Cache-Aware Compositional Analysis of Real-Time Multicore Virtualization Platforms

Meng Xu · Linh Thi Xuan Phan · Oleg Sokolsky · Sisu Xi · Chenyang Lu · Christopher Gill · Insup Lee

Abstract Multicore processors are becoming ubiquitous, and it is becoming increasingly common to run multiple real-time systems on a shared multicore platform. While this trend helps to reduce cost and to increase performance, it also makes it more challenging to achieve timing guarantees and functional isolation. One approach to achieving functional isolation is to use virtualization. However, virtualization also introduces many challenges to the multicore timing analysis; for instance, the overhead due to cache misses becomes harder to predict, since it depends not only on the direct interference between tasks but also on the indirect interference between virtual processors and the tasks executing on them.

In this paper, we present a *cache-aware* compositional analysis technique that can be used to ensure timing guarantees of components scheduled on a multicore virtualization platform. Our technique improves on previous multicore compositional analyses by accounting for the cache-related overhead in the components' interfaces, and it addresses the new virtualization-specific challenges in the overhead analysis. To demonstrate the utility of our technique, we report results from an extensive evaluation based on randomly generated workloads.

Keywords Compositional analysis · Interface · Cache-aware · Multicore · Virtualization

1 Introduction

Modern real-time systems are becoming increasingly complex and demanding; at the same time, the microprocessor industry is offering more computation power in the form of an exponentially growing number of cores. Hence, it is becoming more and more common to run multiple system components on the same multicore platform, rather than deploying them separately on different processors. This shift towards shared computing platforms enables system designers to reduce cost and to increase performance; however, it also makes it significantly more challenging to achieve separation of concerns and to maintain timing guarantees.

One approach to achieve separation of concerns is through virtualization technology. On a virtualization platform, such as Xen [Barham et al., 2003], multiple system components with different functionalities can be deployed in *domains* (virtual machines) that can each run their own operating system. These domains provide a clean isolation between components, and they preserve the components' functional behavior. However, existing virtualization platforms are designed to provide good *average* performance – they are not designed

Meng Xu · Linh Thi Xuan Phan · Insup Lee · Oleg Sokolsky
University of Pennsylvania
E-mail: {mengxu, linhphan, lee, sokolsky}@cis.upenn.edu

Sisu Xi · Chenyang Lu · Christopher Gill
Washington University in St. Louis
E-mail: {xis, cdgill, lu}@cse.wustl.edu

to provide real-time guarantees. To achieve the latter, a virtualization platform would need to ensure that each domain meets its real-time performance requirements. There are on-going efforts towards this goal, e.g., [Bruns et al., 2010; Crespo et al., 2010; Lee et al., 2012], but they primarily focus on single-core processors.

In this paper, we present a framework that can provide timing guarantees for multiple components running on a shared multicore virtualization platform. Our approach is based on *multicore compositional analysis*, but it takes the unique characteristics of virtualization platforms into account. In our approach, each component—i.e., a set of tasks and their scheduling policy—is mapped to a domain, which is executed on a set of virtual processors (VCPUs). The VCPUs of the domains are then scheduled on the underlying physical cores. The schedulability analysis of the system is compositional: we first abstract each component into an *interface* that describes the minimum processing resources needed to ensure that the component is schedulable, and then we compose the resulting interfaces to derive an interface for the entire system. Based on the system's interface, we can compute the minimum number of physical cores that are needed to schedule the system.

A number of compositional analysis techniques for multi-core systems have been developed (for instance, [Baruah and Fisher, 2009; Easwaran et al., 2009; Lipari and Bini, 2010]), but existing theories assume a somewhat idealized platform in which all overhead is negligible. In practice, the platform overhead—especially the cost of cache misses—can substantially interfere with the execution of tasks. As a result, the computed interfaces can underestimate the resource requirements of the tasks within the underlying components. Our goal is to remove this assumption by accounting for the platform overhead in the interfaces. In this paper, we focus on cache-related overhead, as it is among the most prominent in the multicore setting.

Cache-aware compositional analysis for multicore virtualization platforms is challenging because virtualization introduces additional overhead that is difficult to predict. For instance, when a VCPU resumes after being preempted by a higher-priority VCPU, a task executing on it may experience a cache miss, since its cache blocks may have been evicted from the cache by the tasks that were executing on the preempting VCPU. Similarly, when a VCPU is migrated to a new core, all its cached code and data remain in the old core; therefore, if the tasks later access content that was cached before the migration, the new core must load it from memory rather than from its cache.

Another challenge comes from the fact that cache misses that can occur when a VCPU finishes its budget and stops its execution. For instance, suppose a VCPU is currently running a task τ_i that has not finished its execution when the VCPU finishes its budget, and that τ_i is migrated to another VCPU of the same domain that is either idle or executing a lower-priority task τ_j (if one exists). Then τ_i can incur a cache miss if the new VCPU is on a different core, *and* it can trigger a cache miss in τ_j when τ_j resumes. This type of overhead is difficult to analyze, since it is in general not possible to determine statically when a VCPU finishes its budget or which task is affected by the VCPU completion.

In this paper, we address the above virtualization-related challenges, and we present a *cache-aware* compositional analysis for multicore virtualization platforms. Specifically, we make the following contributions:[1]

- We present a new supply bound function for the existing multiprocessor resource periodic (MPR) model that is tighter than the original supply bound function proposed in [Easwaran et al., 2009], thus enabling more resource-efficient interfaces for components (Section 3);
- we introduce DMPR, a deterministic extension of the multiprocessor resource periodic model to better represent component interfaces on multicore virtualization platforms (Section 4);
- we present a DMPR-based compositional analysis for systems without cache-related overhead (Section 5);
- we characterize different types of events that cause cache misses in the presence of virtualization (Section 6); and
- we propose three methods (BASELINE, TASK-CENTRIC-UB, and MODEL-CENTRIC) to account for the cache-related overhead (Sections 7.1, 7.2 and 8);
- we analyze the relationship between the proposed cache-related overhead analysis methods, and we develop a cache-aware compositional analysis method based on a hybrid of these methods (Section 9).

[1] A preliminary version of this paper has appeared in the Real-Time Systems Symposium (RTSS'13) [Xu et al., 2013].

To demonstrate the applicability and the benefits of our proposed cache-aware analysis, we report results from an extensive evaluation on randomly generated workloads using simulation as well as by running them on a realistic platform.

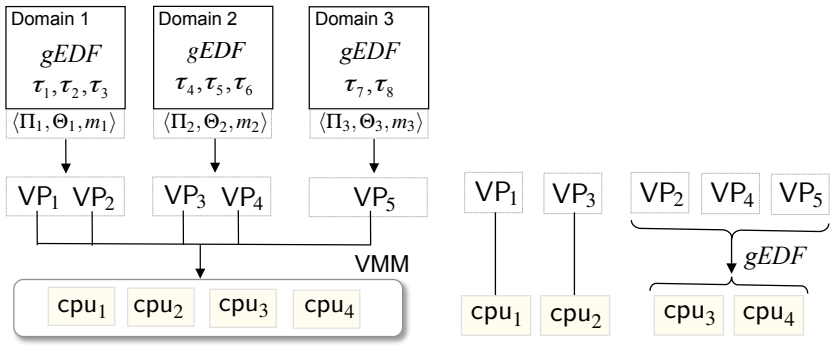
2 System Descriptions

The system we consider consists of multiple real-time components that are scheduled on a multicore virtualization platform, as is illustrated in Fig. 1(a). Each component corresponds to a *domain* (virtual machine) of the platform and consists of a set of tasks; these tasks are scheduled on a set of virtual processors (VCPUs) by the domain's scheduler. The VCPUs of the domains are then scheduled on the physical cores by the virtual machine monitor (VMM).

Each task τ_i within a domain is an explicit-deadline periodic task, defined by $\tau_i = (p_i, e_i, d_i)$, where p_i is the period, e_i is the worst-case execution time (WCET), and d_i is the relative deadline of τ_i. We require that $0 < e_i \leq d_i \leq p_i$ for all τ_i.

Each VCPU is characterized by $\mathsf{VP}_j = (\Pi_j, \Theta_j)$, where Π_j is the VCPU's period and Θ_j is the resource budget that the VCPU services in every period, with $0 \leq \Theta_j \leq \Pi_j$. We say that VP_j is a *full* VCPU if $\Theta_j = \Pi_j$, and a *partial* VCPU otherwise. We assume that each VCPU is implemented as a periodic server [Sha et al., 1986] with period Π_j and maximum budget time Θ_j. The budget of a VCPU is replenished at the beginning of each period; if the budget is not used when the VCPU is scheduled to run, it is wasted. We assume that each VCPU can execute only one task at a time. Like in most real-time scheduling research, we follow the conventional real-time task model in which each task is a single thread in this work; an extension to parallel task models is an interesting but also challenging research direction, which we plan to investigate in our future work.

We assume that all cores are identical and have unit capacity, i.e., each core provides t units of resource (execution time) in any time interval of length t. Each core has a private cache[2], all cores share the same memory, and the size of the memory is sufficiently large to ensure that all tasks (from all domains) can reside in memory at the same time, without conflicts.



(a) Task and VCPU scheduling. (b) Scheduling of VCPUs.

Fig. 1 Compositional scheduling on a virtualization platform.

Scheduling of tasks and VCPUs. We consider a hybrid version of the Earliest Deadline First (EDF) strategy. As is shown in Fig. 1, tasks within each domain are scheduled on the domain's VCPUs under the global EDF

[2] In this work, we assume that the cores either do not share a cache, or that the shared cache has been partitioned into cache sets that are each accessed exclusively by one core [Kim et al., 2012]. We believe that an extension to shared caches is possible, and we plan to consider it in our future work.

(gEDF) [Baruah and Baker, 2008] scheduling policy. The VCPUs of all the domains are then scheduled on the physical cores under a semi-partitioned EDF policy: *each full VCPU is pinned (mapped) to a dedicated core, and all the partial VCPUs are scheduled on the remaining cores under gEDF*. In the example from Fig. 1(b), VP_1 and VP_3 are full VCPUs, which are pinned to the physical cores cpu_1 and cpu_2, respectively. The remaining VCPUs are partial VCPUs, and are therefore scheduled on the remaining cores under gEDF.

Cache-related overhead. When two code sections are mapped to the same cache set, one section can evict the other section's cache blocks from the cache, which causes a cache miss when the former resumes. If the two code sections belong to the same task, this cache miss is an *intrinsic* cache miss; otherwise, it is an *extrinsic* cache miss [Basumallick and Nilsen, 1994]. The overhead due to intrinsic cache misses of a task can typically be statically analyzed based solely on the task; however, extrinsic cache misses depend on the interference between tasks during execution. In this paper, we assume that the tasks' WCETs already include intrinsic cache-related overhead, and we will focus on the extrinsic cache-related overhead. In the rest of this paper, we use the term 'cache' to refer to 'extrinsic cache'.

We use $\Delta_{\tau_i}^{\mathsf{crpmd}}$ to denote the maximum time needed to re-load all the useful cache blocks (i.e., cache blocks that will be reused) of a preempted task τ_i when that task resumes (either on the same core or on a different core).[3] Since the overhead for reloading the cache content of a preempted VCPU (i.e., a periodic server) upon its resumption is insignificant compared to the task's, we will assume here that it is either zero or is already included in the overhead due to cache misses of the running task inside the VCPU.

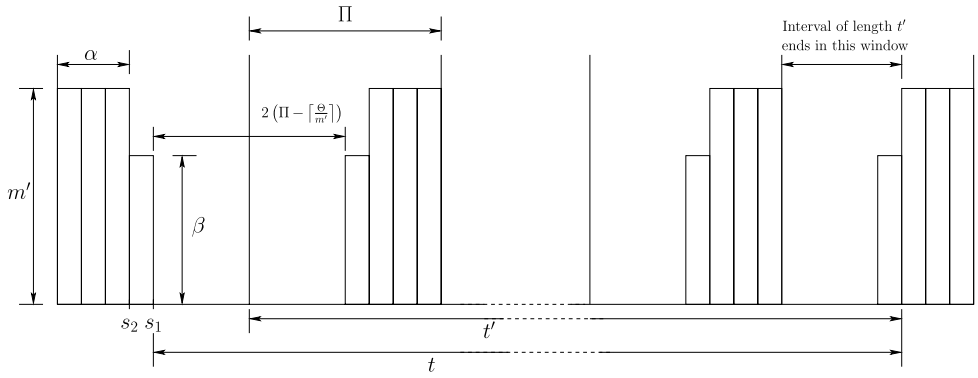
Objectives. In the above setting, our goal is to develop a cache-aware compositional analysis framework for the system. This framework consists of two elements: (1) an interface representation that can succinctly capture the resource requirements of a component (i.e., a domain or the entire system); and (2) an interface computation method for computing a minimum-bandwidth cache-aware interface of a component (i.e., an interface with the minimum resource bandwidth that guarantees the schedulability of a component in the presence of cache-related overhead).

Assumptions. We assume that (1) all VCPUs of each domain j share a single period Π_j; (2) all Π_j are known a priori; and (3) each Π_j is available to all domains. These assumptions are important to make the analysis tractable. Assumption 1 is equivalent to using a time-partitioned approach; we make this assumption to simplify the cache-aware analysis in Section 8, but it should be easy to extend the analysis to allow different periods for the VCPUs. Assumption 2 is made to reduce the search space, which is common in existing work (e.g., [Easwaran et al., 2009]); it can be relaxed by first establishing an upper bound on the optimal period (i.e., the period of the minimum-bandwidth interface) of each domain j, and then searching for the optimal period value based on this bound. Finally, Assumption 3 is necessary to determine how often different events that cause cache-related overhead happen (c.f. Section 6), which is crucial for the cache-aware interface computation in Sections 7 and 8. One approach to relaxing this assumption is to treat the period of the VCPUs of a domain as an input parameter in the computation of the overhead that another domain experiences. Such a parameterized interface analysis approach is very general, but making it efficient remains an interesting open problem for future research. We note, however, that although each assumption can be relaxed, the consequence of relaxing all three assumptions requires a much deeper investigation.

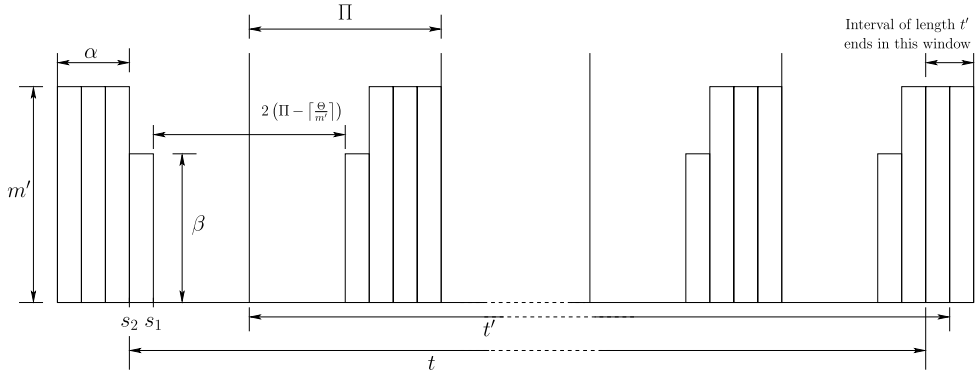
3 Improvement on Multiprocessor Periodic Resource Model

Recall that, when representing a platform, a resource model specifies the characteristics of the resource supply that is provided by that platform; when representing a component's interface, it specifies the total resource requirements of the component that must be guaranteed to ensure the component's schedulability. The resource

[3] We are aware that using a constant maximum value to bound the cache-miss overhead of a task may be conservative, and extensions to a finer granularity, e.g., using program analysis, may be possible. However, as the first step, we keep this assumption to simplify the analysis in this work, and we defer such extensions to our future work.



(a) Case 1



(b) Case 2

Fig. 2 Worst case resource supply of MPR model.

provided by a resource model R can also be captured by a supply bound function (SBF), denoted by $\mathsf{SBF}_R(t)$, that specifies the minimum number of resource units that R provides over any interval of length t.

In this section, we first describe the existing multiprocessor periodic resource (MPR) model [Shin et al., 2008], which serves as a basis for our proposed resource model for multicore virtualization platforms. We then present a new SBF for the MPR model that improves upon the original SBF given in [Shin et al., 2008], thus enabling tighter MPR-based interfaces for components and more efficient use of resource.

3.1 Background on MPR

An MPR model $\Gamma = (\tilde{\Pi}, \tilde{\Theta}, m')$ specifies that a multiprocessor platform with a number of identical, unit-capacity CPUs provides $\tilde{\Theta}$ units of resources in every period of $\tilde{\Pi}$ time units, with concurrency at most m' (in other words, at any time instant at most m' physical processors are allocated to this resource model), where $\tilde{\Theta} \leq m'\tilde{\Pi}$. Its resource bandwidth is given by $\tilde{\Theta}/\tilde{\Pi}$.

The worst-case resource supply scenario of the MPR model is shown in Fig. 2 [Easwaran et al., 2009]. Based on this worst-case scenario, the authors in [Easwaran et al., 2009] proposed an SBF that bounds the resource supplied by the MPR model $\Gamma = (\tilde{\Pi}, \tilde{\Theta}, m')$, which is defined as follows:

$$\tilde{\mathsf{SBF}}_{\Gamma}(t) = \begin{cases} 0, & \text{if } t' < 0 \\ \lfloor t'/\tilde{\Pi} \rfloor \tilde{\Theta} + \max\{0, m'x - (m'\tilde{\Pi} - \tilde{\Theta})\}, & \text{if } t' \geq 0 \wedge x \in [1, y] \\ \lfloor t'/\tilde{\Pi} \rfloor \tilde{\Theta} + \max\{0, m'x - (m'\tilde{\Pi} - \tilde{\Theta})\} - (m' - \beta), & \text{if } t' \geq 0 \wedge x \notin [1, y] \end{cases} \quad (1)$$

where $\alpha = \left\lfloor \frac{\tilde{\Theta}}{m'} \right\rfloor$, $\beta = \tilde{\Theta} - m'\alpha$, $t' = t - \left(\tilde{\Pi} - \left\lceil \frac{\tilde{\Theta}}{m'} \right\rceil\right)$, $x = t' - \tilde{\Pi}\left\lfloor \frac{t'}{\tilde{\Pi}} \right\rfloor$ and $y = \tilde{\Pi} - \left\lfloor \frac{\tilde{\Theta}}{m'} \right\rfloor$.

