IPP Hurray!

www.hurray.isep.ipp.pt

Technical Report

Optimal Virtual Cluster-based Multiprocessor Scheduling

Arvind Easwaran

Insik Shin

Insup Lee

Optimal Virtual Cluster-based Multiprocessor Scheduling

Arvind Easwaran[1] and Insik Shin[2] and Insup Lee[3]

[1]IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.hurray.isep.ipp.pt

[2]Department of Computer Science

KAIST, Daejeon, Republic of Korea.

Tel.: +82-42-350-3524, Fax.: +82-42-350-3510

E-mail: insik.shin@cs.kaist.ac.kr

[3]Department of CIS

University of Pennsylvania, PA, 19104, USA.

Tel.: +1-215-898-3532, Fax: +1-215-573-7362

E-mail: lee@cis.upenn.edu

Abstract

Scheduling of constrained deadline sporadic task systems on multiprocessor platforms is an area which has received much attention in the recent past. It iswidely believed that finding an optimal scheduler is hard, and therefore most studies have focused on developing algorithms with good processor utilization bounds. These algorithms can be broadly classified into two categories: partitioned scheduling in which tasks are statically assigned to individual processors, and global scheduling in which each task is allowed to execute on any processor in the platform. In this paper we consider a third, more general, approach called cluster-based scheduling. In this approach each task is statically assigned to a processor cluster, tasks in each cluster are globally scheduled among themselves, and clusters in turn are scheduled on the multiprocessor platform. We develop techniques to support such cluster-basedscheduling algorithms, and also consider properties that minimize total processor utilization of individual clusters. In the last part of this paper, we develop new virtual cluster-based scheduling algorithms. For implicit deadline sporadic task systems, we develop an optimal scheduling algorithm that is neither Pfair nor ERfair. We also show that the processor utilization bound of US-EDFm/(2m−1) can be improved by using virtual clustering. Since neither partitioned nor global strategies dominate over the other, cluster-based scheduling is a natural direction for research towards achieving improved processor utilization bounds.

Optimal Virtual Cluster-based Multiprocessor Scheduling

Arvind Easwaran · **Insik Shin** · **Insup Lee**

Abstract Scheduling of constrained deadline sporadic task systems on multiprocessor platforms is an area which has received much attention in the recent past. It is widely believed that finding an optimal scheduler is hard, and therefore most studies have focused on developing algorithms with good processor utilization bounds. These algorithms can be broadly classified into two categories: partitioned scheduling in which tasks are statically assigned to individual processors, and global scheduling in which each task is allowed to execute on any processor in the platform. In this paper we consider a third, more general, approach called cluster-based scheduling. In this approach each task is statically assigned to a processor cluster, tasks in each cluster are globally scheduled among themselves, and clusters in turn are scheduled on the multiprocessor platform. We develop techniques to support such cluster-based scheduling algorithms, and also consider properties that minimize total processor utilization of individual clusters. In the last part of this paper, we develop new virtual cluster-based scheduling algorithms. For implicit deadline sporadic task systems, we develop an optimal scheduling algorithm that is neither Pfair nor ERfair. We also

Arvind Easwaran
Department of CIS, University of Pennsylvania, PA, 19104, USA.
Current affiliation: CISTER/IPP-HURRAY, Polytechnic Institute of Porto, Portugal.
Tel.: +351-22-834-0529, Fax.: +351-22-834-0509, E-mail: aen@isep.ipp.pt

Insik Shin
Department of Computer Science, KAIST, Daejeon, Republic of Korea.
Tel.: +82-42-350-3524, Fax.: +82-42-350-3510, E-mail: insik.shin@cs.kaist.ac.kr

Insup Lee
Department of CIS, University of Pennsylvania, PA, 19104, USA.
Tel.: +1-215-898-3532, Fax.: +1-215-573-7362, E-mail: lee@cis.upenn.edu

show that the processor utilization bound of US-EDF$\{m/(2m-1)\}$ can be improved by using virtual clustering. Since neither partitioned nor global strategies dominate over the other, cluster-based scheduling is a natural direction for research towards achieving improved processor utilization bounds.

Keywords Multiprocessor scheduling · Virtual processor clustering · Hierarchical scheduling · Compositional schedulability analysis

1 Introduction

With rapid development in microprocessor technology, multiprocessor and multi-core designs are becoming an attractive solution to fulfill increasing performance demands. In the real-time systems community, there has been a growing interest in multiprocessor scheduling theories. In general, existing approaches over m identical, unit-capacity processors can fall into two categories: *partitioned* and *global* scheduling. Under partitioned scheduling each task is statically assigned to a single processor and is allowed to execute on that processor only. Under global scheduling tasks are allowed to dynamically migrate across m processors and execute on any of them.

In this paper we consider another approach using a notion of *processor cluster*. A cluster is a set of m' processors, where $1 \le m' \le m$. Under cluster-based scheduling, tasks are statically assigned to a cluster and then globally scheduled within the cluster. This scheduling strategy can be viewed as a generalization of partitioned and global scheduling; it is equivalent to partitioned scheduling at one extreme end where we assign tasks to m clusters each of size one, and global scheduling at the other extreme end where we assign tasks to a single cluster of size m. Cluster-based scheduling can be further classified into two types: *physical* and *virtual* depending on how a cluster is mapped to processors in the platform. A physical cluster holds a static one-to-one mapping between its m' processors and some m' out of m processors in the platform [Calandrino et al., 2007]. A virtual cluster allows a dynamic one-to-many mapping between its m' processors and the m processors in the platform. Scheduling tasks in this virtual cluster can be viewed as scheduling them globally on all the m processors in the platform with amount of concurrency at most m', *i.e.*, at any time instant at most m' of the m processors are used by the cluster. A key difference is that physical clusters share no processors in the platform, while virtual clusters can share some.

