $\tau_{i,j}^{k+1}$ and $\tau_{i,j+1}^{k+1}$). This is because in terms of each functionality, the video frames must be processed in sequential order since they are captured sequentially.

In order to apply the EDF[1] scheduling algorithm, we assign a deadline parameter D_i to each dataflow task τ_i, where $D_i = P_i$. Thus, any job $\tau_{i,j}$ has a deadline at time $r_{i,j} + D_i$. Under EDF, jobs are prioritized by deadlines, where jobs with shorter deadlines have higher priorities. Note that we apply EDF scheduling on all types of processors. Thus, all subjobs of any job $\tau_{i,j}$ inherit the priority of $\tau_{i,j}$, regardless of the type of processor on which they are executed. We allow job preemption and migration (but only migrating to a processor of the correct type), i.e., a higher-priority job that requires type-k processors may preempt

[1] We have verified that both preemption and the dynamic job-level priority-based EDF scheduler can possibly be implemented in a couple of industrial dataflow-based cellular network systems seen in practice.

the execution of a lower-priority job on a type-k processor and this lower-priority job may migrate to another type-k processor if such a processor is available.[2]

The utilization of τ_i on type-k processors is denoted u_i^k (i.e., the utilization of subtask τ_i^k), which is given by $\frac{e_i^k}{P_i}$, and we require

$$u_i^k \leq 1; \tag{1}$$

for otherwise the response time of τ_i^k may grow unboundedly. Since a task periodically releases jobs, the concept of task utilization is important as it characterizes the percentage of a single processor's capacity a task (or a subtask) requires in the long term.

The total utilization of a task system τ on type-k processors is defined as $U_{sum}^k(\tau) = \sum_{i=1}^{n} u_i^k$. We place no constraint on total utilization except that

$$U_{sum}^k(T) \leq M^k. \tag{2}$$

A scheduling algorithm is considered to be optimal if it can schedule any task system τ that satisfies Eq. (2) with bounded response times guaranteed for all tasks in τ.

The goal of this paper is to derive response time bounds for a dataflow task system scheduled under EDF. The response time of a task is defined to be the maximum response time of any of its released jobs. A job's response time is defined to be $f_{i,j} - r_{i,j}$, where $f_{i,j}$ denotes the time at which $\tau_{i,j}$ completes and $r_{i,j}$ is the release time of $\tau_{i,j}$. Instead of deriving the response time bound directly, we derive a tardiness bound for any dataflow task scheduled under EDF. The tardiness of a job $\tau_{i,j}$ is defined to be $\max\{0, f_{i,j} - d_{i,j}\}$, where $d_{i,j}$ is the deadline of $\tau_{i,j}$. The tardiness of a task is defined to be the maximum tardiness of any of its released jobs. Since $d_{i,j} = r_{i,j} + P_i$, the response time bound of any task τ_i can be obtained by simply using the derived tardiness bound of the task plus a P_i value.

4 Analysis Overview

Our goal now is to derive a tardiness bound for each dataflow task τ_i scheduled under EDF. Since a job released by any dataflow task consists of a chain of sub-tasks, instead of directly bounding the tardiness of each task, our analysis technique seeks to (i) bound the tardiness of any first subtask τ_i^1 (Sec. 4.1), (ii) bound the tardiness of any second subtask τ_i^2 (Sec. 5), and (iii) finally bound the tardiness of any k^{th} ($k > 2$) subtask τ_i^k based on the tardiness bound derived for τ_i^{k-1} (Sec. 6). In the following, we first bound the tardiness of any τ_i^1 as it is quite straightforward, and then describe the challenges in bounding the tardiness for other subtasks.

4.1 Tardiness Bound on Type 1 Processors

According to our dataflow task model, every dataflow task τ_i releases its jobs periodically with a minimum job inter-arrival gap of P_i, and any subjob $\tau_{i,j+1}^1$ of job $\tau_{i,j+1}$ can start execution at time t if $r_{i,j+1} \leq t$ and subjob $\tau_{i,j}^1$ completes by t. Thus, the first subtask τ_i^1 of each task τ_i scheduled on type-1 processors can be viewed as a sporadic task[3] scheduled on a

[2] Note that, although in many hardware accelerators frequent job preemptions are allowed but discouraged due to overhead consideration, we allow preemptions in this paper as this is the first attempt resolving this scheduling problem. We leave the further issue of enforcing non-preemptive executions on certain types of processors as future work.

[3] Under the sporadic task model, each sporadic task continuously releases jobs and can be specified by (e_i, P_i), where e_i denotes the WCET of any released job and p_i specified the minimum amount of time that must elapse between the releases of successive jobs of this task.

homogeneous multiprocessor with M^1 processors: the period and WCET of each task τ_i^1 are P_i and e_i^1, respectively.

As reviewed earlier, Devi and Anderson [6] have shown that EDF scheduling is capable of scheduling sporadic task sets with probably bounded tardiness while incurring no capacity loss, and the tardiness bound can be calculated using the following closed-form expression:

$$Tardiness(\tau_i^1) = \frac{\sum_{\tau_i^1 \in \varepsilon_{max}(\tau^1, M^1-1)} e_i^1 - e_{min}^1}{M^1 - \sum_{\tau_i^1 \in \mathcal{U}_{max}(\tau^1, M^1-1)} u_i^1} + e_i^1, \tag{3}$$

where $\mathcal{U}_{max}(\tau^1, M^1-1)$ ($\varepsilon_{max}(\tau^1, M^1-1)$, respectively) denotes a subset of (M^1-1) tasks with the highest utilization (largest WCET, respectively) in τ^1 where τ^1 denotes the set of the first subtasks of all dataflow tasks. To ease our description, in the rest of this paper, we use TB_i^k to denote tardiness bound of subtask τ_i^k which is scheduled on type-k processors.

4.2 Challenges on Deriving Tardiness Bound for Any Subtask τ_i^k ($k > 1$)

It is straightforward to derive a tardiness bound for any subtask τ_i^1 because such subtasks can be naturally modeled as sporadic tasks, thus allowing existing analysis to be directly applied. However, since any subjob $\tau_{i,j}^k$ ($k > 1$) cannot start execution until $\tau_{i,j}^{k-1}$ completes, the releasing pattern of any such subtask τ_i^k is hard to characterize. Note that due to dependencies, subjob $\tau_{i,j}^k$ ($k > 1$) is said to be released when $\tau_{i,j}^{k+1}$ completes execution. That is, $r_{i,j}^k = f_{i,j}^{k+1}$, where $f_{i,j}^{k+1}$ denotes the completion time of subjob $\tau_{i,j}^{k+1}$. A straightforward approach is to force such subtasks to be modeled as sporadic tasks by calculating the minimum separation between any two consecutive subjob releases by τ_i^k. However, we use an example illustrating that such an approach would result in significant pessimism and could be invalid in many cases.

▶ **Example 2.** Fig. 2 shows an example to illustrate the pessimism induced by ordinary sporadic model. $\tau_{i,j}^k$ is released at 0 and completes execution at 5, when $\tau_{i,j}^{k+1}$ is released. $\tau_{i,j+1}^k$ is released at 6 and completes execution at 8, when $\tau_{i,j+1}^{k+1}$ is released. Thus, the time interval between the release time of these two consecutive subjobs of τ_i^{k+1} is 3 time units. The minimum time interval between any two consecutive subjob releases of τ_i^{k+1} is thus 3 time units. However, since $e_i^{k+1} = 4$ in this example, the utilization of τ_i^{k+1} would be forced to become $4/3 > 1$, which implies that τ_i^{k+1} is unschedulable and thus the whole task system is unschedulable.

As seen in the above example, intuitively, the approach of forcing any subtask τ_i^2 does not work due to dependencies. The minimum time separation between any two consecutive subjob releases by τ_i^2 could be too small due to the facts that any subjob $\tau_{i,j}^2$ is released at the time when $\tau_{i,j}^1$ completes and $\tau_{i,j}^1$ may complete later than its deadline due to tardiness. This may dramatically increase τ_i^2's utilization in the analysis and thus artificially create more workloads due to τ_i^2 than it actually contributes, which is too pessimistic and may even make the resulting schedulability test invalid. To resolve this challenge, we next present a set of novel analysis techniques to bound any subtask τ_i^2's tardiness by analyzing the actual workload contributed by each such subtask.

5 Tardiness Bound on Type 2 Processors

In this section, we derive the tardiness bound for any subtask τ_i^2 which is scheduled on type-2 processors. Our analysis techniques involve comparing the resource allocation to τ^2, which

denotes the set of all subtasks τ_i^2 in the system, in a processor sharing (PS) schedule (defined below) and an actual EDF schedule of interest for τ^2, both on M^2 type-2 processors, and quantifying the difference between the two. We analyze allocations on a per-subtask basis.

The time interval $[t_1, t_2)$, where $t_2 > t_1$, consists of all time instances t, where $t_1 \leq t < t_2$, and is of length $t_2 - t_1$. For any time $t > 0$, the notation τ^- denotes the time $t - \epsilon$ in the limit $\epsilon- > 0+$, and the notation t^+ is used to denote the time $t + \epsilon$ in the limit $\epsilon- > 0+$.

▶ **Definition 3** (active subjob)**.** A subtask τ_i^2 is active at time t in a schedule $\mathcal{S}$ if there exists a job $\tau_{i,j}^2$ (called τ_i^2's active job at t) such that $f_{i,j}^1 \leq t < d_{i,j}$ (as defined earlier, $f_{i,j}^1$ denotes the completion time of subjob $\tau_{i,j}^1$ as well as the time $\tau_{i,j}^2$ is released). By our task model, every subtask has at most one active subjob at any time.