3.2 Improved SBF of the MPR model

We observe that, although the function $\tilde{\mathsf{SBF}}_{\Gamma}$ given in Eq. (1) is a valid SBF for the MPR model Γ, it is conservative. Specifically, the minimum amount of resource provided by Γ over a time window of length t (see Fig. 2) can be much larger than $\tilde{\mathsf{SBF}}_{\Gamma}(t)$ when (i) the resource bandwidth of Γ is equal to its maximum concurrency level (i.e., $\tilde{\Theta}/\tilde{\Pi} = m'$), or (ii) $x \leq 1$, where x is defined in Eq. (1). We demonstrate these cases using the two examples below.

Example 1 Let $\Gamma_1 = \langle \tilde{\Pi}, \tilde{\Theta}, m' \rangle$, where $\tilde{\Theta} = \tilde{\Pi}m'$, and Π and m' are any two positive integer values. By the definition of the MPR model, Γ_1 represents a multiprocessor platform with exactly m' identical, unit-capacity CPUs that are fully available. In other words, Γ_1 provides $m't$ time units in every t time units. However, according to Eq. (1), we have $\alpha = \left\lfloor \frac{\tilde{\Theta}}{m'} \right\rfloor = \tilde{\Pi}$, $\beta = \tilde{\Theta} - m'\alpha = 0$, $t' = t - \left(\tilde{\Pi} - \left\lceil \frac{\tilde{\Theta}}{m'} \right\rceil\right) = t$, $x = t' - \tilde{\Pi}\left\lfloor \frac{t'}{\tilde{\Pi}} \right\rfloor$, and $y = \tilde{\Pi} - \left\lfloor \frac{\tilde{\Theta}}{m'} \right\rfloor = 0$. Whenever $x \notin [1, y]$, for all $t = t' \geq 0$,

$$\tilde{\mathsf{SBF}}_{\Gamma_1}(t) = \lfloor t'/\tilde{\Pi} \rfloor \tilde{\Theta} + \max\{0, m'x - (m'\tilde{\Pi} - \tilde{\Theta})\} - (m' - \beta) = m't - m'.$$

As a result, $\tilde{\mathsf{SBF}}_{\Gamma_1}(t) < m't$ for all for all t such that $x \notin [1, y]$.

Example 2 Let $\Gamma_2 = \langle \Pi = 20, \Theta = 181, m' = 10 \rangle$ and consider $t = 21.1$. From Eq. (1), we obtain $\alpha = 18$, $\beta = 1$, $t' = t - 1 = 20.1$, $x = 0.1$, and $y = 2$. Since $x \notin [1, y]$, we have

$$\begin{aligned} \tilde{\mathsf{SBF}}_{\Gamma_2}(t) &= \lfloor \frac{t'}{\tilde{\Pi}} \rfloor \tilde{\Theta} + \max\{0, m'x - (m'\tilde{\Pi} - \tilde{\Theta})\} - (m' - \beta) \\ &= \lfloor \frac{20.1}{20} \rfloor 181 + \max\{0, 10 \times 0.1 - (10 \times 20 - 181)\} - (10 - 1) = 172. \end{aligned}$$

We reply on the worst-case resource supply scenario of the MPR model shown in Fig. 2 to compute the worst-case resource supply of Γ_2 during a time interval of length t. We first compute the worst-case resource supply when $t = 21.1$ based on **Case 1** in Fig. 2:

- t starts at the time point s_1;
- During the time interval $[s_1, s_1 + (\tilde{\Pi} - \alpha - 1)]$, i.e., $[s_1, s_1 + 1]$, Γ_2 supplies 0 time unit;
- During the time interval $[s_1 + (\tilde{\Pi} - \alpha - 1), s_1 + (\tilde{\Pi} - \alpha - 1) + \tilde{\Pi}]$, i.e., $[s_1 + 1, s_1 + 21]$, Γ_2 supplies $\Theta = 181$ time units;
- During the time interval $[s_1 + (\tilde{\Pi} - \alpha - 1) + \Pi, s_1 + t]$, i.e., $[s_1 + 21, s_1 + 21.1]$, Γ_2 supplies 0 time unit.

Therefore, Γ_2 supplies 181 time units during a time interval of length $t = 21.1$ based on **Case 1** in Fig. 2.

Next, we compute the worst-case resource supply when $t = 21.1$ based on **Case 2** in Fig. 2:

- t starts at the time point s_2;
- During the interval $[s_2, s_2 + (\tilde{\Pi} - \alpha)]$, i.e., $[s_2, s_2 + 2]$ Γ supplies $\beta = 1$ time unit;
- During the interval $[s_2 + (\tilde{\Pi} - \alpha), s_2 + 2(\tilde{\Pi} - \alpha)]$, i.e., $[s_2 + 2, s_2 + 4]$, Γ supplies $\beta = 1$ time unit;

– During the interval $[s_2 + 2(\tilde{\Pi} - \alpha), s_2 + t]$, i.e., $[s_2 + 4, s_2 + 21.1]$, Γ supplies $(21.1 - 4) \times m' = 171$ time units.

Therefore, Γ_2 supplies $1 + 1 + 171 = 173$ time units during any time interval of length t based on **Case 2** in Fig. 2. Because the two cases in Fig. 2 are the only two possible worst-case scenarios of the MPR resource model [Easwaran et al., 2009], the worst-case resource supply of Γ_2 during any time interval of length $t = 21.1$ is 173 time units. Since $\mathsf{SBF}_{\Gamma_2}(t) = 172$, the value computed by Eq. (1) under-estimates the actual resource provided by Γ_2.

Based on the above observations, we introduce a new SBF that can better bound the resource supply of the MPR model. This improved SBF is computed based on the worst-case resource supply scenarios shown in Fig. 2.

Lemma 1 *The amount of resource provided by the MPR model $\Gamma = \langle \tilde{\Pi}, \tilde{\Theta}, m' \rangle$ over any time interval of length t is at least $\mathsf{SBF}_\Gamma(t)$, where*

$$\mathsf{SBF}_\Gamma(t) = \begin{cases} 0, & t' < 0 \\ \lfloor \frac{t'}{\tilde{\Pi}} \rfloor \tilde{\Theta} + \max\{0, m'x' - (m'\tilde{\Pi} - \tilde{\Theta})\}, & t' \geq 0 \wedge x' \in [1 - \frac{\beta}{m'}, y] \\ \max\{0, \beta(t - 2(\tilde{\Pi} - \lfloor \frac{\tilde{\Theta}}{m'} \rfloor))\} & t' \in [0, 1] \wedge x' \notin [1 - \frac{\beta}{m'}, y] \\ \lfloor \frac{t''}{\tilde{\Pi}} \rfloor \tilde{\Theta} + \max\{0, m'x'' - (m'\tilde{\Pi} - \tilde{\Theta}) - (m' - \beta)\}, & t' \geq 1 \wedge x' \notin [1 - \frac{\beta}{m'}, y] \end{cases} \quad (2)$$

where

$$\alpha = \lfloor \frac{\tilde{\Theta}}{m'} \rfloor; \qquad \beta = \begin{cases} \tilde{\Theta} - m'\alpha, & \tilde{\Theta} \neq \Pi m' \\ m', & \tilde{\Theta} = \Pi m' \end{cases}; \quad t' = t - (\tilde{\Pi} - \lceil \frac{\tilde{\Theta}}{m'} \rceil); \quad t'' = t' - 1;$$

$$x' = (t' - \tilde{\Pi} \lfloor \frac{t'}{\tilde{\Pi}} \rfloor); \quad x'' = (t'' - \tilde{\Pi} \lfloor \frac{t''}{\tilde{\Pi}} \rfloor) + 1; \quad y = \tilde{\Pi} - \lfloor \frac{\tilde{\Theta}}{m'} \rfloor.$$

Proof We will prove that the function $\mathsf{SBF}_\Gamma(t)$ is a valid SBF of Γ based on the worst-case resource supply patterns of Γ shown in Fig. 2.

Consider the time interval of length t' (called time interval t') and the black-out interval (during which the resource supply is zero) in Fig. 2. By definition, x' is the remaining time of the time interval t' in the last period of Γ, and y is half the length of the black-out interval plus one. There are four cases of x, which determine whether $\mathsf{SBF}_\Gamma(t)$ corresponds to the resource supply of Γ in Case 1 or Case 2 in Fig. 2:

– $x' \in [1, y]$: It is easy to show that the value of $\mathsf{SBF}_\Gamma(t)$ in Case 1 is no larger than its value in Case 2. Note that if we shift the time interval of length t in Case 1 by one time unit to the left, we obtain the scenario in Case 2. In doing so, $\mathsf{SBF}_\Gamma(t)$ will be increased by β time units from the first period but decreased by at most β time units from the last period. Therefore, the pattern in Case 2 supplies more resource than the pattern in Case 1 when $x' \in [1, y]$.
– $x' \in [1 - \frac{\beta}{m'}, 1]$: As above, if we shift the time interval of length t in Case 1 by one time unit to the left, we obtain the scenario in Case 2. Recall that x' is the remaining time of the time interval of length t' in the last period, $x' \leq 1$ and $y \geq 1$. In shifting the time interval of length t, $\mathsf{SBF}_\Gamma(t)$ will lose $(1 - x')m'$ time units while gaining β time units from the first period. Because $x' \geq 1 - \frac{\beta}{m'}$, $\beta - (1 - x')m' \geq 0$. Therefore, $\mathsf{SBF}_\Gamma(t)$ gains $\beta - (1 - x')m' \geq 0$ time units in transferring the scenario in Case 1 to the scenario in Case 2. Hence, Case 1 is the worst-case scenario when $x' \in [1 - \frac{\beta}{m'}, 1]$.
– $x' \in [0, 1 - \frac{\beta}{m'})$: It is easy to show that Γ supplies less resource in Case 2 than in Case 1 when we shift the time interval of length t of Case 1 to left by one time unit to get Case 2. Therefore, Case 2 is the worst-case scenario when $x' \in [0, 1 - \frac{\beta}{m'}]$.
– $x' > y$: We can easily show that $\mathsf{SBF}_\Gamma(t)$ is no larger in Case 2 than in Case 1. Because $x' > y$, when we shift the time interval t of Case 1 to left by one time unit to get the scenario in Case 2, Γ loses m' time units from the last period but only gains β time units, where $\beta \leq m'$. Therefore, Case 2 is the worst-case scenario when $x' > y$.

From the above, we conclude that Case 1 is the worst-case resource supply scenario when $x' \in [1 - \frac{\beta}{m'}, y]$, and Case 2 is the worst-case resource supply scenario when $x' \notin [1 - \frac{\beta}{m'}, y]$.

Based on the worst-case resource supply scenario under different conditions above, we can derive Eq. 2 as follows:

- When $t' < 0$: It is obvious that $\mathsf{SBF}_\Gamma(t) = 0$ because Γ supplies no resource in the black-out interval.
- When $t' \geq 0$ and $x' \in [1 - \frac{\beta}{m'}, y]$: Based on the worst-case resource supply scenario in Case 1, Γ has $\lfloor \frac{t'}{\tilde{\Pi}} \rfloor$ periods and provides $\tilde{\Theta}$ time units in each period. Γ has x' remaining time in the last period, which provides $\max\{0, m'x'' - (m'\Pi - \Theta) - (m' - \beta)\}$ time units. Therefore, Γ supplies $\lfloor \frac{t'}{\tilde{\Pi}} \rfloor \tilde{\Theta} + \max\{0, m'x'' - (m'\Pi - \Theta) - (m' - \beta)\}$ time units during time interval t.
- When $t' \in [0, 1]$ and $x' \notin [1 - \frac{\beta}{m'}, y]$: Because $t' \in [0, 1]$, $t \in [\Pi - \lceil \frac{\tilde{\Theta}}{m'} \rceil, \Pi - \lceil \frac{\tilde{\Theta}}{m'} \rceil + 1]$. Therefore, $t < 2(\Pi - \lceil \frac{\tilde{\Theta}}{m'} \rceil) + 2$, where $2(\Pi - \lceil \frac{\tilde{\Theta}}{m'} \rceil)$ is the length of the black-out interval. Hence, the worst-case resource supply of Γ during time interval t is $\max\{0, \beta(t - 2(\Pi - \lfloor \frac{\Theta}{m'} \rfloor))\}$.
- When $t' > 1$ and $x' \notin [1 - \frac{\beta}{m'}, y]$, the worst-case resource supply scenario is Case 2. Γ has $\lfloor \frac{t''}{\tilde{\Pi}} \rfloor$ periods and provides $\tilde{\Theta}$ time units in each period. Γ supplies $\max\{0, m'x'' - (m'\tilde{\Pi} - \tilde{\Theta}) - (m' - \beta)\}$ time units during its first and last periods. Therefore, $\mathsf{SBF}_\Gamma(t) = \lfloor \frac{t''}{\tilde{\Pi}} \rfloor \Theta + \max\{0, m'x'' - (m'\tilde{\Pi} - \tilde{\Theta}) - (m' - \beta)\}$.

The lemma follows from the above results. □

It is easy to verify that, under the two scenarios described in Examples 1 and 2, $\mathsf{SBF}_{\Gamma_1}(t)$ and $\mathsf{SBF}_{\Gamma_2}(t)$ correspond to the actual minimum resource that Γ_1 and Γ_2 provide, respectively. It is also worth noting that, for the scenario described in Example 1, the compositional analysis for the MPR model [Easwaran et al., 2009] is compatible[4] with the underlying gEDF schedulability test under the improved SBF but not under the original SBF in Eq. (1). In the next example, we further demonstrate the benefits of the improved SBF in terms of resource bandwidth saving.

Example 3 Consider a component C with a taskset $\tau = \{\tau_1 = \cdots = \tau_4 = (200, 100, 200)\}$ that is scheduled under gEDF, and the period of the MPR interface of C is fixed to be 40. Following the interface computation method in [Easwaran et al., 2009], the corresponding minimum-bandwidth MPR interfaces, Γ_1 and Γ_2, of C when using the original SBF in Eq. (1) and when using the improved SBF in Eq. (2) are obtained as follows: $\Gamma_1 = \langle 40, 145, 4 \rangle$ and $\Gamma_2 = \langle 40, 120, 3 \rangle$. Thus, the MPR interface of C corresponding to the improved SBF can save $145/40 - 120/40 = 0.625$ cores compared to the interface corresponding to the original SBF proposed in [Easwaran et al., 2009].

4 Deterministic Multiprocessor Periodic Resource Model

In this section, we introduce the deterministic multiprocessor resource model (DMPR) for representing the interfaces. The MPR model described in the previous section is simple and highly flexible because it represents the collective resource requirements of components without fixing the contribution of each processor a priori. However, this flexibility also introduces some extra overhead: it is possible that all processors stop providing resources at the same time, which results in a long worst-case starvation interval (it can be as long as $2(\tilde{\Pi} - \lceil \tilde{\Theta}/m' \rceil)$ time units [Easwaran et al., 2009]). Therefore, to ensure schedulability in the worst case, it is necessary to provide more resources than strictly required. However, we can minimize this overhead by restricting the supply pattern of some of the processors. This is a key element of the deterministic MPR that we now propose.

A DMPR model is a deterministic extension of the MPR model, in which all of the processors but one always provide resource with full capacity. It is formally defined as follows.

[4] We say that a compositional analysis method is compatible with the underlying component's schedulability test it uses if whenever a component C with a taskset τ is deemed schedulable on m cores by the schedulability test, then C is also deemed schedulable under an interface with bandwidth no larger than m by the compositional analysis method.

Definition 1 A DMPR $\mu = \langle \Pi, \Theta, m \rangle$ specifies a resource that guarantees m full (dedicated) unit-capacity processors, each of which provides t resource units in any time interval of length t, and one partial processor that provides Θ resource units in every period of Π time units, where $0 \leq \Theta < \Pi$ and $m \geq 0$.

By definition, the resource bandwidth of a DMPR $\mu = \langle \Pi, \Theta, m \rangle$ is $\mathsf{bw}_\mu = m + \frac{\Theta}{\Pi}$. The total number of processors of μ is $m_\mu = m + 1$, if $\Theta > 0$, and $m_\mu = m$, otherwise.

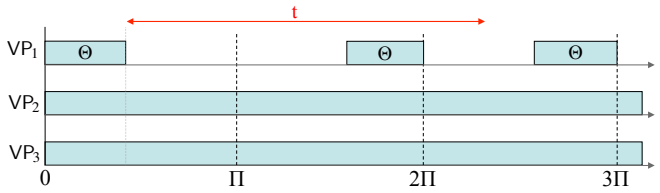


Fig. 3 Worst-case resource supply pattern of $\mu = \langle \Pi, \Theta, m \rangle$.