Motivating example. We now illustrate the capabilities of cluster-based scheduling using an example. Consider a sporadic task system comprised of 6 tasks as follows: $\tau_1 = \tau_2 = \tau_3 = \tau_4 = (3,2,3)$, $\tau_5 = (6,4,6)$ and $\tau_6 = (6,3,6)$. The notation followed here is (T, C, D), where T denotes the minimum release separation between successive instances of the task, C denotes the maximum required processor capacity for each instance and D denotes the relative deadline. Let this task set be scheduled on a multiprocessor platform comprised of 4 processors. It is easy to see that this task set is not schedulable under any partitioned scheduling algorithm, because no processor can be allocated more than one task. Figure 1 shows the schedule of this task set under global Earliest Deadline First (gEDF) [Liu, 1969], EDZL [Cho et al.,
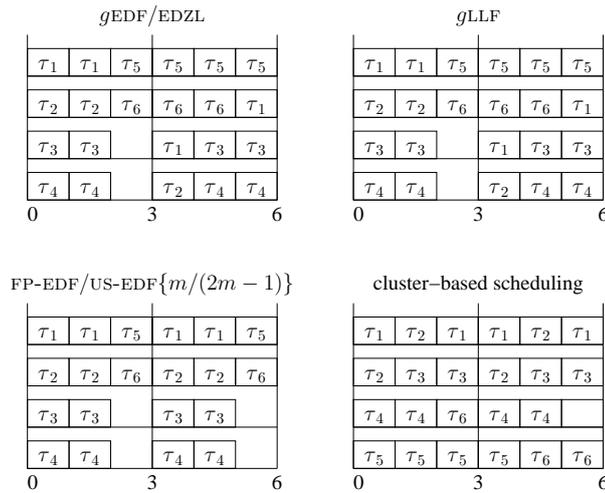
gEDF/EDZL

τ_1	τ_1	τ_5	τ_5	τ_5	τ_5
τ_2	τ_2	τ_6	τ_6	τ_6	τ_1
τ_3	τ_3		τ_1	τ_3	τ_3
τ_4	τ_4		τ_2	τ_4	τ_4

0   3   6

gLLF

τ_1	τ_1	τ_5	τ_5	τ_5	τ_5
τ_2	τ_2	τ_6	τ_6	τ_6	τ_1
τ_3	τ_3		τ_1	τ_3	τ_3
τ_4	τ_4		τ_2	τ_4	τ_4

0   3   6

FP-EDF/US-EDF$\{m/(2m-1)\}$

τ_1	τ_1	τ_5	τ_1	τ_1	τ_5
τ_2	τ_2	τ_6	τ_2	τ_2	τ_6
τ_3	τ_3		τ_3	τ_3	
τ_4	τ_4		τ_4	τ_4	

0   3   6

cluster–based scheduling

τ_1	τ_2	τ_1	τ_1	τ_2	τ_1
τ_2	τ_3	τ_3	τ_2	τ_3	τ_3
τ_4	τ_4	τ_6	τ_4	τ_4	
τ_5	τ_5	τ_5	τ_5	τ_6	τ_6

0   3   6

Fig. 1 Motivating example

2002], Least Laxity First (gLLF) [Leung, 1989], FP-EDF [Baruah, 2004] and US-EDF$\{m/(2m-1)\}$ [Srinivasan and Baruah, 2002] scheduling algorithms. As shown in the figure, the task set is not schedulable under any of these algorithms. Now consider cluster-based scheduling as follows: tasks τ_1, τ_2 and τ_3 are executed under gLLF on a cluster $\mathscr{C}_1$ comprised of 2 processors, and tasks τ_4, τ_5 and τ_6 are executed under gEDF on another cluster $\mathscr{C}_2$ comprised of 2 processors. The resulting schedule is shown in Figure 1, and as can be seen all the task deadlines are met.

In addition to being more general than physical clustering, virtual clustering is also less sensitive to task-processor mappings. This can be explained using the same example as above with an additional task $\tau_7 = (6, 1, 6)$. Just for comparison, suppose τ_7 is assigned to the first cluster $\mathscr{C}_1$ along with tasks τ_1, τ_2 and τ_3. Then physical cluster-based scheduling cannot accommodate those two clusters on 4 processors. On the other hand, virtual clustering has a potential to accommodate them on 4 processors by dynamically re-allocating slack from cluster $\mathscr{C}_2$ to cluster $\mathscr{C}_1$ (time interval (5,6]).

Clustering can also be useful as a mechanism to place a restriction on the amount of concurrency. Suppose m tasks can thrash a L2 cache in a multi-core platform, if they run in parallel at the same time. Then one may consider allowing at most m' of these m tasks to run in parallel, in order to prevent them from thrashing the L2 cache. This can be easily done if the m tasks are assigned to a cluster of m' processors. A similar idea was used in [Anderson et al., 2006].