▶ **Definition 4** (pending subjob)**.** A subjob $\tau_{i,j}^2$ is pending at time t in a schedule $\mathcal{S}$ if $f_{i,j}^1 \leq t$, and $\tau_{i,j}^2$ has not completed execution by t in $\mathcal{S}$. A subtask is pending at time t if any of its subjobs are pending at time t.

▶ **Definition 5** (ready subjob)**.** A pending subjob $T_{i,j}^2$ is ready at time t in a schedule $\mathcal{S}$ if $f_{i,j}^1 \leq t$, $T_{i,j}^2$ has not yet completed at t, and all its predecessor subjobs (i.e., $\tau_{i,j}^1$ and $\tau_{i,j-1}^2$) have completed execution by t in $\mathcal{S}$.

Let $A(\tau_{i,j}^2, t_1, t_2, \mathcal{S})$ denote the total processing time allocated to $\tau_{i,j}^2$ in an arbitrary schedule $\mathcal{S}$ in $[t_1, t_2)$. Then, the total time allocated to all jobs of τ_i^2 in S is given by

$$A(\tau_i^2, t_1, t_2, \mathcal{S}) = \sum_{j \geq 1} A(\tau_{i,j}^2, t_1, t_2, \mathcal{S}) \tag{4}$$

Now consider a PS schedule PS defined below.

▶ **Definition 6** (processor sharing)**.** PS is a processor-sharing (PS) schedule on M^2 type-2 processors for all subtasks in τ^2. In such a schedule, τ_i^2 executes with the rate u_i^2 when it is active.

Thus, if τ_i^2 is active throughout $[t_1, t_2)$, then

$$A(\tau_i^2, t_1, t_2, PS) = (t_2 - t_1) \cdot u_i^2. \tag{5}$$

The difference between the allocation to a subjob $\tau_{i,j}^2$ up to time t in a PS schedule and an arbitrary schedule S, denoted *the lag of subjob* $\tau_{i,j}^2$ *in schedule* S, is defined by

$$lag(\tau_{i,j}^2, t, S) = A(\tau_i^2, 0, t, PS) - A(\tau_i^2, 0, t, S). \tag{6}$$

The concept of lag is important because, if lags remain bounded, then tardiness is bounded as well. In order to bound lags for subjobs of subtasks $\tau_i^2 \in \tau^2$ on type-2 processors, PS plays a pivotal role. Intuitively, we can imagine that PS represent the schedule of a system of n type-2 processors, where the i^{th} processor has a capacity of $\frac{e_i^2}{P_i}$ and is dedicated to executing subjobs of τ_i^2. Thus, we can see that in a PS schedule subjobs released by different subtaskτ_i^2 will not interfere with each other and subjobs released by the same subtask execute at a constant rate sequentially.

The concept of PS is originally introduced in [6] for sporadic task systems. For the sporadic case, PS represents an "ideal" schedule since each job released at time t is expected to complete at its deadline, which is $t + P_i$. Thus, bounding the lag can be achieved through comparing the allocation to an arbitrary schedule S and the allocation in PS. However, a key difference between the sporadic task model and our dataflow task model is that each

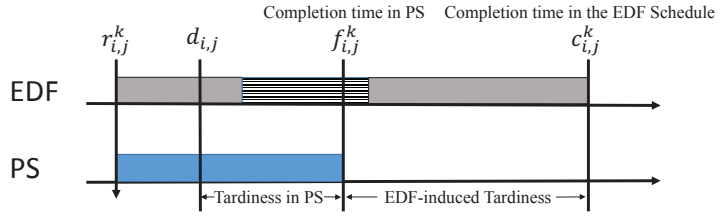


Figure 3 $\tau_{i,j}^k$'s tardiness is comprised of its tardiness in PS and the EDF-induced tardiness.

subjob has two types of dependencies. Due to dependencies, a subjob may not be able to ideally complete by its deadline in PS. Specifically, due to the facts that the release time $r_{i,j}^2$ of any subjob $\tau_{i,j}^2$ is the completion time $f_{i,j}^1$ of the subtask $\tau_{i,j}^1$ executed on type-1 processors ($f_{i,j}^1 > d_{i,j}$ is possible due to tardiness according to Eq. 3), and $\tau_{i,j}^1$ and $\tau_{i,j}^2$ have the same deadline $d_{i,j}$, $\tau_{i,j}^2$ thus may not complete by its deadline at $d_{i,j}$ in PS.

The above discussions highlight a key point that under the dataflow task model, the tardiness of τ_i^2 is due to two sources: (i) the tardiness due to using an arbitrary scheduler which can be less ideal than the PS scheduler (e.g., EDF in this paper), which can be bounded by comparing the allocation to τ_i^2 in the corresponding arbitrary schedule against the allocation in PS, and (ii) the tardiness of τ_i^2 seen in PS, which exists due to the potential late completion of subjobs of τ_i^1 (i.e., its subjobs complete after the corresponding deadlines) executed on type-1 processors (note again that this PS-induced tardiness does not exist for the ordinary sporadic task model.) This is also illustrated in Fig. 3.

Motivated by this key observation, we seek to bound the tardiness of τ_i^2 by bound two types of tardiness: PS-induced tardiness (Sec. 5.1) and EDF-induced tardiness (Sec. 5.2). Finally we obtain a tardiness bound for any τ_i^2 by combining these two types of tardiness (Sec. 5.3).

5.1 *PS*-induced Tardiness

In this section, we derive a bound on the PS-induced tardiness for τ_i^2; in particular, the following theorem shows that the tardiness of any subjob of τ_i^2 in PS can be bounded by a constant that depends only on the dataflow task τ_i's period and and the tardiness bound of τ_i^1 executed on type 1 processors.

▶ **Theorem 7.** *τ_i^2's tardiness in PS is bounded by $TB_i^1 + P_i$.*

Proof. For the first subjob $\tau_{i,1}^2$ of τ_i^2, its release time $r_{i,1}^2$ is no later than $r_{i,1}^1 + P_i + TB_i^1$. This is because the completion time of $\tau_{i,1}^1$ executed on type-1 processor is no later than $r_{i,1}^1 + P_i + TB_i^1$ according to Eq. 3. According to the definition of PS, we know that $\tau_{i,1}^2$ begins execution no later than $r_{i,1}^1 + P_i + TB_i^1$ and will complete no later than $r_{i,1}^1 + P_i + TB_i^1 + P_i$ in PS. Since the deadline of $\tau_{i,1}^2$ is $r_{i,1}^1 + P_i$, $\tau_{i,1}^2$'s tardiness is no greater than $TB_i^1 + P_i$ in PS.

We now prove this theorem's correctness for all the other jobs $\tau_{i,j}^2$ ($j > 1$) by contradiction. Let us assume that the tardiness of some job of τ_i^2 is larger than $TB_i^1 + P_i$ in PS. Let $\tau_{i,x}^2$ denote the first such job.

According to the definition of PS, all subjobs of τ_i^2 execute sequentially on type-2 processors at a constant rate $\frac{e_i^2}{P_i}$. Since the tardiness of $\tau_{i,x}^2$ in PS is greater than $TB_i^1 + P_i$, $\tau_{i,x}^2$ must begin execution after $r_{i,x}^1 + P_i + TB_i^1$. Note that $r_{i,x}^1$, which is one predecessor job of $\tau_{i,x}^2$, must complete by $r_{i,x}^1 + P_i + TB_i^1$ according to Eq. 3. Thus, this implies that $\tau_{i,x-1}^2$

does not complete at $r^1_{i,x} + P_i + TB^1_i$. Since the deadline of $\tau^2_{i,x-1}$ is $r^1_{i,x-1} + P_i$, the tardiness of $\tau^2_{i,x-1}$ in PS is greater than $(r^1_{i,x} + P_i + TB^1_i) - (r^1_{i,x-1} + P_i) = r^1_{i,x} + TB^1_i - r^1_{i,x-1}$. Since $r^1_{i,x} - r^1_{i,x-1} \geq P_i$, the tardiness of $\tau^2_{i,x-1}$ in PS is greater than $TB^1_i) + P_i$. This implies that $\tau^2_{i,x}$ is not the first subjob of τ^2_i that has a tardiness greater than $TB^1_i) + P_i$. A contradiction is reached and the theorem is thus proved. ◀

5.2 EDF-induced Tardiness

In this section, we bound EDF-induced tardiness for subtask τ^2_i by comparing any subjob $\tau^2_{i,j}$'s completion time under EDF and its completion time under PS. Since we only focus on subtasks and subjobs that are scheduled on type-2 processors in this section, to make the description easier, we call τ^2_i a task and $\tau^2_{i,j}$ a job in this section.

First we quantify the allocation to job $\tau^2_{i,j}$ in an interval $[t_1, t_2)$ in PS. For this, we have defined $A(S, \tau^2_{i,j}, t_1, t_2)$ to denote the total processing time allocated to $\tau^2_{i,j}$ in an arbitrary schedule $\mathcal{S}$ in $[t_1, t_2)$ in Eq. 4 and we have $A(\tau^2_{i,j}, t_1, t_2, PS) \leq (t_2 - t_1) \cdot u^2_i$ according to Eq. 5.

We relate the allocation to a job $\tau^2_{i,j}$ under PS to its allocation under EDF using the notation of lag, which is defined as

$$lag(\tau^2_{i,j}, t, EDF) = A(\tau^2_{i,j}, 0, t, PS) - A(\tau^2_{i,j}, 0, t, EDF). \tag{7}$$

Then, task lags can be defined in a similar way:

$$lag(\tau^2_i, t, EDF) = \sum_{j \geq 1} (A(\tau^2_{i,j}, 0, t, PS) - A(\tau^2_{i,j}, 0, t, EDF)). \tag{8}$$

The lag of task τ^2_i at t in EDF schedule indicates the difference between the allocation to τ^2_i in EDF and PS in interval $[0, t)$. If $lag(\tau^2_i, t, EDF)$ is positive, then schedule EDF has performed less work on the jobs of τ^2_i until t than PS, and more work if $lag(\tau^2_i, t, EDF)$ is negative.