Observe that the partial processor of μ is represented by a single-processor periodic resource model $\Omega = (\Pi, \Theta)$ [Shin and Lee, 2003]. (However, it can also be represented by any other single processor resource model, such as EDP model [Easwaran et al., 2007].) Based on this characteristic, we can easily derive the worst-case supply pattern of μ (shown in Figure 3) and its supply bound function, which is given by the following lemma:

Lemma 2 *The supply bound function of a DMPR model $\mu = \langle \Pi, \Theta, m \rangle$ is given by:*

$$\mathsf{SBF}_\mu(t) = \begin{cases} mt, & \text{if } \Theta = 0 \ \vee \ (0 \leq t \leq \Pi - \Theta) \\ mt + y\Theta + \max\{0, t - 2(\Pi - \Theta) - y\Pi\}, & \text{otherwise} \end{cases}$$

where $y = \left\lfloor \frac{t-(\Pi-\Theta)}{\Pi} \right\rfloor$, for all $t > \Pi - \Theta$.

Proof Consider any interval of length t. Since the full processors of μ are always available, μ provides the minimum resource supply iff the partial processor provides the worst-case supply. Since the partial processor is a single-processor periodic resource model $\Omega = (\Pi, \Theta)$, its minimum resource supply in an interval of length t is given by [Shin and Lee, 2003]: $\mathsf{SBF}_\Omega(t) = 0$, if $\Theta = 0$ or $0 \leq t \leq \Pi - \Theta$; otherwise, $\mathsf{SBF}_\Omega(t) = y\Theta + \max\{0, t - 2(\Pi - \Theta) - y\Pi\}$ where $y = \left\lfloor \frac{t-(\Pi-\Theta)}{\Pi} \right\rfloor$. In addition, the m full processors of μ provides a total of mt resource units in any interval of length t. Hence, the minimum resource supply of μ in an interval of length t is $mt + \mathsf{SBF}_\Omega(t)$. This proves the lemma. □

It is easy to show that, when a DMPR μ and an MPR Γ have the same period, bandwidth, and total number of processors, then $\mathsf{SBF}_\mu(t) \geq \mathsf{SBF}_\Gamma(t)$ for all $t \geq 0$, and the worst-case starvation interval of μ is always shorter than that of Γ.

5 Overhead-free Compositional Analysis

In this section, we present our method for computing the minimum-bandwidth DMPR interface for a component, assuming that the cache-related overhead is negligible. The overhead-aware interface computation is considered in the next sections. We first recall some key results for components that are scheduled under gEDF [Easwaran et al., 2009].

5.1 Component schedulability under gEDF

The demand of a task τ_i in a time interval $[a, b]$ is the amount of computation that must be completed within $[a, b]$ to ensure that all jobs of τ_i with deadlines within $[a, b]$ are schedulable. When $\tau_i = (p_i, e_i, d_i)$ is scheduled under gEDF, its demand in any interval of length t is upper bounded by [Easwaran et al., 2009]:

$$\begin{aligned} \mathsf{dbf}_i(t) &= \Big\lfloor \frac{t + (p_i - d_i)}{p_i} \Big\rfloor e_i + CI_i(t), \text{ where} \\ CI_i(t) &= \min\Big\{e_i, \max\Big\{0, t - \Big\lfloor \frac{t + (p_i - d_i)}{p_i} \Big\rfloor p_i\Big\}\Big\}. \end{aligned} \tag{3}$$

In Eq. (3), $CI_i(t)$ denotes the maximum carry-in demand of τ_i in any time interval $[a, b]$ with $b - a = t$, i.e., the maximum demand generated by a job of τ_i that is released prior to a but has not finished its execution requirement at time a.

Consider a component C with a taskset $\tau = \{\tau_1, ...\tau_n\}$, where $\tau_i = (p_i, e_i, d_i)$, and suppose the tasks in C are schedulable under gEDF by a multiprocessor resource with m' processors. From [Easwaran et al., 2009], the worst-case demand of C that must be guaranteed to ensure the schedulability of τ_k in a time interval $(a, b]$, with $b - a = t \geq d_k$ is bounded by:

$$\mathsf{DEM}(t, m') = m' e_k + \sum_{\tau_i \in \tau} \hat{I}_{i,2} + \sum_{i: i \in L_{(m'-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}) \tag{4}$$

$$\begin{aligned} \text{where} \quad \hat{I}_{i,2} &= \min\big\{ \mathsf{dbf}_i(t) - CI_i(t),\ t - e_k \big\},\ \forall\, i \neq k, \\ \hat{I}_{k,2} &= \min\big\{ \mathsf{dbf}_k(t) - CI_k(t) - e_k,\ t - d_k \big\}; \\ \bar{I}_{i,2} &= \min\big\{ \mathsf{dbf}_i(t),\ t - e_k \big\},\ \forall\, i \neq k, \\ \bar{I}_{k,2} &= \min\big\{ \mathsf{dbf}_k(t) - e_k,\ t - d_k \big\}; \end{aligned}$$

and $L_{(m'-1)}$ is the set of indices of all tasks τ_i that have $\bar{I}_{i,2} - \hat{I}_{i,2}$ being one of the $(m' - 1)$ largest such values for all tasks.[5] This leads to the following schedulability test for C:

Theorem 1 ([Easwaran et al., 2009]) *A component C with a task set $\tau = \{\tau_1, ...\tau_n\}$, where $\tau_i = (p_i, e_i, d_i)$, is schedulable under gEDF by a multiprocessor resource model R with m' processors in the absence of overhead if, for each task $\tau_k \in \tau$ and for all $t \geq d_k$, $\mathsf{DEM}(t, m') \leq \mathsf{SBF}_R(t)$, where $\mathsf{DEM}(t, m')$ is given by Eq.* (4) *and $\mathsf{SBF}_R(t)$ gives the minimum total resource supply by R in an interval of length t.*

5.2 DMPR interface computation

In the absence of cache-related overhead, the minimum resource supply provided by a DMPR model $\mu = \langle \Pi, \Theta, m \rangle$ in any interval of length t is $\mathsf{SBF}_\mu(t)$, which is given by Lemma 2. Since each domain schedules its tasks under gEDF, the following theorem follows directly from Theorem 1.

Theorem 2 *A domain $\mathcal{D}$ with a task set $\tau = \{\tau_1, ...\tau_n\}$, where $\tau_i = (p_i, e_i, d_i)$, is schedulable under gEDF by a DMPR model $\mu = (\Pi, \Theta, m)$ if, for each $\tau_k \in \tau$ and for all $t \geq d_k$,*

$$\mathsf{DEM}(t, m_\mu) \leq \mathsf{SBF}_\mu(t), \tag{5}$$

where $m_\mu = m + 1$ if $\Theta > 0$, and $m_\mu = m$ otherwise.

We say that μ is a feasible DMPR for $\mathcal{D}$ if it guarantees the schedulability of $\mathcal{D}$ according to Theorem 2.

The next theorem derives a bound of the value t that needs to be checked in Theorem 2.

[5] Here, d_k and t refer to D_k and $A_k + D_k$ in [Easwaran et al., 2009], respectively.

Theorem 3 *If Eq.* (5) *is violated for some value t, then it must also be violated for a value that satisfies the condition*

$$t < \frac{C_\Sigma + m_\mu e_k + U + B}{\frac{\Theta}{\Pi} + m - U_T} \tag{6}$$

where C_Σ is the sum of the $m_\mu - 1$ largest e_i; $U = \sum_{i=1}^{n}(p_i - d_i)\frac{e_i}{p_i}$; $U_T = \sum_{i=1}^{n}\frac{e_i}{p_i}$; and $B = 2\frac{\Theta}{\Pi}(\Pi - \Theta)$.

Proof The proof follows a similar line with the proof of Theorem 2 in [Easwaran et al., 2009]. Recall that $\mathsf{DEM}(t, m_\mu)$ is given by Eq. (4). According to Eq. (4), we have

$$\hat{I}_{i,2} \leq \lfloor \frac{t + (p_i - d_i)}{p_i} \rfloor e_i \leq \frac{t + (p_i - d_i)}{p_i} e_i \leq t\frac{e_i}{p_i} + \frac{p_i - d_i}{p_i} e_i.$$

Therefore,

$$\sum_{i=1}^{n} \hat{I}_{i,2} \leq \sum_{i=1}^{n} t\frac{e_i}{p_i} + \sum_{i=1}^{n} \frac{p_i - d_i}{p_i} e_i = tU_T + U.$$

Because the carry-in workload of τ_i is no more than e_i, we derive $\sum_{i:i\in L_{(m_\mu-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}) \leq C_\Sigma$. Thus,

$$\mathsf{DEM}(t, m_\mu) \leq m_\mu e_k + tU_T + U + C_\Sigma.$$

Further, $\mathsf{SBF}_\mu(t)$ gives the worst-case resource supply of the DMPR model $\mu = \langle \Pi, \Theta, m\rangle$ over any interval of length t. Based on Lemma 2, the resource supply of μ is total resource supply of one partial VCPU (Π, Θ) and m full VCPUs. From [Shin and Lee, 2003], the resource supply of the partial VCPU (Π, Θ) over any interval of length t is at least $\frac{\Theta}{\Pi}(t - 2(\Pi - \Theta))$. In addition, the resource supply of m full VCPUs over any interval of length t is mt. Hence, the resource supply of μ over any interval of length t is at least $mt + \frac{\Theta}{\Pi}(t - 2(\Pi - \Theta))$. In other words,

$$\mathsf{SBF}_\mu(t) \geq mt + \frac{\Theta}{\Pi}(t - 2(\Pi - \Theta)).$$

Suppose Eq. (5) is violated, i.e., $\mathsf{DEM}(t, m_\mu) > \mathsf{SBF}_\mu(t)$ for some value t. Then, combine with the above results, we imply

$$m_\mu e_k + tU_T + U + C_\Sigma > mt + \frac{\Theta}{\Pi}(t - 2(\Pi - \Theta)),$$

which is equivalent to

$$t < \frac{C_\Sigma + m_\mu e_k + U + B}{\frac{\Theta}{\Pi} + m - U_T}.$$

Hence, if Eq. (5) is violated for some value t, then t must satisfy Eq. (6). This proves the theorem. □

The next lemma gives a condition for the minimum-bandwidth DMPR interface with a given period Π.

Lemma 3 *A DMPR model $\mu^* = \langle \Pi, \Theta^*, m^*\rangle$ is the minimum-bandwidth DMPR with period Π that can guarantee the schedulability of a domain $\mathcal{D}$ only if $m^* \leq m$ for all DMPR models $\mu = \langle \Pi, \Theta, m\rangle$ that can guarantee the schedulability of a domain $\mathcal{D}$.*

Proof Suppose $m^* > m$ for some DMPR $\mu = \langle \Pi, \Theta, m\rangle$. Then, $m^* \geq m + 1$ and, hence, $\mathsf{bw}_{\mu^*} = m^* + \Theta^*/\Pi \geq m + 1 + \Theta^*/\Pi \geq m + 1$. Since $\Theta < \Pi$, $\mathsf{bw}_\mu = m + \Theta/\Pi < m + 1$. Thus, $\mathsf{bw}_{\mu^*} > \mathsf{bw}_\mu$, which implies that m^* cannot be the minimum-bandwidth DMPR with period Π. Hence the lemma.

Computing the domains' interfaces. Let $\mathcal{D}_i$ be a domain in the system and Π_i be its given VCPU period (c.f. Section 2). The minimum-bandwidth interface of $\mathcal{D}_i$ with period Π_i is the minimum-bandwidth DPRM model $\mu_i = \langle \Pi_i, \Theta_i, m_i\rangle$ that is feasible for $\mathcal{D}_i$. To obtain μ_i, we perform binary search on the number of

full processors m'_i, and, for each value m'_i, we compute the smallest value of Θ'_i such that $\langle \Theta'_i, \Pi_i, m'_i \rangle$ is feasible for $\mathcal{D}_i$ (using Theorem 2).[6] Then m_i is the smallest value of m'_i for which a feasible interface is found, and, Θ_i is the smallest budget Θ'_i computed for m_i.

Computing the system's interface. The interface of the system can be obtained by composing the interfaces μ_i of all domains $\mathcal{D}_i$ in the system under the VMM's semi-partitioned EDF policy (c.f. Section 2). Let D denote the number of domains of the platform.

Observe that each interface $\mu_i = \langle \Pi_i, \Theta_i, m_i \rangle$ can be transformed directly into an equivalent set of m_i full VCPUs (with budget Π_i and period Π_i) and, if $\Theta_i > 0$, a partial VCPU with budget Θ_i and period Π_i. Let $\mathcal{C}$ be a component that contains all the partial VCPUs that are transformed from the domains' interfaces. Then the VCPUs in $\mathcal{C}$ are scheduled together under gEDF, whereas all the full VCPUs are each mapped to a dedicated core.

Since each partial VCPU in $\mathcal{C}$ is implemented as a periodic server, which is essentially a periodic task, we can compute the minimum-bandwidth DMPR interface $\mu_{\mathcal{C}} = \langle \Pi_{\mathcal{C}}, \Theta_{\mathcal{C}}, m_{\mathcal{C}} \rangle$ that is feasible for $\mathcal{C}$ by the same technique used for domains. Combining $\mu_{\mathcal{C}}$ with the full VCPUs of the domains, we can see that the system must be guaranteed $m_{\mathcal{C}} + \sum_{1 \le i \le D} m_i$ full processors and a partial processor, with budget $\Theta_{\mathcal{C}}$ and period $\Pi_{\mathcal{C}}$, to ensure the schedulability of the system. The next theorem directly follows from this observation.

Theorem 4 *Let $\mu_i = \langle \Pi_i, \Theta_i, m_i \rangle$ be the minimum-bandwidth DMPR interface of domain $\mathcal{D}_i$, for all $1 \le i \le D$. Let $\mathcal{C}$ be a component with the taskset*

$$\tau_{\mathcal{C}} = \{(\Pi_i, \Theta_i, \Pi_i) \mid 1 \le i \le D \wedge \Theta_i > 0\},$$

which are scheduled under gEDF. Then the minimum-bandwidth DMPR interface with period $\Pi_{\mathcal{C}}$ of the system is given by: $\mu_{\mathsf{sys}} = \langle \Pi_C, \Theta_C, m_{\mathsf{sys}} \rangle$, where $\mu_{\mathcal{C}} = \langle \Pi_{\mathcal{C}}, \Theta_{\mathcal{C}}, m_{\mathcal{C}} \rangle$ is a minimum-bandwidth DMPR interface with period $\Pi_{\mathcal{C}}$ of $\mathcal{C}$ and $m_{\mathsf{sys}} = m_{\mathcal{C}} + \sum_{1 \le i \le D} m_i$.

Based on the system's interface, one can easily derive the schedulability of the system as follows (the lemma comes directly from the interface's definition):

Lemma 4 *Let M be the number of physical cores of the platform. The system is schedulable if $M \ge m_{\mathsf{sys}} + 1$, or, $M = m_{\mathsf{sys}}$ and $\Theta_C = 0$, where $\langle \Pi_C, \Theta_C, m_{\mathsf{sys}} \rangle$ is the minimum-bandwidth DMPR system's interface.*

The results obtained above assume that the cache-related overhead is negligible. We will next develop the analysis in the presence of cache-related overhead.

6 Cache-Related Overhead Scenarios

In this section, we characterize the different events that cause cache-related overhead; this is needed for the cache-aware analysis in Sections 7 and 8.

Cache-related overhead in a multicore virtualization platform is caused by (1) task preemption within the same domain, (2) VCPU preemption, and (3) VCPU exhaustion of budget. We discuss each of them in detail below.

6.1 Event 1: Task-preemption event

Since tasks within a domain are scheduled under gEDF, a newly released higher-priority task preempts a currently executing lower-priority task of the *same* domain, if none of the domain's VCPUs are idle. When a preempted task resumes its execution, it may experience cache misses: its cache content may have been

[6] Note that the number of full processors is always bounded from below by $\lfloor U_i \rfloor$, where U_i is the total utilization of the tasks in $\mathcal{D}_i$, and bounded from above by the number of tasks in $\mathcal{D}_i$ or the number of physical platform (if given), whichever is smaller.

evicted from the cache by the preempting task (or tasks with a higher priority than the preempting task, if a nested preemption occurs), or the task may be resumed on a different VCPU that is running on a different core, in which case the task's cache content may not be present in the new core's cache. Hence the following definition:

Definition 2 (Task-preemption event) A task-preemption event of τ_i is said to occur when a job of another task τ_j in the same domain is released and this job can preempt the current job of τ_i.

Fig. 4 illustrates the worst-case scenario of the overhead caused by a task-preemption event. In the figure, a preemption event of τ_1 happens at time $t = 3$ when τ_3 is released (and preempts τ_1). Due to this event, τ_1 experiences a cache miss at time $t = 5$ when it resumes. Since τ_1 resumes on a different core, all the cache blocks it will reuse have to be reloaded into new core's cache, which results in cache-related preemption/migration overhead on τ_1. (Note that the cache content of τ_1 is not necessarily reloaded all at once, but rather during its remaining execution after it has been resumed; however, for ease of exposition, we show the combined overhead at the beginning of its remaining execution).

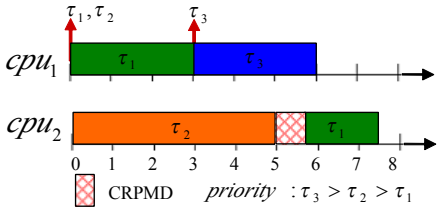


Fig. 4 Cache-related overhead of a task-preemption event.

Since gEDF is work-conserving, tasks do not suspend themselves, and each task resumes at most once after each time it is preempted. Therefore, each task τ_k experiences the overhead caused by each of its task-preemption events at most once, and this overhead is bounded from above by $\Delta_{\tau_k}^{\mathsf{crpmd}}$.

