Reducing Response-Time Bounds for DAG-Based Task Systems on Heterogeneous Multicore Platforms *

Kecheng Yang, Ming Yang, and James H. Anderson
Department of Computer Science
University of North Carolina at Chapel Hill
{yangk,yang,anderson}@cs.unc.edu

ABSTRACT

This paper considers for the first time end-to-end response-time analysis for DAG-based real-time task systems implemented on heterogeneous multicore platforms. The specific analysis problem that is considered was motivated by an industrial collaboration involving wireless cellular base stations. The DAG-based systems considered herein allow intra-task parallelism: while each invocation of a task (i.e., DAG node) is sequential, successive invocations of a task may execute in parallel. In the proposed analysis, this characteristic is exploited to reduce response-time bounds. Additionally, there is some leeway in choosing how to set tasks' relative deadlines. It is shown that by resolving such choices holistically via linear programming, response-time bounds can be further reduced. Finally, in the considered use case, DAGs are defined based upon just a few templates and individually often have quite low utilizations. It is shown that, by combining many such DAGs into one of higher utilization, response-time bounds can often be drastically lowered. The effectiveness of these techniques is demonstrated via both case-study and schedulability experiments.

1. INTRODUCTION

The multicore revolution is currently undergoing a second wave of innovation in the form of heterogeneous hardware platforms. In the domain of real-time embedded systems, such platforms may be desirable to use for a variety of reasons. For example, ARM's big.LITTLE multicore architecture [8] enables performance and energy concerns to be balanced by providing a mix of relatively slower, low-power cores and faster, high-power ones. Unfortunately, the move towards greater heterogeneity is further complicating software design processes that were already being challenged on account of the significant parallelism that exists in "conventional" multicore platforms with identical processors. Such complications are impeding advancements in the embedded computing industry today.

*Work supported by NSF grants CNS 1563845, CPS 1239135, CNS 1409175, and CPS 1446631, AFOSR grant FA9550-14-1-0161, ARO grant W911NF-14-1-0499, and funding from both General Motors and FutureWei Corp.

RTNS '16, October 19 - 21, 2016, Brest, France

ISBN 978-1-4503-4787-7/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2997465.2997486

Problem considered herein. In this paper, we report on our efforts towards solving a particular real-time analysis problem concerning heterogeneity motivated by an industrial collaboration. This problem pertains to the processing done by cellular base stations in wireless networks. Due to space constraints, we refrain from delving into specifics regarding this particular application domain, opting instead for a more abstract treatment of the problem at hand.

This problem involves the scheduling of real-time dataflows on heterogeneous computational elements (CEs), such as CPUs, digital signal processors (DSPs), or one of many types of hardware accelerators. Each dataflow is represented by a DAG, the nodes (resp., edges) of which represent tasks (resp., producer/consumer relationships). A given task is restricted to run on a specific CE type. Task preemption may be impossible for some CEs and should in any case be discouraged. Each DAG has a single source task that is invoked periodically. Intra-task parallelism is allowed in the sense that consecutive jobs (*i.e.*, invocations) of the same task can execute in parallel (but each job executes sequentially). In fact, a later job can finish earlier due to variations in running times.[1] The DAGs to be supported are defined using a relatively small number of "templates," *i.e.*, many DAGs may exist that are structurally identical. The challenge is to devise a multi-resource, real-time scheduler for supporting dataflows as described here with accompanying per-dataflow end-to-end response-time analysis. That is, an upper bound on the response time of a single invocation of each DAG in a system comprised of such DAGs is required.

Related work. The literature on real-time systems includes much work pertaining to the scheduling of DAG-based task systems on identical multiprocessor platforms [1, 2, 4, 5, 6, 7, 9, 10, 13, 15, 17, 18, 19, 20, 21, 22, 23, 24]. However, we are not aware of any corresponding work directed at heterogeneous platforms. Moreover, the problem above has facets—such as allowing intra-task parallelism and defining potentially many DAGs based upon relatively few templates—that have not been fully explored. (Intra-task parallelism has been considered in a limited context in DAG-related work pertaining to identical multiprocessors [7, 9, 23].) Additionally, in prior work on the real-time scheduling of task systems—which are not DAG-based—upon heterogeneous platforms, task-to-CE assignments have been a paramount concern. In our setting, this is a non-issue, as this assignment is pre-determined based on CE functionalities.

Beyond the real-time systems community, DAG-based systems implemented on heterogeneous platforms have been considered before (*e.g.*, [3, 16, 26]). However, all such work known to us focuses on one-shot, aperiodic DAG-based jobs, rather than periodic

[1] In the considered application domain, the data produced by subsequent jobs can be buffered until all prior jobs of the same task have completed.

or sporadic DAG-based task systems. Moreover, real-time issues are considered only obliquely from the perspectives of job admission control or job makespan minimization.

Contributions. In this paper, we formalize the problem described above and then address it by proposing a scheduling approach and associated end-to-end response-time analysis. In the first part of the paper, we attack the problem by presenting a transformation process whereby successive task models are introduced such that: **(i)** the first task model directly formalizes the problem above; **(ii)** prior analysis can be applied to the last model to obtain response-time bounds under earliest-deadline-first (EDF) scheduling; and **(iii)** each successive model is a refinement of the prior one in the sense that all DAG-based precedence constraints are preserved. Such a transformation approach was previously used by Liu and Anderson [20] in work on DAG-based systems, but that work focused on identical multiprocessors. Moreover, our work differs from theirs in that we allow intra-task parallelism. This enables much smaller end-to-end response-time bounds to be derived.

After presenting this transformation process, we discuss two techniques that can reduce the response-time bounds enabled by this process. The first technique exploits the fact that some leeway exists in setting tasks' relative deadlines. By setting more aggressive deadlines for tasks along "long" paths in a DAG, the overall end-to-end response-time bound of that DAG can be reduced. We show that such deadline adjustments can be made by solving a linear program.

The second technique exploits the fact that, in the considered context, DAGs are defined using relatively few templates and typically have quite low utilizations. These facts enable us to reduce response-time bounds by combining many DAGs into one of larger utilization. As a very simple example, two DAGs with a period of 10 time units might be combined into one with a period of 5 time units. A response-time-bound reduction is enabled because these bounds tend to be proportional to periods. In the considered application domain, the extent of combining can be much more extensive: upwards of 40 DAGs may be combinable.

As a final contribution, we evaluate our proposed techniques via case-study and schedulability experiments. These experiments show that our techniques can significantly reduce response-time bounds. Furthermore, our analysis supports "early releasing" [12] (see Sec. 7) to improve observed end-to-end response times. We experimentally demonstrate the efficacy of this as well.

Organization. In the rest of this paper, we formalize the considered problem (Sec. 2), present the refinements mentioned above that enable the use of prior analysis (Secs. 3–4), show that the bounds arising from this analysis can be improved via linear programming (Sec. 5) and DAG combining (Sec. 6), discuss early releasing (Sec. 7), present our case-study (Sec. 8) and schedulability (Sec. 9) experiments, and conclude (Sec. 10).

2. SYSTEM MODEL

In this section, we formalize the dataflow-scheduling problem described in Sec. 1 and introduce relevant terminology. Each dataflow is represented by a DAG, as discussed earlier.

We specifically consider a system $G = \{G_1, G_2, \ldots, G_N\}$ comprised of N DAGs. The DAG G_i consists of n_i nodes, which correspond to n_i tasks, denoted $\tau_i^1, \tau_i^2, \ldots, \tau_i^{n_i}$. Each task τ_i^v releases a (potentially infinite) sequence of *jobs* $J_{i,1}^v$, $J_{i,2}^v, \ldots$. The edges in G_i reflect producer/consumer relationships. A particular task τ_i^v's *producers* are those tasks with outgoing edges directed to τ_i^v, and its *consumers* are those with incoming edges directed from τ_i^v. The j^{th} job of task τ_i^v, $J_{i,j}^v$, cannot commence execution until the j^{th} jobs of all of its producers have completed; this ensures that its necessary input data is available. Such job dependencies only exist with respect to the *same invocation* of a DAG, and not across different invocations. That is, while jobs must execute sequentially, intra-task parallelism is allowed.

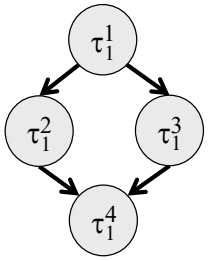


Figure 1: A DAG G_1.

Example 1. Fig. 1 shows an example DAG, G_1. Task τ_1^4's producers are tasks τ_1^2 and τ_1^3, thus for any j, $J_{1,j}^4$ needs input data from each of $J_{1,j}^2$ and $J_{1,j}^3$, so it must wait until those jobs complete. Because intra-task parallelism is allowed, $J_{1,j}^4$ and $J_{1,j+1}^4$ could potentially execute in parallel.

To simplify analysis, we assume that each DAG G_i has exactly one *source task* τ_i^1, which has only outgoing edges, and one *sink task* $\tau_i^{n_i}$, which has only incoming edges. Multi-source/multi-sink DAGs can be supported with the addition of singular "virtual" sources and sinks that connect multiple sources and sinks, respectively. Virtual sources and sinks have a worst-case execution time (WCET) of zero.

We consider the scheduling of DAGs as just described on a heterogeneous hardware platform consisting of different types of CEs. A given CE might be a CPU, DSP, or some specialized hardware accelerator (HAC). The CEs are organized in M CE *pools*, where each CE pool π_k consists of m_k *identical* CEs. Each task τ_i^v has a parameter P_i^v that denotes the particular CE pool on which it must run, *i.e.*, $P_i^v = \pi_k$ means that each job of τ_i^v must be scheduled on a CE in the CE pool π_k. The WCET of task τ_i^v is denoted C_i^v.

Although the problem description in Sec. 1 indicated that source tasks are released periodically, we generalize this to allow sporadic releases, *i.e.*, for the DAG G_i, the job releases of τ_i^1 have a minimum separation time, denoted T_i. A non-source task τ_i^v $(v > 1)$ releases its j^{th} job $J_{i,j}^v$ when the j^{th} jobs of all its producer tasks in G_i have completed. That is, letting $a_{i,j}^v$ and $f_{i,j}^v$ denote the release (or arrival) and finish times of $J_{i,j}^v$, respectively,

$$a_{i,j}^v = \max\{f_{i,j}^w \mid \tau_i^w \text{ is a producer of } \tau_i^v\}. \tag{1}$$

The *response time* of job $J_{i,j}^v$ is defined as $f_{i,j}^v - a_{i,j}^v$, and the *end-to-end response time* of the DAG G_i as $f_{i,j}^{n_i} - a_{i,j}^1$.

Example 2. Fig. 2 depicts an example schedule for the DAG G_1 in Fig. 1, assuming task τ_1^2 is required to execute on a DSP and the other tasks are required to execute on a CPU. The first (resp., second) job of each task has a lighter (resp., darker) shading to make them easier to distinguish. Tasks τ_1^2 and τ_1^3 have only one producer, τ_1^1, so when task τ_1^1 finishes a job at times 3 and 8, τ_1^2 and τ_1^3 release a job immediately. In contrast, task τ_1^4 has two producers, τ_1^2 and τ_1^3. Therefore, τ_1^4 cannot release a job at time 5 when τ_1^3 finishes a job, but rather must wait until time 6 when τ_1^2 also finishes a job. Note that consecutive jobs of the same task might execute in parallel (*e.g.*, $J_{1,1}^4$ and $J_{1,2}^4$ execute in parallel during $[11, 12)$). Furthermore, for a given task, a later-released job (*e.g.*, $J_{1,2}^4$) may even finish earlier than an earlier-released one (*e.g.*, $J_{1,1}^4$) due to execution-time variations.

Scheduling. Since many CEs are non-preemptible, we use the non-preemptive global EDF (G-EDF) scheduling algorithm within each CE pool. The *deadline* of job $J_{i,j}^v$ is given by

$$d_{i,j}^v = a_{i,j}^v + D_i^v, \tag{2}$$

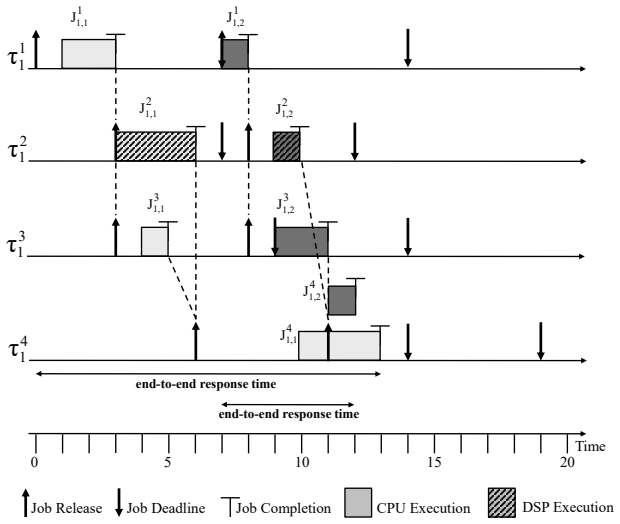


Figure 2: Example schedule for the DAG in G_1 in Fig. 1.

where D_i^v is the *relative deadline* of task τ_i^v. For example, in the example schedule in Fig. 2, relative deadlines of $D_1^1 = 7$, $D_1^2 = 4$, $D_1^3 = 6$, and $D_1^4 = 8$ are assumed.