The total lag for a finite job set Θ at t, denoted $LAG(\Theta, t, EDF)$, is given by the sum of the lags of all jobs in Θ. That is,

$$\begin{aligned} LAG(\Theta, t, EDF) &= \sum_{\tau^2_{i,j} \in \Theta} lag(\tau^2_{i,j}, t, EDF) \\ &= \sum_{\tau^2_{i,j} \in \Theta} (A(\tau^2_{i,j}, 0, t, PS) - A(\tau^2_{i,j}, 0, t, EDF)). \end{aligned} \tag{9}$$

Since both $A(\tau^2_{i,j}, 0, 0, PS)$ and $A(\tau^2_{i,j}, 0, 0, EDF)$ are zero, $\forall\, i$ and $\forall\, j$, $LAG(\Theta, 0, EDF)$ is zero. By Eq. 7 and Eq. 9, we have the following for $t_2 > t_1$:

$$\begin{aligned} LAG(\Theta, t_2, EDF) = LAG(\Theta, t_1, EDF) \\ + A(\Theta, t_1, t_2, PS) - A(\Theta, t_1, t_2, EDF). \end{aligned} \tag{10}$$

▶ **Definition 8** (busy/none-busy interval)**.** A time interval $[t_1, t_2)$ is busy for a job set Θ, if, at each time-instant $t \in [t_1, t_2)$, all processors executes jobs from Θ, and is non-busy for Θ otherwise. An interval $[t_1, t_2)$ is maximally non-busy for Θ if it is non-busy for Θ at every time instant within it and either $t_1 = 0$ or t_1^- is a busy instant for Θ.

If $[t_1, t_2)$ is a busy interval in an EDF schedule for Θ, then the tasks in Θ receive a total allocation of $M^2(t_2 - t_1)$ time in that interval in an EDF schedule. By Eq. 8, the total allocation to Θ cannot exceed $M^2(t_2 - t_1)$ in PS. Thus, by Eq. 9, the LAG of Θ at t_2 is no larger than that at t_1, and the following lemma holds.

▶ **Lemma 9.** *For any time interval* $[t_1, t_2)$ *that is busy for* Θ, $LAG(\Theta, t_2, EDF) \leq LAG(\Theta, t_1, EDF)$.

Lemma 9 implies that the total lag for a job set can only increase across non-busy intervals, which causes tardiness of jobs. Next, we bound EDF-induced tardiness on type 2 processors using lags. For the rest of this section, let τ^2 denote the task system scheduled on type 2 processors. τ^2 has a PS schedule and satisfies Eq. 1 and Eq. 2. All jobs $\tau^2_{i,j} \in \tau^2$ have the following information:

- release time $r^2_{i,j}$
- execution time e^2_i
- completion time $f^2_{i,j}$ for $\tau^2_{i,j}$ in PS

▶ **Definition 10.** $\hat{f}^2_{i,j} = \max\{d_{i,j}, f^2_{i,j}\}$ denotes the earliest time at or after $d_{i,j}$ by which $\tau^2_{i,j}$ has completed in PS.
Note that $\hat{f}^2_{i,j} \geq f^2_{i,j}$ and job $\tau^2_{i,j}$'s tardiness in PS equals $\hat{f}^2_{i,j} - d_{i,j}$.

▶ **Definition 11.** Let $c^2_{i,j}$ denotes the completion time of $\tau^2_{i,j}$ in the EDF schedule. **EDF-induced tardiness** equals $\max\{c^2_{i,j} - \hat{f}^2_{i,j}, 0\}$, as illustrated earlier in Fig. 3.

In the rest of this section, we bound EDF-induced tardiness by leveraging and extending the general analysis framework first developed in [6]. Let

$$\rho = \max_{\tau^1_i \in \tau^1} \{TB^1_i\}. \tag{11}$$

ρ denotes the largest tardiness bound among all subtasks τ^1_i that ae scheduled on type-1 processors. We assume that the EDF schedule has the following property.

▶ **Property 12.** *The EDF-induced tardiness of every job of every task* τ^2_k *in* τ^2 *with deadline less than* $d_{i,j}$ *is at most* $x + e^2_k$, *where* $x \geq \rho$.

Our goal is to determine the smallest $x \geq \rho$ such that the tardiness of $\tau^2_{i,j}$ remains at most $x + e^2_i$. Such a result would by induction imply a tardiness of at most $x + e^2_k$ for all jobs of every task $\tau^2_k \in \tau^2$. Because τ^2 is arbitrary, the tardiness bound will hold for every concrete instantiation of τ^2. The objective is easily met if $\tau^2_{i,j}$ completes by its deadline, $d_{i,j}$, so assume otherwise. The completion time of $\tau^2_{i,j}$ then depends on the amount of work that can compete with $\tau^2_{i,j}$ after $d_{i,j}$. Hence, a value for x can be determined via the following steps.

1. Compute an upper bound on pending work for tasks in τ^2 (including $\tau^2_{i,j}$) that can compete with $\tau^2_{i,j}$ after $d_{i,j}$.
2. Determine the amount of such work necessary for the tardiness of $\tau^2_{i,j}$ to exceed $x + e^2_i$.
3. Determine the smallest $x \geq \rho$ such that the tardiness of $\tau^2_{i,j}$ is at most $x + e^2_i$ using the upper bound in Step 1 and the necessary condition in Step 2.

To reason about the tardiness of $\tau^2_{i,j}$ we need to determine how other jobs preempt/delay its execution. We classify such jobs based on the relation between their deadlines and completion time on PS and those of $\tau^2_{i,j}$, as follows.

- $\mathbf{fd} = \{\tau^2_{l,k} : d_{l,k} \leq d_{i,j} \wedge f^2_{l,k} \leq \hat{f}^2_{i,j}\}$
- $\mathbf{fD} = \{\tau^2_{l,k} : d_{l,k} > d_{i,j} \wedge f^2_{l,k} \leq \hat{f}^2_{i,j}\}$
- $\mathbf{Fd} = \{\tau^2_{l,k} : d_{l,k} \leq d_{i,j} \wedge f^2_{l,k} > \hat{f}^2_{i,j}\}$
- $\mathbf{FD} = \{\tau^2_{l,k} : d_{l,k} > d_{i,j} \wedge f^2_{l,k} > \hat{f}^2_{i,j}\}$

In this notation, f and F denote jobs' completion time in PS at most and greater than $\hat{f}_{i,j}$, respectively. d denotes that $\tau^2_{l,k}$'s deadline is no later than that of $\tau^2_{i,j}$, and D denotes that $\tau^2_{l,k}$'s deadline is later than that of $\tau^2_{i,j}$. Note that $\tau^2_{i,j} \in fd$.

The set of jobs with completion time in PS at most $\hat{f}^2_{i,j}$ is further referred to as $\Theta = fd \cup fD$. This set of jobs do not execute beyond $\hat{f}^2_{i,j}$ in the PS schedule. Note that because the jobs in $fd \cup Fd$ might have the priority at least that of $\tau^2_{i,j}$ (at some time instant), the execution of $\tau^2_{i,j}$ might be postponed (in the worst case) until there are at most m ready jobs in $fd \cup Fd$ including $\tau^2_{i,j}$. Based on this observation, we derive the EDF-induced tardiness bound in three steps.

5.2.1 Step 1: An upper bound on competing work

In this section, we determine an upper bound on competing work for $\tau^2_{i,j}$, which is denoted by $UB(fd \cup Fd, \hat{f}_{i,j}, EDF)$.

Because jobs in $fd \cup Fd$ can have priorities at least that of $\tau^2_{i,j}$, the competing work due to $fd \cup Fd$ for $\tau^2_{i,j}$ beyond $\hat{f}^2_{i,j}$, $UB(fd \cup Fd, \hat{f}_{i,j}, EDF)$, is bounded by the sum of (i) the amount of work pending at $\hat{f}^2_{i,j}$ for jobs in fd, and (ii) the amount of work $W(Fd, \hat{f}^2_{i,j}, EDF)$ required by jobs in Fd that can compete with $\tau^2_{i,j}$ after $\hat{f}^2_{i,j}$. For the pending work mentioned in (i), because jobs from Θ have completion time in PS at most $\hat{f}^2_{i,j}$, they do not execute in the PS schedule beyond $\hat{f}^2_{i,j}$. Thus, the work pending for jobs in fd is given by $LAG(fd, \hat{f}^2_{i,j}, EDF)$, which must be positive in order for $\tau^2_{i,j}$ to exceed its completion time in PS at $\hat{f}_{i,j}$.

Instead of bounding $LAG(fd, \hat{f}^2_{i,j}, EDF)$, we try to bound $LAG(\Theta, \hat{f}^2_{i,j}, EDF)$. This is because $LAG(fd, \hat{f}^2_{i,j}, EDF) \leq LAG(\Theta, \hat{f}^2_{i,j}, EDF)$. This inequality holds because $\Theta = fd \cup fD$, and $LAG(fD, \hat{f}^2_{i,j}, EDF)$ is non-negative because according to the definition of fD, the jobs in fD cannot perform more work by time $\hat{f}^2_{i,j}$ in EDF than they have performed in the PS schedule. Thus, $W(fd \cup Fd, \hat{f}^2_{i,j}, EDF) \leq LAG(\Theta, \hat{f}^2_{i,j}, EDF) + W(Fd, \hat{f}^2_{i,j}, EDF)$. Therefore, $W(fd \cup Fd, \hat{f}^2_{i,j}, EDF)$ can be obtained by determining upper bounds for $LAG(\Theta, \hat{f}^2_{i,j}, EDF)$ and $W(Fd, \hat{f}^2_{i,j}, EDF)$ individually.