Lemma 5 *A newly released job of τ_j preempts a job of τ_i under gEDF only if $d_j < d_i$.*

Proof Suppose $d_j \geq d_i$ and a newly released job J_j of τ_j preempts a job J_i of τ_i. Then, J_j must be released later than J_i. As a result, the absolute deadline of J_j is later than J_i's (since $d_j \geq d_i$), which contradicts the assumption that J_j preempts J_i under gEDF. This proves the lemma. □

The maximum number of task-preemption events in each period of τ_i is given by the next lemma.

Lemma 6 (Number of task-preemption events) *The maximum number of task-preemption events of τ_i under gEDF during each period of τ_i, denoted by $N_{\tau_i}^1$, is bounded by*

$$N_{\tau_i}^1 \leq \sum_{\tau_j \in \mathsf{HP}(\tau_i)} \left\lceil \frac{d_i - d_j}{p_j} \right\rceil \tag{7}$$

where $\mathsf{HP}(\tau_i)$ is the set of tasks τ_j within the same domain with τ_i with $d_j < d_i$.

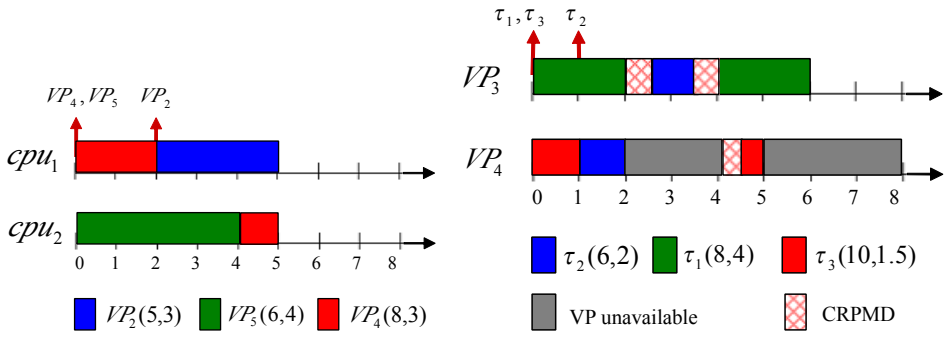
Proof Let τ_i^c be the current job of τ_i in a period of τ_i, and let r_i^c be its release time. From Lemma 5, only jobs of a task τ_j with $d_j < d_i$ and in the same domain can preempt τ_i^c. Further, for each such τ_j, only the jobs that are released after τ_i^c and that have absolute deadlines no later than τ_i^c's can preempt τ_i^c. In other words, only jobs that are released within the interval $(r_i^c, r_i^c + d_i - d_j]$ can preempt τ_i^c. As a result, the maximum number of task-preemption events of τ_i under gEDF is no more than $\sum_{\tau_j \in \mathsf{HP}(\tau_i)} \left\lceil \frac{d_i - d_j}{p_j} \right\rceil$. □

6.2 VCPU-preemption event

Definition 3 (VCPU-preemption event) A VCPU-preemption event of VP_i occurs when VP_i is preempted by a higher-priority VCPU VP_j of another domain.

When a VCPU VP_i is preempted, the currently running task τ_l on VP_i may migrate to another VCPU VP_k of the same domain and may preempt the currently running task τ_m on VP_k. This can cause the tasks running on VP_k experiences cache-related preemption or migration overhead twice in the worst case, as is illustrated in the following example.

Example 4 The system consists of three domains $\mathcal{D}_1$-$\mathcal{D}_3$. $\mathcal{D}_1$ has VCPUs VP_1 (full) and VP_2 (partial); $\mathcal{D}_2$ has VCPUs VP_3 (full) and VP_4 (partial); and $\mathcal{D}_3$ has one partial VCPU VP_5. The partial VCPUs of the domains – $\mathsf{VP}_2(5,3)$, $\mathsf{VP}_4(8,3)$ and $\mathsf{VP}_5(6,4)$ – are scheduled under gEDF on cpu_1 and cpu_2, as is shown in Fig. 5(a). In addition, domain $\mathcal{D}_2$ consists of three tasks, $\tau_1(8,4,8)$, $\tau_2(6,2,6)$ and $\tau_3(10,1.5,10)$, which are scheduled under gEDF on its VCPUs (Fig. 5(b)).



(a) Scheduling scenario of VCPUs. (b) Cache overhead of tasks in $\mathcal{D}_2$.

Fig. 5 Cache overhead due to a VCPU-preemption event.

As is shown in Fig. 5(a), a VCPU-preemption event occurs at time $t = 2$, when VP_4 (of $\mathcal{D}_2$) is preempted by VP_2. Observe that, within $\mathcal{D}_2$ at this instant, τ_2 is running on VP_4 and τ_1 is running on VP_3. Since τ_2 has an earlier deadline than τ_1, it is migrated to VP_3 and preempts τ_1 there. Since VP_3 is mapped to a different core from cpu_1, τ_2 has to reload its useful cache content to the cache of the new core at $t = 2$. Further, when τ_1 resumes at time $t = 3.5$, it has to reload the useful cache blocks that may have been evicted from the cache by τ_2. Hence, the VCPU-preemption event of VP_4 causes overhead for both of the tasks in its domain.

Lemma 7 *Each VCPU-preemption event causes at most two tasks to experience a cache miss. Further, the cache-related overhead it causes is at most* $\Delta_{\mathsf{C}}^{\mathsf{crpmd}} = \max_{\tau_i \in C} \Delta_{\tau_i}^{\mathsf{crpmd}}$*, where* C *is the component that has the preempted VCPU.*

Proof At most one task is running on a VCPU at any time. Hence, when a VCPU VP_i of C is preempted, at most one task (τ_m) on VP_i is migrated to another VCPU VP_j, and this task preempts at most one task (τ_l) on VP_j. As a result, at most two tasks (i.e., τ_m and τ_l) incur a cache miss because of the VCPU-preemption event. (Note that τ_l cannot immediately preempt another task τ_n because otherwise, τ_m would have migrated to the VCPU on which τ_n is running and preempted τ_n instead.) Further, since the overhead caused by each cache miss in C is at most $\Delta_{\mathsf{C}}^{\mathsf{crpmd}} = \max_{\tau_i \in C} \Delta_{\tau_i}^{\mathsf{crpmd}}$, the maximum overhead caused by the resulting cache misses is at most $2\Delta_{\mathsf{C}}^{\mathsf{crpmd}}$. □

Since the partial VCPUs are scheduled under gEDF as implicit-deadline tasks (i.e., the task periods are equal to their relative deadlines), the number of VCPU-preemption events of a partial VCPU VP_i during each VP_i's period also follows Lemma 6. The next lemma is implied directly from this observation.

Lemma 8 (Number of VCPU-preemption events) *Let* $\mathsf{VP}_i = (\Pi_i, \Theta_i)$ *for all partial VCPUs* VP_i *of the domains. Let* $\mathsf{HP}(\mathsf{VP}_i)$ *be the set of* VP_j *with* $0 < \Theta_j < \Pi_j < \Pi_i$*. Denote by* $N^2_{\mathsf{VP}_i}$ *and* $N^2_{\mathsf{VP}_i,\tau_k}$ *the maximum number of VCPU-preemption events of* VP_i *during each period of* VP_i *and during each period of* τ_k *inside* VP_i*'s domain, respectively. Then,*

$$N^2_{\mathsf{VP}_i} \leq \sum_{\mathsf{VP}_j \in \mathsf{HP}(\mathsf{VP}_i)} \left\lceil \frac{\Pi_i - \Pi_j}{\Pi_j} \right\rceil \tag{8}$$

$$N^2_{\mathsf{VP}_i,\tau_k} \leq \sum_{\mathsf{VP}_j \in \mathsf{HP}(\mathsf{VP}_i)} \left\lceil \frac{p_k}{\Pi_j} \right\rceil. \tag{9}$$

6.3 VCPU-completion event

Definition 4 (VCPU-completion event) A VCPU-completion event of VP_i happens when VP_i exhausts its budget in a period and stops its execution.

Like in VCPU-preemption events, each VCPU-completion event causes at most two tasks to experience a cache miss, as given by Lemma 9.

Lemma 9 *Each VCPU-completion event causes at most two tasks to experience a cache miss.*

Proof The effect of a VCPU-completion event is very similar to that of a VCPU-preemption event. When VP_i finishes its budget and stops, the running task τ_m on VP_i may migrate to another running VCPU VP_j, and, τ_m may preempt at most one task τ_l on VP_j. Hence, at most two tasks incur a cache miss due to a VCPU-preemption event. □

Lemma 10 (Number of VCPU-completion events) *Let* $N^3_{\mathsf{VP}_i}$ *and* $N^3_{\mathsf{VP}_i,\tau_k}$ *be the number of VCPU-completion events of* VP_i *in each period of* VP_i *and in each period of* τ_k *inside* VP_i*'s domain. Then,*

$$N^3_{\mathsf{VP}_i} \leq 1 \tag{10}$$

$$N^3_{\mathsf{VP}_i,\tau_k} \leq \left\lceil \frac{p_k - \Theta_i}{\Pi_i} \right\rceil + 1 \tag{11}$$

Proof Eq. (10) holds because VP_i completes its budget at most once every period. Further, observe that τ_i experiences the worst-case number of VCPU-preemption events when (1) its period ends at the same time as the budget finish time of VP_i's current period, and (2) VP_i finishes its budget as soon as possible (i.e., B_i time units from the beginning of the VCPU's period) in the current period and as late as possible (i.e., at the end of the VCPU's period) in all its preceding periods. Eq. (11) follows directly from this worst-case scenario. □

VCPU-stop event. Since a VCPU stops its execution when its VCPU-completion or VCPU-preemption event occurs, we define a *VCPU-stop event* that includes both types of events. That is, a VCPU-stop event of VP_i occurs when VP_i stops its execution because its budget is finished *or* because it is preempted by a higher-priority VCPU. Since VCPU-stop events include both VCPU-completion events and VCPU-preemption events, the maximum number of VCPU-stop events of VP_i during each VP_i's period, denoted as $N^{\mathsf{stop}}_{\mathsf{VP}_i}$, satisfies

$$N^{\mathsf{stop}}_{\mathsf{VP}_i} = N^2_{\mathsf{VP}_i} + N^3_{\mathsf{VP}_i} \leq \sum_{\mathsf{VP}_j \in \mathsf{HP}(\mathsf{VP}_i)} \left\lceil \frac{\Pi_i - \Pi_j}{\Pi_j} \right\rceil + 1 \tag{12}$$

Overview of the overhead-aware compositional analysis. Based on the above quantification, in the next two sections we develop two different approaches, task-centric and model-centric, for the overhead-aware interface computation. Although the obtained interfaces by both approaches are safe and can each be used independently, we combine them to obtain the interface with the smallest bandwidth as the final result.

7 Task-centric Compositional Analysis

This section introduces two task-centric analysis methods to account for the cache-related overhead in the interface computation. The first, denoted as BASELINE, accounts for the overhead by inflating the WCET of every task in the system with the maximum overhead it experiences within each of its periods. The second, denoted as TASK-CENTRIC-UB, combines the result of the first method using an upper bound on the number of VCPUs that each domain needs in the presence of cache-related overhead. We describe each method in detail below.

7.1 BASELINE: Analysis based on WCET-inflation

As was discussed in Section 6, the overhead that a task experiences during its lifetime is composed of the overhead caused by task-preemption events, VCPU-preemption events and VCPU-completion events. In addition, when one of the above events occurs, each task τ_k experiences at most one cache miss overhead and, hence, a delay of at most $\Delta_{\tau_k}^{\mathsf{crpmd}}$. From [Brandenburg, 2011], the cache overhead caused by a task-preemption event can be accounted for by inflating the higher-priority task τ_i of the event with the maximum cache overhead caused by τ_i. From Lemmas 8 and 10, we conclude that the maximum overhead τ_k experiences within each period is

$$\delta_{\tau_k}^{\mathsf{crpmd}} = \max_{\tau_i \in \mathsf{LP}(\tau_k)} \{\Delta_{\tau_i}^{\mathsf{crpmd}}\} + \Delta_{\tau_k}^{\mathsf{crpmd}} (N_{\mathsf{VP}_i,\tau_k}^2 + N_{\mathsf{VP}_i,\tau_k}^3)$$

where $\mathsf{LP}(\tau_k)$ is the set of tasks τ_i within the same domain with τ_k with $d_i > d_k$ and VP_i is the partial VCPU of the domain of τ_k. As a result, the worst-case execution time of τ_k in the presence of cache overhead is at most

$$e'_k = e_k + \delta_{\tau_k}^{\mathsf{crpmd}}. \tag{13}$$

Thus, we can state the following theorem:

Theorem 5 *A component with a taskset $\tau = \{\tau_1, ...\tau_n\}$, where $\tau_k = (p_k, e_k, d_k)$, is schedulable under gEDF by a DMPR model μ in the presence of cache-related overhead if its inflated taskset $\tau' = \{\tau'_1, ...\tau'_n\}$ is schedulable under gEDF by μ in the absence of cache-related overhead, where $\tau'_k = (p_k, e'_k, d_k)$, and e'_k is given by Eq. 13.*

Based on Theorem 5, we can compute the DMPR interfaces of the domains and the system by first inflating the WCET of each task τ_k in each domain with the overhead $\delta_{\tau_k}^{\mathsf{crpmd}}$ and then applying the same method as the overhead-free interface computation in Section 5.2.[7]

7.2 TASK-CENTRIC-UB: Combination of BASELINE with an upper bound on the number of VCPUs

Recall from Section 6 that, VCPU-preemption events and VCPU-completion events happen only when the component has a partial VCPU. Therefore, the taskset in a component with no partial VCPU experiences only the cache overhead caused by task-preemption events. Recall that when a task-preemption event happens, the corresponding lower-priority task τ_i experiences a cache miss delay of at most $\Delta_{\tau_i}^{\mathsf{crpmd}}$. Thus, the maximum cache overhead that a high-priority task τ_k causes to any preempted task is $\max_{\tau_i \in \mathsf{LP}(\tau_k)} \Delta_{\tau_i}^{\mathsf{crpmd}}$, where $\mathsf{LP}(\tau_k)$ is the set of tasks τ_i within the same domain with τ_k that have $d_i > d_k$. As a result, the worst-case execution time of τ_k in the presence of cache overhead caused by task-preemption events is at most

$$e''_k = e_k + \max_{\tau_i \in \mathsf{LP}(\tau_k)} \Delta_{\tau_i}^{\mathsf{crpmd}}, \tag{14}$$

where $\tau_i \in \mathsf{LP}(\tau_k)$ if $d_i > d_k$. This implies the following lemma:

[7] Note that we inflate only the tasks' WCETs and not the VCPUs' budgets, since $\delta_{\tau_k}^{\mathsf{crpmd}}$ includes the overhead for reloading the useful cache content of a preempted VCPU when it resumes.

Lemma 11 *A component with a taskset $\tau = \{\tau_1, ..., \tau_n\}$, where $\tau_k = (p_k, e_k, d_k)$, is schedulable under gEDF by a DMPR model $\bar{\mu} = \langle \Pi, 0, \bar{m} \rangle$ in the presence of cache-related overhead if its inflated taskset $\tau'' = \{\tau_1'', ..., \tau_n''\}$ is schedulable under gEDF by $\mu'' = \langle \Pi, \Theta'', m'' \rangle$ in the absence of cache-related overhead, where $\tau_k'' = (p_k, e_k'', d_k)$, e_k'' is given by Eq. 14, and $\bar{m} = m'' + \lceil \frac{\Theta''}{\Pi} \rceil$. Further, the maximum number of full VCPUs of the interface of the taskset τ in the presence of cache overhead is $\bar{m}$.*

Proof First, observe that the inflated taskset τ'' safely accounts for all the cache overhead experienced by τ. This is because (1) inflating the worst-cache execution time of each task τ_k with $\max_{\tau_i \in \mathsf{LP}(\tau_k)} \Delta_{\tau_i}^{\mathsf{crpmd}}$ is safe to account for the cache overhead delay caused by task-preemption events (as was proven in [Brandenburg, 2011]), and (2) the DMPR model $\bar{\mu}$ has no partial VCPU and thus, τ does not experience any cache overhead caused by VCPU-preemption events or VCPU-completion events. Further, based on Lemma 2, one can easily show that the resource supply bound function $\mathsf{SBF}_{\mu}(t)$ of a DMPR model $\mu = \langle \Pi, \Theta, m \rangle$ is monotonically non-decreasing with the budget of μ when the period of μ is fixed. In other words, $\mathsf{SBF}_{\bar{\mu}}(t) \geq \mathsf{SBF}_{\mu''}(t)$ for all t. Combine the above observations, we imply that τ is schedulable under the resource model $\bar{\mu}$ in the presence of cache overhead if τ'' is schedulable under the resource model μ'' in the absence of cache overhead. This proves the first part of the lemma.

Since τ is schedulable under the resource model $\bar{\mu}$ in the presence of cache overhead, the number of full VCPUs of the overhead-aware interface of τ is always less than or equal to the ceiling of the bandwidth of $\bar{\mu}$, which is exactly $\bar{m}$. □

Note that the maximum number of full VCPUs given by Lemma 11 can be larger or smaller than the interface bandwidth computed by the BASELINE method, as is illustrated in the following two examples.

Example 5 Consider a system Sys_1 consisting of two domains, C_1 and C_2, with workloads $\tau_{C_1} = \{\tau_1^1 = \cdots = \tau_1^3 = (100, 40, 100)\}$ and $\tau_{C_2} = \{\tau_2^1 = \cdots = \tau_2^3 = (100, 40, 100)\}$, respectively. Suppose that Sys_1 employs the hybrid EDF scheduling strategy described in Section 2; the periods of DMPR interfaces of C_1, C_2 and Sys_1 are set to $80, 40$ and 20, respectively; and the cache overhead per task is 1. Then, the DMPR cache-aware interface of C_1 computed using the BASELINE method is $\mu_{C_1} = \langle 80, 76, 1 \rangle$, which has a bandwidth of $1 + 76/80 = 1.95$.

