12-2013

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

Meng Xu
University of Pennsylvania, mengxu@cis.upenn.edu

Linh T.X. Phan
University of Pennsylvania, linhphan@cis.upenn.edu

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Oleg Sokolsky
University of Pennsylvania, sokolsky@cis.upenn.edu

Sisu Xi
Washington University in St Louis, xis@cse.wustl.edu

See next page for additional authors

Recommended Citation

Meng Xu, Linh T.X. Phan, Insup Lee, Oleg Sokolsky, Sisu Xi, Chenyang Lu, and Christopher Gill, "Cache-Aware Compositional Analysis of Real-Time Multicore Virtualization Platforms", *34th IEEE Real-Time Systems Symposium (RTSS 2013)* , 1-10. December 2013. http://dx.doi.org/10.1109/RTSS.2013.9

IEEE Real-Time Systems Symposium (RTSS 2013), Vancouver, Canada, December 3-6, 2013.
An extended version of this paper is available at http://repository.upenn.edu/cis_papers/786/

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

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
RT-Xen, cache-aware, compositional analysis, interface, multicore, real-time, virtualization

Disciplines
Computer Engineering | Computer Sciences

Comments
IEEE Real-Time Systems Symposium (RTSS 2013), Vancouver, Canada, December 3-6, 2013.

An extended version of this paper is available at http://repository.upenn.edu/cis_papers/786/

Author(s)
Meng Xu, Linh T.X. Phan, Insup Lee, Oleg Sokolsky, Sisu Xi, Chenyang Lu, and Christopher Gill

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

Meng Xu Linh T.X. Phan Insup Lee Oleg Sokolsky
University of Pennsylvania
{mengxu, linhphan, lee, sokolsky}@cis.upenn.edu

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

***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.

I. 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 [1], 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 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., [8], [10], [16], but they primarily focus on single-core processors.

*The published version of this paper contains a typo in Eq. 13 that has been fixed in this author version.

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 multicore systems have been developed (e.g., [3], [12], [18]), 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:

- We introduce DMPR, a deterministic extension of the multiprocessor resource periodic model to better represent component interfaces on multicore virtualization platforms (Section III);
- we present a DMPR-based compositional analysis for systems without cache-related overhead (Section IV);
- we characterize different types of events that cause cache misses in the presence of virtualization (Section V); and
- we propose two approaches, task-centric and model-centric, to account for the cache-related overhead. Based on the results, we develop the corresponding cache-aware compositional analysis methods (Sections VI and VII).

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.

II. System Descriptions

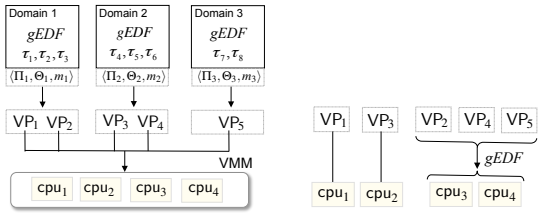
The system we consider consists of multiple real-time components that are scheduled on a multicore virtualization platform, as is illustrated in Fig. 1(a). Each component corresponds to a *domain* (virtual machine) of the platform and consists of a set of tasks; these tasks are scheduled on a set of 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 [20] 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 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[1], 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.

[1]We assume that either there is no shared cache, or the shared cache has been partitioned into cache sets so that each core has exclusive access to some number of cache sets [15].



(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 (gEDF) [2] 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 [5]. 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 Δ^{crpmd} to denote the maximum time needed to reload all the useful cache blocks (i.e., cache blocks that will be reused) of a preempted task when that task resumes (either on the same core or on a different core).[2] 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). We assume that all VCPUs of each

[2]For ease of exposition, we use a single bound on the cache-related overhead; however, our analysis can easily be extended to consider different bounds for the preemption overhead and the migration overhead of each task.

domain j share a single period Π_j, which is given a priori and available to all the other domains.

III. Deterministic Multiprocessor Periodic Resource Model

In this section, we introduce the deterministic multiprocessor resource model (DMPR) for representing the interfaces.

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 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.

Several resource models have been developed for multiprocessor platforms. In this paper, we propose a deterministic extension of the multiprocessor periodic resource (MPR) model [21], since it is the most efficient and suitable for our virtualization platform.

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 $\tilde{\Theta} \leq m'\tilde{\Pi}$.

The MPR model 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 [12]). 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.

Deterministic multiprocessor resource model (DMPR). 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.

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.

Observe that the partial processor of μ is represented by a single-processor periodic resource model $\Omega = (\Pi, \Theta)$ [22]. (However, it can also be represented by any other single processor resource model, such as EDP model [11].) Based on this characteristic, we can easily derive the worst-case supply pattern of μ (shown in Figure 2) and its supply bound function, which is given by the following lemma:

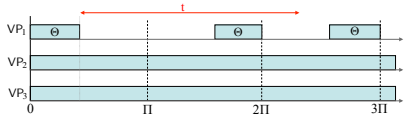


Fig. 2: Worst-case resource supply pattern of $\mu = \langle \Pi, \Theta, m \rangle$.