Upper bound on $LAG(\Theta, \hat{f}^2_{i,j}, EDF)$. Since we are deriving this bound w.r.t. Θ, we assume that all busy and non-busy intervals considered are w.r.t. Θ and the EDF schedule.

By Lemma 9, if no non-busy interval for Θ exists in $[0, \hat{f}^2_{i,j})$, then $LAG(\Theta, \hat{f}^2_{i,j}, EDF) \leq LAG(\Theta, 0, EDF) = 0$. Thus, we consider the more interesting case where some non-busy interval exists in $[0, \hat{f}^2_{i,j})$. An interval for jobs in Θ could be non-busy for two reasons:

- Simply not enough ready jobs in Θ exists that can occupy all available processors. Thus, it does not matter whether jobs from Fd or FD execute during the interval. Such an interval is called non-busy non-occupation.
- There exists ready jobs in Θ that cannot execute within some sub-intervals in $[0, \hat{f}^2_{i,j})$, since jobs in Fd occupy one or more processors and they have higher priorities. Such an interval is called non-busy occupation.

Let the carry-in job $\tau^2_{c,h}$ of a task τ^2_c be defined as the job, if any, for which $r^2_{c,h} \leq \hat{f}^2_{i,j} < f^2_{c,h}$ holds. Note that at most one such job could exist for each task τ^2_c and only such carry-in jobs can prevent the execution of jobs in Θ before time $\hat{f}^2_{i,j}$, thus increasing the LAG for Θ.

▶ **Definition 13.** Let τ^2_H be the set of tasks that have carry in jobs in Fd.

▶ **Definition 14.** Let δ_c be the amount of work performed by a carry-in job $\tau^2_{c,h}$ in EDF by time $\hat{f}^2_{i,j}$.

As discussed earlier, LAG for Θ can increase only across non-busy intervals. In much of the rest of the analysis, we focus on a time t_2 defined as follows: if there exists a non-busy non-occupation interval before $\hat{f}^2_{i,j}$, across which LAG for Θ increases, then let $[t_1, t_2]$ denote the latest such interval; otherwise, $t_1 = t_2 = 0$.

The following two lemmas have been proved previously in [6, 18] for ordinary sporadic task systems (note again that no PS-induced tardiness exists for sporadic task systems). Note that when we calculate EDF-induced tardiness, the value of $LAG(\Theta, \hat{f}^2_{i,j}, EDF) + W(FD, \hat{f}^2_{i,j}, EDF)$ depends only on allocations in the PS schedule and allocations to jobs in $\Theta \cup FD$ by time $\hat{f}^2_{i,j}$ in the EDF schedule, which can compete processing time with $\tau^2_{i,j}$ after $\hat{f}^2_{i,j}$. Thus, the tardiness in PS does not affect the derivation of the EDF-induced tardiness. Also, Property 12 alone is sufficient for determining how much work any job in $\Theta \cup FD$ other than $\tau^2_{i,j}$ completes before $\hat{f}^2_{i,j}$. For these reasons, Lemmas 15 and Lemma 16 continue to hold for τ^2 task systems.

▶ **Lemma 15.** $LAG(\Theta, \hat{f}^2_{i,j}, EDF) \leq LAG(\Theta, t_2, EDF) + \sum_{\tau^2_c \in \tau^2_H} \delta_c(1 - u^2_c)$.

Proof. By Eq. (9) and Eq. (10),

$$\begin{aligned} &LAG(\Theta, \hat{f}_{l,j}, EDF) \\ &\leq LAG(\Theta, t_2, EDF) + A(\Theta, t_2, \hat{f}_{l,j}, PS) \\ &- A(\Theta, t_2, \hat{f}_{l,j}, EDF). \end{aligned} \tag{12}$$

We split $[t_2, \hat{f}_{i,j}]$ into b non-overlaping intervals $[t_{p_i}, t_{q_i})$, where $1 \leq i \leq b$, such that $t_2 = t_{p_1}$, $t_{q_{(i-1)}} = t_{p_i}$ and $t_{q_b} = \hat{f}_{i,j}$. Each interval $[t_{p_i}, t_{q_i})$ is either busy, non-busy occupation or non-busy non-occupation. We assume that any occupation interval $[t_{p_i}, t_{q_i})$ is defined so that if a task $\tau^2_c \in \tau^2_H$ executes at some point in the interval, then it executes continuously throughout the interval. Note that such a task τ^2_c does not necessarily execute continuously throughout $[t_2, \hat{f}_{i,j})$. For each occupation interval $[t_{p_i}, t_{q_i})$, we define a subset of tasks $\alpha^2_c \subseteq \tau^2_H$ that execute continuously throughout $[t_{p_i}, t_{q_i})$. The allocation difference for Θ throughout the interval $[t_2, \hat{f}^2_{i,j})$ is thus:

$$\begin{aligned} &A(\Theta, t_2, \hat{f}^2_{l,j}, PS) - A(\Theta, t_2, \hat{f}^2_{l,j}, EDF) \\ &= \sum_{i=1}^{b} (A(\Theta, t_{p_i}, t_{q_i}, PS) - A(\Theta, t_{p_i}, t_{q_i}, EDF)). \end{aligned} \tag{13}$$

We now bound the difference between the work performed in the PS schedule and the schedule EDF across each of these intervals $[t_{p_i}, t_{q_i})$. The sum of these bounds gives us a bound on the total allocation difference throughout $[t_2, \hat{f}^2_{i,j})$. Depending on the nature of the interval $[t_{p_i}, t_{q_i})$, three cases are possible.

Case 1. $[t_{p_i}, t_{q_i})$ **is busy.** Because in EDF all processors are occupied by jobs in Θ, $A(\Theta, t_{p_i}, t_{q_i}, EDF) = M^2 \times (t_{q_1} - t_{p_i})$. In PS, $A(\Theta, t_{p_i}, t_{q_i}, PS) \leq U^2_{sum}(t_{q_1} - t_{p_i})$. Since $U^2_{sum} \leq M^2$, we have

$$A(\Theta, t_{p_i}, t_{q_i}, PS) - A(\Theta, t_{p_i}, t_{q_i}, EDF) \leq 0. \tag{14}$$

Case 2. $[t_{p_i}, t_{q_i})$ **is non-busy non-occupation.** By the selection of $[t_1, t_2]$, LAG does not increase for Θ across $[t_{p_i}, t_{q_i})$. Therefore, from Eq. (10), we have

$$A(\Theta, t_{p_i}, t_{q_i}, PS) - A(\Theta, t_{p_i}, t_{q_i}, EDF) \leq 0. \tag{15}$$

Case 3. $[t_{p_i}, t_{q_i})$ **is non-busy occupation.** The cumulative utilization of all tasks $\tau^2_c \in \alpha^2_c$, which execute continuously throughout $[t_{p_i}, t_{q_i})$, is $\sum_{\tau^2_c \in \alpha^2_c} u_c$. The carry-in jobs of these tasks do not belong to Θ, by the definition of Θ. Therefore, the allocation of jobs

in Θ during $[t_{p_i}, t_{q_i})$ in PS is at most $A(\Theta, t_{p_i}, t_{q_i}, PS) = (t_{q_i} - t_{p_i})(M^2 - \sum_{\tau_c^2 \in \alpha_c^2} u_c)$. All processors are occupied at every time instant in the interval $[t_{p_i}, t_{q_i})$, because it is occupation. Thus, $A(\Theta, t_{p_i}, t_{q_i}, EDF) = (t_{q_i} - t_{p_i})(M^2 - |\alpha_c^2|)$. Therefore, the allocation difference for jobs in Θ throughout the interval is

$$\begin{aligned} & A(\Theta, t_{p_i}, t_{q_i}, PS) - A(\Theta, t_{p_i}, t_{q_i}, EDF) \\ & \leq (t_{q_i} - t_{p_i})((M^2 - \sum_{\tau_c^2 \in \alpha_c^2} u_c^2) - (M^2 - \mid \alpha_c^2 \mid)) \\ & = (t_{q_i} - t_{p_i}) \sum_{\tau_c^2 \in \alpha_c^2} (1 - u_c^2)). \end{aligned} \tag{16}$$

To complete our proof, define $\alpha_c^2 =$ null for all intervals $[t_{p_i}, t_{q_i})$ that are either busy or non-busy non-occupation. Then, summing the allocation differences for all the intervals $[t_{p_i}, t_{q_i})$ given by Eq. (14), Eq. (15), and Eq. (16), we have

$$\begin{aligned} & A(\Theta, t_{p_i}, t_{q_i}, PS) - A(\Theta, t_{p_i}, t_{q_i}, EDF) \\ & \leq \sum_{i=1}^{b} \sum_{\tau_c^2 \in \alpha_c^2} (t_{q_i} - t_{p_i})(1 - u_c^2)). \end{aligned} \tag{17}$$

For each task $\tau_c^2 \in \tau_H^2$, the sum of the lengths of the intervals $[t_{p_i}, t_{q_i})$, in which the carry-in job of τ_c^2 executes continuously is at most δ_c^2. $A(\Theta, t_{p_i}, t_{q_i}, PS) - A(\Theta, t_{p_i}, t_{q_i}, EDF) \leq \sum_{\tau_c^2 \in \tau_H^2} \delta_c^2(1-u_c^2)$. Setting this value into Eq. (12), we get $LAG(\Theta, \hat{f}_{i,j}, EDF) \leq LAG(\Theta, t_2, EDF) + \sum_{\tau_c^2 \in \tau_H^2} \delta_c^2(1 - u_c^2)$. ◀