In contrast, if we only consider the cache overhead caused by task-preemption events, then the interface of the system is given by $\mu''_{C_1} = \langle 80, 64, 1 \rangle$. Based on Lemma 11, the maximum number of full VCPUs of C_1 is $1 + 64/80 = 2$, and the corresponding DMPR interface is $\bar{\mu}_{C1} = \langle 80, 0, 2 \rangle$. Thus, the interface computed by the BASELINE method has a smaller bandwidth than the maximum number of full VCPUs given by Lemma 11.

Example 6 Consider a system Sys_2 that is identical to the system Sys_1 in Example 5, except that the cache overhead for each task is 5 instead of 1. In this case, the cache-aware interface of C_1 computed using the BASELINE method is $\bar{\mu}_{C_1} = \langle 80, 72, 2 \rangle$, which has a bandwidth of $2 + 72/80 = 2.9$. In contrast, if we only consider only the cache overhead caused by task-preemption events, then the interface of the system is given by $\mu''_{C_1} = \langle 80, 74, 1 \rangle$. Based on Theorem 11, the maximum number of full VCPUs is $1 + 74/80 = 2$. Therefore, the interface computed by the BASELINE method has a larger bandwidth than the maximum number of full VCPUs given by Lemma 11.

Since the interface $\bar{\mu}$ given by Lemma 11 does not always have a smaller bandwidth than the interface computed using the BASELINE method, we combine the two interfaces to derive the minimum-bandwidth DMPR interface in the presence of overhead, as is given by Theorem 6. The correctness of this theorem is derived directly from the correctness of Lemma 11 and Theorem 5.

Theorem 6 *Let C be a component with a taskset $\tau = \{\tau_1, ..., \tau_n\}$ that is schedulable by the gEDF scheduler, where $\tau_k = (p_k, e_k, d_k)$ for all $1 \leq k \leq n$. Suppose $\mu'_C = \langle \Pi, \Theta', m' \rangle$ is the feasible DMPR interface given by Theorem 5, and m'' is the maximum number of full VCPUs of C given by Lemma 11. Then, the component C is schedulable under the DMPR interface μ_C, where $\mu_C = \mu'_C$ if $m'' > m' + \frac{\Theta'}{\Pi}$, and $\mu_C = \langle \Pi, 0, m'' \rangle$ otherwise.*

Interface computation under the TASK-CENTRIC-UB method: Based on the above results, the overhead-aware interface for a system can be obtained by first computing the interface for each domain using Theorem 6, and then computing the system's interface by applying the overhead-free interface computation in Section 5.

7.3 TASK-CENTRIC-UB vs. BASELINE

As was discussed in Section 7.2, the interface of a domain computed by the TASK-CENTRIC-UB method always has a bandwidth no larger than the bandwidth of the interface computed by the BASELINE method. We will show that this relationship also holds for the interfaces at the system level. We first define the dominance relation between any two analysis methods as follows:

Definition 5 A compositional analysis method CSA is said to dominate another compositional analysis method CSA' iff for any system S, the interface bandwidth of S when computed using CSA is always less than or equal to the interface bandwidth of S when computed using CSA'.

Lemma 12 *The* TASK-CENTRIC-UB *method always dominates the* BASELINE *method.*

Proof Consider a system S with D domains, $\{C_1, ..., C_D\}$. Let $\mu_{C_i} = \langle \Pi_i, \Theta_i, m_i \rangle$ and $\mu'_{C_i} = \langle \Pi_i, \Theta'_i, m'_i \rangle$ be the minimum-bandwidth DMPR interfaces of C_i under the TASK-CENTRIC-UB method and the BASELINE method, respectively. We have the following:

- Under the TASK-CENTRIC-UB method, the system has a set of partial VCPUs, $\mathsf{VP}_{\mathsf{part}} = \{\mathsf{VP}_1 = (\Pi_1, \Theta_1), ..., \mathsf{VP}_D = (\Pi_D, \Theta_D)\}$, and $(m_1 + ... + m_D)$ full VCPUs. Based on the analysis in Section 5, the minimum-bandwidth DMPR interface of S is given by $\mu_S = \langle \Pi_C, \Theta_C, m_S \rangle$, where $\mu_C = \langle \Pi_C, \Theta_C, m_C \rangle$ is the minimum-bandwidth DMPR interface for $\mathsf{VP}_{\mathsf{part}}$ and $m_S = m_C + \sum_{1 \le i \le D} m_i$.
- Under the BASELINE method, the system has a set of partial VCPUs, $\mathsf{VP}'_{\mathsf{part}} = \{\mathsf{VP}'_1 = (\Pi_1, \Theta'_1)$, ...,$\mathsf{VP}_D = (\Pi_D, \Theta'_D)\}$ and $(m'_1 + ... + m'_D)$ full VCPUs. Therefore, the minimum-bandwidth DMPR interface system is given by $\mu'_S = \langle \Pi_C, \Theta'_C, m'_S \rangle$, where $\mu'_C = \langle \Pi, \Theta'_C, m'_C \rangle$ is the minimum-bandwidth DMPR interface of the partial VCPU set $\mathsf{VP}'_{\mathsf{part}}$, and $m'_S = m'_C + \sum_{1 \le i \le D} m'_i$.

From Theorem 6, there are two cases for the relationship between μ_{C_i} and μ'_{C_i}:

1. $\Theta_i = \Theta'_i$ and $m_i = m'_i$, if the interface bandwidth computed by the BASELINE method is less than or equal to the maximum number of full VCPUs of C_i given by Lemma 11 (i.e., $m'_i + \frac{\Theta'_i}{\Pi} \le m_i + \frac{\Theta_i}{\Pi}$);
2. $\Theta_i = 0$ and $m_i \le m'_i$, otherwise.

We can conclude from the above cases that for all partial VCPUs VP_i and VP'_i computed respectively by the TASK-CENTRIC-UB method and the BASELINE method, $\mathsf{VP}_i = \mathsf{VP}'_i$, or VP_i has budget equal to 0 whereas VP'_i has budget larger than 0. In other words, $\mathsf{VP}_{\mathsf{part}} \subseteq \mathsf{VP}'_{\mathsf{part}}$.

Because $\mathsf{VP}_{\mathsf{part}}$ is only a subset of $\mathsf{VP}'_{\mathsf{part}}$, we can derive from Eq. (4) that the resource demand of $\mathsf{VP}_{\mathsf{part}}$ is always less than or equal to the resource demand of $\mathsf{VP}'_{\mathsf{part}}$. Therefore, if $\mathsf{VP}'_{\mathsf{part}}$ is schedulable under the DMPR interface μ'_C, then $\mathsf{VP}_{\mathsf{part}}$ is also schedulable under μ'_C. Because μ_C is the bandwidth-optimal DMPR interface of $\mathsf{VP}_{\mathsf{part}}$, the bandwidth of μ_C is no larger than the bandwidth of μ'_C, i.e., $\frac{\Theta_C}{\Pi_C} + m_C \le \frac{\Theta'_C}{\Pi_C} + m'_C$. In addition, $\sum_{1 \le i \le D} m_i \le \sum_{1 \le i \le D} m'_i$, because $m_i \le m'_i$. Hence, the bandwidth of μ_S, which is equal to $\frac{\Theta_C}{\Pi_C} + m_C + \sum_{1 \le i \le D} m_i$, is no larger than the bandwidth of μ'_S, which is $\frac{\Theta'_C}{\Pi_C} + m'_C + \sum_{1 \le i \le D} m'_i$. This proves the lemma. □

8 Model-centric Compositional Analysis

Recall from Section 6 that each VCPU-stop event (i.e., VCPU-preemption or VCPU-completion event) of VP_i causes at most one cache miss overhead for at most two tasks of the same domain. However, since it is

unknown which two tasks may be affected, the BASELINE method in Section 7 assumes that *every* task τ_k of the same domain is affected by *all* the VCPU-stop events of VP_i (and thus includes all of the corresponding overheads in the inflated WCET of the task). While this approach is safe, it is very conservative, especially when the number of tasks or the number of events is high.

In this section, we propose an alternative method, called MODEL-CENTRIC, that avoids the above assumption to minimize the pessimism of the analysis. The idea is to account for the total overhead due to VCPU-stop events that is incurred by all tasks in a domain, rather than by each task individually. This combined overhead is the overhead that *the domain as a whole* experiences due to VCPU-stop events under a given DMPR interface μ of the domain (since the budget of the partial VCPU of a domain is determined by the domain's interface). Therefore, the effective resource supply that a domain receives from a DMPR interface μ in the presence of VCPU-stop events is the total resource supply that μ provides, less the combined overhead.

8.1 Challenge: Resource parallel supply problem

Based on the overhead scenarios in Section 6, at first it seems possible to account for the overhead of the VCPU-preemption and VCPU-completion events by inflating the budget of an overhead-free interface with the cache-related overhead caused by the VCPU-preemption and VCPU-completion events that occur within a period of the overhead-free interface. However, this interface budget inflation approach is unsafe, due to the resource parallel supply under multicore interfaces. We illustrate this via the following scenario.

Example 7 Consider a system with a single component C that has a workload $\tau = \{\tau_1 = \tau_2 = (2, 0.1, 2), \tau_3 = (2, 1.81, 2)\}$, which is scheduled under $gEDF$. We assume that ties are broken based on increasing order of tasks' indices, i.e., a task with a smaller index has a higher priority. Suppose the cache overhead for each task is given by $\Delta_{\tau_1}^{\mathsf{crpmd}} = \Delta_{\tau_2}^{\mathsf{crpmd}} = 0.05$ and $\Delta_{\tau_3}^{\mathsf{crpmd}} = 0.2$. (The time unit is ms.) In this example, we consider only the cache overhead caused by VCPU-preemption and VCPU-completion events and assume that there are no other types of overhead.

Based on the overhead-free anlaysis in Section 5, the taskset τ is schedulable under the DMPR interface $\mu = \langle 2, 1.01, 2\rangle$. Since the interface has only one partial VCPU and this partial VCPU is not preempted by any other (full) VCPUs, the taskset τ in C experiences no VCPU-preemption event. In addition, at most one VCPU-completion event happens in a period of the DMPR interface μ. Further, based on Section 6, each VCPU-completion event causes at most two tasks to experience a cache miss. Therefore, the total cache overhead delay in a DMPR interface's period is at most $2 \max_{1 \le i \le 3}\{\Delta_{\tau_i}^{\mathsf{crpmd}}\} = 0.4$.

Suppose we inflate the budget of the overhead-free DMPR interface μ with the total cache overhead delay of 0.4. Then, we obtain the DMPR interface $\mu' = \langle 2, 1.41, 2\rangle$. However, the taskset τ is *not schedulable* under μ', as is illustrated by Fig. 6.

Fig. 6(a) shows the resource supply pattern of μ', and Fig. 6(b) shows the release and schedule patterns of the tasks in τ. Here, the tasks τ_1, τ_2, and τ_3 are released at $t = 1.01$. τ_3 migrates from VCPU_3 to VCPU_2 at $t = 1.41$ and occurs a delay of $\Delta_{\tau_3}^{\mathsf{crpmd}} = 0.2$ time units to reload its cache content (because VCPU_3 completes its budget at $t = 1.41$). τ_3 keeps running on VCPU_2 for 1.41 time units and finishes its execution at $t = 3.02$. Since τ_3's absolute deadline is $t = 3.01$, τ_3 misses its deadline.

The flaw in the cache-aware analysis approach that naïvely inflates the interface's budget comes from the resource parallel supply problem of the global multicore scheduling. In the above scenario, when τ_3 experiences cache overhead, its worst-case execution time is enlarged and thus, it needs more CPU time to execute. However, inflating the budget of the interface cannot guarantee that τ_3 receives the inflated budget, e.g., when part of the inflated budget is assigned to a VCPU that supplies resource in parallel with the VCPU on which τ_3 is running. Because τ_3 is not a parallel task and cannot execute on two cores at the same time, τ_3 does not fully utilize the inflated budget. As a result, although the extra budget is enough to account for the cache overhead τ_3 experiences, the inflated budget is not enough to guarantee the schedulability of the taskset under the resource model with inflated budget.

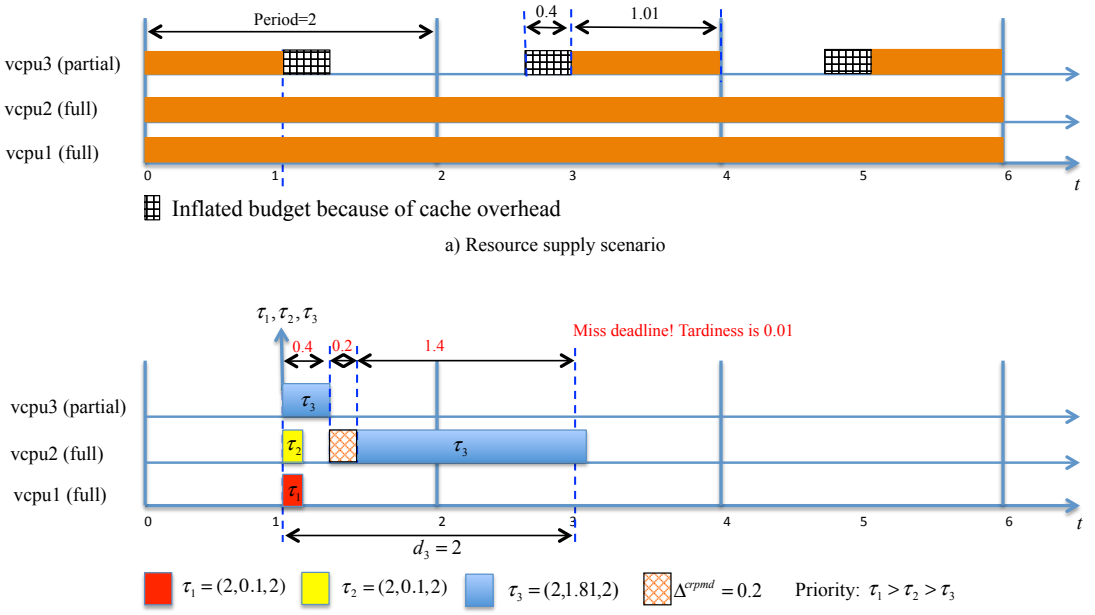


Fig. 6 Scenario of unsafe analysis of inflating interface's budget.

It is worth noting that the above overhead-aware analysis based on interface budget inflation is only safe under the assumption that the resource demand of a taskset is independent of the resource supply of the interface. However, this assumption is incorrect in the multicore setting: both the resource demand of a taskset in Eq. 4 and the resource supply of a resource mdoel in Lemma 2 depend on the number of VCPUs of a component, and they are coupled in terms of the number of VCPUs.

In the next section, we present an alternative approach that explicitly considers the effect of cache overhead on the SBF of the interface of *each* VCPU.

8.2 Cache-aware effective resource supply of a DMPR model

We first analyze the effective resource supply of a DMPR model μ, i.e., the supply it provides to a domain in the presence of the overhead caused by VCPU-stop events. We then combine the results with the overhead caused by task-preemption events to derive the schedulability and the interface of a domain.

Consider a DMPR interface $\mu = (\Pi, \Theta, m)$ of a domain $\mathcal{D}_i$, and recall that μ provides one partial VCPU $\mathsf{VP}_i = (\Pi, \Theta)$ and m full VCPUs to $\mathcal{D}_i$. Then, in the presence of overhead due to VCPU-stop events, the effective resource supply of μ consists of the effective resource supply of VP_i and the effective resource supply of m full processors. Here, the effective budget (resource) of a VCPU is the budget (resource) that is used solely to execute the tasks running on the VCPU, rather than to handle the cache misses that are caused by VCPU-stop events. We quantify each of them below.

For ease of exposition, we say that a VCPU incurs a CRPMD if the task running on the VCPU incurs the overhead caused by a VCPU-stop event, and we call a time interval $[a, b]$ an *overhead interval* of a VCPU if the effective resource the VCPU provides during $[a, b]$ is zero. (Note that the first overhead interval of VP_i in a period cannot start before VP_i begins its execution.) Finally, we call $[a, b]$ a *black-out interval* of a VCPU if it consists of overhead intervals or intervals during which the VCPU provides no resources.

Effective resource supply of the partial VCPU VP_i of μ. Recall that $N^{\mathsf{stop}}_{\mathsf{VP}_i}$ denotes the maximum number of VCPU-stop events of VP_i during each period Π. The next lemma states a worst-case condition for the effective resource supply of VP_i:

Lemma 13 *The worst-case effective resource supply of VP_i in each period occurs when VP_i has $N^{\mathsf{stop}}_{\mathsf{VP}_i}$ VCPU-stop events.*

Proof Because VP_i has a constant budget of Θ in each period Π, the more cache-related overhead it incurs in a period, the fewer effective resources it can supply to (the actual execution of) the tasks in the domain. Since the overhead that a domain's tasks incur in a period of VP_i is highest when VP_i stops its execution as many times as possible, the worst-case effective resource supply of VP_i in a period occurs when VP_i has the maximum number of VCPU-stop events, which is $N^{\mathsf{stop}}_{\mathsf{VP}_i}$ events. Hence, the lemma. □

Based on this lemma, we can construct the worst-case scenario during which the effective resource supply of VP_i is minimal, and we can derive the effective supply bound function according to this worst-case scenario.