Hierarchical scheduling. Physical clustering requires intra-cluster scheduling only. This is because clusters are assigned disjoint physical processors, and hence tasks in different clusters cannot interfere with each others executions. However, the notion of virtual clustering inherently requires a two-level hierarchical scheduling framework; inter- and intra-cluster scheduling. In inter-cluster scheduling physical processors are dynamically assigned to virtual clusters. In intra-cluster scheduling
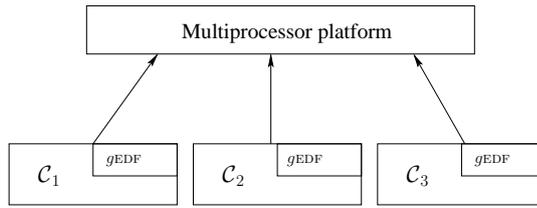
Fig. 2 Example virtual clustering framework

processor allocations given to a virtual cluster are assigned to tasks in that cluster. Consider the example shown in Figure 2. Let a task set be divided into three clusters $\mathscr{C}_1$, $\mathscr{C}_2$ and $\mathscr{C}_3$, each employing gEDF scheduling strategy. If we use physical clustering, then each cluster can be separately analyzed using existing techniques for gEDF. On the other hand if we use virtual clustering, then in addition to intra-cluster schedulability analysis, there is a need to develop techniques for scheduling the clusters on the multiprocessor platform. Therefore, supporting hierarchical multiprocessor scheduling is cardinal to the successful development of virtual clustering.

There have been considerable studies on hierarchical uniprocessor scheduling. Denoting a collection of tasks and a scheduler as a *component*, these studies employed the notion of a component interface to specify resources required for scheduling the component's tasks [Mok et al., 2001; Shin and Lee, 2003; Easwaran et al., 2007]. Analogously, we denote a cluster along with the tasks and scheduler assigned to it as a *component* in hierarchical multiprocessor schedulers. To support inter-cluster scheduling, this paper proposes a component interface that specifies resources required by the tasks in the component's cluster. Inter-cluster scheduler can allocate processor supply to the cluster based on its interface. Intra-cluster scheduler can then use this processor supply to schedule the tasks in the cluster. Many new issues arise to adopt the notion of a component interface from uniprocessor to multiprocessor scheduling. One of them is how to enable a component interface to carry information about concurrent execution of tasks in the component. For example, suppose a single task cannot execute in parallel. Then multiple processors cannot be used concurrently to satisfy the execution requirement of this single task. Such an issue needs to be handled for the successful development of component interfaces. In this paper we present one solution to this issue. Our approach is to capture in a component's interface, all the task-level concurrency constraints in that component. The interface demands enough processor supply from inter-cluster scheduler so that the intra-cluster scheduler can handle task-level concurrency constraints. As a result the inter-cluster scheduler does not have to worry about this issue.

Contributions. The contributions of this paper are five-fold. First, we introduce the notion of general hierarchical multiprocessor schedulers to support virtual cluster-based scheduling. Second, we present an approach to specify the task-level concurrency constraints in a component's interface. In Section 2 we introduce a multiprocessor resource model based interface that not only captures the task-level concurrency constraints, but also specifies the total resource requirements of the component. This

enables the inter-cluster scheduler to schedule clusters using their interfaces alone. Third, since such interfaces represent *partitioned resource supplies*[1] as opposed to *dedicated resource supplies*[2], we also extend existing schedulability conditions for gEDF in this direction[3] (see Section 4). Such extensions to schedulability conditions are essential for supporting development of component interfaces. Fourth, we consider the optimization problem of minimizing the total resource requirements of the component interface. In Section 5, we present an efficient solution to this problem based on the following property of our gEDF schedulability condition: total processor utilization required by a component interface to schedule tasks in the component increases, as number of processors allocated to the component's cluster increases. Thus an optimal solution is obtained when we find the smallest number of processors that guarantee schedulability of the component. Fifth, in Section 6 we develop an overhead free inter-cluster scheduling framework based on McNaughton's algorithm [McNaughton, 1959]. Using this framework we present a new algorithm, called *V*irtual *C*lustering - *I*mplicit *D*eadline *T*asks (VC-IDT), for scheduling implicit deadline sporadic task systems on identical, unit-capacity multiprocessor platforms. We show that VC-IDT is an optimal scheduling algorithm, that does not satisfy the property of P-fairness [Baruah et al., 1996] or ER-fairness [Anderson and Srinivasan, 2000]. The latter feature of our algorithm, as we will see in Section 6.2.1, translates into better bounds on the number of preemptions. As an illustration of the capabilities of general task-processor mappings supported by virtual clustering, we also show that the processor utilization bound of US-EDF$\{m/(2m-1)\}$ can be improved by using this framework. In our previous work [Shin et al., 2008] we presented the first four contributions listed above. In this paper we elaborate on (and extend) those contributions, and in the process develop new virtual cluster-based scheduling algorithms (fifth contribution described above).

2 Task and resource models

In this section we describe our task model and the multiprocessor platform. We also introduce multiprocessor resource models which we use as component interfaces.

2.1 Task and platform models

Task model. We assume a constrained deadline sporadic task model [Baruah et al., 1990]. In this model a sporadic task is specified as $\tau_i = (T_i, C_i, D_i)$, where T_i is the minimum release separation, C_i is the maximum processor capacity requirement and D_i is the relative deadline. These task parameters satisfy the property $C_i \leq D_i \leq T_i$.[4] Successive instances of τ_i are released with a minimum separation of T_i time

---

[1] If a processor can be used by a cluster only in some time intervals and not all, then its supply is said to be partitioned.

[2] If a processor can be used by a cluster at all times, then its supply is said to be dedicated.

[3] We have chosen to focus on one scheduling algorithm in this paper. However the issues are the same for other schedulers, and hence techniques developed here are applicable to other schedulers as well.