▶ **Lemma 16.** *$lag(\tau_k^2, t, EDF) \leq x \times u_k^2 + e_k^2$ for any task τ_k^2 and $t \in [0, \hat{f}_{i,j}^2]$.*

Proof. Let $\hat{f}_{k,j}^2$ be the completion time of the earliest pending job of τ_k^2, $\tau_{k,j}^2$, in the PS schedule at time t. Let γ_k be the amount of work $\tau_{k,j}^2$ performs before t in the EDF schedule. We first prove the lemma for the case $\hat{f}_{k,j}^2 < t$. By Eq. (7) and the selection of $\tau_{k,j}^2$,

$$\begin{aligned} & lag(\tau_k^2, t, EDF) \\ & = \sum_{h \geq j} lag(\tau_{k,h}^2, t, EDF) \\ & = \sum_{h \geq j} (A(\tau_{k,h}^2, 0, t, PS) - A(\tau_{k,h}^2, 0, t, EDF)) \end{aligned} \tag{18}$$

$A(\tau_{k,h}^2, 0, t, EDF) = A(\tau_{k,h}^2, r_{k,h}^2, t, EDF)$, because no job executes before its release time. Thus,

$$\begin{aligned} & lag(\tau_k^2, t, EDF) \\ & = \sum_{h > j} (A(\tau_{k,h}^2, r_{k,h}^2, t, PS) - A(\tau_{k,h}^2, r_{k,h}^2, t, EDF)) \\ & + A(\tau_{k,j}^2, r_{k,j}^2, t, PS) - A(\tau_{k,j}^2, r_{k,j}^2, t, EDF). \end{aligned} \tag{19}$$

$A(\tau_{k,j}^2, r_{k,j}^2, t, PS) = e_k^2$ and $\sum_{h>j} A(\tau_{k,h}^2, r_{k,h}^2, t, PS) \leq u_k^2(t - \hat{f}_{k,j}^2)$, by the definition of PS. $A(\tau_{k,j}^2, r_{k,j}^2, t, EDF) = \gamma_k$ and $\sum_{h>j} A(\tau_{k,h}^2, r_{k,h}^2, t, EDF) = 0$, by the selection of $\tau_{k,j}^2$. Setting these values into Eq. (19), we have

$$lag(\tau_k^2, t, EDF) \leq u_k^2(t - \hat{f}_{k,j}^2) + e_k^2 - \gamma_k \tag{20}$$

By Property (P), $\tau^2_{k,j}$ has EDF-induced tardiness at most $x+e^2_k$, so $t+e^2_k-\gamma_k \le \hat{f}^2_{k,j}+x+e^2_k$. Thus, $t-\hat{f}^2_{k,j} \le x+e^2_k+\gamma_k-e^2_k$. From 20, we have

$$lag(\tau^2_k,t,EDF) \le u^2_k(t-\hat{f}^2_{k,j})+e^2_k-\gamma_k = u^2_k(x+e^2_k+\gamma_k-e^2_k)+e^2_k-\gamma_k$$
$$\le x\times u^2_k+e^2_k.$$

Then, we prove the lemma for the case $\hat{f}^2_{k,j} \ge t$. By Eq. (7) and the selection of $\tau^2_{k,j}$,

$$lag(\tau^2_k,t,EDF) = \sum_{h\le j} lag(\tau^2_{k,h},t,EDF) = \sum_{h\le j}(A(\tau^2_{k,h},0,t,PS)-A(\tau^2_{k,h},0,t,EDF))$$
$$= \sum_{h<j}(A(\tau^2_{k,h},r^2_{k,h},t,PS)-A(\tau^2_{k,h},r^2_{k,h},t,EDF))$$
$$+A(\tau^2_{k,j},r^2_{k,j},t,PS)-A(\tau^2_{k,j},r^2_{k,j},t,EDF). \quad (21)$$

By the definition of PS, $\sum_{h<j} A(\tau^2_{k,h},r^2_{k,h},t,PS) \le \sum_{h<j} e^2_k$; since $\tau_{k,j}$ is the earliest pending job of τ^2_k in the schedule EDF at time t, $\sum_{h<j} A(\tau^2_{k,h},r_{k,h},t,EDF) = \sum_{h<j} e^2_k$. Also, $A(\tau^2_{k,j},r^2_{k,j},t,PS) \le e^2_k$ and $A(\tau^2_{k,j},r^2_{k,j},t,EDF) = \gamma_k \ge 0$, and setting these values into Eq. (21):

$$lag(\tau^2_k,t,EDF) \le (\sum_{h<j} e^2_k - \sum_{h<j} e^2_k)+e^2_k-\gamma_k \le e^2_k$$

Therefore, $lag(\tau^2_k,t,EDF) \le x\times u^2_k+e^2_k$ for any task τ^2_k and $t\in[0,\hat{f}_{i,j}]$ is proved. ◀

We now prove that there is an upper bound on Θ's LAG at time t_2. Let $k'=\max\{k : \tau_{i,k}\in\Theta, r_{i,k}\le\hat{f}_{l,j}\}$. Define

▶ **Definition 17.** Let $U(\tau^2,y)$ ($E(\tau^2,y)$) be the set of at most $\min\{|\tau^2|,y\}$ tasks from τ^2 of highest utilization (execution cost), where $|\tau^2|$ is the number of tasks in τ^2, and let

$$E^2_L = \sum_{\tau^2_i\in E(\tau^2,M^2-1)} e^2_i \quad (22)$$

and

$$U^2_L = \sum_{\tau^2_i\in U(\tau^2,M^2-1)} u^2_i \quad (23)$$

▶ **Lemma 18.** $LAG(\Theta,t_2,EDF) \le x\times U^2_L+E^2_L$.

Proof. To bound $LAG(\Theta,t_2,EDF)$, we sum individual task lags at t_2. If $t_2=0$, then $LAG(\Theta,t_2,EDF)=0$ and the lemma holds trivially. So assume that $t_2>0$. Consider the set of tasks $\chi=\{\tau^2_i : \exists\tau^2_{i,j}\in\Theta$ such that $\tau^2_{i,j}$ is pending at $t_2\}$. Because the instant t_2 is non-busy non-occupation, $|\chi|\le(M^2-1)$. If a task has no pending jobs at t_2, then $lag(\tau^2_i,t_2,S)\le 0$. Therefore, by Eq. (7) and Lemma 16, we have

$$LAG(\Theta,t_2,EDF)$$
$$= \sum_{\tau^2_i:\tau^2_{i,j}\in\Theta} lag(\tau^2_i,t_2,EDF) \le \sum_{\tau^2_i\in\chi} lag(\tau^2_i,t_2,EDF)$$
$$\le \sum_{\tau^2_i\in\chi}(u^2_i\times x+e^2_i) \le E^2_L+x\times U^2_L \quad (24)$$

◀

Thus, based on Lemma 15 and Lemma 18, we have the desired upper bound $LAG(\Theta,\hat{f}_{i,j},S) \le x\times U^2_L+E^2_L+\sum_{\tau^2_c\in\tau^2_H}\delta_c(1-u^2_c)$.

Upper bound on $W(Fd, \hat{f}^2_{i,j}, EDF)$. To compute a bound on the requirement of jobs that can compete with $\tau^2_{i,j}$ after $\hat{f}^2_{i,j}$, $W(Fd, \hat{f}^2_{i,j}, EDF)$, we first find the latest release time of such a job. Intuitively, if a job is released at a time instant which is far behind $\hat{f}^2_{i,j}$, the job's priority may not be higher than $\tau^2_{l,j}$'s priority due to the constrained range of its tardiness bound on the type 1 processors.

▶ **Lemma 19.** *If $\tau^2_{l,k} \in Fd \cup fd$, then $r^2_{l,k} \leq \hat{f}^2_{i,j} + TB^1_l$.*

Proof. Given that $\hat{f}^2_{i,j} \geq d_{i,j}$ holds for any job $\tau^2_{i,j}$, if $\tau^2_{l,k} \in Fd \cup fd$, then $\hat{f}^2_{i,j} \geq d_{i,j} \geq d_{l,k}$ holds. According to Eq. (3), we have $r^2_{l,k} \leq d_{l,k} + TB^1_l$. Thus, $r^2_{l,k} \leq \hat{f}^2_{i,j} + TB^1_l$ is proved. ◀

▶ **Corollary 20.** *All jobs in $Fd \cup fd$ are released at or before $\hat{f}^2_{i,j} + \rho$, where $\rho = \max_{\tau^1_i \in \tau^1}\{TB^1_i\}$ according to Eq. 11.*

According to the dataflow task model, a subjob's release time on type 2 processors is the completion time of the corresponding subjob released by the same dataflow job on type 1 processors. Then, the minimum inter-arrival time of any two consecutive subjobs released by the same dataflow task on type 2 processors is the dataflow task's execution time on type 1 processors. Thus, we have the following corollary.