Lemma 1. *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 $y = \lfloor \frac{t-(\Pi-\Theta)}{\Pi} \rfloor$, 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 [22]: $\mathsf{SBF}_\Omega(t) = 0$, 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 $y = \lfloor \frac{t-(\Pi-\Theta)}{\Pi} \rfloor$. 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.

IV. 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 [12].

A. 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 [12]:

$$\begin{aligned} \mathsf{dbf}_i(t) &= \left\lfloor \frac{t + (p_i - d_i)}{p_i} \right\rfloor e_i + CI_i(t), \text{ where} \\ CI_i(t) &= \min\left\{ e_i, \max\left\{ 0, t - \left\lfloor \frac{t + (p_i - d_i)}{p_i} \right\rfloor p_i \right\} \right\}. \end{aligned} \tag{1}$$

In Eq. (1), $CI_i(t)$ 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 [12], 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:

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

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

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.[3] This leads to the following schedulability test for C:

Theorem 2 ([12])**.** *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. (2) and $\mathsf{SBF}_R(t)$ gives the minimum total resource supply by R in an interval of length t.*

B. 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 1. Since each domain schedules its tasks under gEDF, the following theorem follows directly from Theorem 2.

Theorem 3. *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 $t \geq d_k$,*

$$\mathsf{DEM}(t, m_\mu) \leq \mathsf{SBF}_\mu(t), \quad (3)$$

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 3.

The next theorem derives a bound of the value t that needs to be checked in Theorem 3. The proof is similar to that of Theorem 2 in [12], and we omit it due to space constraints.

Theorem 4. *If Eq. 3 is violated for some value t, then it must also be violated for a value that satisfies the condition*

$$t \leq \frac{C_\Sigma + m_\mu e_k + U + B}{\frac{\Theta}{\Pi} + m - U_T} \quad (4)$$

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 $B = 2\frac{\Theta}{\Pi}(\Pi - \Theta)$.

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

Lemma 5. *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 II). 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 3).[4] 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 II). 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 $m_\mathcal{C} + \sum_{1 \leq i \leq D} m_i$ 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 6. *Let $\mu_i = \langle \Pi_i, \Theta_i, m_i \rangle$ be the minimum-bandwidth DMPR interface of domain $\mathcal{D}_i$, for all $1 \leq i \leq D$. Let $\mathcal{C}$ be a component with the taskset*

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

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 $m_{\mathsf{sys}} = m_\mathcal{C} + \sum_{1 \leq i \leq D} m_i$.

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):

[3] Here, d_k and t refer to D_k and $A_k + D_k$ in [12], respectively.

[4] 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.

Lemma 7. *Let M be the number of physical cores of the platform. The system is schedulable if $M \geq m_{\mathsf{sys}}+1$, 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.

V. 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 VI and VII.

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.

A. 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 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. 3 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).

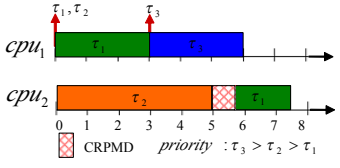


Fig. 3: Cache-related overhead of a task-preemption event.

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 experiences the overhead caused by each of its task-preemption events at most once, and this overhead is bounded from above by Δ^{crpmd}.

Lemma 8. *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; its proof is available in the Appendix.

Lemma 9 (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 $N^1_{\tau_i}$, is bounded by*

$$N^1_{\tau_i} \leq \sum_{\tau_j \in \mathsf{HP}(\tau_i)} \left\lceil \frac{d_i - d_j}{p_j} \right\rceil \tag{5}$$

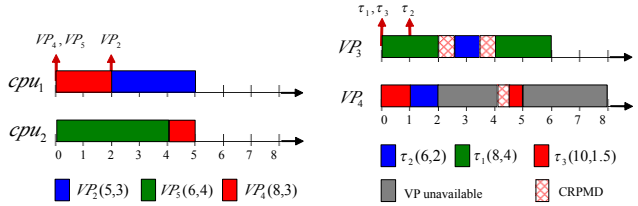
where $\mathsf{HP}(\tau_i)$ is the set of tasks τ_j within the same domain with τ_i with $d_j < d_i$.

B. 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 1. *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. 4(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. 4(b)).*



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

Fig. 4: Cache overhead due to a VCPU-preemption event.

As is shown in Fig. 4(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 10. *Each VCPU-preemption event causes at most two tasks to experience a cache miss.*

The proof of the lemma is available in the Appendix.

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 9. The next lemma is implied directly from this observation (its proof is straightforward and is omitted here due to space constraints):

Lemma 11 (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,*

$$N^2_{\mathsf{VP}_i} \leq \sum_{\mathsf{VP}_j \in \mathsf{HP}(\mathsf{VP}_i)} \left\lceil \frac{\Pi_i - \Pi_j}{\Pi_j} \right\rceil \tag{6}$$

$$N^2_{\mathsf{VP}_i,\tau_k} \leq \sum_{\mathsf{VP}_j \in \mathsf{HP}(\mathsf{VP}_i)} \left\lceil \frac{p_k}{\Pi_j} \right\rceil. \tag{7}$$

C. 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.

Lemma 12 (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,*

$$N^3_{\mathsf{VP}_i} \leq 1 \tag{8}$$

$$N^3_{\mathsf{VP}_i,\tau_k} \leq \left\lceil \frac{p_k - \Theta_i}{\Pi_i} \right\rceil + 1 \tag{9}$$

Proof Sketch: Eq. (8) 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. (9) 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

$$N^{\mathsf{stop}}_{\mathsf{VP}_i} = N^2_{\mathsf{VP}_i} + N^3_{\mathsf{VP}_i} \leq \sum_{\mathsf{VP}_j \in \mathsf{HP}(\mathsf{VP}_i)} \left\lceil \frac{\Pi_i - \Pi_j}{\Pi_j} \right\rceil + 1 \tag{10}$$

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.