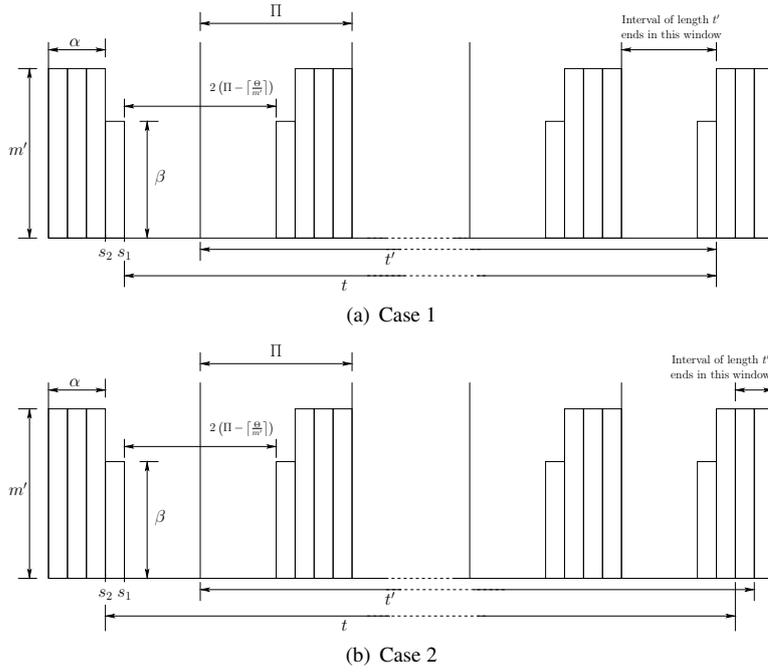[4] If $D_i = T_i$ then the task is called implicit deadline task.

(a) Case 1

(b) Case 2

Fig. 3 Schedule of μ w.r.t sbf$_\mu(t)$

units. We refer to each such instance as a *real-time job*. Each job of τ_i must receive C_i units of processor capacity within D_i time units from its release. These C_i units must be supplied sequentially to the job. This restriction is useful in modeling many real-world systems, because in general, all portions of a software program cannot be parallelized.

Multiprocessor platform and scheduling strategy. In this paper we assume an identical, unit-capacity multiprocessor platform having *m* processors. Each processor in this platform has a resource bandwidth of one, *i.e.*, it can provide *t* units of processor capacity in every time interval of length *t*. We also assume that a job can be preempted on one processor and may resume execution on another processor with negligible preemption and migration overheads, as in the standard literature of global scheduling [Goossens et al., 2003; Baker, 2005a; Bertogna et al., 2005a; Baruah, 2007]. We assume such a global scheduling strategy within each cluster, and in particular, we assume that the strategy is global EDF (denoted as gEDF). At each time instant, if m' denotes the number of physical processors allocated to the cluster, then gEDF schedules unfinished jobs that have the m' earliest relative deadlines.

2.2 Multiprocessor resource model

A *resource model* is a model for specifying the characteristics of processor supply. When these models represent component interfaces, they specify total processor re-

quirements of the component. Periodic [Shin and Lee, 2003], EDP [Easwaran et al., 2007], bounded-delay [Feng and Mok, 2002], etc., are examples of resource models that have been extensively used for analysis of hierarchical uniprocessor schedulers. These resource models can also be used as component interfaces in hierarchical multiprocessor schedulers. One way to achieve this is to consider m' identical resource models as a component interface, where m' is the number of processors allocated to the component's cluster. However, this interface is restrictive because each processor contributes the same amount of resource to the component as any other processor in the cluster. It is desirable to be more flexible in that interfaces should be able to represent the collective processor requirements of clusters, without fixing the contribution of each processor a priori. Apart from increased flexibility, such interfaces can also improve processor utilization in the system.

We now introduce a *multiprocessor resource* model that specifies the characteristics of processor supply provided by an identical, unit-capacity multiprocessor platform. This resource model does not fix the contribution of each processor a priori, and hence is a suitable candidate for cluster interfaces.

Definition 1 (Multiprocessor periodic resource model (MPR)) A *multiprocessor periodic resource* model $\mu = \langle \Pi, \Theta, m' \rangle$ specifies that an identical, unit-capacity multiprocessor platform collectively provides Θ units of resource in every Π time units, where the Θ units are supplied with concurrency at most m'; at any time instant at most m' physical processors are allocated to this resource model. $\frac{\Theta}{\Pi}$ denotes the *resource bandwidth* of model μ.

It is easy to see from the above definition that a *feasible MPR model* must satisfy the condition $\Theta \leq m'\Pi$. The supply bound function of a resource model (sbf) lower bounds the amount of processor supply that the model guarantees in a given time interval. Specifically, $\text{sbf}_R(t)$ is equal to the minimum amount of processor capacity that model R is guaranteed to provide in any time interval of duration t. In uniprocessor systems, sbf is used in schedulability conditions to generate resource model based component interfaces. Extending this approach to multiprocessors, in this paper we derive similar schedulability conditions to generate MPR model based component interfaces. Hence we now present the sbf for a MPR model $\mu = \langle \Pi, \Theta, m' \rangle$. Figure 3 shows the schedule for μ that generates this minimum supply in a time interval of duration t, where $\alpha = \left\lfloor \frac{\Theta}{m'} \right\rfloor$ and $\beta = \Theta - m'\alpha$. As can be seen, length of the largest time interval with no supply is equal to $2\Pi - 2\left\lceil \frac{\Theta}{m'} \right\rceil$ (shown in the figures). sbf_μ[5] is given by the following equation.

$$\text{sbf}_\mu(t) = \begin{cases} 0 & t' < 0 \\ \left\lfloor \frac{t'}{\Pi} \right\rfloor \Theta + \max\left\{0, m'x - (m'\Pi - \Theta)\right\} & t' \geq 0 \text{ and } x \in [1, y] \\ \left\lfloor \frac{t'}{\Pi} \right\rfloor \Theta + \max\left\{0, m'x - (m'\Pi - \Theta)\right\} - (m' - \beta) & t' \geq 0 \text{ and } x \notin [1, y] \end{cases}$$

$$\text{where } t' = t - \left(\Pi - \left\lceil \frac{\Theta}{m'} \right\rceil\right), x = \left(t' - \Pi \left\lfloor \frac{t'}{\Pi} \right\rfloor\right) \text{ and } y = \Pi - \left\lfloor \frac{\Theta}{m'} \right\rfloor \qquad (1)$$

---

[5] A correction has been made to sbf_μ from its original publication in [Shin et al., 2008].