In the context of this paper, deadlines mainly serve the purpose of determining jobs' priorities rather than strict timing constraints for individual jobs. Therefore, deadline misses are acceptable as long as the end-to-end response time of each DAG can be reasonably bounded.

Utilization. We denote the *utilization* of task τ_i^v by

$$u_i^v = \frac{C_i^v}{T_i}. \tag{3}$$

We also use Γ_k to denote the set of tasks that are required to execute on the CE pool π_k, *i.e.*,

$$\Gamma_k = \{\tau_i^v \mid P_i^v = \pi_k\}. \tag{4}$$

The overutilization of a CE pool could cause unbounded response times, so we require for each k,

$$\sum_{\tau_i^v \in \Gamma_k} u_i^v \leq m_k. \tag{5}$$

3. OFFSET-BASED INDEPENDENT TASKS

In this section, we present a second task model, which is a refinement of that just presented, as will be shown in Sec. 4. The prior model is somewhat problematic because of difficult-to-analyze dependencies among jobs. In particular, by (1), the release times of jobs of non-source tasks depend on the finish times of other jobs, and hence on their execution times. By (2), deadlines (and hence priorities) of jobs are affected by similar dependencies.

In order to ease analysis difficulties associated with such job dependencies, we introduce here the *offset-based independent task (obi-task) model*. Under this model, tasks are partitioned into groups. The i^{th} such group consists of tasks denoted $\tau_i^1, \tau_i^2, \ldots, \tau_i^{n_i}$, where τ_i^1 is a designated *source* task that releases jobs sporadically with a minimum separation of T_i. That is, for any positive integer j,

$$a_{i,j+1}^1 - a_{i,j}^1 \geq T_i. \tag{6}$$

Job releases of each non-source task τ_i^v are governed by a new parameter Φ_i^v, called the *offset* of τ_i^v. Specifically, τ_i^v releases its j^{th} job exactly Φ_i^v time units after the release time of the j^{th} job of the source task τ_i^1 of its group. That is,

$$a_{i,j}^v = a_{i,j}^1 + \Phi_i^v. \tag{7}$$

For consistency, we define

$$\Phi_i^1 = 0. \tag{8}$$

Under the obi-task model, a job of a task τ_i^v can be scheduled at any time after its release *independently* of the execution of any other jobs, *even jobs of the same task* τ_i^v.

The definitions so far have dealt with job releases. Additionally, the two per-task parameters C_i^v and P_i^v from Sec. 2 are retained with the same definitions.

The following property shows that every obi-task τ_i^v has a minimum job-release separation of T_i.

Property 1. *For any obi-task* τ_i^v, $a_{i,j+1}^v - a_{i,j}^v \geq T_i$.

Proof.

$$\begin{aligned} a_{i,j+1}^v - a_{i,j}^v = \; & \{\text{by (7)}\} \\ & (a_{i,j+1}^1 + \Phi_i^v) - (a_{i,j}^1 + \Phi_i^v) \\ \geq \; & \{\text{by (6)}\} \\ & T_i \end{aligned}$$

□

4. RESPONSE-TIME BOUNDS

In this section, we establish two results that enable prior work to be leveraged to establish response-time bounds for DAG-based task systems. First, we show that, under the obi-task model with arbitrary offset settings, per-task response-time bounds can be derived by exploiting prior work pertaining to a task model called the *npc-sporadic task model* ("npc" stands for "no precedence constraints"—this refers to the lack of precedence constraints among jobs of the same task) [14, 27]. Second, we show that, by properly setting offsets, any DAG-based task system can be transformed to a corresponding obi-task system.

4.1 Response-Time Bounds for Obi-Tasks

An *npc-sporadic task* τ_i is specified by (C_i, T_i, D_i), where C_i is its WCET, T_i is the minimum separation time between consecutive job releases of τ_i, and D_i is its relative deadline. As before, τ_i's utilization is $u_i = C_i/T_i$.

The main difference between the conventional sporadic task model and the npc-sporadic task model is that the former requires successive jobs of each task to execute in sequence while the latter allows them to execute in parallel. That is, under the conventional sporadic task model, job $J_{i,j+1}$ cannot commence execution until its predecessor $J_{i,j}$ completes, even if $a_{i,j+1}$, the release time of $J_{i,j+1}$, has elapsed. In contrast, under the npc-sporadic task model, any job can execute as soon as it is released. Note that, although we allow intra-task parallelism, each individual job still must execute sequentially.

Yang and Anderson [27] investigated the G-EDF scheduling of npc-sporadic tasks on uniform heterogeneous multiprocessor platforms where different processors may have different speeds. By setting each processor's speed to be 1.0, the following theorem follows from their work.

Theorem 1. *(Follows from Theorem 4 in [27]) Consider the scheduling of a set of npc-sporadic tasks* τ *on* m *identical multiprocessors. Under non-preemptive G-EDF, each npc-task* $\tau_i \in \tau$

has the following response-time bound, provided $\sum_{\tau_l \in \tau} u_l \leq m$.

$$\frac{1}{m}\left(D_i \cdot \sum_{\tau_l \in \tau} u_l + \sum_{\tau_l \in \tau}\left(u_l \cdot \max_{\tau_l \in \tau}\{0, T_l - D_l\}\right)\right) + \max_{\tau_l \in \tau}\{C_l\} + \frac{m-1}{m}C_i.$$

We now show that Theorem 1 can be applied to obtain per-task response-time bounds for any obi-task set.

Concrete vs. non-concrete. A *concrete* sequence of job releases that satisfies a task's specification (under either the obi- or npc-sporadic task model) is called an *instantiation* of that task. An instantiation of a task *set* is defined similarly. In contrast, a task or a task set that can have multiple (potentially infinite) instantiations satisfying its specification (*e.g.*, minimum release separation) is called *non-concrete*.

By Property 1, *any* instantiation of an obi-task τ_i^v is an instantiation of the npc-sporadic task $\tau_i^v = (C_i^v, T_i, D_i^v)$. Hence, any instantiation of an obi-task set $\{\tau_i^v \mid P_i^v = \pi_k\}$ is an instantiation of the npc-sporadic task set $\{\tau_i^v \mid P_i^v = \pi_k\}$. Also, since obi-tasks execute independently of one another, obi-tasks executing in different CE pools cannot affect each other. Since each CE pool π_k has m_k identical processors, the problem we must consider is that of scheduling an instantiation of the npc-sporadic task set $\{\tau_i^v \mid P_i^v = \pi_k\}$ on m_k identical processors. Since Theorem 1 applies to a non-concrete npc-sporadic task set, it applies to every concrete instantiation of such a task set. Thus, we have the following response-time bound for each obi-task τ_i^v:

$$R_i^v = \frac{1}{m_k}\left(D_i^v \cdot \sum_{\tau_l^w \in \Gamma_k} u_l^w + \sum_{\tau_l^w \in \Gamma_k}(u_l^w \cdot \max\{0, T_l - D_l^w\})\right) + \max_{\tau_l^w \in \Gamma_k}\{C_l^w\} + \frac{m_k - 1}{m_k}C_i^v, \quad (9)$$

where $\Gamma_k = \{\tau_l^w \mid P_l^w = \pi_k\}$.

Note that (9) is applicable as long as all relative deadlines are non-negative, and applies assuming *any arbitrary* offset setting.

4.2 From DAG-Based Task Sets to Obi-Task Sets

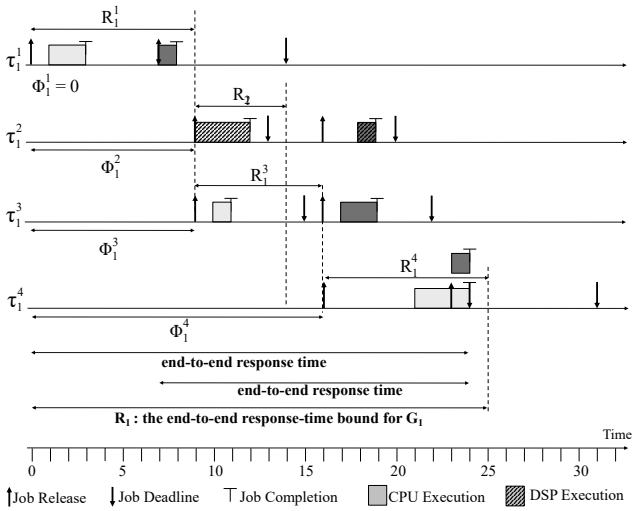
We now show that, by properly setting offsets, any DAG-based task set can be transformed to an obi-task set, and per-DAG end-to-end response-time bounds can be derived by leveraging the obi-task response-time bounds just stated.

Any DAG-based task set can be implemented by an obi-task set in an obvious way: each DAG becomes an obi-task group with the same task designated as its source, and all T_i, C_i^v, and P_i^v parameters are retained without modification. What is less obvious is how to define task offsets under the obi-task model. This is done by setting each Φ_i^v $(v \neq 1)$ parameter to be a *constant* such that

$$\Phi_i^v \geq \max_{\tau_i^k \in prod(\tau_i^v)}\{\Phi_i^k + R_i^k\}, \quad (10)$$

where $prod(\tau_i^v)$ denotes the set of obi-tasks corresponding to the DAG-based tasks that are the producers of the DAG-based task τ_i^v in G_i, and R_i^k denotes a *response-time bound* for the obi-task τ_i^k. For now, we assume that R_i^k is known, but later, we will show how to compute it.

Example 3. Consider again the DAG G_1 in Fig. 1. Assume that, after applying the above transformation, the obi-tasks have response-time bounds of $R_1^1 = 9$, $R_1^2 = 5$, $R_1^3 = 7$, and $R_1^4 = 9$, respectively. Then, we can set $\Phi_1^1 = 0$, $\Phi_1^2 = 9$, $\Phi_1^3 = 9$, and $\Phi_1^4 = 16$, respectively, and satisfy (10). With these response-time bounds, the end-to-end response-time bound that can be guaranteed is determined by R_1^1, R_1^3, and R_1^4 and is given by $R_1 = 25$. Fig. 3 depicts a possible schedule for these obi-tasks and illustrates the transformation. Like in Fig. 2, the first (resp., second) job of each task has a lighter (resp., darker) shading, and intra-task parallelism is possible (*e.g.*, $J_{1,1}^4$ and $J_{1,2}^4$ in time interval $[23, 24)$).



(Assume depicted jobs are scheduled alongside other jobs, which are not shown.)

Figure 3: Example schedule of the obi-tasks corresponding to the DAG-based tasks in G_1 in Fig. 1.

The following properties follow from this transformation process. According to Property 2, a DAG-based task set can be implemented by a corresponding set of obi-tasks, and all producer/consumer constraints in the DAG-based specification will be implicitly guaranteed, provided the offsets of the obi-tasks are properly set (*i.e.*, satisfy (10)).

Property 2. *If τ_i^k is a producer of τ_i^v in the DAG-based task system, then for the j^{th} jobs of the corresponding two obi-tasks, $f_{i,j}^k \leq a_{i,j}^v$.*

Proof. By (7), $a_{i,j}^k = a_{i,j}^1 + \Phi_i^k$, and by the definition of R_i^k, $f_{i,j}^k \leq a_{i,j}^k + R_i^k$. Thus,

$$f_{i,j}^k \leq a_{i,j}^1 + \Phi_i^k + R_i^k. \quad (11)$$

By (7), $a_{i,j}^v = a_{i,j}^1 + \Phi_i^v$, and by (10), $\Phi_i^v \geq \Phi_i^k + R_i^k$. Thus,

$$a_{i,j}^v \geq a_{i,j}^1 + \Phi_i^k + R_i^k. \quad (12)$$

By (11) and (12), $f_{i,j}^k \leq a_{i,j}^v$. □

Property 3 shows how to compute an end-to-end response-time bound R_i.