VI. Task-centric Compositional Analysis

In this section, we present a task-centric approach to account for the overhead in the interface computation. The idea is to inflate the WCET of every task in the system with the maximum overhead it experiences within each of its periods.

As was discussed in Section V, 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 and, hence, a delay of at most Δ^{crpmd}. From Lemmas 9, 11 and 12, we conclude that the maximum overhead τ_k experiences within each period is

$$\delta^{\mathsf{crpmd}}_{\tau_k} = \Delta^{\mathsf{crpmd}}\big(N^1_{\tau_k} + N^2_{\mathsf{VP}_i,\tau_k} + N^3_{\mathsf{VP}_i,\tau_k}\big)$$

where 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

$$e'_k = e_k + \delta^{\mathsf{crpmd}}_{\tau_k} \tag{11}$$

Thus, we can state the following theorem:

Theorem 13. *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 where* e'_k *is given by Eq. 11.*

Based on Theorem 13, 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 $\Delta^{\mathsf{crpmd}}_{\tau_k}$ and then applying the same method as the overhead-free interface computation in Section IV-B.[5]

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

VII. Model-centric Compositional Analysis

Recall from Section V that each VCPU-stop event (i.e., VCPU-preemption or VCPU-completion event) of VP_i causes at most one cache miss for at most two tasks of the same domain. However, since it is unknown which two tasks may be affected, the task-centric approach in Section VI 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 approach 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.

In the following, 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.

A. Cache-aware effective resource supply of a DMPR model

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 *blackout 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 14. *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 15. *The effective supply bound function of the partial VCPU $\mathsf{VP}_i = (\Pi, \Theta)$ of a resource model $\mu = (\Pi, \Theta, m)$ is*

$$\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}(t) = \begin{cases} y\Theta^* + \max\{0, t - x - y\Pi - z\} & \text{if } \Theta > N^{\mathsf{stop}}_{VP_i}\Delta^{\mathsf{crpmd}} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

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

Proof Sketch: Let $\mathcal{I}$ be any interval of length t. Then 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 blackout intervals that overlap with $\mathcal{I}$ is maximal. Fig. 11 illustrates this scenario, 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 $[t_7, t_8]$:

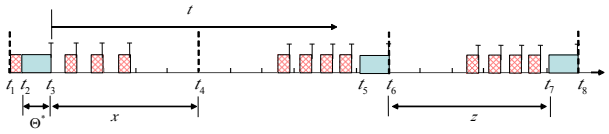


Fig. 5: Worst-case effective resource supply of $\mathsf{VP}_i = (\Pi, \Theta)$.

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 14 that the maximum number of VCPU-stop events of VP_i in a period Π is $N^{\mathsf{stop}}_{\mathsf{VP}_i}$. Further, Δ^{crpmd} is the maximum overhead a task experiences due to a VCPU-stop event. Hence, the effective budget is $\Theta^* = t_3 - t_2 \leq \Theta - N^{\mathsf{stop}}_{\mathsf{VP}_i}\Delta^{\mathsf{crpmd}}$. In addition, we have $t_2 - t_1 \leq \Delta^{\mathsf{crpmd}}$, $x = t_4 - t_3 \geq \Pi - \Delta^{\mathsf{crpmd}} - \Theta^*$ and $z = t_7 - t_6 = \Pi - \Theta^*$. Based on this information, we can derive the minimum effective resource supply during the interval $\mathcal{I}$, as is given by Eq. 12. ∎

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 their effective resource supply. Its proof is available in the Appendix.

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.

Lemma 17. *The effective resource supply bound function of the m full VCPUs of μ is given by:*

$$\mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VPs}}(t) = \begin{cases} m\big(y\Theta' + \max\{0, t - y\Pi - 2x\}\big) & \text{if } \Theta \neq 0 \\ mt & \text{if } \Theta = 0 \end{cases} \tag{13}$$

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

The proof of the lemma can be found in the Appendix.

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 17.

Lemma 18. *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:*

$$\mathsf{SBF}^{\mathsf{stop}}_{\mu} = \mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VP}_i}(t) + \mathsf{SBF}^{\mathsf{stop}}_{\mathsf{VPs}}(t),\ \forall\, t \geq 0. \tag{14}$$

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. (12)*, 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.* (13)*.*

B. Model-centric DMPR interface computation

Based on the effective supply function, we can develop the component schedulability test as follows. The proof of the theorem is available in the Appendix.

Theorem 19. *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 $e''_k = e_k + \Delta^{\mathsf{crpmd}} N^1_{\tau_k}$ for all $1 \leq k \leq n$. Then, $\mathcal{D}_i$ is 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 $\mathsf{SBF}^{\mathsf{stop}}_{\mu}(t)$ in the absence of overhead.*

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.