Lemma 14 *The effective resource supply that VP_i provides during $\mathcal{I}$ is minimal when (1) VP_i provides its budget as early as possible in the current period and as late as possible in the subsequent periods, (2) VP_i has as many VCPU-stop events as possible in each period, and (3) the interval $\mathcal{I}$ begins in the current period of VP_i and the total length of the black-out intervals that overlap with $\mathcal{I}$ is maximal.*

Proof Suppose VP_i provides Θ resource units in each of its period. Denote by ScenarioA and ScenarioB the effective resource supply scenarios described in Claim 1 and the worst-case supply scenario. Further, denote by $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}(t)$ and $\overline{\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}}(t)$ the effective resource supply of VP_i over any interval of length t in ScenarioA and ScenarioB, respectively. Then, $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}(t) \geq \overline{\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}}(t)$. Let the effective resource supply in each period of VP_i in ScenarioB be $\overline{\Theta^*}$. Because there is at most $N^{\mathsf{stop}}_{\mathsf{VP}_i}$ cache misses during each period of VP_i, $\overline{\Theta^*} \geq \Theta - N^{\mathsf{stop}}_{\mathsf{VP}_i}\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i} = \Theta^*$, where Θ^* is the effective budget that VP_i provides in each period in ScenarioA. There are two cases:
Case 1) $\Theta \leq N^{\mathsf{stop}}_{\mathsf{VP}_i}\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i}$**:** We have $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}(t) = 0$. Because $\overline{\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}}(t) \leq \mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}(t)$, VP_i can provide at most Θ^* effective budget in each period under ScenarioB, where $\Theta^* = \Theta - N^{\mathsf{stop}}_{\mathsf{VP}_i}\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i}$. In other words, $\overline{\Theta^*} \leq \Theta^*$. Since $\Theta^* \leq \overline{\Theta^*}$, we obtain $\overline{\Theta^*} = \Theta^*$.

Case 2) $\Theta > N^{\mathsf{stop}}_{\mathsf{VP}_i}\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i}$**:** There are five sub-cases, as follows:

(a) $t \leq x + z$: We have $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}(t) = 0$. Because $\overline{\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}}(t) \leq \mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}(t)$, VP_i in ScenarioB must provide its budget as early as possible in the current period and as late as possible in the next period (as is shown in the interval $[t_3, t_5]$ in ScenarioA), so that it can guarantee that $\overline{\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}}(t) = 0$. Further, because VP_i must provide at most Θ^* time units during each period Π, VP_i always provides effective resource when t is enlarged. Therefore, the maximum length of the black-out interval is $x + z$.
(b) $x + z < t \leq x + z + \Theta^*$: Since VP_i provides $\overline{\Theta^*}$ resource units in each period and the whole second period of ScenarioB overlaps with the interval $\mathcal{I}$, VP_i must provide Θ^* resource units at the end of the Θ^* time unit interval of the second period. Thus, ScenarioB is the same as ScenarioA during the interval $[t_5, t_6]$.
(c) $x + z + \Theta^* < t \leq x + 2z + \Theta^*$: $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}(t) = \Theta^*$ and VP_i in ScenarioA provides no effective resource during $[t_6, t_7]$. Therefore, VP_i in ScenarioB also provides no effective resource during $[t_6, t_7]$ (since $\overline{\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}}(t) \leq \Theta^*$).
(d) $x + 2z + \Theta^* < t \leq x + 2z + 2\Theta^*$: Similar to the sub-case (b) above, VP_i in ScenarioB must provide Θ^* time units during $[t_7, t_8]$ (because otherwise, it cannot provide Θ^* time units in each period).
(e) By repeating the sub-cases (c) and (d), we can prove that VP_i in ScenarioB provides no less effective resource than that in ScenarioA.

From the above, we imply that ScenarioA is the worst-case effective resource supply scenario of VP_i. Hence, the lemma. □

Lemma 15 *The effective supply bound function of the partial VCPU* $\mathsf{VP}_i = (\Pi, \Theta)$ *of a resource model* $\mu = (\Pi, \Theta, m)$ *of a component* C *is*

$$\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}(t) = \begin{cases} y\Theta^* + \max\{0, t - x - y\Pi - z\}, & \text{if } \Theta > N^{\mathsf{stop}}_{\mathsf{VP}_i}\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i} \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

where $\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i} = \max_{\tau_i \in C}\{\Delta^{\mathsf{crpmd}}_{\tau_i}\}$, $\Theta^* = \Theta - N^{\mathsf{stop}}_{\mathsf{VP}_i}\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i}$, $x = \Pi - \Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i} - \Theta^*$, $y = \lfloor \frac{t-x}{\Pi} \rfloor$ *and* $z = \Pi - \Theta^*$.

Proof Let $\mathcal{I}$ be any interval of length t. We will prove the lemma based on the worst-case resource supply scenario given by Lemma 14.

Fig. 7 illustrates the worst-case scenario described in Lemma 14, where $\mathcal{I}$ begins at time t_3 and the intervals during which VP_i provides effective resources are $[t_2, t_3]$, $[t_5, t_6]$ and $[t_7, t_8]$:

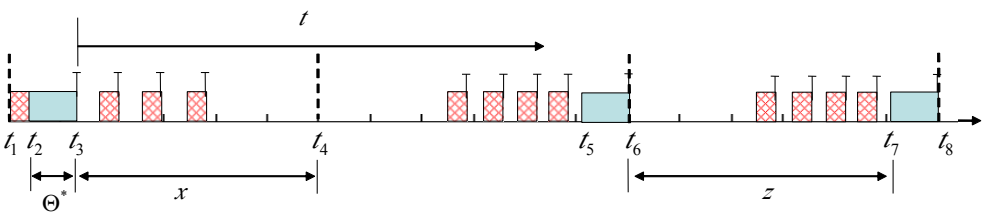


Fig. 7 Worst-case effective resource supply of $\mathsf{VP}_i = (\Pi, \Theta)$.

In the figure, the first overhead interval of VP_i in a period starts when VP_i first begins its execution in that period. This first overhead interval is caused by the VCPU-completion event of VP_i that occurs in the previous period. Recall from Lemma 13 that the maximum number of VCPU-stop events of VP_i in a period Π is $N^{\mathsf{stop}}_{\mathsf{VP}_i}$. Further, according to the gEDF scheduling of component C, any task in C may run the partial VCPU and experience the cache overhead caused by the VCPU-stop event. Therefore, the maximum overhead a task in component C experiences due to a VCPU-stop event of VP_i is $\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i} = \max_{\tau_i \in C}\{\Delta^{\mathsf{crpmd}}_{\tau_i}\}$. As a result, the effective budget is $\Theta^* \geq \Theta - N^{\mathsf{stop}}_{\mathsf{VP}_i}\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i}$. Further, we have:

$$\begin{aligned} t_3 - t_2 &\geq \Theta - (N^{\mathsf{stop}}_{\mathsf{VP}_i} - 1)\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i} - (t_2 - t_1) = \Theta^* + \Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i} - (t_2 - t_1); \\ x &= t_4 - t_3 = (t_4 - t_1) - (t_3 - t_2) - (t_2 - t_1) \leq \Pi - \Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i} - \Theta^*; \\ z &= t_7 - t_6 = (t_8 - t_6) - (t_8 - t_7) \leq \Pi - \Theta^*. \end{aligned}$$

Based on this information, we can derive the minimum effective resource supply during the interval $\mathcal{I}$ as follows: if $\Theta \leq N^{\mathsf{stop}}_{\mathsf{VP}_i}\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i}$, then $\Theta^* = 0$ and $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i} = 0$; otherwise, $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}(t) = y\Theta^* + \max\{0, t - x - y\Pi - z\}$. In addition, $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}(t)$ is minimal when $\Theta^* = \Theta - N^{\mathsf{stop}}_{\mathsf{VP}_i}\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i}$ and $x = \Pi - \Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i} - \Theta^*$. Therefore, Equation 15 gives the minimum effective resource supply of the worst-case effective resource supply scenario described in Lemma 14. This proves the lemma. □

Effective resource supply of all m full VCPUs of μ. Similar to the partial-VCPU case, we can also establish a worst-case condition for the total effective resource supply of the full VCPUs:

Lemma 16 *The* m *full VCPUs provide the worst-case total effective resource supply when they incur* $N^{\mathsf{stop}}_{\mathsf{VP}_i}$ *CRPMDs in total during each period* Π *of the partial* VP_i *of* μ.

Proof Because the total resource supply of m full VCPUs in any interval of length t is always mt, these VCPUs together provide the least effective resource supply when they incur the maximum number of CRPMDs. Recall from Section 6 that, when a VCPU-stop event of the partial VCPU VP_i of a domain $\mathcal{D}_i$ occurs, it

causes one CRPMD in a full VCPU of the same domain. Hence, the total number of CRMPDs that these full VCPUs incur together is the number of VCPU-stop events of the partial VCPU VP_i of the same domain. The lemma then follows from a combination with Lemma 13. □

The next lemma gives the worst-case supply scenarios of m full VCPUs. Fig. 8 illustrates one of the conditions under this worst-case scenario.

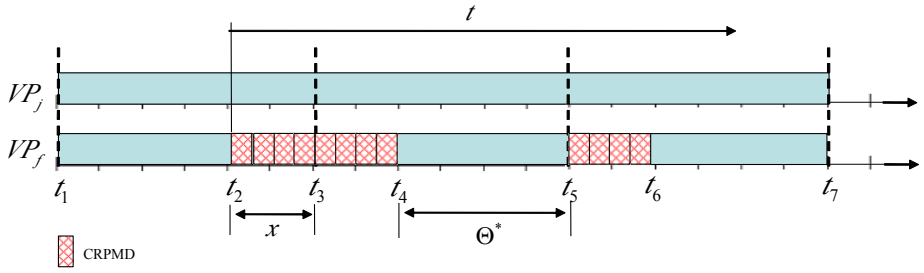


Fig. 8 Worst-case resource supply of m full VCPUs of μ.

Lemma 17 *The worst-case effective resource supply of m full VCPUs of μ in any interval $\mathcal{I}$ of length t occurs when (1) all the $N^{\mathsf{stop}}_{\mathsf{VP}_i}$ CRPMDs are experienced by one full VCPU VP_f in each period Π of VP_i, (2) VP_f incurs the overhead as late as possible in the first period and as early as possible in the rest of periods of VP_i, (3) the maximum overhead cost of each CRPMD overhead is $\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i}$, and (4) the interval $\mathcal{I}$ begins when the first CRPMD occurs in the first period.*

Proof We denote by ScenarioA the effective resource supply scenario given by Lemma 17 (see Fig. 8), and let ScenarioB be a worst-case effective resource supply scenario of the m full VCPUs. Let $x = N^{\mathsf{stop}}_{\mathsf{VP}_i}\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i}$. We will prove that the m full VCPUs provides no less effective resource in ScenarioB than in ScenarioA with the following arguments:

1. While a full VCPU VP_f is experiencing a CRPMD, the resource provided by any other full VCPU VP_j is unavailable to the task currently running on VP_f (since this task cannot execute on more than one VCPUs at any given time). Since it is unknown which exact task in the domain is running on VP_f, it is unknown whether VP_j is available to a given task. Hence, we consider VP_j as unavailable to every task while VP_f is experiencing the overhead, so as to guarantee the safety of the schedulability analysis. Recall from Lemma 16 that, all m full VCPUs incur $N^{\mathsf{stop}}_{\mathsf{VP}_i}$ CRPMDs in each period. The unavailable intervals of each period Π is maximized when all these $N^{\mathsf{stop}}_{\mathsf{VP}_i}$ CRPMDs are incurred by one full VCPU VP_f in each period Π of VP_i. Hence, ScenarioB must obey Condition (1).
2. The maximum total length of the unavailable intervals of m full VCPUs in each period is $x = N^{\mathsf{stop}}_{\mathsf{VP}_i}\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i}$. The maximum black-out interval happens when the unavailable intervals in two periods are consecutive and the maximum cost of each CRPMD is $\Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i}$. Therefore, the full VCPU VP_f should incur the overhead as late as possible in the first period and as early as possible in the second period of VP_i in order for the black-out interval to be maximized. In addition, the interval $\mathcal{I}$ should begin when the first CRPMD occurs in the first period. Hence, ScenarioB should obey the conditions (3) and (4), and the m full VCPUs provide no less effective resource in ScenarioB than in ScenarioA when $t \leq 2x$.
3. When $x + k\Pi < t < 2x + k\Pi$ ($k \in N$), because m full VCPUs must provide $m(\Pi - x)$ effective resource units in each period and the interval t has k periods, the m full VCPUs in ScenarioB should provide at least $km(\Pi - x)$ effective resource units during a time interval of length t. Because $t > x + k\Pi$, the m full VCPUs in ScenarioB have already provided $km(\Pi - x)$ effective resource units during the interval of length $x + k\Pi$. Therefore, they must provide no effective resource in the remaining time interval of length $t - (x + k\Pi)$ (otherwise, the m full VCPUs would provide more effective resource in

ScenarioB than in ScenarioA.) Hence, VP_f should incur the overhead as early as possible in all periods (except for the first period) of VP_i. Hence, by combining the the arguments (2) and (3), we imply that ScenarioB must obey Condition (2) and the m full VCPUs provide no less effective resource in ScenarioB than in ScenarioA when $x + k\Pi < t < 2x + k\Pi$.

4. When $2x + k\Pi < t < x + (k+1)\Pi$ ($k \in N$), the m full VCPUs in ScenarioB provides no effective resource during $[x + k\Pi, 2x + k\Pi]$ according to the argument (3). In addition, the m full VCPUs in ScenarioB must provide $m(\Pi - x)$ effective resource units during $[x + k\Pi, x + (k+1)\Pi]$, i.e., the $(k+1)^{th}$ period of VP_i, in order to guarantee $m(\Pi - x)$ effective resource units during the $(k+1)^{th}$ period of VP_i. Therefore, the m VCPUs in ScenarioB always provides the same effective resource during $[2x + k\Pi, x + (k+1)\Pi]$ as in ScenarioA. Hence, they provide no less effective resource in ScenarioB than in ScenarioA when $2x + k\Pi < t < x + (k+1)\Pi$.

Because the m full VCPUs provide no less effective resource in ScenarioB than in ScenarioA, and ScenarioB is a worst-case effective resource supply scenario, we imply that ScenarioA is also a worst-case effective resource supply scenario of the m full VCPUs. Hence, the lemma. □

The next lemma gives the effective SBF of the m full VCPUs of μ based on the worst-case scenario described in Lemma 17.

Lemma 18 *The effective resource supply bound function of the m full VCPUs of μ is given by:*

$$\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VPs}}(t) = \begin{cases} m\big(y\Theta' + \max\{0, t - y\Pi - 2x\}\big) & \text{if } \Theta \neq 0 \\ mt & \text{if } \Theta = 0 \end{cases} \tag{16}$$

where $x = N^{\mathsf{stop}}_{\mathsf{VP}_i} \Delta^{\mathsf{crpmd}}_{\mathsf{VP}_i}$, $y = \lfloor \frac{t-x}{\Pi} \rfloor$ and $\Theta' = \Pi - x$.

Proof The effective resource supply bound function $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VPs}}(t)$ of the resource supply scenario given by Lemma 17 is given by: When $t < 2x$, $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VPs}}(t) = 0$; When $x + k\Pi < t < 2x + k\Pi$, $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VPs}}(t) = km(\Pi - x)$; When $2x + k\Pi < t < x + (k+1)\Pi$, $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VPs}}(t) = km(\Pi - x) + m(t - 2x - k\Pi)$. Equation 16 is derived by rearranging the equations of $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VPs}}(t)$. Since the resource supply scenario given by Lemma 17 is a worst-case scenario, $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VPs}}(t)$ is the effective resource supply bound function of the m full VCPUs of μ. □

Effective resource supply of a DMPR model The next lemma gives the effective resource supply that a DMPR interface $\mu = (\Pi, \Theta, m)$ provides to a domain $\mathcal{D}_i$ after having accounted for the overhead due to VCPU-stop events. The lemma is a direct consequence of Lemmas 15 and 18.

Lemma 19 *The effective resource supply of a DMPR interface $\mu = \langle \Pi, \Theta, m \rangle$ of a domain $\mathcal{D}_i$ after having accounted for the overhead due to VCPU-stop events is given by:*

$$\mathsf{SBF}^{\mathsf{stop}}_{\mu}(t) = \mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}(t) + \mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VPs}}(t), \ \forall\, t \geq 0. \tag{17}$$

Here, $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}(t)$ is the effective resource supply of the partial VCPU $\mathsf{VP}_i = (\Pi, \Theta)$, which is given by Eq. (15)*, and $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VPs}}(t)$ is the effective resource supply of the m full VCPUs of μ, which is given by Eq.* (16)*.*

Proof Since the resource supply of a DMPR interface is the total effective resource supply of its partial VCPU and full VCPUs, the lemma directly follows from the definition of $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}(t)$ and $\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VPs}}(t)$. □

Note that, when no partial VCPU exists for interface $\mu = \langle \Pi, 0, m \rangle$, the effective resource supply of μ is equal to the resource supply of μ, i.e., $\mathsf{SBF}^{stop}_{\mu}(t) = mt$.

8.3 DMPR interface computation under MODEL-CENTRIC method

Based on the effective supply function, we can develop the component schedulability test as follows.

Theorem 7 *Consider a domain $\mathcal{D}_i$ with a taskset $\tau = \{\tau_1, ...\tau_n\}$, where $\tau_k = (p_k, e_k, d_k)$. Let $\tau'' = \{\tau''_1, ...\tau''_n\}$, where $\tau''_k = (p_k, e''_k, d_k)$ and $e''_k = e_k + max_{\tau_i \in LP(\tau_k)} \Delta^{\mathsf{crpmd}}_{\tau_i}$*[8] *for all $1 \leq k \leq n$. Then, $\mathcal{D}_i$ is*

[8] Recall that $\mathsf{LP}(\tau_k) = \{\tau_i | d_i > d_k\}$

schedulable under gEDF by a DMPR model μ in the presence of cache-related overhead, if the inflated taskset τ'' is schedulable under gEDF by the effective resource supply $\mathsf{SBF}_{\mu}^{\mathsf{stop}}(t)$ in the absence of overhead.