There are two main cases to consider for sbf_μ. If t' is as shown in Figure 3(a), then the interval that generates the minimum supply starts from time instant s_1 shown in the same figure. On the other hand, if t' is as shown in Figure 3(b), then the interval that generates the minimum supply starts from time instant s_2 shown in the same figure. In uniprocessor systems although schedulability conditions with sbf have been derived, a linear approximation of sbf is often used to reduce the time-complexity of the interface generation process. Hence, in anticipation, we present the following linear lower bound for sbf_μ[6]. Functions sbf_μ and lsbf_μ are plotted in Figure 4.

$$\mathrm{lsbf}_\mu(t) = \frac{\Theta}{\Pi}\left(t - \left[2\left(\Pi - \frac{\Theta}{m'}\right) + 2\right]\right) \tag{2}$$

The following lemma proves that lsbf_μ is indeed a lower bound for sbf_μ.

Lemma 1 $\mathrm{lsbf}_\mu(t) \le \mathrm{sbf}_\mu(t)$ *for all* $t \ge 0$.

Proof Consider Figure 4. Observe that $\mathrm{lsbf}_\mu(t) = 0$ for all $t \le t_4$. Therefore it is sufficient to show that $\mathrm{lsbf}_\mu(t) \le \mathrm{sbf}_\mu(t)$ for all $t > t_4$. Suppose $\mathrm{sbf}_\mu(t_4) = 2\beta + \varepsilon$ for some $\varepsilon \ge 0$.

We now show that $\mathrm{lsbf}_\mu(t) \le \mathrm{sbf}_\mu(t)$ for all t such that $t_4 < t \le t_8$, where $t_8 = t_4 + \Pi$. The following statements are true by definition: 1) $\mathrm{sbf}_\mu(t_8) = \Theta + 2\beta + \varepsilon$, and 2) $\mathrm{lsbf}_\mu(t_8) = \Theta$. Further, because the slope of sbf_μ in the interval $(t_4, t_5]$ is at least as much as the slope of lsbf_μ ($\frac{\Theta}{\Pi} \le m'$), $\mathrm{lsbf}_\mu(t) \le \mathrm{sbf}_\mu(t)$ for all t such that $t_4 < t \le t_5$. From the figure, we can see that $\mathrm{sbf}_\mu(t_6) = \Theta = \mathrm{lsbf}_\mu(t_8)$ and $t_6 \le t_8$. Therefore $\mathrm{lsbf}_\mu(t) \le \mathrm{sbf}_\mu(t)$ for all t such that $t_6 \le t \le t_8$. The last statement follows from the fact that sbf_μ is a non-decreasing function. This combined with the facts that $t_6 = t_5 + 1$ and lsbf_μ is a linear function, implies $\mathrm{lsbf}_\mu(t) \le \mathrm{sbf}_\mu(t)$ for all t such that $t_4 < t \le t_8$.

Observe that in every successive time interval of length Π starting from t_4, the following holds: 1) both sbf_μ and lsbf_μ increase by exactly Θ, and 2) they both have slope characteristics identical to those in the interval $(t_4, t_8]$. Therefore the arguments from the previous paragraph hold for each such time interval of length Π. The result of the lemma then follows. $\square$

Uniprocessor resource models, such as periodic or EDP, allow a view that a component executes over an exclusive share of a physical uniprocessor platform. Extending this notion, MPR models allow a view that a component, and hence the corresponding cluster, executes over an exclusive share of a physical multiprocessor platform. Although this view guarantees a minimum total processor share given by sbf, it does not enforce any distribution of this share over the processors in the platform, apart from the concurrency bound m'. In this regard MPR models are general and hence our candidate for component interfaces.

---

[6] lsbf_μ has also been modified from its original publication in [Shin et al., 2008], in order to be consistent with the new sbf_μ.