C. Hybrid cache-aware DMPR interface

The minimum interface of a component can be obtained from the ones computed using the task-centric approach (Section VI) and the model-centric approach described above. The theorem is trivially true, since both interfaces are safe.

Theorem 20 (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 and the model-centric approaches, respectively.*

Discussion. We observe that the schedulability analysis under gEDF in the absence of overhead (Theorem 2) 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).

VIII. Evaluation

To evaluate the benefits of our proposed interface model and cache-aware compositional analysis, we performed simulations using randomly generated workloads. We had three main objectives for our evaluation: (1) Determine how much resource bandwidth the DMPR model can save compared to the MPR model; (2) evaluate the relative performance of the hybrid analysis approach (which combines model-centric and task-centric) and the task-centric analysis approach; and (3) study the impact of task parameters (e.g., the number of tasks in a taskset) on both the overhead-aware DMPR interfaces.

A. Experimental setup

Workload. Our evaluation is based on a set of synthetic real-time workloads. Each workload contained a set of randomly generated periodic task sets. The tasks' periods were chosen uniformly at random between 350ms and 850ms, and the tasks' deadlines were equal to their periods. The tasks' utilizations followed one of four distributions: a uniform distribution within the range $[0.001, 0.1]$ and three bimodal distributions, where the utilizations were distributed uniformly over either $[0.1, 0.4]$ 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).[6] Each generated workload was then distributed uniformly into four different domains.

Overhead value measurement. For our measurements, we used a Dell Optiplex-980 quad-core workstation with the RT-Xen 1.0 platform [23]; each domain was running LITMUS^{RT} 2012.3 [9]. The scheduler was gEDF in the domains and semi-partitioned EDF in the VMM, as described in Section II. We ran cache-intensive programs (as tasks) in the domains, and we measured the cache-related preemption and migration delay each task incurred using the feather-trace tool from $LITMUS^{RT}$. We then used the maximum observed values of all tasks as the value of Δ^{crpmd} for the theoretical analysis.

B. Results

DMPR vs MPR. For this experiment, we generated 625 tasksets with taskset utilizations ranging from 0.1 to 5, with a step of 0.2. For each taskset utilization, there were 25 independently generated tasksets; the task utilizations were

[6]The bimodal distribution probabilities are similar to the ones used in [7].

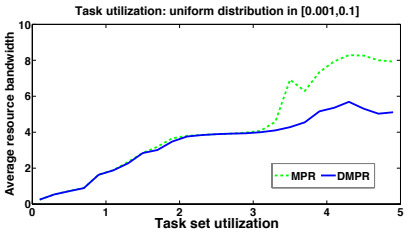


Fig. 6: DMPR vs. MPR.

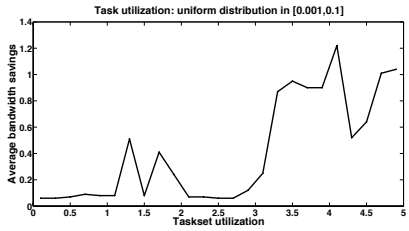


Fig. 7: Savings per taskset utilization.

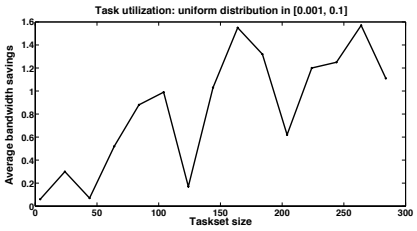
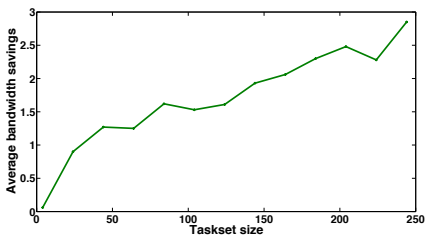
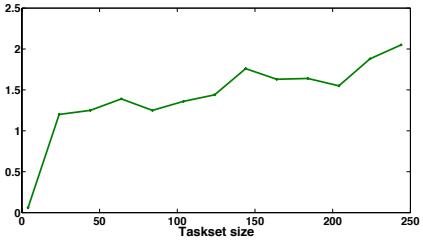


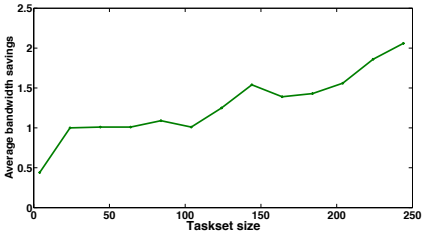
Fig. 8: Savings per taskset size.



(a) Bimodal-light distribution.



(b) Bimodal-medium distribution.



(c) Bimodal-heavy distribution.

Fig. 9: Average resource bandwidth saved with different task utilization.