Property 3. *In the obi-task system, for each j, all jobs $J_{i,j}^1, J_{i,j}^2, \cdots, J_{i,j}^{n_i}$ finish their execution within R_i time units after a_i^1, where*

$$R_i = \Phi_i^{n_i} + R_i^{n_i}. \quad (13)$$

Proof. By (7) and the definition of R_i^v, $J_{i,j}^v$ finishes by time $a_{i,j}^1 + \Phi_i^v + R_i^v$. Thus, $J_{i,j}^{n_i}$ in particular finishes within $\Phi_i^{n_i} + R_i^{n_i} = R_i$ time units after a_i^1. Also, by (10), $\Phi_i^v + R_i^v \leq \Phi_i^{n_i}$, since $\tau_i^{n_i}$ is the *single* sink in G_i. Because $\Phi_i^{n_i} \leq R_i$, this implies that, for any v,

$J_{i,j}^v$ finishes within R_i time units after a_i^1. □

Thus, a DAG-based task set can be transformed to an obi-task set with the same per-task parameters. Given these per-task parameters, a response-time bound for each obi-task can be computed by (9) for *any* arbitrary offset setting. Then, we can properly set the offsets for each obi-task according to (10) by considering the corresponding tasks in each DAG in topological order [11], starting with $\Phi_i^1 = 0$ for each source task τ_i^1, by (8). By Property 2, the resulting obi-task set satisfies all requirements of the original DAG-based task system, and by Property 3, an end-to-end response-time bound R_i can be computed for each DAG G_i.

(Note that the response-time bound for a virtual source/sink is not computed by (9), but is zero by definition, since its WCET is zero. Any job of such a task completes in zero time as soon as it is released.)

5. SETTING RELATIVE DEADLINES

In the prior sections, we showed that, by applying our proposed transformation techniques, an end-to-end response-time bound for each DAG can be established, given arbitrary but fixed relative-deadline settings. That is, given $D_i^v \geq 0$ for any i, v, we can compute corresponding end-to-end response-time bounds (*i.e.*, R_i for each i) by (9), (8), (10), and (13).

Similar DAG transformation approaches have been presented previously [13, 20], but under the assumption that intra-task precedence constraints exist (*i.e.*, jobs of the same task must execute in sequence). Moreover, in this prior work, per-task relative deadlines have been defined in a DAG-oblivious way. By considering the actual structure of such a DAG, it may be possible to reduce its end-to-end response-time bound by setting its tasks' relative deadlines so as to favor certain critical paths.

Consider, for example, the DAG illustrated in Fig. 4. Suppose that the prior analysis yields a response-time bound of 10 for each task, as depicted within each node. The corresponding end-to-end response-time bound would then be 40 and is obtained by considering the right-side path. Now, suppose that we alter the tasks' relative-deadline settings to favor the tasks along this path at the possible expense of the remaining task on the left. Further, suppose this modification changes the per-task response-time bounds to be as depicted in parentheses. Then, this modification would have the impact of reducing the end-to-end bound to 32.

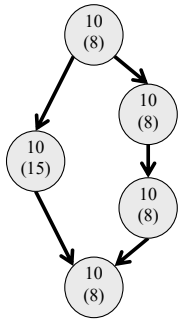


Figure 4: More highly prioritizing the right-side path in this DAG decreases its end-to-end response-time bound.

In this section, we show that the problem of determining the "best" relative-deadline settings can be cast as a linear-programming problem, which can be solved in polynomial time. The proposed linear program (LP) is developed in the next two subsections.

5.1 Linear Program

In our LP, there are three variables per task τ_i^v: D_i^v, Φ_i^v, and R_i^v. The parameters T_i, C_i^v, and m_k are viewed as constants. Thus, there are $3|V|$ variables in total, where $|V|$ is the total number of tasks (*i.e.*, nodes) across all DAGs in the system. Before stating the required constraints, we first establish the following theorem, which shows that a relative-deadline setting of $D_x^y > T_x$ is pointless to consider.

Theorem 2. *If $D_x^y > T_x$, then by setting $D_x^y = T_x$, R_x^y, the response-time bound of task τ_x^y, will decrease, and each other task's response-time bound will remain the same.*

Proof. To begin, note that, by (9), τ_x^y does not impact the response-time bounds of those tasks executing on CE pools other than P_x^y. Therefore, the response-time bounds $\{R_i^v \mid P_i^v \neq P_x^y\}$ for such tasks are not altered by any change to D_x^y.

In the remainder of the proof, we consider a task τ_i^v such that $P_i^v = P_x^y$. Let R_i^v and $R_i'^v$ denote the response-time bounds for τ_i^v before and after, respectively, reducing D_x^y to T_x. If $i = x$ and $v = y$, then by (9),

$$\begin{aligned} R_x^y - R_x'^y =& \frac{D_x^y - T_x}{m_k} \cdot \sum_{\tau_l^w \in \tau^k} u_l^w + \\ & \frac{u_x^y}{m_k}(\max\{0, T_x - D_x^y\} - \max\{0, T_x - T_x\}) \\ >& \{\text{since } D_x^y > T_x\} \\ & 0. \end{aligned}$$

Alternatively, if $i \neq x$ or $v \neq y$, then by (9),

$$\begin{aligned} R_i^v - R_i'^v =& \frac{u_x^y}{m_k}(\max\{0, T_x - D_x^y\} - \max\{0, T_x - T_x\}) \\ =& \{\text{since } D_x^y > T_x\} \\ & 0. \end{aligned}$$

Thus, the theorem follows. □

By Theorem 2, the reduction of D_x^y mentioned in the theorem does not increase the response-time bound for any task. By (10), this implies that none of the offsets, $\{\Phi_i^v\}$, needs to be increased. Therefore, by Property 3, no end-to-end response-time bound increases. These properties motivate our first set of linear constraints.

Constraint Set (i): For each task τ_i^v,

$$0 \leq D_i^v \leq T_i.$$

$2|V|$ individual linear inequalities arise from this constraint set, where $|V|$ is the total number of tasks.

Another issue we must address is that of ensuring that the offset settings, given by (10), are encoded in our LP. This gives rise to the next constraint set.

Constraint Set (ii): For each edge from τ_i^w to τ_i^v in a DAG G_i,

$$\Phi_i^v \geq \Phi_i^w + R_i^w.$$

There are $|E|$ distinct constraints in this set, where $|E|$ is the total number of edges in all DAGs in this system.

Finally, we have a set of constraints that are linear equality constraints.

Constraint Set (iii): With Constraints Set (i), it is clear that we can re-write (9) as follows, for each task τ_i^v,

$$\begin{aligned} R_i^v =& \frac{1}{m_k}\left(D_i^v \cdot \sum_{\tau_l^w \in \Gamma_k} u_l^w + \sum_{\tau_l^w \in \Gamma_k} \left(u_l^w \cdot (T_l - D_l^w)\right)\right) \\ & + \max_{\tau_l^w \in \Gamma_k} \{C_l^w\} + \frac{m_k - 1}{m_k} C_i^v, \end{aligned} \tag{14}$$

where $\Gamma_k = \{\tau_l^w | P_l^w = \pi_k\}$. Moreover, by (8), for each DAG G_i,

$$\Phi_i^1 = 0.$$

Constraint Set (iii) yields $|V| + |G|$ linear equations, where $|G|$ denotes the number of DAGs.

Constraint Sets (i), (ii), and (ii) fully specify our LP, with the exception of the objective function. In this LP, there are $3|V|$ variables, $2|V|+|E|$ inequality constraints, and $|V|+|G|$ linear equality constraints.

5.2 Objective Function

Different objective functions can be specified for our LP that optimize end-to-end response-time bounds in different senses. Here, we consider a few examples.

Single-DAG systems. For systems where only a single DAG exists, the optimization criterion is rather clear. In order to optimize the end-to-end response-time bound of the single DAG, the objective function should minimize the end-to-end response-time bound of the only DAG, G_1. That is, the desired LP is as follows.

$$\begin{array}{ll} \text{minimize} & \Phi_1^{n_1} + R_1^{n_1} \\ \text{subject to} & \text{Constraint Sets (i), (ii), and (iii)} \end{array}$$

Multiple-DAG systems. For systems containing multiple DAGs, choices exist as to the optimization criteria to consider. We list three here.

Minimizing the average end-to-end response-time bound:

$$\begin{array}{ll} \text{minimize} & \sum_i (\Phi_i^{n_i} + R_i^{n_i}) \\ \text{subject to} & \text{Constraint Sets (i), (ii), and (iii)} \end{array}$$

Minimizing the maximum end-to-end response-time bound:

$$\begin{array}{ll} \text{minimize} & Y \\ \text{subject to} & \forall i: \quad \Phi_i^{n_i} + R_i^{n_i} \leq Y \\ & \text{Constraint Sets (i), (ii), and (iii)} \end{array}$$

Minimizing the maximum proportional end-to-end response-time bound:

$$\begin{array}{ll} \text{minimize} & Y \\ \text{subject to} & \forall i: \quad (\Phi_i^{n_i} + R_i^{n_i})/T_i \leq Y \\ & \text{Constraint Sets (i), (ii), and (iii)} \end{array}$$

6. DAG COMBINING

In the application domain that motivates our work, the DAGs to be scheduled are typically of quite low utilizations and are defined based on a relatively small number of templates that define various computational patterns. Two DAGs defined using the same template are structurally identical: they are defined by graphs that are isomorphic, corresponding nodes from the two graphs perform identical computations, the source nodes are released at the same time, *etc.* Such structurally identical graphs can be combined into one graph with a reduced period and larger utilization, as long any overutilization of the underlying hardware platform is avoided. Such combining can be a very effective technique, because as the experiments presented later show, our response-time bounds tend to be proportional to periods.

We illustrate this idea with a simple example. Consider two DAGs G_1 and G_2 with a common period of T that are structurally identical. A schedule of these two DAGs is illustrated abstractly at the top of Fig. 5. As illustrated at the bottom of the figure, if these two DAGs are combined, then they are replaced by a structurally identical graph, denoted here as $G_{[1,2]}$, with a period of $T/2$. With this change, the provided response-time bounds have to be slightly adjusted. For example, if $G_{[1,2]}$ has a response-time bound of $R_{[1,2]}$, then this would also be a response-time bound for G_1, but that for G_2 would be $R_{[1,2]} + \frac{T}{2}$, because in combining the two graphs, the releases of G_2 are effectively shifted forward by $\frac{T}{2}$ time units. While this graph combining idea is really quite simple, the experiments presented later suggest that it can have a profound impact in the considered application domain. In particular, in that domain, per-DAG utilizations are low enough that upwards of 40 DAGs can be combined into one. Thus, the actual period reduction is not merely by a factor of $\frac{1}{2}$ but by a factor as high as $\frac{1}{40}$.

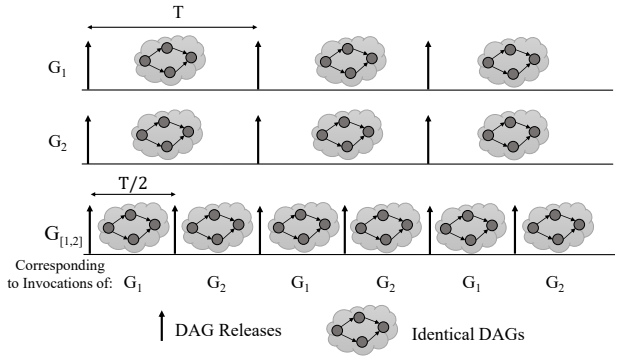


Figure 5: Illustration of DAG combining.

7. EARLY RELEASING

Transforming a DAG-based task system to a corresponding obi-task system enabled us to derive an end-to-end response-time *bound* for each DAG. However, such a transformation may actually cause *observed* end-to-end response times at runtime to increase, because the offsets introduced in the transformation may prevent a job from executing even if all of its producers have already finished. For example, in Fig. 3, $J_{1,1}^3$ cannot execute until time 9, even though its corresponding producer job, $J_{1,1}^1$, has finished by time 3.

Observed response times can be improved under deadline-based scheduling without altering analytical response-time bounds by using a technique called *early releasing* [12]. When early releasing is allowed, a job is eligible for execution as soon as all of its corresponding producer jobs have finished, even if this condition is satisfied before its actual release time.

Early releasing does not affect the response-time analysis for npc-sporadic tasks presented previously [27] because this analysis is based on the total demand for processing time due to jobs with deadlines at or before a particular time instant. Early releasing does not change upper bounds on such demand, because every job's actual release time and hence deadline are unaltered by early releasing. Thus, the response-time bounds and therefore the end-to-end response-time bounds previously established without early releasing still hold with early releasing.