Fig. 4 sbf_μ and lsbf_μ
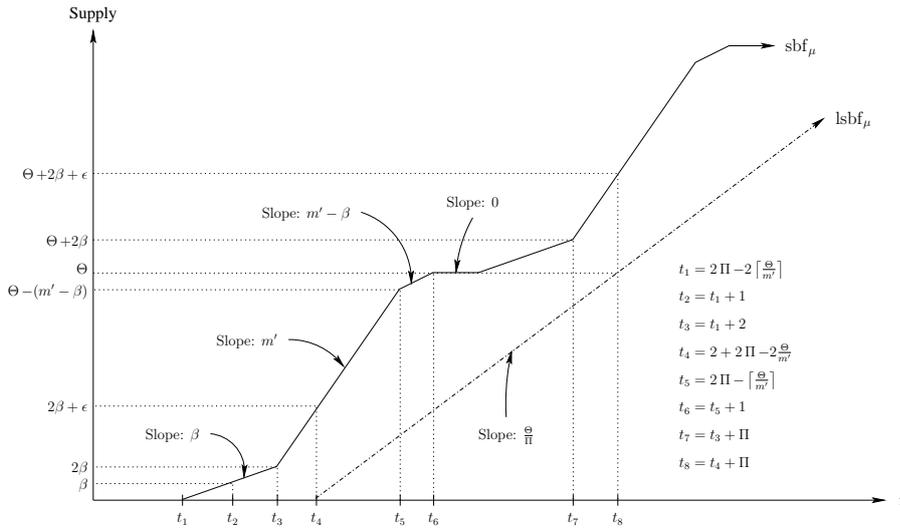
3 Related work

Multiprocessor scheduling. In general, studies on real-time multiprocessor scheduling theory can fall into two categories: *partitioned* and *global* scheduling. Under partitioned scheduling each task is statically assigned to a single processor and uniprocessor scheduling algorithms are used to schedule tasks. Under global scheduling tasks are allowed to migrate across processors and algorithms that simultaneously schedule on all the processors are used. Many partitioning algorithms and their analysis [Oh and Baker, 1998; López et al., 2001; Baruah and Fisher, 2006; Fisher et al., 2006], and global scheduling algorithms and their analysis [Baruah et al., 1996; Andersson et al., 2001; Cho et al., 2002; Srinivasan and Baruah, 2002; Zhu et al., 2003; Goossens et al., 2003; Baker, 2003; Baruah, 2004; Baker, 2005a, 2006; Bertogna et al., 2005a; Cho et al., 2006; Baruah, 2007; Cirinei and Baker, 2007; Bertogna and Cirinei, 2007; Baruah and Fisher, 2007; Baruah and Baker, 2008a,b; Funaoka et al., 2008], have been proposed in the past.

For implicit deadline task systems, both Earliest Deadline First (EDF) [López et al., 2001] and Rate Monotonic (RM) [Oh and Baker, 1998] based partitioned scheduling have been proposed along with processor utilization bounds. These studies have since been extended for constrained deadline task systems, and EDF [Baruah and Fisher, 2006] and fixed-priority [Fisher et al., 2006] based scheduling have been developed for them. Under global scheduling of implicit deadline task systems, several optimal algorithms such as Pfair [Baruah et al., 1996], BoundaryFair [Zhu et al., 2003], LNREF [Cho et al., 2006], and NVNLF [Funaoka et al., 2008], have been proposed. To reduce the relatively high preemptions in these algorithms and to support constrained deadline task systems, processor utilization bounds and worst-case response time analysis for EDF [Goossens et al., 2003; Baker, 2003, 2005a; Bertogna

et al., 2005a; Baruah, 2007; Bertogna and Cirinei, 2007; Baruah and Baker, 2008a,b] and Deadline Monotonic (DM) [Baker, 2003, 2006; Bertogna and Cirinei, 2007; Baruah and Fisher, 2007] based global scheduling strategies have been developed. Towards better processor utilization, new global algorithms such as dynamic-priority EDZL [Cho et al., 2002; Cirinei and Baker, 2007] and US-EDF$\{m/(2m-1)\}$ [Srinivasan and Baruah, 2002], and fixed-priority RM-US$\{m/(3m-2)\}$ [Andersson et al., 2001] and FP-EDF [Baruah, 2004], have also been proposed. Partitioned scheduling suffers from an inherent performance limitation in that a task may fail to be assigned to any processor, although the total available processor capacity across the platform is larger than the task's requirements. Global scheduling has been developed to overcome this limitation. However global algorithms are either not known to utilize processors optimally (like in the case of constrained deadline task systems), or if they are known to be optimal, then they have high number of preemptions (like in the case of implicit deadline task systems). Moreover, for constrained deadline tasks, simulations conducted by Baker [Baker, 2005b] have shown that partitioned scheduling performs much better than global scheduling on an average. These simulations reflect the large pessimism in current schedulability tests for global algorithms. To eliminate the performance limitation of partitioned scheduling and to achieve high processor utilization without incurring high preemption costs, we consider the more general task-processor mappings that virtual cluster-based scheduling proposes.

Algorithms that support slightly more general task-processor mappings than either partitioned or global scheduling have been proposed in the past. Andersson *et al.* [Andersson and Tovar, 2006; Andersson and Bletsas, 2008; Andersson et al., 2008] and Kato and Yamasaki [Kato and Yamasaki, 2007] have developed algorithms that allow a task to be scheduled on at most two processors in the platform. Virtual cluster-based scheduling framework that we propose generalizes all these task-processor mappings and therefore can lead to higher processor utilization. Baruah and Carpenter [Baruah and Carpenter, 2003] introduced an approach that restricts processor migration of jobs, in order to alleviate the performance limitation of partitioned scheduling and the processor migration overheads of global scheduling. It has been shown that the worst-case processor utilization of this approach is no better than partitioned scheduling (roughly 50%). Calandrino *et al.* [Calandrino et al., 2007] presented a physical clustering framework in which tasks are first assigned to physical processor clusters and then scheduled globally within those clusters. They experimentally evaluated this framework to show that cache-access related overheads can be reduced in comparison to both partitioned and global scheduling strategies. Virtual clustering is again a generalization of this framework, and moreover, unlike their work, we develop efficient schedulability analysis techniques with a focus on achieving high processor utilization. Recently, virtual clustering has also been considered in the context of tardiness guarantees for soft real-time systems [Leontyev and Anderson, 2008].

Moir and Ramamurthy [Moir and Ramamurthy, 1999] and Anderson *et al.* [Holman and Anderson, 2001; Anderson et al., 2006] presented an approach that upper bounds the amount of concurrent execution within a group of tasks. They developed their approach using a two-level Pfair-based scheduling hierarchy. These studies are most related to our work on virtual clustering, but they differ from our technique

mainly in the following aspect. We introduce a multiprocessor resource model that makes it possible to clearly separate intra- and inter-cluster scheduling. This allows development of schedulability analysis techniques for virtual clustering that are easily extensible to many different schedulers. However their approaches do not employ such a notion. Therefore their analysis techniques are bound to Pfair scheduling, and do not generalize to other algorithms and task models such as the one considered in this paper. This flexibility provides a powerful tool for the development of various task-processor mappings and intra- and inter-cluster scheduling algorithms.