	MPR	RT-Xen	DMPR	RT-Xen	Cache-aware Hybrid	RT-Xen	Cache-aware Task-centric	RT-Xen
Schedulable	Yes	No	Yes	No	No	No	No	No
Deadline miss ratio	-	78%	-	78%	-	0.07%	-	7%

TABLE I: Performance in theory vs. in practice.

uniformly distributed, as described earlier. For each generated taskset, we distributed the tasks into the four domains, and we then computed the overhead-free interface of the system using the DMPR interface computation method (Section IV) as well as using the existing MPR interface computation method from [12]. Fig. 6 shows the average resource bandwidths with respect to taskset utilization of the DMPR interfaces and the MPR interfaces. We observe that the DMPR interface always requires either the same or less resource bandwidth than the MPR interface across all taskset utilizations. We observe that the DMPR interfaces save a significant amount of resources for more than 40% of the tasksets. For instance, when the taskset utilization is 4.5, it saves up to 3.5 cores.

Hybrid cache-aware vs. task-centric cache-aware. To compare the performance of the two overhead-aware analysis approaches, we used the same tasksets and system configuration as in the previous experiment. We then computed the respective DMPR interface of the system for each taskset using the respective approach.

Fig. 7 shows the average resource bandwidth savings of the hybrid approach compared to the task-centric approach for each task set utilization. The results show that the hybrid approach reduces the resource bandwidth for 64% of the tasksets. We observe that, as the taskset utilization increases, there are more cores being saved. However, the performance in general varies significant across taskset utilizations. One reason for this behavior 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. As a result, when the taskset utilization increases, the interface bandwidth can decrease.

Impact of the number of tasks. We further investigated the impact of the number of tasks (i.e., the taskset size) on the average bandwidth savings of the hybrid approach compared to the task-centric approach. For this experiment, we generated a set of tasksets with sizes between 4 to 300, with a step of 20, and with 25 tasksets per size. We tried both the uniform distribution and the three bi-modal distributions discussed in the experiment setup.

Figure 8 shows the average resource bandwidth savings of the hybrid approach compared to the task-centric approach for the uniform distribution. We observe that the hybrid approach almost always outperforms the task-centric approach.

Fig. 9(a)–9(c) show the average resource bandwidth savings for different taskset sizes with the three bi-modal distributions. We observe that, across all three cases, the hybrid approach consistently outperforms the task-centric approach. In addition, as the number of tasks increases, the savings also increase. This is expected because the task-centric 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.

Performance in theory vs. in practice. We also validated the invalidity of the overhead-free interfaces and the correctness of the cache-aware interface in practice. For this experiment, we first computed the domains' interfaces, and then, we ran the generated tasks on our RT-Xen experimental platform. The periods and budgets of the domains in RT-Xen were assigned to be that 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 I shows the schedulability and deadline miss ratios of these methods.

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, there were 78% of the jobs missing their deadlines. The experimental results also confirm that our cache-aware analysis correctly estimated the resource requirements of the system in practice. In addition, we also observe that the hybrid approach also outperformed (had fewer deadline misses than) the task-centric approach.

IX. Related Work

Several compositional analysis techniques for multicore platforms have been developed (see e.g., [3], [12], [17], [18]) 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., [7]), 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 [19].

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 [6], 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 [17], 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, [17] 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 [14] can be used as an input to our analysis; studies on cache-related preemption and migration delay [4] can be used to obtain the value of Δ^{crpmd} used in our analysis; and finally, cache-aware scheduling, such as [13], can be used to reduce the additional cache-related overhead in the compositional/virtualization setting.

X. 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 two different approaches, model-centric and task-centric, for analyzing the cache-related overhead and for testing the schedulability of components in the presence of cache overhead. We have also introduced a deterministic extension of the MPR model, which improves the interface resource efficiency, as well as accompanying overhead-aware interface computation methods. Our evaluation on synthetic workloads shows that our interface model can help reduce resource bandwidth by a significant factor compared to the MPR model, and that the model-centric approach achieves significant resource savings compared to the task-centric approach (which is based on WCET inflation).

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP*, 2003.

[2] S. Baruah and T. Baker. Schedulability analysis of global EDF. *Real-Time Systems*, 38(3):223–235, 2008.

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

[4] A. Bastoni, B. B. Brandenburg, and J. H. Anderson. Cache-Related Preemption and Migration Delays: Empirical Approximation and Impact on Schedulability. In *OSPERT*, 2010.

[5] S. Basumallick and K. Nilsen. Cache issues in real-time systems. In *LCTES*, 1994.

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

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

[8] 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.

[9] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. H. Anderson. LITMUS RT: A testbed for empirically comparing real-time multiprocessor schedulers. In *RTSS*, 2006.

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

[11] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using edp resource models. In *RTSS*, 2007.

[12] A. Easwaran, I. Shin, and I. Lee. Optimal virtual cluster-based multiprocessor scheduling. *Real-Time Systems*, 43(1):25–59, 2009.

[13] N. Guan, M. Stigge, W. Yi, and G. Yu. Cache-aware scheduling and analysis for multicores. In *EMSOFT*, 2009.

[14] D. Hardy, T. Piquet, and I. Puaut. Using bypass to tighten wcet estimates for multi-core processors with shared instruction caches. In *RTSS*, 2009.

[15] T. Kim, M. Peinado, and G. Mainar-Ruiz. System-level protection against cache-based side channel attacks in the cloud. In *USENIX Security*, 2012.

[16] 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.

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

[18] G. Lipari and E. Bini. A framework for hierarchical scheduling on multiprocessors: From application requirements to run-time allocation. In *RTSS*, 2010.