Example 4. Considering G_1 in Fig. 1 again, Fig. 3 is a possible schedule, without early releasing, for the obi-tasks that implement G_1, as discussed earlier. When we allow early releasing, we do not change any release times or deadlines, but simply allow a job to become eligible for execution before its release time provided its producers have finished. Fig. 6 depicts a possible schedule where early releasing is allowed, assuming the same releases and deadlines as in Fig. 3. Several jobs (*e.g.*, $J_{1,1}^2$, $J_{1,2}^2$, $J_{1,2}^3$, $J_{1,1}^4$, and $J_{1,2}^4$) now commence execution before their release times. As a result, observed end-to-end response times are reduced, while still retaining all response-time bounds (per-task and end-to-end).

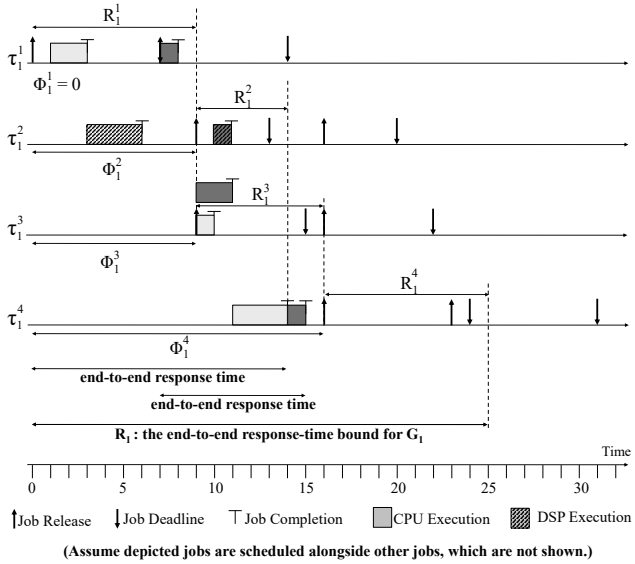


Figure 6: Example schedule of the obi-tasks corresponding to the DAG-based tasks in G_1 in Fig. 1, when early releasing is allowed.

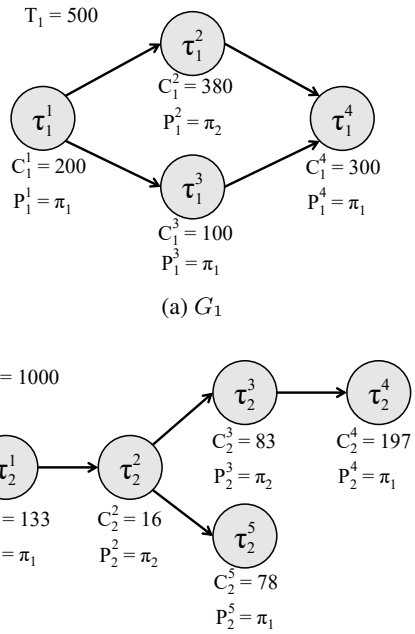


Figure 7: DAGs in the case-study system. G_2 has two sinks, so to analyze it, a virtual sink τ_2^6 must be added that has a WCET of 0 and a response-time bound of 0. We show the resulting graph in Fig. 8.

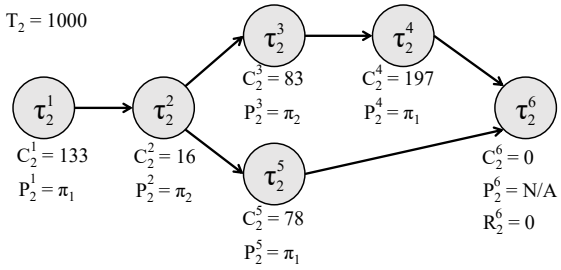


Figure 8: G_2', where a virtual sink is created for G_2.

8. CASE STUDY

To illustrate the computational details of our analysis, we consider here a case-study system consisting of three DAGs, G_1, G_2, and G_3, which are specified in Fig. 7. π_1 is a CE pool consisting of two identical CPUs, and π_2 is a CE pool consisting of two identical DSPs. Thus, $m_1 = m_2 = 2$. These three DAGs have fewer nodes and higher utilizations than typically found in our considered application domain. However, one can imagine that these graphs were obtained from combining many identical graphs of lower utilization. While it would have been desirable to consider larger graphs with more nodes, graphs from our chosen domain typically have tens of nodes, and this makes them rather unwieldy to discuss. Still, the general conclusions we draw here are applicable to larger graphs.

Utilization check. First, we must calculate the total utilization of all tasks assigned to each CE pool to make sure that neither is overutilized. We have $\sum_{\tau_i^v \in \Gamma_1} u_i^v = (200 + 100 + 300)/500 + (133+78+197+73+5)/1000 = 1.686 < 2$, and $\sum_{\tau_i^v \in \Gamma_2} u_i^v = 380/500 + (16 + 83 + 242)/1000 = 1.101 < 2$.

Virtual source/sink. Note that all DAGs have a single source and sink, except for G_2, which has two sinks. For it, we connect its two sinks to a single virtual sink τ_2^6, which has a WCET of 0 and a response-time bound of 0. We call the resulting DAG G_2', which is shown in Fig. 8.

Implicit deadlines. We now show how to compute response-time bounds assuming implicit deadines, *i.e.*, $D_1^v = 500$ for $1 \leq v \leq 4$, $D_2^v = 1000$ for $1 \leq v \leq 5$ (the relative deadline of the virtual sink is irrelevant), and $D_3^v = 1000$ for $1 \leq v \leq 3$. In order to derive an end-to-end response-time bound, we first transform the original DAG-based tasks into obi-tasks as described in Sec. 3. Next, we calculate a response-time bound R_i^v for each obi-task τ_i^v by (9). The resulting task response-time bounds, $\{R_i^v\}$, are listed in Table 1. Note that, as a virtual sink, the response-time bound for the virtual sink τ_2^6 does not need to be computed by (9), but is 0 by definition. By (8) and (10), the offsets of the obi-tasks can now be computed in topological order with respect to each DAG. The resulting offsets, $\{\Phi_i^v\}$, are also shown in Table 1. Finally, by Property 3, we have an end-to-end response-time bound for each DAG: $R_1 = \Phi_1^4 + R_1^4 = 2538.25$, $R_2 = \Phi_2^6 + R_2^6 = 4361.5$, and $R_3 = \Phi_3^3 + R_3^3 = 3376.5$.

LP-based deadline settings. If we use LP techniques to optimize end-to-end response-time bounds, then choices exist regarding the objective function, because our system has multiple DAGs. We consider three choices here.

Minimizing the average end-to-end response-time bound. For this choice, relative-deadline settings, obi-task response-time bounds, and obi-task offsets are as shown in Table 2(a). The resulting end-to-end response-time bounds are $R_1 = \Phi_1^4 + R_1^4 = 3134.5$, $R_2 = \Phi_2^6 + R_2^6 = 2341.2$, and $R_3 = \Phi_3^3 + R_3^3 = 1736.2$.

R_1^1	R_1^2	R_1^3	$\mathbf{R_1^4}$	R_2^1	R_2^2	R_2^3	R_2^4	R_2^5	$\mathbf{R_2^6}$	R_3^1	R_3^2	$\mathbf{R_3^3}$
821.5	845.25	771.5	**871.5**	1209.5	938.5	972	1241.5	1182	**0**	1179.5	1051.5	**1145.5**
Φ_1^1	Φ_1^2	Φ_1^3	$\mathbf{\Phi_1^4}$	Φ_2^1	Φ_2^2	Φ_2^3	Φ_2^4	Φ_2^5	$\mathbf{\Phi_2^6}$	Φ_3^1	Φ_3^2	$\mathbf{\Phi_3^3}$
0	821.5	821.5	**1666.75**	0	1209.5	2148	3120	2148	**4361.5**	0	1179.5	**2231**

Table 1: Case-study task response-time bounds and obi-task offsets assuming implicit deadlines. Bold entries denote sinks.

D_1^1	D_1^2	D_1^3	$\mathbf{D_1^4}$	D_2^1	D_2^2	D_2^3	D_2^4	D_2^5	$\mathbf{D_2^6}$	D_3^1	D_3^2	$\mathbf{D_3^3}$
500	500	500	**500**	0	0	0	0	772.84	**0**	0	0	**0**
R_1^1	R_1^2	R_1^3	$\mathbf{R_1^4}$	R_2^1	R_2^2	R_2^3	R_2^4	R_2^5	$\mathbf{R_2^6}$	R_3^1	R_3^2	$\mathbf{R_3^3}$
1034.4	1015.7	984.36	**1084.4**	579.36	558.5	592	611.36	1203.4	**0**	549.36	671.5	**515.36**
Φ_1^1	Φ_1^2	Φ_1^3	$\mathbf{\Phi_1^4}$	Φ_2^1	Φ_2^2	Φ_2^3	Φ_2^4	Φ_2^5	$\mathbf{\Phi_2^6}$	Φ_3^1	Φ_3^2	$\mathbf{\Phi_3^3}$
0	1034.4	1049.2	**2050.1**	0	579.36	1137.9	1729.9	1137.9	**2341.2**	0	549.36	**1220.9**

(a)

D_1^1	D_1^2	D_1^3	$\mathbf{D_1^4}$	D_2^1	D_2^2	D_2^3	D_2^4	D_2^5	$\mathbf{D_2^6}$	D_3^1	D_3^2	$\mathbf{D_3^3}$
0	500	359.06	**500**	0	0	0	584.52	1000	**0**	505.63	1000	**0**
R_1^1	R_1^2	R_1^3	$\mathbf{R_1^4}$	R_2^1	R_2^2	R_2^3	R_2^4	R_2^5	$\mathbf{R_2^6}$	R_3^1	R_3^2	$\mathbf{R_3^3}$
642.06	894.75	894.75	**1113.6**	608.56	437.5	471	1133.3	1424.1	**0**	1004.8	1101	**544.56**
Φ_1^1	Φ_1^2	Φ_1^3	$\mathbf{\Phi_1^4}$	Φ_2^1	Φ_2^2	Φ_2^3	Φ_2^4	Φ_2^5	$\mathbf{\Phi_2^6}$	Φ_3^1	Φ_3^2	$\mathbf{\Phi_3^3}$
0	642.06	642.06	**1536.8**	0	608.56	1046.1	1517.1	1128.8	**2650.4**	0	1004.8	**2105.8**

(b)

D_1^1	D_1^2	D_1^3	$\mathbf{D_1^4}$	D_2^1	D_2^2	D_2^3	D_2^4	D_2^5	$\mathbf{D_2^6}$	D_3^1	D_3^2	$\mathbf{D_3^3}$
0	0	200.53	**0**	1000	0	253.21	1000	1000	**0**	1000	1000	**1000**
R_1^1	R_1^2	R_1^3	$\mathbf{R_1^4}$	R_2^1	R_2^2	R_2^3	R_2^4	R_2^5	$\mathbf{R_2^6}$	R_3^1	R_3^2	$\mathbf{R_3^3}$
679.95	798.99	798.99	**729.95**	1489.4	616.99	789.89	1521.4	1461.9	**0**	1459.4	1280.5	**1425.4**
Φ_1^1	Φ_1^2	Φ_1^3	$\mathbf{\Phi_1^4}$	Φ_2^1	Φ_2^2	Φ_2^3	Φ_2^4	Φ_2^5	$\mathbf{\Phi_2^6}$	Φ_3^1	Φ_3^2	$\mathbf{\Phi_3^3}$
0	679.95	679.95	**1478.9**	0	1489.4	2106.4	2896.3	2552.7	**4417.8**	0	1508.3	**2835.6**

(c)

Table 2: Case-study relative-deadline settings, obi-task response-time bounds, and obi-task offsets when using linear programming to **(a)** minimize average end-to-end response-time bounds, **(b)** minimize maximum end-to-end response-time bounds, and **(c)** minimize maximum proportional end-to-end response-time bounds. Bold entries denote sinks.

Minimizing the maximum end-to-end response-time bound. For this choice, relative-deadline settings, obi-task response-time bounds, and obi-task offsets are as shown in Table 2(b). The resulting end-to-end response-time bounds are $R_1 = \Phi_1^4 + R_1^4 = 2650.4$, $R_2 = \Phi_2^6 + R_2^6 = 2650.4$, and $R_3 = \Phi_3^3 + R_3^3 = 2650.4$.