Hierarchical scheduling. For uniprocessor platforms there has been a growing attention to hierarchical scheduling frameworks. Since a two-level framework was introduced [Deng and Liu, 1997], its schedulability has been analyzed under fixed-priority [Kuo and Li, 1999] and EDF-based [Lipari et al., 2000] scheduling. For multi-level frameworks many resource model based component interfaces such as bounded-delay [Mok et al., 2001; Shin and Lee, 2004], periodic [Lipari and Bini, 2003; Shin and Lee, 2003, 2008] and EDP [Easwaran et al., 2007], have been introduced, and schedulability conditions have been derived under fixed-priority and EDF scheduling [Feng and Mok, 2002; Lipari and Bini, 2003; Shin and Lee, 2003; Almeida and Pedreiras, 2004; Davis and Burns, 2005; Easwaran et al., 2007]. As discussed in the introduction, these studies do not provide any technique to capture task-level concurrency constraints in interfaces, and therefore are not well suited for virtual clustering.

4 Component schedulability condition

In this section we develop a schedulability condition for components in hierarchical multiprocessor schedulers, such that this condition accommodates the notion of a partitioned resource supply. Specifically, we extend existing gEDF schedulability conditions for dedicated resource, with the supply bound function of a MPR model. Any MPR model that satisfies this condition can be used as an interface for the component.

We consider a component comprising of cluster $\mathscr{C}$ and sporadic tasks $\mathscr{T} = \{\tau_1 = (T_1, C_1, D_1), \ldots, \tau_n = (T_n, C_n, D_n)\}$ scheduled under gEDF. To keep the presentation simple, we use notation $\mathscr{C}$ to refer to the component as well. We now develop a schedulability condition for $\mathscr{C}$ assuming it is scheduled using MPR model $\mu = \langle \Pi, \Theta, m' \rangle$, where m' denotes number of processors in the cluster. This condition uses the total processor demand of task set $\mathscr{T}$ for a given time interval. Existing studies [Bertogna et al., 2005a] have developed an upper bound for this demand which we can use. Only upper bounds are known for this demand, because unlike the synchronous arrival sequence in uniprocessors, no notion of worst-case arrival sequence is known for multiprocessors [Baruah, 2007]. Hence we first summarize this existing demand upper bound and then present our schedulability condition.

4.1 Component demand

Workload. The workload of a task τ_i in an interval $[a, b]$ gives the cumulative length of all intervals in which τ_i is executing, when task set $\mathscr{T}$ is scheduled under $\mathscr{C}$'s
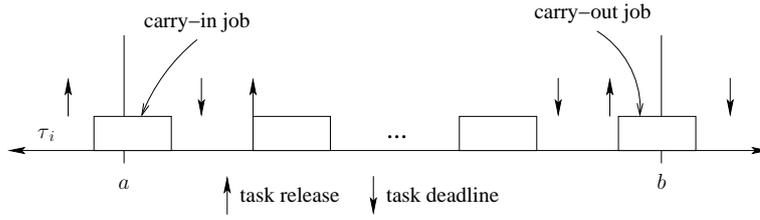
Fig. 5 Workload of task τ_i in interval $[a,b]$

scheduler. This workload consists of three parts (illustrated in Figure 5): (1) the *carry-in* demand generated by a job of τ_i that is released prior to a, but did not finish its execution requirements until a, (2) the demand of a set of jobs of τ_i that are both released and have their deadlines within the interval, and (3) the *carry-out* demand generated by a job of τ_i that is released in the interval $[a,b)$, but does not finish its execution requirements until b.

Workload upper bound for τ_i under gEDF. If workload in an interval $[a,b]$ can be efficiently computed for all $a,b \geq 0$ and for all tasks τ_i, then we can obtain the exact demand of task set $\mathscr{T}$ in all intervals. However, since no such efficient computation technique is known (apart from task set simulation), we use an upper bound for this workload obtained by Bertogna *et al.* [Bertogna et al., 2005a]. This bound is obtained under two assumptions: (1) some job of some task τ_k has a deadline at time instant b, and (2) this job of τ_k misses its deadline. In the schedulability conditions we develop, these assumptions hold for all time instants b that are considered. Hence this is a useful bound and we present it here. Figure 6 illustrates the dispatch pattern corresponding to this bound. A job of task τ_i has a deadline that coincides with time instant b. Jobs of τ_i that are released prior to time b are assumed to be released as late as possible. Also, the job of τ_i that is released before a but has a deadline in the interval $[a,b]$, is assumed to execute as late as possible. This imposes maximum possible interference on the job of τ_k with deadline at b. Let $\mathscr{W}_i(t)$ denote this workload bound for τ_i in a time interval of length t $(=b-a)$. Also let $CI_i(t)$ denote the carry-in demand generated by the execution pattern shown in Figure 6. Then

$$\mathscr{W}_i(t) = \left\lfloor \frac{t+(\mathrm{T}_i-\mathrm{D}_i)}{\mathrm{T}_i} \right\rfloor \mathrm{C}_i + CI_i(t),$$

$$\text{where } CI_i(t) = \min\left\{\mathrm{C}_i, \max\left\{0, t - \left\lfloor \frac{t+(\mathrm{T}_i-\mathrm{D}_i)}{\mathrm{T}_i} \right\rfloor \mathrm{T}_i\right\}\right\} \tag{3}$$