[19] L. T. X. Phan, M. Xu, J. Lee, I. Lee, and O. Sokolsky. Overhead-aware compositional analysis of real-time systems. In *RTAS*, 2013.

[20] L. Sha, J. P. Lehoczky, and R. Rajkumar. Solutions for Some Practical Problems in Prioritized Preemptive Scheduling. In *RTSS*, 1986.

[21] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *ECRTS*, 2008.

[22] 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.

[23] S. Xi, J. Wilson, C. Lu, and C. Gill. Rt-xen: towards real-time hypervisor scheduling in xen. In *EMSOFT*, 2011.

APPENDIX

Proof of Lemma 9: 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 8, 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 $\sum_{\tau_j \in \mathsf{HP}(\tau_i)} \left\lceil \frac{d_i - d_j}{p_j} \right\rceil$. ∎

Proof of Lemma 10: At most one task is running on a VCPU at any time. Hence, when a VCPU VP_i 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.). Since the overhead caused by each cache miss is at most Δ^{crpmd}, the maximum overhead caused by a VCPU-preemption event is at most $2\Delta^{\mathsf{crpmd}}$. ∎

Proof of Lemma 16: 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 V 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 14. ∎

Proof of Theorem 19: 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. ∎

Proof Sketch for Lemma 17: The effective supply that m full VCPUs provide to the domain in the presence of VCPU-stop events can be computed based on their worst-case supply scenario. From Lemma 16, we can see that the worst-case effective resource supply of m full VCPUs of μ in an interval $\mathcal{I}$ of length t occurs when (1) all the $N_{\mathsf{VP}_i}^{\mathsf{stop}}$ CRPMDs are incurred 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, and (3) the interval $\mathcal{I}$ begins when the first CRPMD occurs in the first period. This worst-case scenario is illustrated by Fig. 12:

In addition, while 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

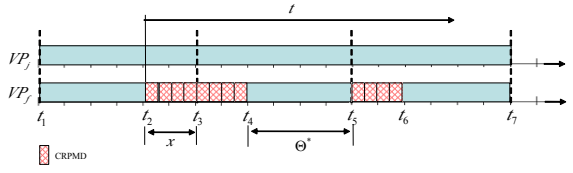


Fig. 10: Worst-case resource supply of m full VCPUs of μ.

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.

Based on the above observation and the worst-case scenario shown in Fig. 12, we derive the minimum effective resource supply the m full VCPUs provide in the interval $\mathcal{I}$, which is given by Eq. 13. In other words, the effective supply bound function of these full VCPUs is given by Eq. 13. ∎

Proof for Lemma 15: Let $\mathcal{I}$ be any interval of length t. We assume that 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 blackout intervals that overlap with $\mathcal{I}$ is maximal. Fig. 11 illustrates this scenario, 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 $[t_7, t_8]$:

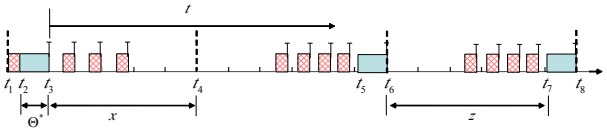


Fig. 11: Worst-case effective resource supply of $\mathsf{VP}_i = (\Pi, \Theta)$.

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 14 that the maximum number of VCPU-stop events of VP_i in a period Π is $N_{\mathsf{VP}_i}^{\mathsf{stop}}$. Further, Δ^{crpmd} is the maximum overhead a task experiences due to a VCPU-stop event. Hence, the effective budget is $\Theta^* \geq \Theta - N_{\mathsf{VP}_i}^{\mathsf{stop}} \Delta^{\mathsf{crpmd}}$. We have $t_3 - t_2 \geq \Theta - (N_{\mathsf{VP}_i}^{\mathsf{stop}} - 1)\Delta^{\mathsf{crpmd}} - (t_2 - t_1) = \Theta^* + \Delta^{\mathsf{crpmd}} - (t_2 - t_1)$, $x = t_4 - t_3 = (t_4 - t_1) - (t_3 - t_2) - (t_2 - t_1) \leq \Pi - \Delta^{\mathsf{crpmd}} - \Theta^*$, and $z = t_7 - t_6 = (t_8 - t_6) - (t_8 - t_7) \leq \Pi - \Theta^*$. Based on this information, we can derive the minimum effective resource supply during the interval $\mathcal{I}$: if $\Theta \leq N_{VP_i}^{\mathsf{stop}} \Delta^{\mathsf{crpmd}}$, $\Theta^* = 0$ and $SBF_{VP_i}^{\mathsf{stop}} = 0$; otherwise, $SBF_{VP_i}^{\mathsf{stop}}(t) = y\Theta^* + \max\{0, t - x - y\Pi - z\}$. Further, $SBF_{VP_i}^{\mathsf{stop}}(t)$ is minimized, when $\Theta^* = \Theta - N_{\mathsf{VP}_i}^{\mathsf{stop}} \Delta^{\mathsf{crpmd}}$ and $x = \Pi - \Delta^{\mathsf{crpmd}} - \Theta^*$. Therefore, Equation 12 is the minimum effective resource supply of the worst case effective resource supply scenario we assumed at the beginning of the proof. Then we will prove the correctness of the assmuption.