Minimizing the maximum proportional end-to-end response-time bound. For this choice, relative-deadline settings, obi-task response-time bounds, and obi-task offsets are as shown in Table 2(c). The resulting end-to-end response-time bounds are $R_1 = \Phi_1^4 + R_1^4 = 2208.9$, $R_2 = \Phi_2^6 + R_2^6 = 4417.8$, and $R_3 = \Phi_3^3 + R_3^3 = 4261.0$.

Early releasing. As discussed in Sec. 7, early releasing can improve observed response times without compromising response-time bounds. The value of allowing early releasing can be seen in the results reported in Table 3. This table gives the largest observed end-to-end response time of each DAG in Fig. 7, assuming implicit deadlines with and without early releasing, in a schedule that was simulated for 50,000 time units. Analytical bounds are shown as well.

	G_1	G_2	G_3
Early releasing	1006	897	453
No early releasing	1966.75	3536.25	2586.0
Bounds	2538.25	4361.5	3376.5

Table 3: Observed end-to-end response times with/without early releasing and analytical end-to-end response-time bounds for the implicit-deadline setting.

9. SCHEDULABILITY STUDIES

In this section, we expand upon the specific case study just described by considering general schedulability trends seen in experiments involving randomly generated task systems.

9.1 Improvements Enabled by Basic Techniques

We first consider the improvements enabled by the basic techniques covered in Secs. 4 and 5 that underlie our work: allowing intra-task parallelism as provided by the npc-sporadic task model, and determining relative-deadline settings by solving an LP.

Random system generation. In our experiments, we considered a heterogeneous platform comprised of three CE pools, each consisting of eight identical CEs. Each pool was assumed to have the same total utilization. We considered all choices of total per-pool utilizations in the range $[1, 8]$ in increments of 0.5.

We generated DAG-based task systems using a method similar to that used by others [4, 17]. These systems were generated by first specifying the number of DAGs in the system, N, and the number of tasks per DAG, n. For each considered pair N and n, we randomly generated 50 *task-system structures*, each comprised of N DAGs with n nodes. Each node in such a structure was randomly assigned to one of the CE pools, and for each DAG in the structure, one node was designated as its source, and one as its sink. Further, each pair of internal nodes (not a source or a sink) was connected

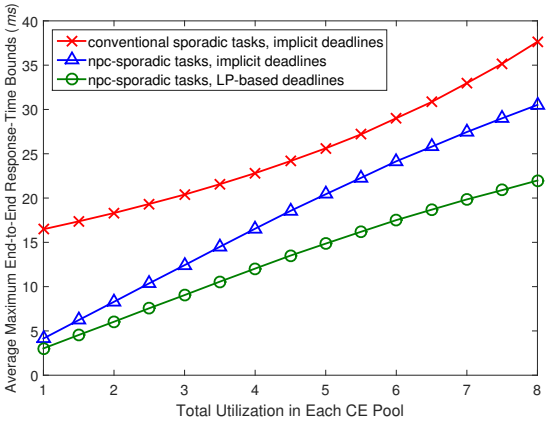


Figure 9: AMERBs as a function of total utilization in each CE pool in the case where each task set has five DAGs, 20 tasks per DAG, and `edgeProb`=0.5.

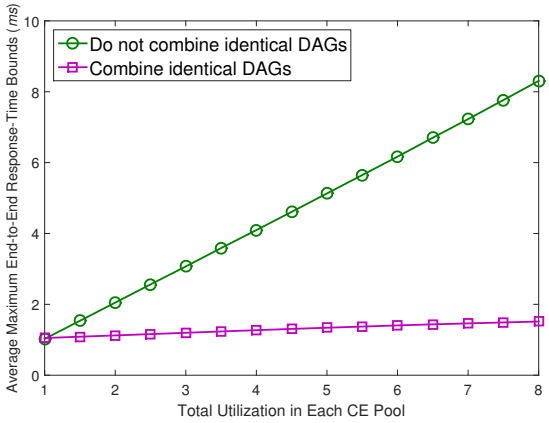


Figure 10: AMERBs as a function of total utilization in each CE pool in the case where the number of identical DAGs per template is fixed to 40.

by an edge with probability `edgeProb`, a settable parameter. Such an edge was directed from the lower-indexed node to the higher-indexed node, to preclude cycles. Finally, an edge was added from the source to each internal node with no incoming edges, and to the sink from each internal node with no outgoing edges.

For each considered per-pool untilization and each generated task-system structure, we randomly generated 50 actual task systems by generating task utilizations using the MATLAB function `randfixedsum()` [25]. According to the application domain that motivates this work,[2] we defined each DAG's period to be $1\ ms$. (A task's WCET is determined by its utilization and period.) For each considered value of N, n, and total per-pool utilization (one point in one of our graphs), we considered 50 (task system structures) $\times$ 50 (utilizations) $= 2,500$ task sets.

Comparison setup. We compared three strategies: **(i)** transforming to a conventional sporadic task system and using implicit relative deadlines, which is a strategy used in prior work on identical platforms [20]; **(ii)** transforming to an npc-sporadic task system and using implicit relative deadlines; and **(iii)** transforming to an npc-sporadic task system and using LP-based relative deadlines. When applying our LP techniques, we chose the objective function that minimizes the maximum end-to-end response-time bound. Although an identical platform was assumed in [20], the techniques from that paper can be extended to heterogeneous platforms in a similar way to this paper.

Results. In all cases that we considered, the two evaluated techniques improved end-to-end response-time bounds, often significantly. Due to space constraints, we present here only the case where $N = 5$, $n = 20$, and `edgeProb` $= 0.5$. For each generated task set, we recorded the maximum end-to-end response-time bound among its five DAGs. For each given total per-pool utilization point, we report here the average of the maximum end-to-end response-time bounds among the $2,500$ task sets generated for that point. We call this metric the *average maximum end-to-end response-time bound (AMERB)*. Fig. 9 plots AMERBs as a function of total per-CE-pool utilization. As seen, the application of both techniques reduced AMERBs by 39.42% to 81.65%.

In addition to this plot, we also considered other cases with different values for N, n, and `edgeProb`. These other results show similar trends and can be found in an online appendix (available at http://cs.unc.edu/~anderson/papers.html).

[2]In applications usually considered in the real-time-systems community, much larger periods are the norm. The considered domain is quite different.

9.2 Improvements Enabled by DAG Combining

As mentioned in Sec. 6, in the application domain that motivates our work, DAGs are usually defined using several well-defined computational templates, and as a result, many identical DAGs will exist. We proposed the technique of DAG combining in Sec. 6 to exploit this fact to further reduce response-time bounds. We now discuss schedulability experiments that we conducted to evaluate this technique.

Random system generation. We employed a process of randomly generating systems that is similar to that discussed in Sec. 9.1, except that, instead of generating task-system structures comprised of N DAGs, we generated structures comprised of N *templates*. Additionally, we introduced a new parameter K that indicates the number of identical DAGs per template. A period of $1\ ms$ was still associated with each DAG.

Comparison setup. We compared two strategies: **(i)** do no combining, and compute end-to-end response-time bounds assuming $N \cdot K$ independent DAGs; **(ii)** combine identical DAGs, and compute end-to-end response-time bounds assuming N DAGs, making adjustments as discussed in Sec. 6 to obtain actual response-time bounds for the DAGs that were combined. Under both strategies, the general techniques evaluated in Sec. 9.1 were applied.

Results. In all cases that we considered, the DAG combining technique improved end-to-end response-time bounds significantly. Due to space constraints, we present here only the case where each system has five templates, each of which has 20 nodes, and `edgeProb` $= 0.5$. Other results can be found online. For the considered case, Fig. 10 plots AMERBs as a function of total per-pool utilization, when the number of identical DAGs per template is fixed to 40 (this number is close to what would be expected in the application domain that motivates this work). Note that the AMERBs in Fig. 10 are much lower than those in Fig. 9, even before applying the DAG combining technique. This is because the systems considered in Fig. 10 have far more DAGs than those in Fig. 9. As a result, for each given total per-pool utilization, the systems in Fig. 10 have much lower per-DAG and per-task utilizations. As also seen in Fig. 10, when DAG combining is applied, the AMERBs are not very much influenced by the total per-CE-pool utilization. That is because DAG combining resulted in quite

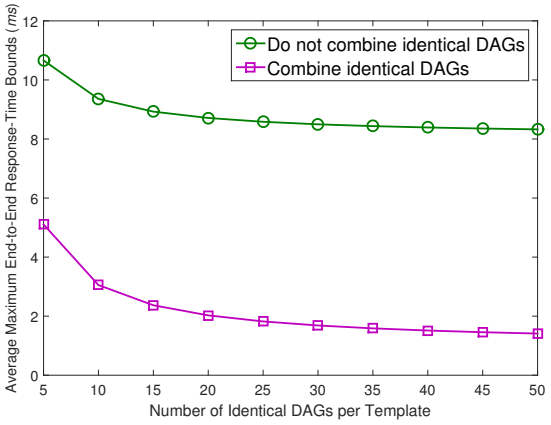


Figure 11: AMERBs as a function of the number of identical DAGs per template in the case where total utilization in each CE pool is fixed to eight.

small response-time bounds by (9), so total end-to-end bounds were mainly impacted by the introduced shifting, rather than total per-CE-pool utilization. Also, Fig. 11 plots AMERBs as a function of the number of identical DAGs per template, when every CE pool is fully utilized (*i.e.*, the total utilization of each pool is eight). In this case, the AMERB metric was calculated over all task sets that have the same number of identical DAGs per template.

According to our industry partners, in the considered application domain, a DAG's end-to-end response-time bound should typically be at most $2.35\ ms$. As observed in Fig. 10, in the absence of DAG combining, AMERBs in this experiment were as high as $8.2\ ms$. However, the introduction of DAG combining enabled a drop to less than $2.0\ ms$, even when the platform was fully utilized. This demonstrates that DAG combining—as simple as it may seem—can have a powerful impact in the targeted domain.

10. CONCLUSION

We presented task-transformation techniques to provide end-to-end response-time bounds for DAG-based tasks implemented on heterogenous multiprocessor platforms where intra-task parallelism is allowed. We also presented an LP-based method for setting relative deadlines and a DAG combining technique that can be applied to improve these bounds. We evaluated the efficacy of these results by considering a case-study task system and by conducting schedulability studies. To our knowledge, this paper is the first to present end-to-end response-time analysis for the considered context. In future work, we intend to extend these techniques to deal with synchronization requirements and to incorporate methods for limiting interference caused by contention for shared hardware components such as caches, memory banks, *etc.*

Acknowledgments: We thank Alan Gatherer, Lee McFearin, and Peter Yan for bringing the problem studied in this paper to our attention and for answering many questions regarding cellular base stations.

References

[1] H. Ali, B. Akesson, and L. Pinho. Generalized extraction of real-time parameters for homogeneous synchronous dataflow graphs. In *23rd PDP*, 2015.

[2] P. Axer, S. Quinton, M. Neukirchner, R. Ernst, B. Dobel, and H. Hartig. Response-time analysis of parallel fork-join workloads with real-time constraints. In *25th ECRTS*, 2013.

[3] R. Bajaj and D. Agrawal. Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques. *IEEE Transactions on Parallel and Distributed Systems*, 15(2):107–118, 2004.

[4] S. Baruah. Improved multiprocessor global schedulability analysis of sporadic DAG task systems. In *26th ECRTS*, 2014.

[5] S. Baruah. The federated scheduling of constrained-deadline sporadic DAG task systems. In *18th DATE*, 2015.

[6] S. Baruah. Federated scheduling of sporadic DAG task systems. In *29th IPDPS*, 2015.

[7] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese. A generalized parallel task model for recurrent real-time processes. In *33rd RTSS*, 2012.

[8] big.LITTLE Processing. http://www.arm.com/products/processors/technologies/biglittleprocessing.php.

[9] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese. Feasibility analysis in the sporadic DAG task model. In *25th ECRTS*, 2013.

[10] H. Chwa, J. Lee, K. Phan, A. Easwaran, and I. Shin. Global EDF schedulability analysis for synchronous parallel tasks on multicore platforms. In *25th ECRTS*, 2013.

[11] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2009.

[12] U. Devi. *Soft Real-Time Scheduling on Multiprocessors*. PhD thesis, University of North Carolina, Chapel Hill, NC, 2006.

[13] G. Elliott, N. Kim, J. Erickson, C. Liu, and J. Anderson. Minimizing response times of automotive dataflows on multicore. In *20th RTCSA*, 2014.