▶ **Corollary 21.** *The minimum inter-arrival time of any two consecutive jobs, i.e. $\tau^2_{l,k}$ and $\tau^2_{l,k+1}$, released by the same task τ^2_l on type 2 processors equals to the corresponding dataflow task's execution time on type 1 processors, i.e. e^1_l.*

▶ **Lemma 22.** *The amount of work $W(Fd, \hat{f}^2_{i,j}, EDF)$ required by jobs in Fd that can compete with $\tau^2_{i,j}$ after $\hat{f}^2_{i,j}$ can be bounded by $\sum_{\tau^2_c \in \tau^2_H}(e^2_c - \delta_c) + \sum_{\tau^2_l \in \tau^2 \setminus \tau^2_i}(\lceil \frac{TB^1_l}{e^1_l} \rceil) e^2_l$.*

Proof. Each job $\tau_{l,k}$ in Fd is either a carry-in job or is released after $\hat{f}_{i,j}$. In the latter case, by Lemma 19, $\tau_{l,k}$ is released in the interval $(\hat{f}^2_{i,j}, \hat{f}^2_{i,j} + TB^1_l]$. Thus, each task τ^2_l may have one carry-in job in Fd and up to $\lceil \frac{TB^1_l}{e^1_l} \rceil$ jobs in Fd released after $\hat{f}^2_{i,j}$ according to corollary. 20 and 21. If τ^2_l has a carry-in job, then τ^2_l is in τ^2_H and the work due to its carry-in job after $\hat{f}^2_{i,j}$ is at most $e^2_l - \delta_i$. The work generated by any job of τ^2_l in Fd released after $\hat{f}^2_{i,j}$ is at most e^2_l. From these facts, the lemma follows. (Note that τ^2_i is excluded from the second summation because it does not have jobs in Fd.) ◀

Upper bound on $W(fd \cup Fd, \hat{f}^2_{i,j}, EDF)$. Since $W(fd \cup Fd, \hat{f}^2_{i,j}, EDF) \leq LAG(\Theta, \hat{f}^2_{i,j}, EDF) + W(Fd, \hat{f}^2_{i,j}, EDF)$, by Lemmas 15, 18, and 22, we have

$$\begin{aligned} W(fd \cup Fd, \hat{f}^2_{i,j}, EDF) &\leq x \times U^2_L + E^2_L + \sum_{\tau^2_c \in \tau^2_H}(\delta_c(1 - u^2_c) + (e^2_c - \delta_c)) \\ &\quad + \sum_{\tau^2_l \in \tau^2 \setminus \tau^2_i}(\lceil \frac{TB^1_l}{e^1_l} \rceil) e^2_l \\ &\leq x \times U^2_L + E^2_L + \sum_{\tau^2_l \in \tau^2 \setminus \tau^2_i}(\lceil \frac{TB^1_l}{e^1_l} \rceil + 1) e^2_l. \end{aligned} \tag{25}$$

5.2.2 Step 2: Determining necessary condition for tardiness to exceed $x + e^2_i$

We now find a lower bound on the amount of competing work that is necessary for $\tau^2_{i,j}$ to miss its deadline by more than $x + e^2_i$.

▶ **Lemma 23.** *If the tardiness of $\tau_{i,j}^2$ exceeds $x + e_i^2$, where $x \geq \rho$ (recall that ρ is defined in Eq. 11), then $W(fd \cup Fd, \hat{f}_{i,j}^2, EDF) > \rho + m \times (x - \rho) + e_i^2$.*

Proof. We prove it by contraposition: we assume that $W(fd \cup Fd, \hat{f}_{i,j}^2, EDF) \leq \rho + m \times (x - \rho) + e_i^2$ holds and show that the tardiness of $\tau_{i,j}^2$ can not exceed $x + e_i^2$. All jobs in $fD \cup FD$ are ignored for this proof, because they can not preempt $\tau_{i,j}^2$ and thus they can not delay $\tau_{i,j}^2$'s completion time. According to Corollary 20, all jobs in $fd \cup Fd$ are released at or before $\hat{f}_{i,j}^2 + \rho$. Thus, the number of tasks with pending jobs in $fd \cup Fd$ definitely decreases after $\hat{f}_{i,j}^2 + \rho$.

Consider the point in time $b_{i,j} = \max\{c_{i,j-1}^2, v_{i,j}^2\}$, where $v_{i,j}^2 = \min\{t \geq \hat{f}_{i,j}^2 : [t, \infty)$ is a non-busy interval$\}$ and $c_{i,j-1}^2$ is the completion time of $\tau_{i,j-1}^2$ in EDF schedule.

At $b_{i,j}$, $\tau_{i,j}^2$ must have begun executing in EDF, because $b_{i,j} \geq v_{i,j}^2 \geq \hat{f}_{i,j}^2 \geq r_{i,j}^2$. Its predecessor has completed (since $b_{i,j} \geq c_{i,j-1}^2$), and there is at least one idle processor after $b_{i,j}$. Therefore, $\tau_{i,j}^2$ will complete by $b_{i,j} + e_i^2$. In other words,

$$c_{i,j}^2 \leq \max\{c_{i,j-1}^2 + e_i^2, v_{i,j}^2 + e_i^2\} \leq \max\{\hat{f}_{i,j-1}^2 + x + e_i^2 + e_i^2, v_{i,j}^2 + e_i^2\} \tag{26}$$

We consider two cases depending on the relationship between $\hat{f}_{i,j-1}^2 + x + e_i^2 + e_i^2$ and $v_{i,j}^2 + e_i^2$.

Case 1. $\boldsymbol{\hat{f}_{i,j-1}^2 + x + e_i^2 + e_i^2 \geq v_{i,j}^2 + e_i^2}$**.** Because jobs of τ_i^2 execute sequentially at a rate of u_i^2 in PS, and do not begin until their predecessors complete, thus,

$$f_{i,j}^2 \geq f_{i,j-1}^2 + e_i^2/u_i^2 \geq f_{i,j-1}^2 + e_i^2. \tag{27}$$

Also, jobs execute in PS after their release times, thus,

$$f_{i,j}^2 \geq r_{i,j}^2 + e_i^2/u_i^2 \geq r_{i,j}^2 + e_i^2 \geq d_{i,j-1}^2 + e_i^2. \tag{28}$$

Combine Eq. (27) with Eq. (28), we get $f_{i,j}^2 \geq \max\{f_{i,j-1}, d_{i,j-1}\} + e_i^2 = \hat{f}_{i,j-1}^2 + e_i^2$. Thus, Eq. (26) becomes

$$\begin{aligned} c_{i,j}^2 &\leq \max\{\hat{f}_{i,j-1} + x + e_i^2 + e_i^2, v_{i,j}^2 + e_i^2\} = \hat{f}_{i,j-1} + x + e_i^2 + e_i^2 \leq f_{i,j} + x + e_i^2 \\ &\leq \hat{f}_{i,j}^2 + x + e_i^2, \end{aligned} \tag{29}$$

which implies that the tardiness of $\tau_{i,j}^2$ does not exceed $x + e_i^2$.

Case 2. $\boldsymbol{\hat{f}_{i,j-1}^2 + x + e_i^2 + e_i^2 < v_{i,j}^2 + e_i^2}$**.** If $v_{i,j}^2 \leq \hat{f}_{i,j}^2 + x$, then by the definition of $v_{i,j}^2$, the tardiness of $\tau_{i,j}^2$ can not exceed $x + e_i^2$. Thus, we assume otherwise, i.e., $v_{i,j}^2 > \hat{f}_{i,j}^2 + x$. In this case, $[\hat{f}_{l,j} + \rho, \hat{f}_{l,j} + x]$ must be a busy interval. This is because according to Corollary 20, all jobs in $fd \cup Fd$ are released at or before $\hat{f}_{i,j}^2 + \rho$. Thus, the number of tasks with pending jobs in $fd \cup Fd$ definitely decreases after $\hat{f}_{i,j}^2 + \rho$. When $v_{i,j}^2 > \hat{f}_{i,j}^2 + x$, $v_{i,j}^2$ must be the earliest non-busy time instance after $\hat{f}_{i,j}^2 + \rho$. The reason is explained in Fig. 4. Since no jobs are released after $\hat{f}_{i,j}^2 + \rho$, once a processor is idle at a non-busy time instant $t' \geq \hat{f}_{i,j}^2 + \rho$, no jobs can be scheduled to it. Then, the time interval after t' becomes non-busy. Thus, by the definition of $v_{i,j}^2$, $v_{i,j}^2$ must equal t'. Thus, the workload in EDF during $[\hat{f}_{i,j}^2 + \rho, \hat{f}_{i,j}^2 + x]$ is at least $(x - \rho) \times m$. Since $W(fd \cup Fd, \hat{f}_{i,j}^2, EDF) \leq \rho + m \times (x - \rho) + e_i^2$, the workload in EDF during $[\hat{f}_{i,j}^2, \hat{f}_{i,j}^2 + \rho]$ and $[\hat{f}_{i,j}^2 + x, \hat{f}_{i,j}^2 + x + e_i^2]$ is at most $\rho + e_i^2$. Even if all this work executes sequentially, it will complete by $\hat{f}_{i,j}^2 + x + e_i^2$ because $x \geq \rho$. Therefore, $c_{i,j}^2 \leq \hat{f}_{i,j}^2 + x + e_i^2$ and hence the contraposition holds. This lemma is proved. ◀

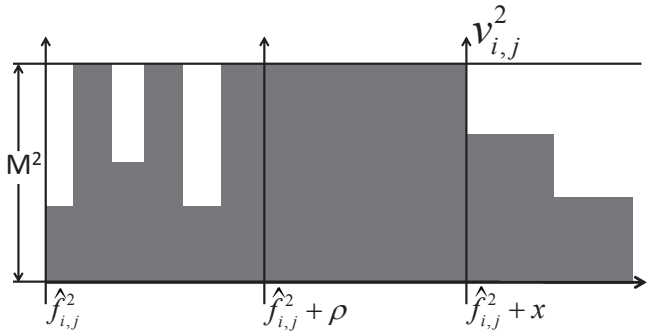


Figure 4 The structure of workload in Lemma 23.