Suppose VP_i provides Θ resource in each period. Let the worst case effective resource supply scenario in practice as

$scenario(b)$ and the assumed worst case effective resource supply scenario as $scenario(a)$, see Fig. 11, to simplify the explanation. $\overline{SBF^{\mathsf{stop}}_{VP_i}}(t) \leq SBF^{\mathsf{stop}}_{VP_i}(t)$, where $\overline{SBF^{\mathsf{stop}}_{VP_i}}(t)$ and $\overline{SBF^{\mathsf{stop}}_{VP_i}}(t)$ are the effective resource supply over interval t in $scenario(b)$ and $scenario(a)$, respectively. Let the effective resource supply in each period of $scenario(b)$ as $\overline{\Theta^*}$. Because there is at most $N^{\mathsf{stop}}_{VP_i}$ cache misses during each period, $\overline{\Theta^*} \geq \Theta - N^{\mathsf{stop}}_{VP_i}\Delta^{\mathsf{crpmd}} = \Theta^*$, where Θ^* is the effective budget that VP_i provides in each period in $scenario(a)$.

If $\Theta \leq N^{\mathsf{stop}}_{VP_i}\Delta^{\mathsf{crpmd}}$, $SBF^{\mathsf{stop}}_{VP_i}(t) = 0$, because $\overline{SBF^{\mathsf{stop}}_{VP_i}}(t) \leq SBF^{\mathsf{stop}}_{VP_i}(t)$, VP_i in $scenario(b)$ can provide at most Θ^* effective budget in each period, where $\Theta^* = \Theta - N^{\mathsf{stop}}_{VP_i}\Delta^{\mathsf{crpmd}}$; That is $\overline{\Theta^*} \leq \Theta^*$. Because $\Theta^* \geq \overline{\Theta^*} \leq \Theta^*$, $\overline{\Theta^*} = \Theta^*$.

If $\Theta > N^{\mathsf{stop}}_{VP_i}\Delta^{\mathsf{crpmd}}$, it has the following situations:

(a) When $t \leq x + z$, $SBF^{\mathsf{stop}}_{VP_i}(t) = 0$, because $\overline{SBF^{\mathsf{stop}}_{VP_i}}(t) \leq SBF^{\mathsf{stop}}_{VP_i}(t)$, VP_i in $scenario(b)$ must provide its budget as early as possible in the current period and as late as possible in the next period as shown in interval $[t_3, t_5]$ in $scenario(a)$, so that it can guarantee that $\overline{SBF^{\mathsf{stop}}_{VP_i}}(t) = 0$; Further, because VP_i must provides at most Θ^* time during each period Pi, VP_i always provides effective resource when t is enlarged; therefore, the maximum bankout interval is $x + z$;

(b) When $x + z < t \leq x + z + \Theta^*$, because VP_i provides $\overline{\Theta^*}$ resource in each period and the whole second period of $scenario(b)$ overlaps with t, VP_i must provides Θ^* resource at the end of Θ^* time of the second period; therefore, $scenario(b)$ has the same scenario during interval $[t_5, t_6]$ with $scenario(a)$;

(c) When $x+z+\Theta^* < t \leq x+2z+\Theta^*$, $SBF^{\mathsf{stop}}_{VP_i}(t) = \Theta^*$ and VP_i in $scenario(a)$ provides no effective resource during $[t_6, t_7]$; therefore, VP_i in $scenario(b)$ must provide no effective resource during $[t_6, t_7]$; otherwise, $\overline{SBF^{\mathsf{stop}}_{VP_i}}(t) > \Theta^*$;

(d) When $x+2z+\Theta^* < t \leq x+2z+2\Theta^*$, it is similar with the situation (b). VP_i in $scenario(b)$ must provide Θ^* time during $[t_7, t_8]$ as it does in $scenario(a)$; otherwise, VP_i in $scenario(b)$ cannot provide Θ^* time in each period;

(e) Repeating the situation (c) and (d), we can prove that VP_i in an arbitrary $scenario(b)$ provides no less effective resource than that in $scenario(a)$. Hence, it is proved that $scenario(a)$ is the worst case effective resource supply scenario.

■

Proof of Lemma 17: We claim that the worst-case effective resource supply of m full VCPUs of μ in an interval $\mathcal{I}$ of length t occurs when (1) all the $N^{\mathsf{stop}}_{\mathsf{VP}_i}$ CRPMDs are incurred 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 cost of each CRPMD overhead is Δ^{crpmd} and (4) the interval $\mathcal{I}$ begins when the first CRPMD occurs in the first period. One of this worst-case scenario is illustrated by Fig. 12.

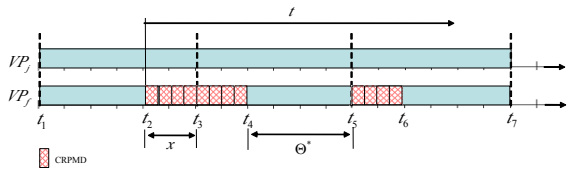


Fig. 12: Worst-case resource supply of m full VCPUs of μ.