[14] J. Erickson and J. Anderson. Response time bounds for G-EDF without intra-task precedence constraints. In *15th OPODIS*, 2011.

[15] J. Fonseca, V. Nelis, G. Raravi, and L. Pinho. A multi-DAG model for real-time parallel applications with conditional execution. In *30th SAC*, 2015.

[16] T. Grandpierre, C. Lavarenne, and Y. Sorel. Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors. In *7th CODES*, 1999.

[17] J. Li, K. Agrawal, C. Lu, and C. Gill. Analysis of global EDF for parallel tasks. In *25th ECRTS*, 2013.

[18] J. Li, A. Saifullah, K. Agrawal, C. Gill, and C. Lu. Analysis of federated and global scheduling for parallel real-time tasks. In *26th ECRTS*, 2014.

[19] J. Li, A. Saifullah, K. Agrawal, C. Gill, and C. Lu. Capacity augmentation bound of federated scheduling for parallel DAG tasks. Technical report, Washington University in St Louis, 2014.

[20] C. Liu and J. Anderson. Supporting soft real-time DAG-based systems on multiprocessors with no utilization loss. In *31st RTSS*, 2010.

[21] C. Maia, M. Bertogna, L. Nogueira, and L. Pinho. Global EDF schedulability analysis for synchronous parallel tasks on multicore platforms. In *22nd RTNS*, 2014.

[22] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. Buttazzo. Response-time analysis of conditional DAG tasks in multiprocessor systems. In *27th ECRTS*, 2015.

[23] A. Parri, A. Biondi, and M. Marinoni. Response time analysis for G-EDF and G-DM scheduling of sporadic DAG-tasks with arbitrary deadline. In *23rd RTNS*, 2015.

[24] A Saifullah, K. Agrawal, C. Lu, and C. Gill. Multi-core real-time scheduling for generalized parallel task models. In *32nd RTSS*, 2011.

[25] R. Stafford. Random vectors with fixed sum. http://www.mathworks.com/matlabcentral/fileexchange/9700-random-vectors-with-fixed-sum.

[26] G. Stavrinides and H. Karatza. Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques. *Simulation Modelling Practice and Theory*, 19(1):540–552, 2011.

[27] K. Yang and J. Anderson. Optimal GEDF-based schedulers that allow intra-task parallelism on heterogeneous multiprocessors. In *ESTIMedia*, 2014.

APPENDIX: ADDITIONAL GRAPHS

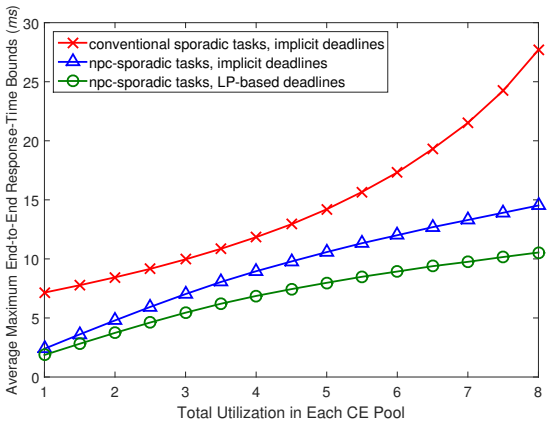


Figure 12: AMERBs as a function of total utilization in each CE pool in the case where each task set has five DAGs, ten tasks per DAG, and `edgeProb`=0.2.

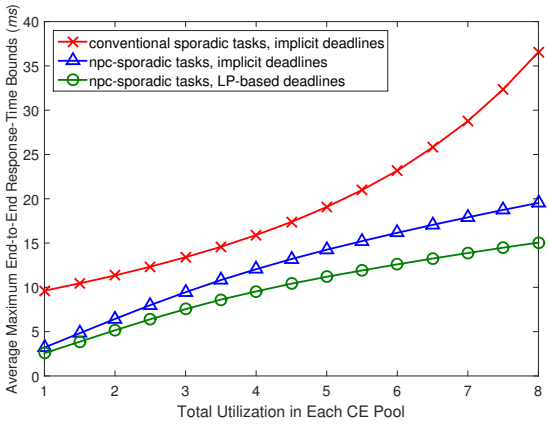


Figure 13: AMERBs as a function of total utilization in each CE pool in the case where each task set has five DAGs, ten tasks per DAG, and `edgeProb`=0.5.

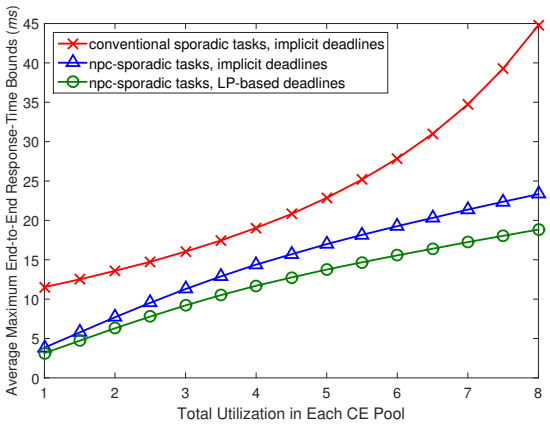


Figure 14: AMERBs as a function of total utilization in each CE pool in the case where each task set has five DAGs, ten tasks per DAG, and `edgeProb`=0.8.

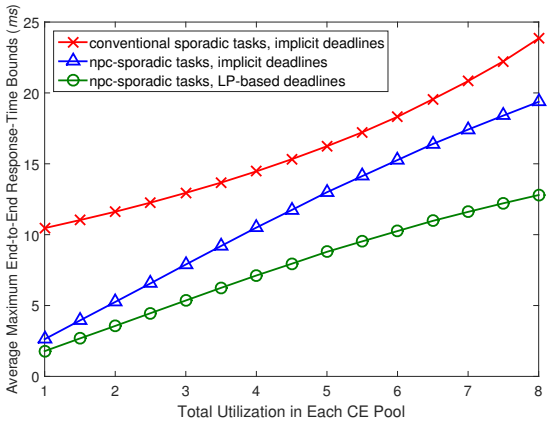


Figure 15: AMERBs as a function of total utilization in each CE pool in the case where each task set has five DAGs, 20 tasks per DAG, and `edgeProb`=0.2.

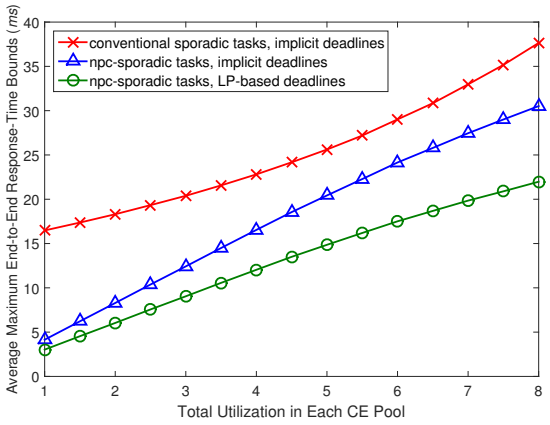


Figure 16: AMERBs as a function of total utilization in each CE pool in the case where each task set has five DAGs, 20 tasks per DAG, and `edgeProb`=0.5.

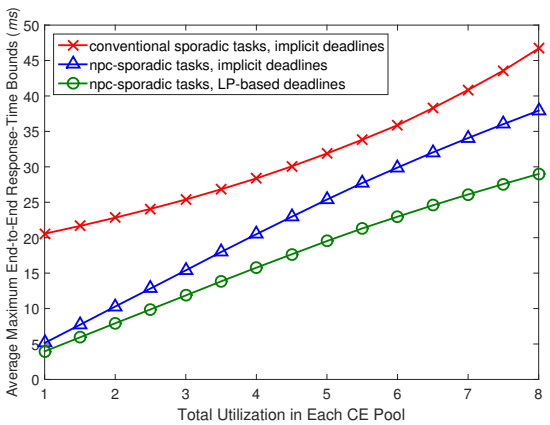


Figure 17: AMERBs as a function of total utilization in each CE pool in the case where each task set has five DAGs, 20 tasks per DAG, and `edgeProb`=0.8.

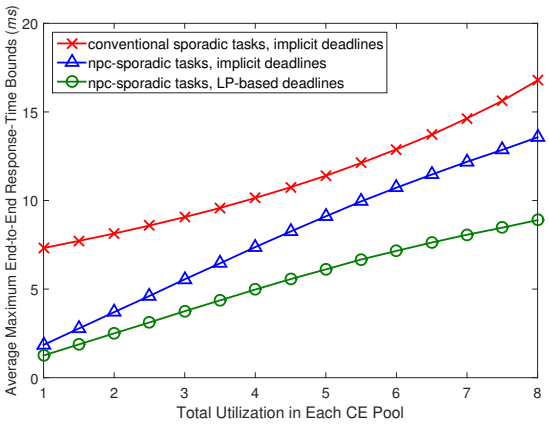


Figure 18: AMERBs as a function of total utilization in each CE pool in the case where each task set has ten DAGs, ten tasks per DAG, and `edgeProb`=0.2.

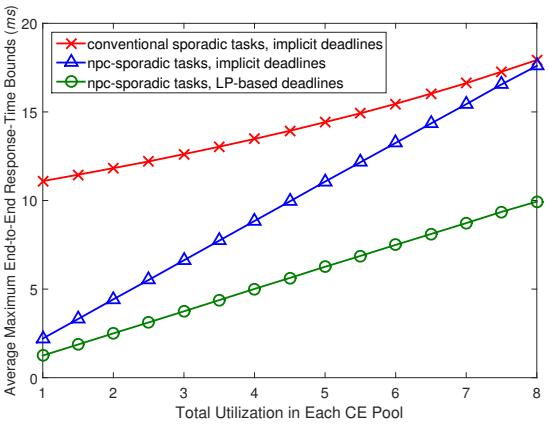


Figure 21: AMERBs as a function of total utilization in each CE pool in the case where each task set has ten DAGs, 20 tasks per DAG, and `edgeProb`=0.2.

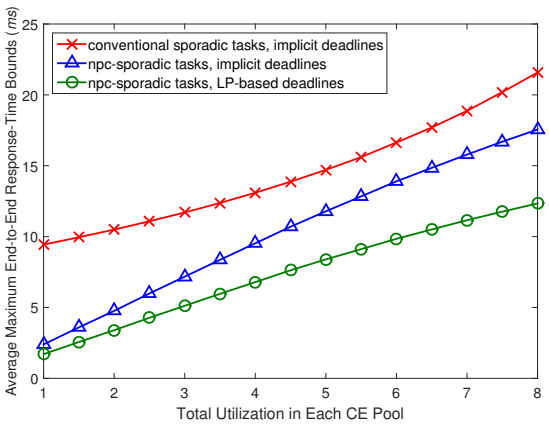


Figure 19: AMERBs as a function of total utilization in each CE pool in the case where each task set has ten DAGs, ten tasks per DAG, and `edgeProb`=0.5.

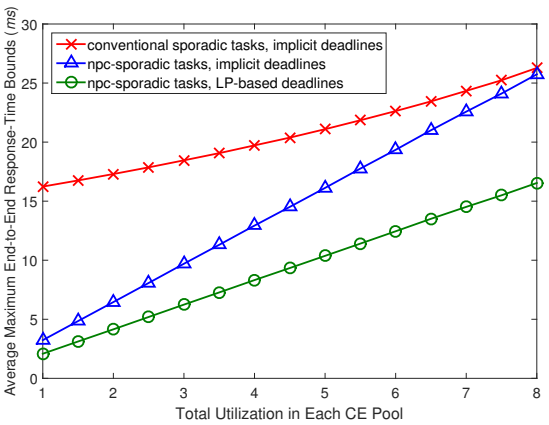


Figure 22: AMERBs as a function of total utilization in each CE pool in the case where each task set has ten DAGs, 20 tasks per DAG, and `edgeProb`=0.5.

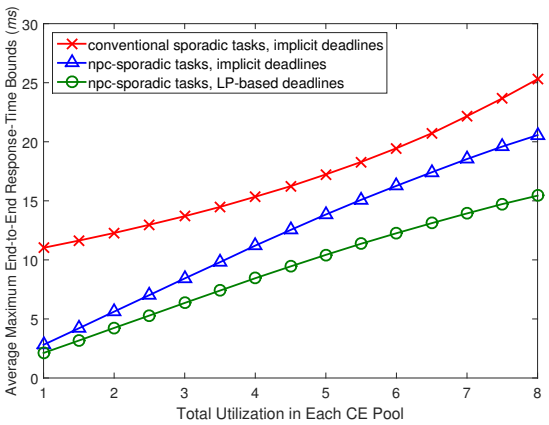


Figure 20: AMERBs as a function of total utilization in each CE pool in the case where each task set has ten DAGs, ten tasks per DAG, and `edgeProb`=0.8.