5.2.3 Step 3: Deriving EDF-induced tardiness bound

By Lemma. 23, setting the upper bound on Eq. (25) to be at most $\rho + m \times (x - \rho) + e_i^2$ will ensure that the EDF-induced tardiness of $\tau_{i,j}^2$ is at most $x + e_i^2$. The resulting inequality is as follows.

$$x \times U_L^2 + E_L^2 + \sum_{\tau_l^2 \in \tau^2 \setminus \tau_i^2} (\lceil \frac{TB_l^1}{e_l^1} \rceil + 1) e_l^2 \leq \rho + m \times (x - \rho) + e_i^2 \tag{30}$$

Thus,

$$x \geq \frac{E_L^2 + D^*}{m - U_L^2} \tag{31}$$

where

$$D^* = (m-1)\rho - e_i^2 + \sum_{\tau_l^2 \in \tau^2 \setminus \tau_i^2} (\lceil \frac{TB_l^1}{e_l^1} \rceil + 1) e_l^2. \tag{32}$$

If x equals the greater of ρ and the right-hand side of Eq. (31) (recall that $x \geq \rho$ is required), then the EDF-induced tardiness of $\tau_{i,j}^2$ will not exceed $x + e_i^2$.

▶ **Theorem 24.** *With x as defined above, the EDF-induced tardiness for a job $\tau_{i,j}^2$ on type 2 processors in EDF schedule is at most $x + e_i^2$.*

5.3 Tardiness Bound of τ_i^2

We have bounded PS-induced tardiness and EDF-induced tardiness of any τ_i^2 in Secs. 5.1 and 5.2, respectively. As shown earlier in Fig. 3, the total tardiness of τ_i^2 scheduled under EDF on type-2 processors can be bounded by combining these two types of tardiness. The following theorem immediately follows.

▶ **Theorem 25.** *The tardiness of any task $\tau_i^2 \in \tau$ scheduled under EDF is at most*

$$TB_i^2 = TB_i^1 + P_i + \frac{E_L^2 + D^*}{m - U_L^2} + e_i^2, \tag{33}$$

where $D^* = (m-1)\rho - e_i^2 + \sum_{\tau_l^2 \in \tau^2 \setminus \tau_i^2} (\lceil \frac{TB_l^1}{e_l^1} \rceil + 1) e_l^2$.

6 Tardiness Bound of τ_i^k ($k > 2$)

In this section, we bound any τ_i^k's ($k > 2$) tardiness based on τ_i^{k-1}'s tardiness on type-$(k-1)$ processors. The proof procedure is exactly the same as how we bound τ_i^2's tardiness based on τ_i^1's tardiness in Sec. 5.

6.1 *PS*-induced Tardiness of τ_k^i

▶ **Theorem 26.** *Tardiness Bound of τ_i^k on PS Schedule is $TB_i^{k-1} + P_i$.*

Proof. We can prove this Theorem by mathematical induction. (i) When $k = 2$, the tardiness Bound of τ_i^2 in PS is $TB_i^1 + P_i$ according to Theorem 7. (ii) When $k > 2$, the tardiness Bound of τ_i^k in PS is $TB_i^{k-1} + P_i$, which can be proved using the same reasoning for proving Theorem 7 (by simply replacing subtask indexes 1 and 2 by indexes $k-1$ and k respectively). Thus, this theorem is proved. ◀

6.2 EDF-induced Tardiness of τ_i^k

In this section, we bound EDF-induced tardiness of τ_i^k exactly as how we bound such tardiness for τ_i^2 described in Sec. 5.2 in the following three steps.

Step 1: We derive an **upper bound** on competing work at $\hat{f}_{i,j}^k$ following the same methods described in Sec. 5.2.1.

$$W(fd \cup Fd, \hat{f}_{i,j}^k, EDF) \leq x \times U_L^k + E_L^k + \sum_{\tau_c^k \in \tau_H^k} (\delta_c(1 - u_c^k) + (e_c^k - \delta_c))$$

$$+ \sum_{\tau_l^k \in \tau^k \setminus \tau_i^k} (\lceil \frac{TB_l^{k-1}}{e_l^{k-1}} \rceil) e_l^k \leq x \times U_L^k + E_L^k + \sum_{\tau_l^k \in \tau^k \setminus \tau_i^k} (\lceil \frac{TB_l^{k-1}}{e_l^{k-1}} \rceil + 1) e_l^k. \tag{34}$$

Step 2: Determining **necessary condition** for tardiness to exceed $x + e_i^k$ following the same methods described in Sec. 5.2.2.

$$W(fd \cup Fd, \hat{f}_{i,j}^k, EDF) > \rho + m \times (x - \rho) + e_i^k, \tag{35}$$

where $\rho = \max_{\tau_i^{k-1} \in \tau^{k-1}} \{TB_i^{k-1}\}$.

Step 3: Deriving **EDF-induced tardiness bound** on type k processors following the same methods described in Sec. 5.2.3. Based on Eq. 34 and Eq. 35, we have

$$x \geq \frac{E_L^k + D^*}{M^k - U_L^k} \tag{36}$$

where

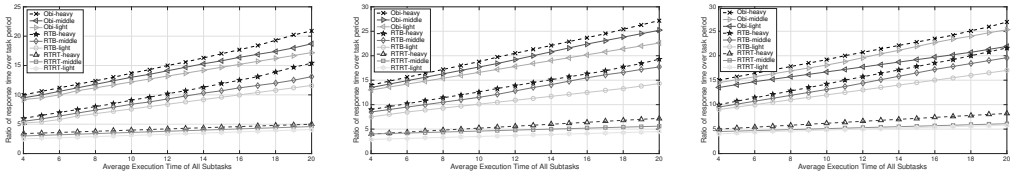
$$D^* = (M^k - 1)\rho - e_i^k + \sum_{\tau_l^k \in \tau^k \setminus \tau_i^k} (\lceil \frac{TB_l^{k-1}}{e_l^{k-1}} \rceil + 1) e_l^k. \tag{37}$$

Thus, the following corollary immediately follows.

▶ **Corollary 27.** *The tardiness of any subtask $\tau_i^k \in \tau$ scheduled under EDF is at most*

$$TB_i^k = TB_i^{k-1} + P_i + \frac{E_L^k + D^*}{M^k - U_L^k} + e_i^k$$

$$= TB_i^1 + (k-1)P_i + \sum_{j=2}^{k} \frac{E_L^j + D^*}{M^j - U_L^j} + \sum_{j=2}^{k} e_i^j, \tag{38}$$



(a) 4 types of processors **(b)** 6 types of processors **(c)** 8 types of processors

Figure 5 Magnitude of our derived analytical response time bounds.

where $D^* = (M^j - 1)\rho - e_i^j + \sum_{\tau_l^j \in \tau^j \setminus \tau_i^j} (\lceil \frac{TB_l^{j-1}}{e_l^{j-1}} \rceil + 1) e_l^j$.

Since the response time bound of any task τ_i can be obtained by simply using the derived tardiness bound of the task plus a P_i value, the response time bound can be easily calculated by the following Theorem 28.

▶ **Theorem 28.** *The response time bound for any dataflow task $\tau_i \in \tau$ with m subtasks scheduled under EDF, denoted RB_i, is*

$$\begin{aligned} RB_i &= TB_i^m + P_i \\ &= TB_i^1 + mP_i + \sum_{j=2}^{m} \frac{E_L^j + D^*}{M^j - U_L^j} + \sum_{j=2}^{m} e_i^j, \end{aligned} \tag{39}$$

where $D^* = (M^j - 1)\rho - e_i^j + \sum_{\tau_l^j \in \tau^j \setminus \tau_i^j} (\lceil \frac{TB_l^{j-1}}{e_l^{j-1}} \rceil + 1) e_l^j$ *and* TB_i^1 *is given by Eq. 3.*

7 Experiments

So far we have shown that EDF is optimal to support dataflow tasks with bounded response times on a heterogeneous computing platform. The magnitude of the analytical response time bound yielded under our analysis is of importance. To assess this, we have conducted experiments using randomly-generated task sets with widely varied parameters to examine how large the magnitude of the analytical response time bound is. Although we can only use synthesized task sets for assessment, we have verified that the generated range of parameters is wide enough to cover a couple of industrial dataflow-based cellular network systems seen in practice.

7.1 Experiment setup

The experiments compare the analytical response time bounds (RTB) given by Corollary 28 and the actual response time (RTRT) observed at runtime for randomly generated dataflow tasks scheduled under EDF in a heterogeneous multiprocessor system with m types of resources. For each resource type, there are 8 identical processors. When we generate subtasks, new subtasks were added until the total utilization of subtasks on each type of resources equals 8. Subtask utilizations were distributed differently for each experiment using three uniform distributions. The ranges for the uniform distributions were [0.005,0.1] (light), [0.1,0.3] (medium), and [0.3,0.8] (heavy). Task execution time were uniformly distributed over $(0ms, 20ms]$. Task periods were calculated from execution time and utilizations. For each task utilization distribution, $1,000,000$ task sets were generated for systems with $m = 4$,

6, or 8. We compare RTB, Obi, and RTRT under three utilization settings: per-subtask utilization is heavy, medium, and light.

7.2 Results

The obtained results are shown in Figure 5. Each curve plots the ratio of response time over task period averaged among all tasks generated in each experiment, as a function of the average execution time of all subtasks.