We denote the claimed worst-case effective resource supply scenario as $scenario(a)$, see Fig. 12, to simplify the explanation. Suppose the actual worst-case effective resource supply scenario is an arbitrary scenario $scenario(b)$. Let $x = N^{\mathsf{stop}}_{VP_i}\Delta^{\mathsf{crpmd}}$. We will prove that the m full VCPUs in $scenario(b)$ provides no less effective resource than that in $scenario(a)$ 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, all m full VCPUs incur $N^{\mathsf{stop}}_{VP_i}$ CRPMDs in each period. All the $N^{\mathsf{stop}}_{VP_i}$ CRPMDs should be incurred by one full VCPU VP_f in eahc period Π of VP_i so that the unavailable intervals of each period Π is maximized. Hence, $scenario(b)$ must obey the condition (1).
2) The maximum unavailable intervals of m full VCPUs is $x = N^{\mathsf{stop}}_{VP_i}\Delta^{\mathsf{crpmd}}$ in total in each period. The maximum bankout interval happens when the unavailable intervals in two periods are consecutive and the cost of each CRPMD is Δ^{crpmd}. 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 so that the bankout interval is maximized. In addition, the interval $\mathcal{I}$ should begin when the first CRPMD occurs in the first period. Hence, the $scenario(b)$ should obey the condition (3) and (4), and the m full VCPUs in $scenario(b)$ provide no less effective resource than that in $scenario(a)$ 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 in each period and the interval t has k periods, the m full VCPUs in $scenario(b)$ should provide at least $km(\Pi - x)$ effective resource during time interval t. Because $t > x + k\Pi$, the m full VCPUs in $scenario(b)$ has already provided $km(\Pi - x)$ effective resource during interval $x + k\Pi$. Therefore, the m full VCPUs must provide no effective resource at the remaining of $t - (x + k\Pi)$ time interval. (otherwise, the m full VCPUs in $scenario(b)$ provides more effective resource that that in $scenario(a)$.) Hence, the VP_f should incur the overhead as early as possible in all periods (except

for the first period) of VP_i. Therefore, $scenario(b)$ must obey the condition (2) by considering the arguments (2) and (3) together, and the m full VCPUs in $scenario(b)$ provide no less effective resource than that in $scenario(a)$ 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 $scenario(b)$ provides no effective resource during $[x + k\Pi, 2x + k\Pi]$ according to argument (3). In addition, the m full VCPUs in $scenario(b)$ must provide $m(\Pi - x)$ effective resource during $[x+k\Pi, x+(k+1)\Pi]$, i.e., the $(k+1)^{th}$ period of VP_i, so that they can provide $m(\Pi - x)$ effective resource during the $(k+1)^{th}$ period of VP_i. Therefore, the m VCPUs in $scenario(b)$ must always provide effective resource during $[2x + k\Pi, x + (k+1)\Pi]$ as in $scenario(a)$. Hence, the m full VCPUs in $scenario(b)$ provide no less effective resource than that in $scenario(a)$ when $2x + k\Pi < t < x + (k+1)\Pi$.

Because the m full VCPUs in an arbitrary $scenario(b)$ provide no less effective resource that that in $scenario(a)$, $scenario(a)$ is one of the worst-case effective resource supply scenario of the m full VCPUs. Hence, the claimed worst-case effective resource supply scenario is proved correct.

We can derive the effective resource supply bound function $SBF^{\mathsf{stop}}_{VPs}(t)$ of the worst-case effective resoruce supply scenario based on the arguments above. When $t < 2x$, $SBF^{\mathsf{stop}}_{VPs}(t) = 0$; When $x + k\Pi < t < 2x + k\Pi$, $SBF^{\mathsf{stop}}_{VPs}(t) = km(\Pi - x)$; When $2x + k\Pi < t < x + (k+1)\Pi$, $SBF^{\mathsf{stop}}_{VPs}(t) = km(\Pi - x) + m(t - 2x - k\Pi)$. Equation 13 is derived by rearranging the equations of $SBF^{\mathsf{stop}}_{VPs}(t)$. ■

Proof of Lemma 18: Suppose the minimum effective resource supply of a DMPR interface μ is $\overline{SBF^{\mathsf{stop}}_{\mu}}$. $\overline{SBF^{\mathsf{stop}}_{\mu}} = \overline{SBF^{\mathsf{stop}}_{VP_i}(t)} + \overline{SBF^{\mathsf{stop}}_{VPs}(t)}$, where $\overline{SBF^{\mathsf{stop}}_{VP_i}(t)}$ is the effective resource supply of the parital VCPU and $\overline{SBF^{\mathsf{stop}}_{VPs}(t)}$ is the effective resoruce supply of the m full VCPUs. Because $\overline{SBF^{\mathsf{stop}}_{VP_i}(t)} \geq SBF^{\mathsf{stop}}_{VP_i}(t)$ and $\overline{SBF^{\mathsf{stop}}_{VPs}(t)} \geq SBF^{\mathsf{stop}}_{VPs}(t)$, $\overline{SBF^{\mathsf{stop}}_{\mu}(t)} \geq SBF^{\mathsf{stop}}_{\mu}(t)$. Hence, it is proved that $SBF^{\mathsf{stop}}_{\mu}(t)$ is the minimum effective resource supply of a DMPR interface μ. ■