Proof Since τ'' includes the overhead that τ incurs due to task-preemption events, if $\mathsf{SBF}_{\mu}^{\mathsf{stop}}(t)$ is sufficient to schedule τ'' assuming negligible overhead, then it is also sufficient to schedule τ in the presence of task-preemption events. As $\mathsf{SBF}_{\mu}^{\mathsf{stop}}(t)$ gives the effective supply that μ provides to τ after having accounted for the overhead due to VCPU-stop events, μ provides sufficient resources to schedule τ in the presence of the overhead from all types of events. This proves the theorem.

Based on the above results, we can generate a cache-aware minimum-bandwidth DMPR interface for a domain in the same manner as in the overhead-free case, except that we use the effective resource supply and the inflated taskset in the schedulability test. Similarly, the system's interface can be computed from the interfaces of the domains in the exact same way as the overhead-free interface computation.

9 Hybrid cache-aware DMPR interface

Recall from Section 7 that the TASK-CENTRIC-UB method always dominates the BASELINE method. However, neither of these analysis methods dominates the MODEL-CENTRIC method, and vice versa. We demonstrate this using two example systems, where the TASK-CENTRIC-UB method gives a smaller interface bandwidth in the first system but a larger interface bandwidth in the second system compared to the interface bandwidth given by the MODEL-CENTRIC method.

Example 8 Let Sys_1 be a system consisting of two domains C_1 and C_2 that are scheduled under the hybrid EDF scheduling strategy (c.f. Section 2) and that have workloads $\tau_{C_1} = \{\tau_1^1 = ... = \tau_1^4 = (200, 100, 200)\}$ and $\tau_{C_2} = \{\tau_2^1 = \tau_2^2 = (200, 100, 200)\}$, respectively. By applying the analysis in Sections 7.2 and 8, the interfaces of the system under TASK-CENTRIC-UB and under MODEL-CENTRIC are computed to be $\mu_{\mathsf{Sys}_1} = \langle 20, 17, 5\rangle$ and $\mu'_{\mathsf{Sys}_1} = \langle 20, 19, 5\rangle$, respectively. Thus, the system's interface under TASK-CENTRIC-UB has a smaller bandwidth than that of the interface computed under MODEL-CENTRIC.

Example 9 Let Sys_2 be a system consisting of two domains C_1 and C_2 that are scheduled under the hybrid EDF scheduling strategy and that have workloads $\tau_{C_1} = \{\tau_1^1, ..., \tau_1^5 = (100, 5, 100)\}$ and $\tau_{C_2} = \{\tau_2^1, ..., \tau_2^5 = (100, 5, 100)\}$, respectively. The interfaces of this system under TASK-CENTRIC-UB and under MODEL-CENTRIC are given by $\mu_{\mathsf{Sys}_2} = \langle 20, 0, 4\rangle$ and $\mu'_{\mathsf{Sys}_2} = \langle 20, 14, 3\rangle$, respectively. Thus, the system's interface under TASK-CENTRIC-UB has a larger bandwidth than that of the interface computed under MODEL-CENTRIC.

One can also show that neither MODEL-CENTRIC nor BASELINE dominates one another. For instance, consider the system Sys_1 in Example 8. The interface of the whole system under the BASELINE method is $\mu''_{\mathsf{Sys}_1} = \langle 20, 17, 5\rangle$, which has a smaller bandwidth than the interface μ'_{Sys_1} computed using the MODEL-CENTRIC method. Further, since the TASK-CENTRIC-UB method dominates the BASELINE method but not the MODEL-CENTRIC method, the BASELINE method also does not dominate the MODEL-CENTRIC method.

From the above observations, we can derive the minimum interface of a component from the ones computed using the TASK-CENTRIC-UB and MODEL-CENTRIC methods (since TASK-CENTRIC-UB method always dominates BASELINE), as stated by Theorem 8. The theorem is trivially true, since both interfaces computed using the TASK-CENTRIC-UB and MODEL-CENTRIC methods are safe. We refer to this analysis as the HYBRID method.

Theorem 8 (Hybrid cache-aware interface) *The minimum cache-aware DMPR interface of a domain $\mathcal{D}_i$ (a system $\mathcal{S}$) is the interface that has a smaller resource bandwidth between μ_{task} and μ_{model}, where μ_{task} and μ_{model} are the minimum-bandwidth DMPR interfaces of $\mathcal{D}_i$ ($\mathcal{S}$) computed using the* TASK-CENTRIC-UB *and the* MODEL-CENTRIC *methods, respectively.*

Discussion. We observe that the schedulability analysis under gEDF in the absence of overhead (Theorem 1) is only a sufficient test, and that its pessimism degree varies significantly with the characteristics of the

taskset. For instance, under the same multiprocessor resource, one taskset with a larger total utilization may be schedulable while another with a smaller total utilization may not be schedulable. As a result, it is possible that the overhead-aware interface of a domain (system) may require less resource bandwidth than the overhead-free interface of the same domain (system).

10 Evaluation

To evaluate the benefits of our proposed interface model and cache-aware compositional analysis, we performed simulations using randomly generated workloads. We had five main objectives for our evaluation: (1) Determine how much resource bandwidth the interfaces computed using the improved SBF (Section 3.2) can save compared to the interfaces computed using the original SBF proposed in [Easwaran et al., 2009]; (2) determine how much resource bandwidth the DMPR model can save compared to the MPR model; (3) evaluate the relative performance of the HYBRID method and the BASELINE method; (4) study the impact of task parameters (e.g., the range of taskset utilization, the distribution of task's utilization, the period range of tasks) on the interfaces under the HYBRID and BASELINE methods; and (5) evaluate the performance of the HYBRID analysis when using a cache overhead value per task and when using the maximum cache overhead value for the entire system.

10.1 Experimental setup

Key factors. We focus on the following five key factors that can affect the performance of a cache-aware compositional analysis:[9]:

- *Utilization of a task set.* Tasks with larger utilizations tend to have a larger number of tasks; thus, each task tends to experience more cache overhead during its lifetime because there are more other tasks that can preempt it.
- *Distribution of task utilizations.* High-utilization tasks are more sensitive to cache overhead and can more easily become unschedulable because of this overhead than tasks with small utilization.
- *Periods of the tasks.* If two tasks have the same utilization and experience the same cache overhead, the task with the smaller period has a higher probability of missing its deadline because of the overhead than the task with the larger period because the former has a smaller relative deadline. Therefore tasks with smaller period are more sensitive to cache overhead.
- *Number of tasks in a task set.* In the BASELINE approach and the task-centric approach from Section 7, when a VCPU-stop event happens, each task's worst-case execution time is inflated by the cache overhead caused by this event, even though at most two tasks actually experience the cache overhead that the event has caused. Hence, these two approaches will become more and more pessimistic as the number of tasks increases.
- *Cost of cache overhead per event.* If the cost of cache overhead increases, tasks will experience longer delays when task-preemption or VCPU-stop events occur.

Workload. In order to evaluate the impact of the above five factors on the performance of overhead-free and overhead-aware compositional analysis, we generated a number of synthetic real-time workloads with randomly generated periodic task sets that span a range of different parameters for each of these factors. Below, we explain how the parameters were chosen.

We picked the *task set utilizations* from the interval $[0, 24]$, with increments of 0.2, to be consistent with the ranges used in [Brandenburg et al., 2011] and [Brandenburg, 2011]. However, we observed that a smaller interval is sufficient to demonstrate the relative performance of overhead-free and overhead-aware compositional analysis; hence, we used the range $[0, 5]$, again with increments of 0.2, when evaluating the impact of the other factors on overhead-aware compositional analysis.

[9] We assume other factors are same when we discuss one factor's impact on the cache-aware analysis

The *tasks' utilizations* were drawn from one of four distributions: one uniform distribution over the range $[0.001, 0.1]$ and three bimodal distributions; in the latter, the utilization was distributed uniformly over either $[0.1, 0.5)$ or $[0.5, 0.9]$, with respective probabilities of 8/9 and 1/9 (light), 6/9 and 3/9 (medium), and 4/9 and 5/9 (heavy). These probabilities are consistent with the ones used in [Bastoni et al., 2010] and [Brandenburg, 2011]. The *periods* of the tasks were drawn from a uniform distribution over one of the following three ranges: $(350ms, 850ms)$, $(550ms, 650ms)$, and $(100ms, 1100ms)$; all periods were integer. These distributions are identical to those used in [Lee et al., 2011]. The *number of tasks* in a task set ranged from $[0, 300]$ with increments of 20.

The *cost of cache overhead per event* was chosen based on the cache overhead ratio, which we define as the cache overhead of a task τ_i divided by the worst-case execution time of τ_i. We picked the cache overhead ratio from the range $[0, 0.1]$ with increments of 0.01. This range was chosen based on measurements of the L2 cache miss overhead of tasks on our experimental platform; we found that the cost of missing the L2 private cache but hitting the L3 shared cache was $0.02ms$ when the working set size was $256KB$ (the L2 private cache size). Because the L3 cache hit latency is very small (less than 100 cycles), the cache overhead per task-preemption or VCPU-stop event is only $0.02ms$. Therefore, the cache overhead ratio was less than 0.02 for any task we measured that had a worst-case execution time of more than $2ms$.

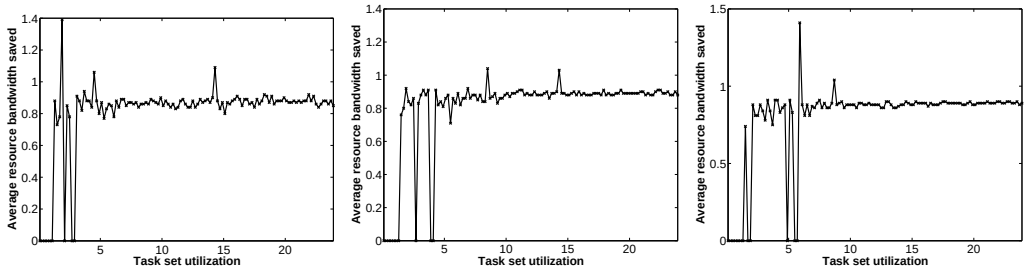
Overhead measurements. For our measurements, we used a Dell Precision T3610 six-core workstation with the RT-Xen 2.0 platform [Xi et al., 2014]; each domain was running LITMUS^{RT} 2012.3 [Calandrino et al., 2006]. The scheduler was gEDF in the domains and semi-partitioned EDF in the VMM, as described in Section 2. We allocated a full-capacity VCPU to one domain and pinned this VCPU to a physical core of its own; this was done to avoid interference from domain 0 (the administrative domain in RT-Xen), which was pinned to a different core. We measured the cache overhead of the cache-intensive program ρ as follows. First we warmed up the cache by accessing all the cache content of the program; then we used the time stamp counter to measure the time l_{hit} it takes to access the same content again. Because the cache was warm, l_{hit} is the cache hit latency of this program. Next, we allocated an array of the same size as the private L2 cache and loaded this into the same core's L2 cache in order to pollute the cache content of ρ. Finally, we again accessed all the cache content of ρ and recorded the cache miss latency l_{miss}. The cache overhead of the program ρ per task-preemption or VCPU-stop event is then $l_{miss} - l_{hit}$.

10.2 Overhead-free analysis

We begin with an empirical comparison of the overhead-free analyses. For this purpose, we set up four domains with harmonic periods, and we randomly generated tasks and uniformly distributed them across the four domains. To be consistent with [Phan et al., 2013], we generated 25 task sets per task set utilization or task set size.

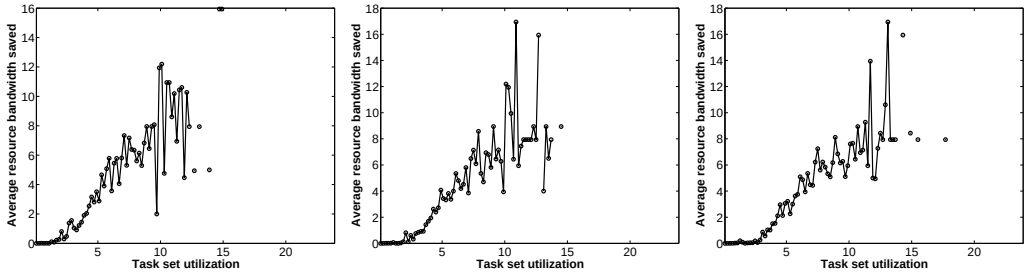
MPR with improved SBF vs. MPR with original SBF. To estimate the impact of the improved SBF, we generated 625 tasksets with taskset utilizations ranging from 0.1 to 24, with increments of 0.2. The task utilizations were drawn from the bimodal-light distribution as described earlier; the tasks' periods were uniformly distributed across $[350ms, 850ms]$. For each taskset we generated, we distributed the tasks into *one domain*, and we then computed the overhead-free interface of the domain using MPR with the improved SBF, as well as using the original MPR. Fig. 9(a) shows the average bandwidth savings due to the improved SBF. We observe that, across all taskset utilizations, MPR with the improved SBF always requires either the same or less resource bandwidth than MPR with the original SBF. We also observe that MPR with the improved SBF saves over 0.8 cores when the taskset utilization is larger than 5. Fig. 9(b) and 9(c) show the average resource bandwidth savings with the other two bi-modal distributions; we observe that, in all three cases, MPR with the improved SBF consistently outperformed MPR with the original SBF.

DMPR vs. MPR with the original SBF. To compare DMPR to MPR with the original SBF on the whole system, we distributed the tasks in each taskset over **four domains** and we then computed the overhead-free



(a) Bimodal-light distribution. (b) Bimodal-medium distribution. (c) Bimodal-heavy distribution.

Fig. 9 Average resource bandwidth saved: MPR with improved SBF vs. MPR with original SBF.



(a) Bimodal-light distribution. (b) Bimodal-medium distribution. (c) Bimodal-heavy distribution.

Fig. 10 Average resource bandwidth saved: DMPR vs. MPR with original SBF.

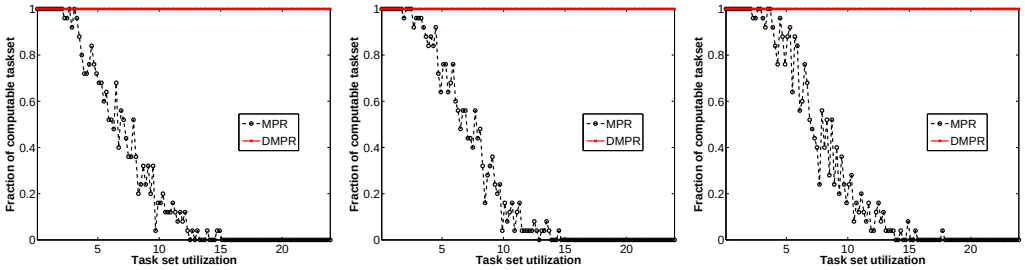
interface of the whole system using both DMPR and MPR with the original SBF. Fig. 10(a) shows the average bandwidth savings of DMPR for different taskset utilizations. Our results show that DMPR consistently saves bandwidth relative to MPR with the original SBF for up to 16 cores. There are very few data points beyond this point because we can only compute the average bandwidth savings when both analyses return valid interfaces for the same taskset; however, for taskset utilizations above 16, MPR generally fails to compute a valid interface for the system.

As shown in Fig. 11(a), the fraction of tasksets with valid interfaces under MPR with the original SBF decreases with increasing taskset utilization. This is because the original SBF of MPR is pessimistic and cannot provide $m't$ time units with interface $\Gamma = \langle \Pi, \Pi m', m' \rangle$. Once the interfaces of the leaf components (i.e., domains) have been computed, these interfaces are transferred to VCPUs as the workload of the top component. When some of those VCPUs have utilization 1, the resource demand increases faster than the resource supply of MPR with the original SBF; hence, MPR cannot find a valid interface. DMPR does not have this problem because it can always supply $m't$ time units with bandwidth m'; hence, the fraction of tasksets with valid interfaces is always 1. As Fig. 11(b) and Fig. 11(c) show, the results for the other two bimodal distributions are similar: DMPR is consistently able to compute interfaces for all tasksets, whereas MPR with the original SBF finds fewer and fewer interfaces as the taskset utilization increases.

10.3 Comparison of HYBRID cache-aware analysis vs. BASELINE cache-aware analysis

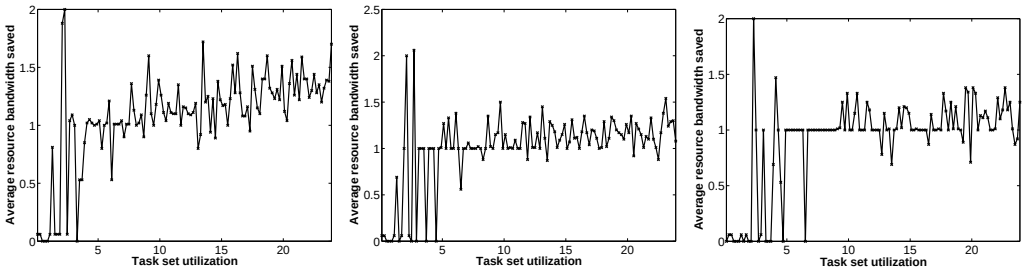
Next, we compared the performance of the two overhead-aware analysis approaches. For this we used the same tasksets and system configuration as for the previous experiment, but we additionally computed DMPR interfaces for each taskset using the respective approach.