As seen in the figures, in all scenarios, RTB yields a much reduced response time bounds compared to Obi. For example, as shown in Fig. 5(a), when the average execution time of all subtasks is 12, RTB (Obi) yields a ratio of response time over task period of 8.6 (13.7), 9.3 (14.5), 10.1 (15.3), under light, medium, and heavy per-subtask utilizations, respectively. On average, RTB yields a 68% reduction on response time bounds compared to Obi. Moreover, in most scenarios, RTB yields reasonably small response time bounds compared to RTRT. For example, as shown in Fig. 5(a), when the average execution time of all subtasks is at most 12, the ratio of response time over task period under RTB ranges from 5 to 10, where the ratio under RTRT ranges from 4 to 6. Even for the worst-case scenario where the average execution time of all subtasks is large, the response time bound yielded under RTB is at most 3 times greater than RTRT. Given that RTRT represents the runtime response time observed in an actual GEDF schedule, we believe that the analytical bounded yielded under RTB is not only safe, but also reasonably tight in most cases. Another observation seen in all tested scenarios is that the analytical response time bounds under RTB become larger when per-subtask utilizations become heavier. This is because the task sets with heavy utilization may have the largest values of U_L^k (defined in Eq. 23), which results the largest response time bound according to Corollary 28.

8 Conclusion

In this paper, we investigate the problem of scheduling dataflow tasks on a heterogeneous computing platform with multiple types of resources with pre-defined affinity of tasks to subgroups of resources, which is motivated by many industrial applications seen in practice. We present a new set of analysis techniques that demonstrate that the classical and simple EDF scheduler can guarantee probably bounded response times for tasks with no capacity loss, thus proving EDF to be an optimal solution for this dataflow scheduling problem. Despite EDF's optimality in terms of schedulability, experiments also demonstrate that the magnitude of the response time bounds calculated under our analysis is reasonably small under all scenarios. This paper demonstrates the potential of applying EDF into practical industrial systems to schedule dataflow workloads with guaranteed bounded response times.

References

1 Ahmed Alhammad and Rodolfo Pellizzoni. Schedulability analysis of global memory-predictable scheduling. In *2014 International Conference on Embedded Software, EMSOFT 2014, New Delhi, India, October 12-17, 2014*, pages 20:1–20:10, 2014. doi:10.1145/2656045.2656070.

2 Björn Andersson, Gurulingesh Raravi, and Konstantinos Bletsas. Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors. In *Proceedings of the 31st IEEE Real-Time Systems Symposium, RTSS 2010, San Diego, California, USA, November 30 – December 3, 2010*, pages 239–248, 2010. doi:10.1109/RTSS.2010.32.

3 Bach Duy Bui, Rodolfo Pellizzoni, Marco Caccamo, Chin F. Cheah, and Andrew Tzakis. Soft real-time chains for multi-hop wireless ad-hoc networks. In *Proceedings of the 13th*

IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2007, April 3-6, 2007, Bellevue, Washington, USA, pages 69–80, 2007. doi:10.1109/RTAS.2007.34.

4 Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, pages 137–150, 2004. URL: http://www.usenix.org/events/osdi04/tech/dean.html.

5 UmaMaheswari C. Devi and James H. Anderson. Fair integrated scheduling of soft real-time tardiness classes on multiprocessors. In *10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004), 25-28 May 2004, Toronto, Canada*, pages 554–561, 2004. doi:10.1109/RTTAS.2004.1317303.

6 UmaMaheswari C. Devi and James H. Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. In *Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS 2005), 6-8 December 2005, Miami, FL, USA*, pages 330–341, 2005. doi:10.1109/RTSS.2005.39.

7 Zheng Dong and Cong Liu. Closing the loop for the selective conversion approach: A utilization-based test for hard real-time suspending task systems. In *2016 IEEE Real-Time Systems Symposium, RTSS 2016, Porto, Portugal, November 29 – December 2, 2016*, pages 339–350, 2016. doi:10.1109/RTSS.2016.040.

8 Piotr Dziurzanski, Amit Kumar Singh, Leandro Soares Indrusiak, and Björn Saballus. Hard real-time guarantee of automotive applications during mode changes. In *Proceedings of the 23rd International Conference on Real Time Networks and Systems, RTNS 2015, Lille, France, November 4-6, 2015*, pages 161–170, 2015. doi:10.1145/2834848.2834859.

9 Tom Z. J. Fu, Jianbing Ding, Richard T. B. Ma, Marianne Winslett, Yin Yang, Zhenjie Zhang, Yong Pei, and Bingbing Ni. Livetraj: Real-time trajectory tracking over live video streams. In *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference, MM'15, Brisbane, Australia, October 26-30, 2015*, pages 777–780, 2015. doi:10.1145/2733373.2807401.

10 Vincenzo Gulisano, Ricardo Jiménez-Peris, Marta Patiño-Martínez, Claudio Soriente, and Patrick Valduriez. Streamcloud: An elastic and scalable data streaming system. *IEEE Trans. Parallel Distrib. Syst.*, 23(12):2351–2365, 2012. doi:10.1109/TPDS.2012.24.

11 Ralf Jahr, Mike Gerdes, Theo Ungerer, Haluk Ozaktas, Christine Rochange, and Pavel G. Zaykov. Effects of structured parallelism by parallel design patterns on embedded hard real-time systems. In *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications, Chongqing, China, August 20-22, 2014*, pages 1–10, 2014. doi:10.1109/RTCSA.2014.6910546.

12 Shunsuke Kamijo, Yasuyuki Matsushita, Katsushi Ikeuchi, and Masao Sakauchi. Traffic monitoring and accident detection at intersections. *IEEE Trans. Intelligent Transportation Systems*, 1(2):108–118, 2000. doi:10.1109/6979.880968.

13 Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy, and Siddarth Taneja. Twitter heron: Stream processing at scale. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 – June 4, 2015*, pages 239–250, 2015. doi:10.1145/2723372.2742788.

14 Kam-yiu Lam, Jiantao Wang, Joseph Kee-Yin Ng, Song Han, Limei Zheng, Calvin Ho Chuen Kam, and Chun Jiang Zhu. Smartmood: Toward pervasive mood tracking and analysis for manic episode detection. *IEEE Trans. Human-Machine Systems*, 45(1):126–131, 2015. doi:10.1109/THMS.2014.2360469.

15 Hennadiy Leontyev and James H. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. In *Proceedings of the 28th IEEE Real-Time Systems Sym-*

posium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA, pages 413–422, 2007. doi:10.1109/RTSS.2007.33.

16 Yang Li, Linh Thi Xuan Phan, and Boon Thau Loo. Network functions virtualization with soft real-time guarantees. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*, pages 1–9, 2016. doi:10.1109/INFOCOM.2016.7524563.

17 Cong Liu and James H. Anderson. Supporting Soft Real-Time DAG-Based Systems on Multiprocessors with No Utilization Loss. In *Proceedings of the 31st IEEE Real-Time Systems Symposium, RTSS 2010, San Diego, California, USA, November 30 – December 3, 2010*, pages 3–13, 2010. doi:10.1109/RTSS.2010.38.

18 Cong Liu and James H. Anderson. An o(m) analysis technique for supporting real-time self-suspending task systems. In *Proceedings of the 33rd IEEE Real-Time Systems Symposium, RTSS 2012, San Juan, PR, USA, December 4-7, 2012*, pages 373–382, 2012. doi:10.1109/RTSS.2012.87.

19 José Marinho, Vincent Nélis, Stefan M. Petters, Marko Bertogna, and Robert I. Davis. Limited pre-emptive global fixed task priority. In *Proceedings of the IEEE 34th Real-Time Systems Symposium, RTSS 2013, Vancouver, BC, Canada, December 3-6, 2013*, pages 182–191, 2013. doi:10.1109/RTSS.2013.26.

20 André Oliveira Maroneze, Sandrine Blazy, David Pichardie, and Isabelle Puaut. A formally verified WCET estimation tool. In *14th International Workshop on Worst-Case Execution Time Analysis, WCET 2014, July 8, 2014, Ulm, Germany*, pages 11–20, 2014. doi:10.4230/OASIcs.WCET.2014.11.

21 Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio C. Buttazzo. Response-time analysis of conditional DAG tasks in multiprocessor systems. In *27th Euromicro Conference on Real-Time Systems, ECRTS 2015, Lund, Sweden, July 8-10, 2015*, pages 211–221, 2015. doi:10.1109/ECRTS.2015.26.

22 Jan Nowotsch and Michael Paulitsch. Quality of service capabilities for hard real-time applications on multi-core processors. In *21st International Conference on Real-Time Networks and Systems, RTNS 2013, Sophia Antipolis, France, October 17-18, 2013*, pages 151–160, 2013. doi:10.1145/2516821.2516826.

23 Alessandro Vittorio Papadopoulos, Martina Maggio, Alberto Leva, and Enrico Bini. Hard real-time guarantees in feedback-based resource reservations. *Real-Time Systems*, 51(3):221–246, 2015. doi:10.1007/s11241-015-9224-1.

24 Keivan Ronasi, Vincent W. S. Wong, and Sathish Gopalakrishnan. Distributed scheduling in multihop wireless networks with maxmin fairness provisioning. *IEEE Trans. Wireless Communications*, 11(5):1753–1763, 2012. doi:10.1109/TWC.2012.030812.110493.

25 Kecheng Yang, Ming Yang, and James H. Anderson. Reducing response-time bounds for dag-based task systems on heterogeneous multicore platforms. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS 2016, Brest, France, October 19-21, 2016*, pages 349–358, 2016. doi:10.1145/2997465.2997486.

26 Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters. In *4th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'12, Boston, MA, USA, June 12-13, 2012*, 2012. URL: https://www.usenix.org/conference/hotcloud12/workshop-program/presentation/zaharia.

27 Haibo Zeng and Marco Di Natale. Outstanding paper award: Using max-plus algebra to improve the analysis of non-cyclic task models. In *25th Euromicro Conference on Real-Time Systems, ECRTS 2013, Paris, France, July 9-12, 2013*, pages 205–214, 2013. doi:10.1109/ECRTS.2013.30.