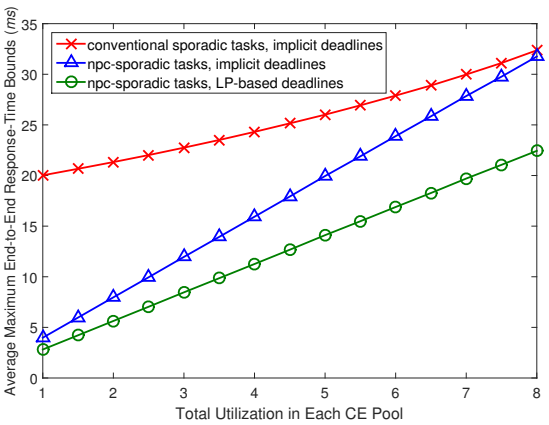


Figure 23: AMERBs as a function of total utilization in each CE pool in the case where each task set has ten DAGs, 20 tasks per DAG, and `edgeProb`=0.8.

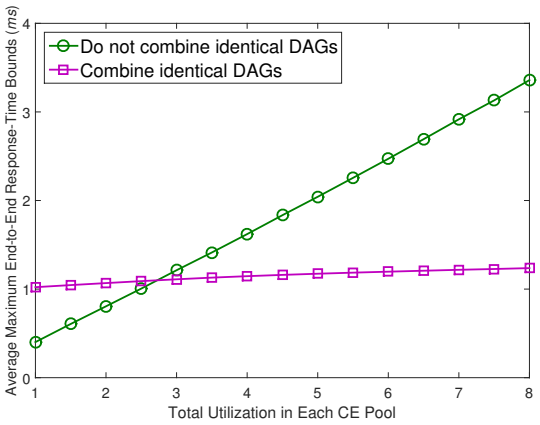


Figure 24: AMERBs as a function of total utilization in each CE pool in the case where the number of identical DAGs per template is fixed to 40, each task set has five templates, each DAG has ten tasks, and `edgeProb`=0.2.

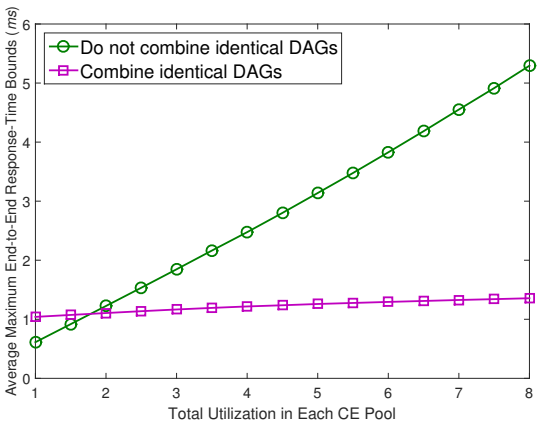


Figure 25: AMERBs as a function of total utilization in each CE pool in the case where the number of identical DAGs per template is fixed to 40, each task set has five templates, each DAG has ten tasks, and `edgeProb`=0.5.

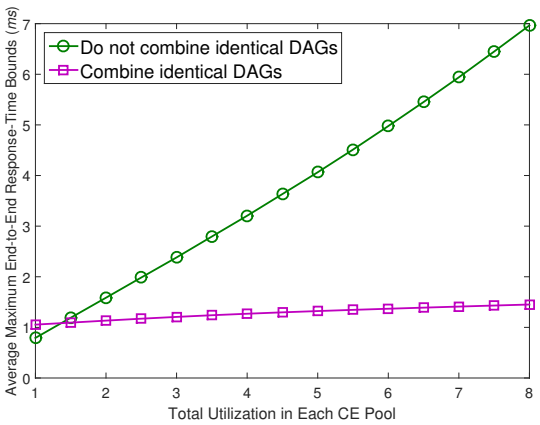


Figure 26: AMERBs as a function of total utilization in each CE pool in the case where the number of identical DAGs per template is fixed to 40, each task set has five templates, each DAG has ten tasks, and `edgeProb`=0.8.

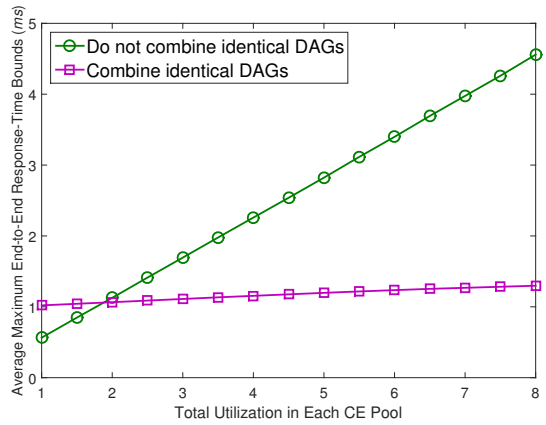


Figure 27: AMERBs as a function of total utilization in each CE pool in the case where the number of identical DAGs per template is fixed to 40, each task set has five templates, each DAG has 20 tasks, and `edgeProb`=0.2.

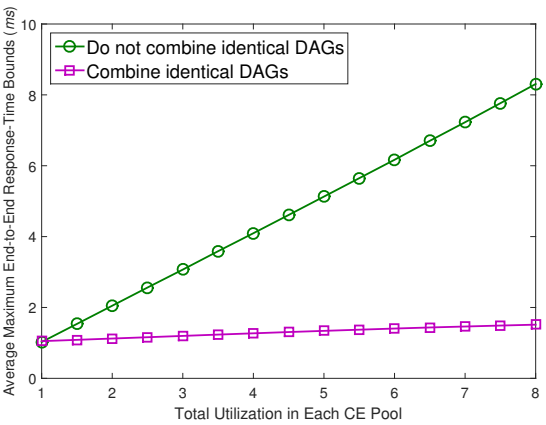


Figure 28: AMERBs as a function of total utilization in each CE pool in the case where the number of identical DAGs per template is fixed to 40, each task set has five templates, each DAG has 20 tasks, and `edgeProb`=0.5.

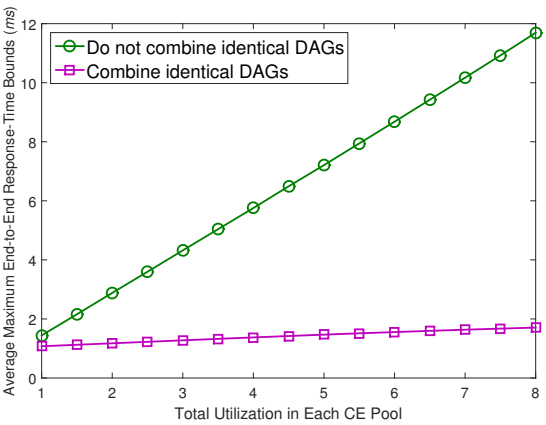


Figure 29: AMERBs as a function of total utilization in each CE pool in the case where the number of identical DAGs per template is fixed to 40, each task set has five templates, each DAG has 20 tasks, and `edgeProb`=0.8.

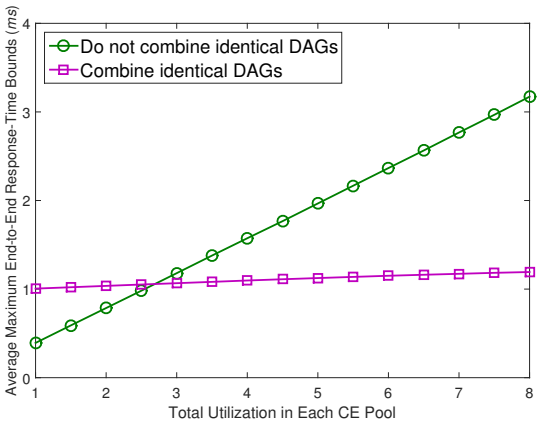


Figure 30: AMERBs as a function of total utilization in each CE pool in the case where the number of identical DAGs per template is fixed to 40, each task set has ten templates, each DAG has ten tasks, and `edgeProb`=0.2.

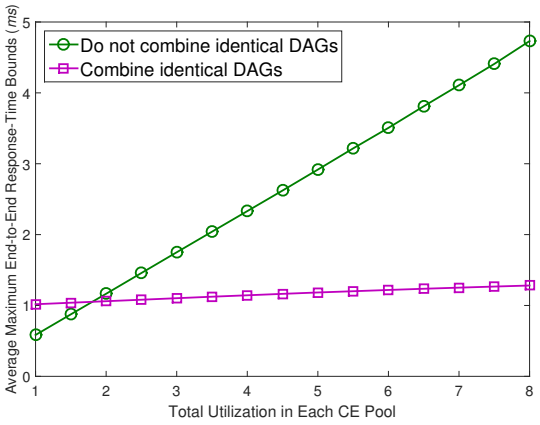


Figure 31: AMERBs as a function of total utilization in each CE pool in the case where the number of identical DAGs per template is fixed to 40, each task set has ten templates, each DAG has ten tasks, and `edgeProb`=0.5.

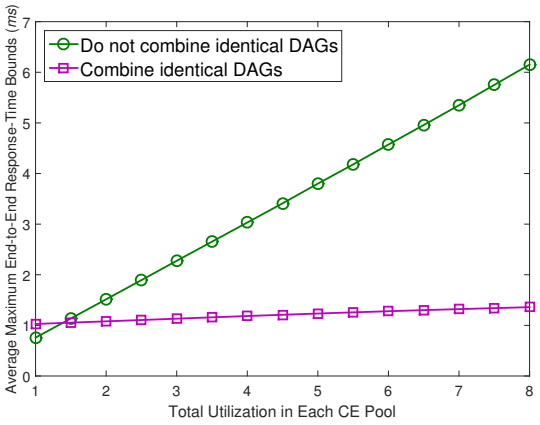


Figure 32: AMERBs as a function of total utilization in each CE pool in the case where the number of identical DAGs per template is fixed to 40, each task set has ten templates, each DAG has ten tasks, and `edgeProb`=0.8.

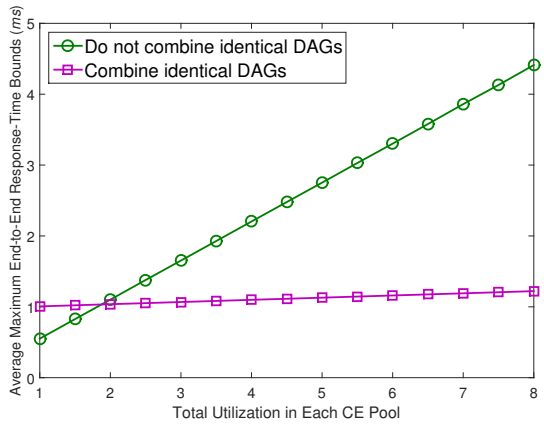


Figure 33: AMERBs as a function of total utilization in each CE pool in the case where the number of identical DAGs per template is fixed to 40, each task set has ten templates, each DAG has 20 tasks, and `edgeProb`=0.2.

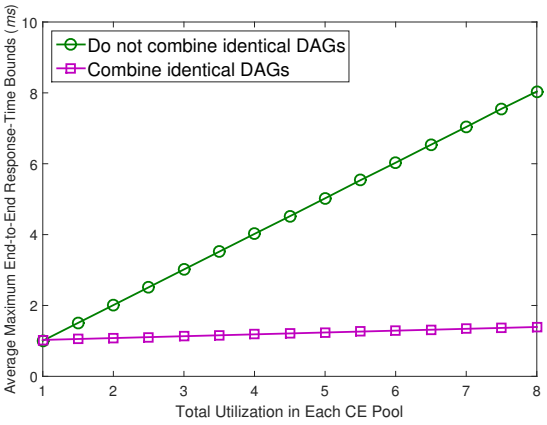


Figure 34: AMERBs as a function of total utilization in each CE pool in the case where the number of identical DAGs per template is fixed to 40, each task set has ten templates, each DAG has 20 tasks, and `edgeProb`=0.5.