Impact of taskset utilization. Fig. 13(a) shows the average resource bandwidth savings of the HYBRID approach compared to the BASELINE approach for each taskset utilization. We observe that a) HYBRID reduced the resource bandwidth in all cases, and that b) more and more cores are being saved as the taskset utilization



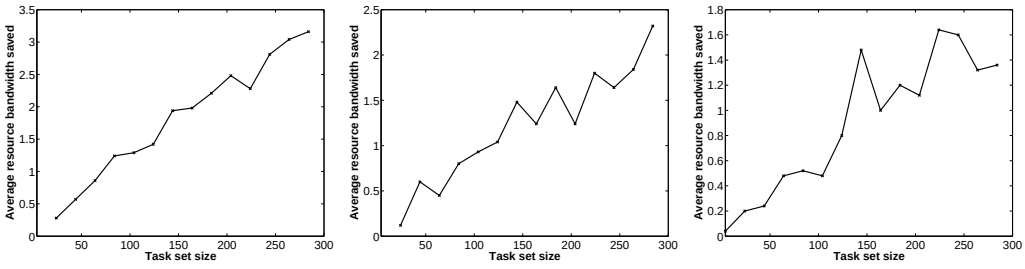
(a) Bimodal-light distribution. (b) Bimodal-medium distribution. (c) Bimodal-heavy distribution.

Fig. 11 Fraction of taskset with valid interfaces: DMPR vs. MPR with original SBF.



(a) Bimodal-light distribution. (b) Bimodal-medium distribution. (c) Bimodal-heavy distribution.

Fig. 12 Average resource bandwidth saved: HYBRID vs. BASELINE.



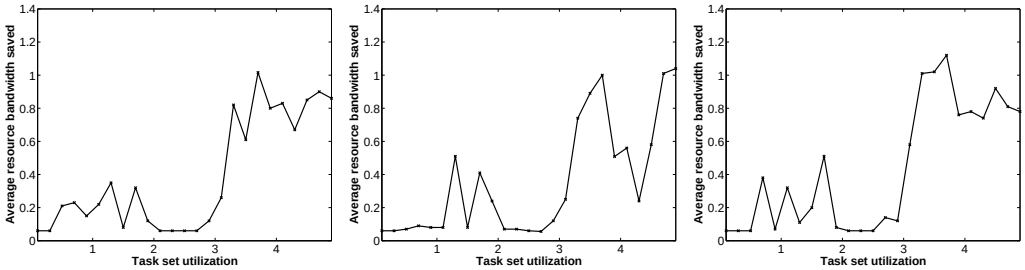
(a) Bimodal-light distribution. (b) Bimodal-medium distribution. (c) Bimodal-heavy distribution.

Fig. 13 Average resource bandwidth saved: HYBRID vs. BASELINE.

increases. Note that, as the taskset utilization increases, the interface bandwidth can sometimes decrease. One reason for this is that the underlying gEDF schedulability test is only sufficient, and is not strictly dependent on the taskset utilization; in other words, it is possible that a taskset with a high utilization is schedulable but another with a lower utilization is not. We also observe that, as discussed earlier, the relative performance of the HYBRID and BASELINE analyses is easy to see even for small taskset utilizations; this is why we only compare the two overhead-aware analysis for taskset utilizations $[0, 5]$ instead of the larger $[0, 24]$ range.

Impact of task utilization. Fig. 13(a)-Fig. 13(c) show the average resource bandwidth savings for different taskset utilizations and each of the three bimodal distributions. We observe that, in all three cases, the HYBRID approach consistently outperformed the BASELINE approach. Further, as the taskset utilization increases, the savings also increase and remain steady at approximately one core once the taskset utilization has reached 10.

Impact of taskset size. We investigated the impact of the number of tasks (i.e., the taskset size) on the average bandwidths saving of the HYBRID approach compared to the BASELINE approach. For this experiment, we



(a) Task period: [100, 1100]ms. (b) Task period: [350, 850]ms. (c) Task period: [550, 650]ms.

Fig. 14 Average resource bandwidth saved under different ranges of tasks' periods

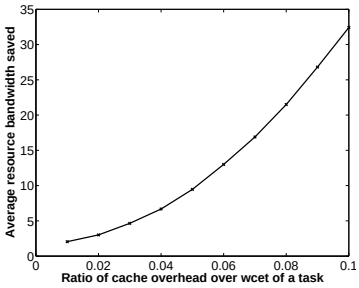


Fig. 15 Average bandwidth saving under different ratios of cache overhead to task WCET.

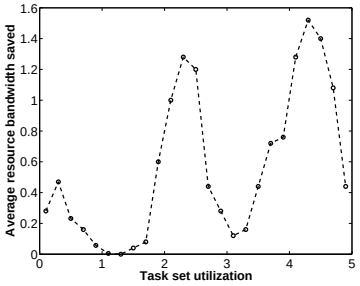


Fig. 16 Average bandwidth saving of HYBRID with cache overhead per task over HYBRID with maximum cache overhead of system (Ratio of overhead over wcet is uniformly in [0,0.1])

generated a set of tasksets with sizes between 4 to 300, with increments of 20, and with 25 tasksets per size. As before, we tried each of the three bimodal distributions we discussed in Section 10.1.

Fig. 13(a)-Fig. 13(c) show the average resource bandwidth savings for different taskset sizes with each of the three bi-modal distributions. We observe that a) the HYBRID approach consistently outperforms the BASELINE approach, and b) the savings increase with the number of tasks. This is expected because the BASELINE technique inflates the WCET of every task with all the cache-related overhead each task experiences; hence, its total cache overhead increases with the size of the taskset.

Impact of task period distribution. We further investigated the impact of the distribution of tasks' periods on the average bandwidth savings of the HYBRID approach compared to the BASELINE approach. For this experiment, we generated a number of tasksets with taskset utilizations in the range $[0, 5]$ with increments of 0.2, and, as usual, 25 tasksets per taskset utilization. The individual tasks' utilizations were drawn from the bi-modal light distribution. For the tasks' periods, we tried each of the three distributions that were discussed in Section 10.1. Fig. 14(a)-Fig. 14(c) show the average resource bandwidth saving for three different distribution of tasks' periods; in all three cases, the HYBRID approach consistently outperforms the BASELINE approach.

Impact of cost of cache overhead. We first generated 25 tasksets with taskset utilization 4.9 and uniformly distributed the tasks of each taskset over four domains with harmonic periods. The tasks' utilizations were uniformly distributed in $[0.001, 0.1]$, and their periods were uniformly distributed in $[350ms, 850ms]$. We then modified the cache overhead of tasks of the 25 tasksets and generated a set of tasksets with cache-related overhead ratio $[0, 0.1]$ with increments of 0.01 based on the 25 tasksets. Recall from Section 10.1 that we define the cache-related overhead ratio of a task τ_i to be the cost of one cache-related overhead of τ_i divided by the worst-case execution time of τ_i.

	Overhead-free MPR		Overhead-free DMPR		HYBRID		BASELINE	
	Theory	RT-Xen	Theory	RT-Xen	Theory	RT-Xen	Theory	RT-Xen
Schedulable	Yes	No	Yes	No	No	No	No	No
Deadline miss ratio		78%		78%		0.07%		7%

Table 1 Performance in theory vs. in practice.

Fig. 15 shows the average resource bandwidth savings of the HYBRID approach over the BASELINE approach for each cache overhead ratio. We observe that the HYBRID approaches saves more resources as the cache-related overhead ratio increases. This is expected because tasks' utilizations are uniformly distributed over $[0.001, 0.1]$ and a taskset has more tasks than the number of VCPUs. Since the BASELINE approach inflates the WCET of every task with all the cache-related overheads any task can experience, its total cache overhead increases as the cost of one cache-related overhead increases.

Impact of per-task cache overheads. When different tasks can have different costs for cache-related overheads, it is pessimistic to simply use the largest cache overhead in the system, as we did in [Xu et al., 2013]. To evaluate the impact of considering cache overheads *per task*, we generated tasks with different cache-related overhead ratios, drawn from an uniform distribution over $[0, 0.1]$. We then calculated the system's interface with the HYBRID analysis using the following two approaches: (1) Using a per-task cost of cache overheads to compute the HYBRID analysis, as we did in this work; and (2) Using the upper bound for the cache overhead in the system as the cost for each task, as we did in [Xu et al., 2013].

Fig. 16 shows the average resource bandwidth savings of the HYBRID approach with per-task cache overheads relative to the more pessimistic approach. We observe that the HYBRID approach with per-task cache overheads consistently outperformed the pessimistic approach; however, the saving does *not* increase as the taskset utilization increases. This is because the TASK-CENTRIC-UB approach only considers the cache overhead caused by task-preemption events, and each task's WCET is only inflated with one cache overhead. Therefore, the pessimistic HYBRID analysis with system's maximum cache overhead may have the same upper-bounded number of full VCPUs as the HYBRID analysis with cache overhead per task. When both analyses use the upper-bounded number of full VCPUs as the components' interface, the HYBRID analysis with per-task cache overheads will have the same interface bandwidth as the pessimistic analysis and thus saves no resources; however, (2) if both HYBRID analyses choose the interfaces computed by the MODEL-CENTRIC analysis, the HYBRID analysis with per-task cache overheads will save resources relative to the pessimistic approach because every time one cache-related overhead happens, the pessimistic approach will have more cache overhead.

10.4 Performance in theory vs. in practice

We also validated the correctness of the cache-aware interfaces (and the invalidity of the overhead-free interfaces) in practice. For this experiment, we first computed the domains' interfaces, and we then ran the generated tasks on our RT-Xen experimental platform. The periods and budgets of the domains in RT-Xen were chosen to be those of the respective computed interfaces. We then computed the schedulability and deadline miss ratios of the tasks, based on the theoretical schedulability test and the measurements on the RT-Xen platform. Table 1 shows the schedulability and deadline miss ratios of these methods.[10]

We observe that the overhead-free MPR and DMPR interfaces significantly underestimate the tasks' resource requirements: even though the tasks were claimed to be schedulable by the computed interfaces, 78% of the jobs missed their deadlines. The experimental results also confirm that our cache-aware analysis correctly estimated the resource requirements of the system in practice: the theory predicted that the tasks would not be schedulable, and this was confirmed in practice by the nonzero deadline miss ratio, which was

[10] We note that the interfaces given by the HYBRID method and the BASELINE method are the same as the interfaces given by the cache-aware hybrid analysis method and task-centric analysis method proposed in the conference version [Xu et al., 2013], respectively.

0.07% for the HYBRID approach and 7% for the task-centric approach. We also observe that the HYBRID approach had fewer deadline misses than, and thus outperformed, the task-centric approach.

11 Related Work

Several compositional analysis techniques for multicore platforms have been developed (see e.g., [Baruah and Fisher, 2009; Easwaran et al., 2009; Leontyev and Anderson, 2008; Lipari and Bini, 2010]) but, unlike this work, they do not consider the platform overhead. There are also methods that account for cache-related overhead in multicore schedulability analysis (e.g., [Brandenburg, 2011]), but they cannot be applied to the virtualization and compositional setting. To the best of our knowledge, the only existing overhead-aware interface analysis is for uniprocessors [Phan et al., 2013].

Prior work has already extended the multiprocessor resource model in a number of ways. Most notably, Bini et al. introduced generalizations such as the parallel supply function [Bini et al., 2009], as well as later refinements. These models capture the resource requirements at each different level of parallelism; thus, they minimize the interface abstraction overhead that the MPR model incurs. However, they also increase the complexity of the interface representation and the interface computation. Our work follows a different approach: instead of adding more information, we make the supply pattern of the resource model more deterministic. As a result, we can improve the worst-case resource supply of the model without increasing its complexity. In addition, this approach helps to reduce the platform overhead that arises when these interfaces are scheduled at the next level.

The semi-partitioned EDF scheduling we use at the VMM level is similar to the strategy proposed for soft real-time tasks by Leontyev and Anderson [Leontyev and Anderson, 2008], in which the bandwidth requirement of a container is distributed to a number of dedicated processors as well as a periodic server, which is globally scheduled onto the remaining processors. The two key differences to our work are that 1) we use gEDF within the domains, which necessitates a different analysis, and that 2) unlike our work, [Leontyev and Anderson, 2008] does not consider cache overhead.

There are other lines of cache-related research that benefit our work. For example, results on intrinsic cache analysis and WCET estimation [Hardy et al., 2009] can be used as an input to our analysis; studies on cache-related preemption and migration delay [Bastoni et al., 2010] can be used to obtain the value of cache-overhead per task value $\Delta_{\tau_i}^{\mathsf{crpmd}}$ used in our analysis; and cache-aware scheduling, such as [Guan et al., 2009], can be used to reduce the additional cache-related overhead in the compositional/virtualization setting.

12 Conclusion

We have presented a cache-aware compositional analysis technique for real-time virtualization multicore systems. Our technique accounts for the cache overhead in the component interfaces, and thus enables a safe application of the analysis theories in practice. We have developed three different approaches, BASELINE, TASK-CENTRIC-UB and MODEL-CENTRIC, for analyzing the cache-related overhead and for testing the schedulability of components in the presence of cache overhead. We have also introduced an improved supply bound function for the MPR model and a deterministic extension of the MPR model, which improve the interface resource efficiency, as well as accompanying overhead-aware interface computation methods. Our evaluation on synthetic workloads shows that our improved SBF and the DMPR interface model can help reduce resource bandwidth by a significant factor compared to the MPR model with the existing SBF, and that a hybrid of TASK-CENTRIC-UB and MODEL-CENTRIC achieves significant resource savings compared to the BASELINE method (which is based solely on WCET inflation).

Acknowledgements This research was supported in part by the ONR N000141310802, NSF CNS-1329984, NSF CNS-1117185, NSF ECCS-1135630, and MKE (The Ministry of Knowledge Economy), Korea, under the Global Collaborative R&D program supervised by the KIAT (M002300089).

References

Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP*, 2003.

S. Baruah and N. Fisher. Component-based design in multiprocessor real-time systems. In *ICESS*, 2009.

Sanjoy Baruah and Theodore Baker. Schedulability analysis of global EDF. *Real-Time Systems*, 38(3): 223–235, 2008.

Andrea Bastoni, Bjorn B. Brandenburg, and James H. Anderson. Cache-Related Preemption and Migration Delays: Empirical Approximation and Impact on Schedulability. In *OSPERT*, 2010.

Swagato Basumallick and Kelvin Nilsen. Cache issues in real-time systems. In *LCTES*, 1994.

E. Bini, M. Bertogna, and S. Baruah. Virtual multiprocessor platforms: Specification and use. In *RTSS*, 2009.

Björn B. Brandenburg. *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. PhD thesis, The University of North Carolina at Chapel Hill, 2011.

Björn B. Brandenburg, Hennadiy Leontyev, and James H. Anderson. An overview of interrupt accounting techniques for multiprocessor real-time systems. *Journal of Systems Architecture*, 57(6):638–654, 2011.

F. Bruns, S. Traboulsi, D. Szczesny, E. Gonzalez, Y. Xu, and A. Bilgic. An Evaluation of Microkernel-Based Virtualization for Embedded Real-Time Systems. In *ECRTS*, 2010.

John M. Calandrino, Hennadiy Leontyev, Aaron Block, UmaMaheswari C. Devi, and James H. Anderson. LITMUS RT: A testbed for empirically comparing real-time multiprocessor schedulers. In *RTSS*, 2006.

A. Crespo, I. Ripoll, and M. Masmano. Partitioned Embedded Architecture Based on Hypervisor: the XtratuM Approach. In *EDCC*, 2010.

Arvind Easwaran, Madhukar Anand, and Insup Lee. Compositional analysis framework using edp resource models. In *RTSS*, 2007.

Arvind Easwaran, Insik Shin, and Insup Lee. Optimal virtual cluster-based multiprocessor scheduling. *Real-Time Systems*, 43(1):25–59, 2009.

Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. Cache-aware scheduling and analysis for multicores. In *EMSOFT*, 2009.

Damien Hardy, Thomas Piquet, and Isabelle Puaut. Using bypass to tighten wcet estimates for multi-core processors with shared instruction caches. In *RTSS*, 2009.

Taesoo Kim, Marcus Peinado, and Gloria Mainar-Ruiz. System-level protection against cache-based side channel attacks in the cloud. In *USENIX Security*, 2012.

J. Lee, S. Xi, S. Chen, L. T. X. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky. Realizing compositional scheduling through virtualization. In *RTAS*, 2012.

Jaewoo Lee, Linh T. X. Phan, Sanjian Chen, Oleg Sokolsky, and Insup Lee. Improving resource utilization for compositional scheduling using DPRM interfaces. *SIGBED Rev.*, 2011.

H. Leontyev and J. H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In *ECRTS*, 2008.

Giuseppe Lipari and Enrico Bini. A framework for hierarchical scheduling on multiprocessors: From application requirements to run-time allocation. In *RTSS*, 2010.

Linh T. X. Phan, Meng Xu, Jaewoo Lee, Insup Lee, and Oleg Sokolsky. Overhead-aware compositional analysis of real-time systems. In *RTAS*, 2013.

Lui Sha, John P. Lehoczky, and Ragunathan Rajkumar. Solutions for Some Practical Problems in Prioritized Preemptive Scheduling. In *RTSS*, 1986.

I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proc. of the 24th IEEE Real-Time Systems Symposium (RTSS)*, Cancun, Maxico, 2003.

Insik Shin, A. Easwaran, and Insup Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *ECRTS*, 2008.

Sisu Xi, Meng Xu, Chenyang Lu, Linh T. X. Phan, Christopher Gill, Oleg Sokolsky, and Insup Lee. Real-time multi-core virtual machine scheduling in xen. In *EMSOFT*, 2014.

Meng Xu, Linh T. X. Phan, Insup Lee, Oleg Sokolsky, Sisu Xi, Chenyang Lu, and Christopher D. Gill. Cache-aware compositional analysis of real-time multicore virtualization platforms. In *RTSS*, 2013.