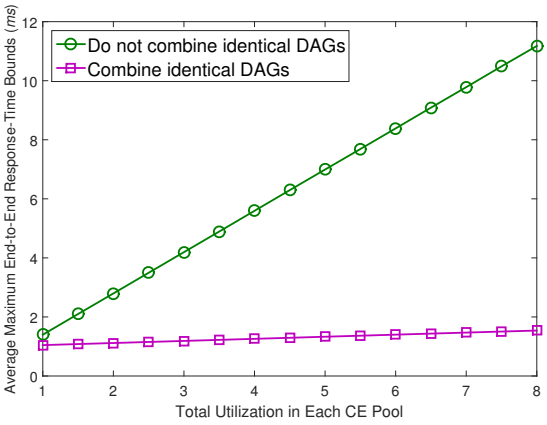


Figure 35: AMERBs as a function of total utilization in each CE pool in the case where the number of identical DAGs per template is fixed to 40, each task set has ten templates, each DAG has 20 tasks, and `edgeProb`=0.8.

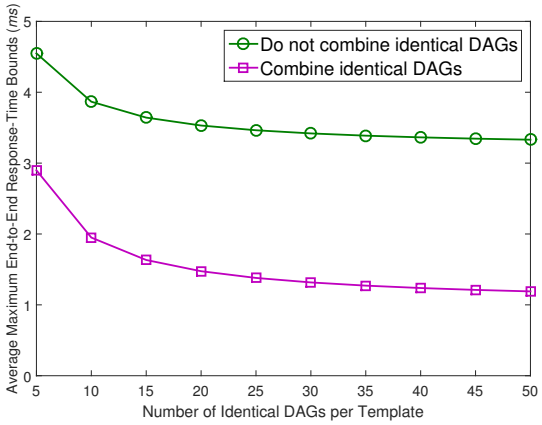


Figure 36: AMERBs as a function of the number of identical DAGs per template in the case where total utilization in each CE pool is fixed to eight, each task set has five templates, each DAG has ten tasks, and `edgeProb`=0.2.

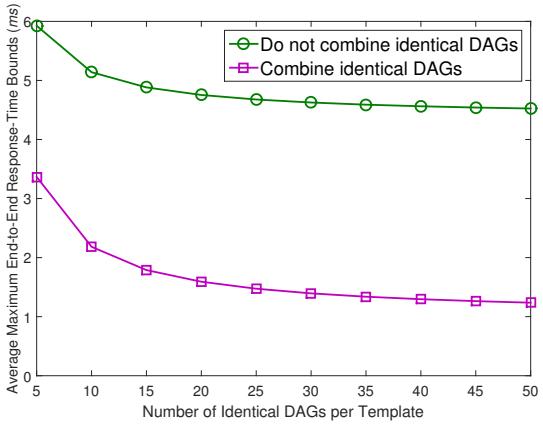


Figure 39: AMERBs as a function of the number of identical DAGs per template in the case where total utilization in each CE pool is fixed to eight, each task set has five templates, each DAG has 20 tasks, and `edgeProb`=0.2.

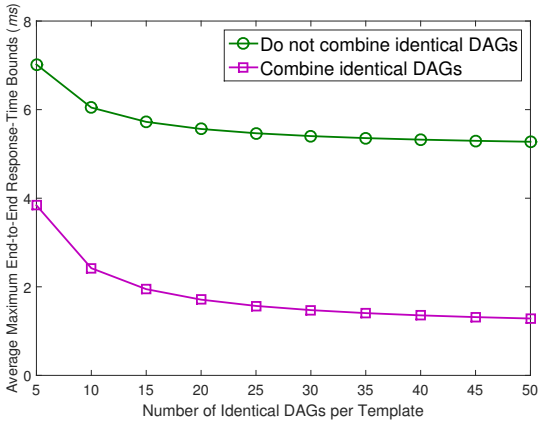


Figure 37: AMERBs as a function of the number of identical DAGs per template in the case where total utilization in each CE pool is fixed to eight, each task set has five templates, each DAG has ten tasks, and `edgeProb`=0.5.

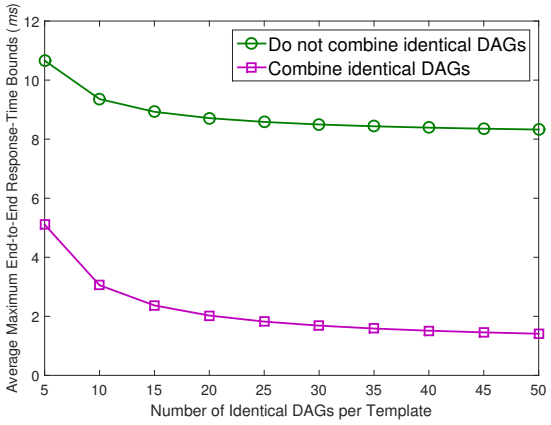


Figure 40: AMERBs as a function of the number of identical DAGs per template in the case where total utilization in each CE pool is fixed to eight, each task set has five templates, each DAG has 20 tasks, and `edgeProb`=0.5.

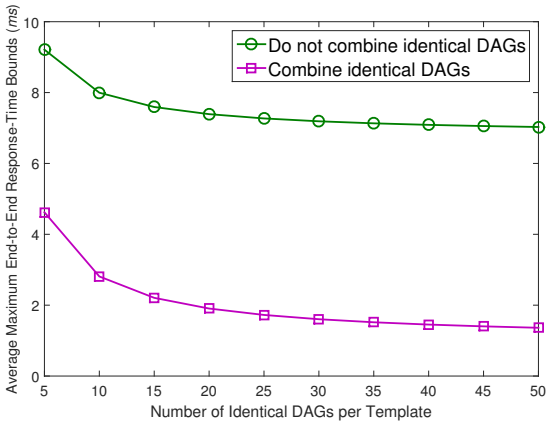


Figure 38: AMERBs as a function of the number of identical DAGs per template in the case where total utilization in each CE pool is fixed to eight, each task set has five templates, each DAG has ten tasks, and `edgeProb`=0.8.

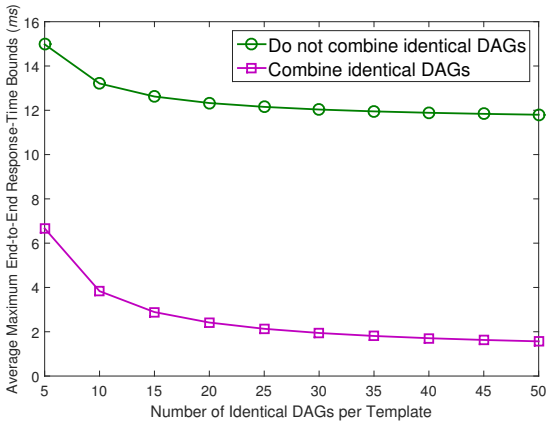


Figure 41: AMERBs as a function of the number of identical DAGs per template in the case where total utilization in each CE pool is fixed to eight, each task set has five templates, each DAG has 20 tasks, and `edgeProb`=0.8.

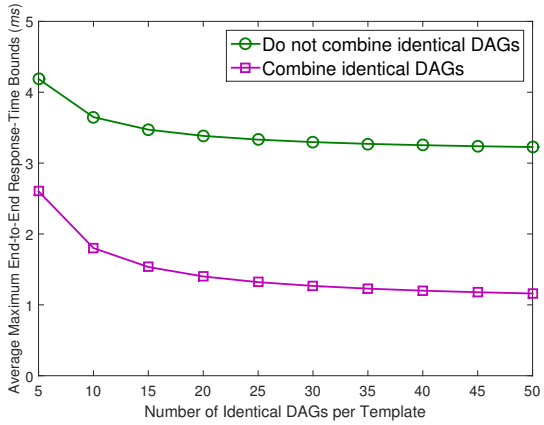


Figure 42: AMERBs as a function of the number of identical DAGs per template in the case where total utilization in each CE pool is fixed to eight, each task set has ten templates, each DAG has ten tasks, and `edgeProb`=0.2.

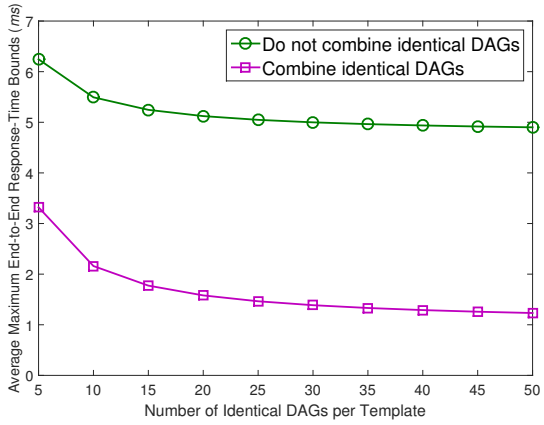


Figure 43: AMERBs as a function of the number of identical DAGs per template in the case where total utilization in each CE pool is fixed to eight, each task set has ten templates, each DAG has ten tasks, and `edgeProb`=0.5.

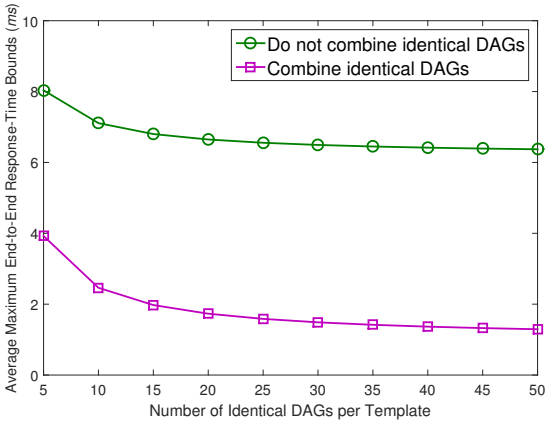


Figure 44: AMERBs as a function of the number of identical DAGs per template in the case where total utilization in each CE pool is fixed to eight, each task set has ten templates, each DAG has ten tasks, and `edgeProb`=0.8.

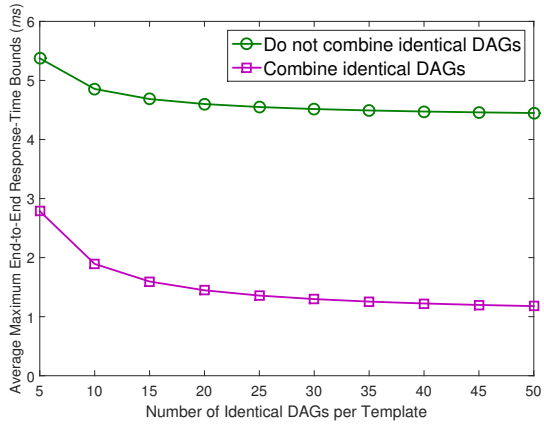


Figure 45: AMERBs as a function of the number of identical DAGs per template in the case where total utilization in each CE pool is fixed to eight, each task set has ten templates, each DAG has 20 tasks, and `edgeProb`=0.2.

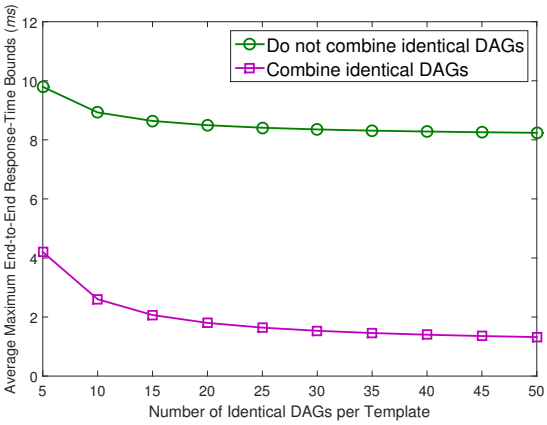


Figure 46: AMERBs as a function of the number of identical DAGs per template in the case where total utilization in each CE pool is fixed to eight, each task set has ten templates, each DAG has 20 tasks, and `edgeProb`=0.5.

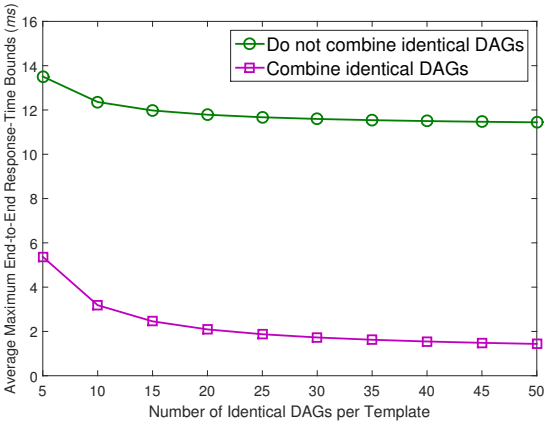


Figure 47: AMERBs as a function of the number of identical DAGs per template in the case where total utilization in each CE pool is fixed to eight, each task set has ten templates, each DAG has 20 tasks, and `edgeProb`=0.8.