It has been shown that the actual workload of τ_i can never exceed $\mathscr{W}_i(b-a)$ in the interval $[a,b]$, provided tasks are scheduled under gEDF and a deadline miss occurs for that job of τ_k whose deadline is at b [Bertogna et al., 2005a]. This follows from the observation that no job of τ_i with deadline greater than b can execute in the interval $[a,b]$. In the following section we develop a schedulability condition for $\mathscr{C}$ using this workload bound.
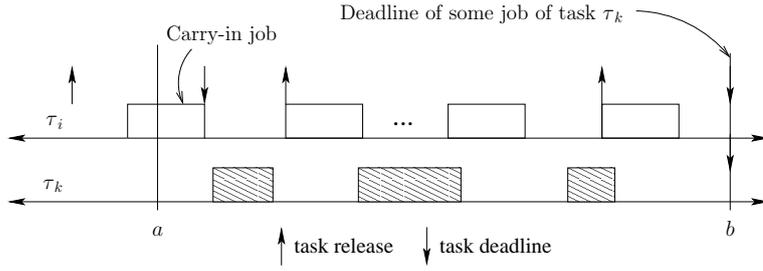
Fig. 6 Dispatch and execution pattern of task τ_i for $\mathscr{W}_i(b-a)$
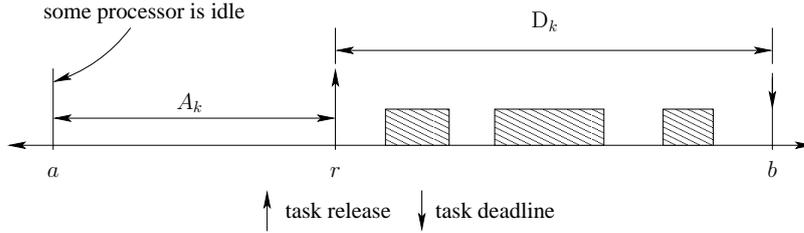

Fig. 7 Example time instant a under dedicated resource

4.2 Schedulability condition

We now present a schedulability condition for component $\mathscr{C}$ when it is scheduled using MPR model $\mu = \langle \Pi, \Theta, m' \rangle$. For this purpose we extend (with the notion of sbf_μ) an existing condition that checks the schedulability of $\mathscr{C}$ on a dedicated resource comprised of m' unit-capacity processors.

When task τ_k is scheduled on m' unit-capacity processors under gEDF, existing work identifies different time intervals that must be checked to guarantee schedulability of τ_k [Baruah, 2007]. In particular, it assumes b denotes the missed deadline of some job of task τ_k (henceforth denoted as job τ_k^b), and then specifies different values of a, corresponding to the interval $[a,b]$, that need to be considered. Figure 7 gives one such time instant a. It corresponds to a point in time such that: (1) at least one of the m' processors is idle at that instant, (2) it is prior to the release time of job τ_k^b (r in the figure), and (3) no processor is idle in the interval $(a,r]$. Observe that at each such time instant a there can be at most $m'-1$ tasks that contribute towards carry-in demand. This is because at most $m'-1$ processors are executing jobs at a. This observation is used to develop an efficient schedulability condition in the dedicated resource case. Informally, the study derives a condition on the total higher priority workload in the interval $[a,b]$ that guarantees a deadline miss for τ_k^b. In the following discussion we extend this notion of time instant a for the case when τ_k is scheduled under the partitioned resource supply μ.

When task τ_k is scheduled using μ, we denote a time instant as t_{idle} if at least one of the m' processors is idle at that instant, even though it is available for use as per supply μ. Figure 8 illustrates one such time instant, where r denotes the release time of job τ_k^b, A_k denotes the length of the interval $(a,r]$ and A_k+D_k denotes the
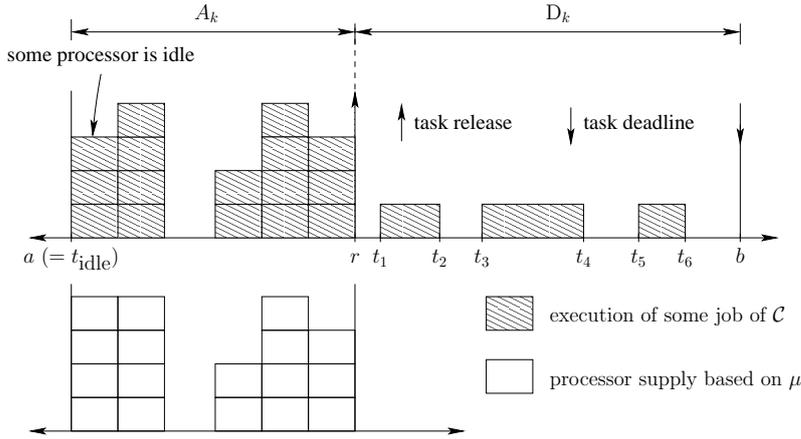
Fig. 8 Example time instant t_{idle}

length of the interval $(a,b]$. To check schedulability of task τ_k we consider all time instants a such that: (1) a is t_{idle}, (2) $a \le r$, and (3) no time instant in the interval $(a,r]$ is t_{idle}. The time instant illustrated in Figure 8 satisfies these properties.

To derive the schedulability condition for component $\mathscr{C}$, we consider all intervals $[a,b]$ as explained above and derive conditions under which a deadline miss occurs for job τ_k^b. If τ_k^b misses its deadline, then the total workload of jobs having priority at least τ_k^b must be greater than the total processor supply available to $\mathscr{C}$ in $[a,b]$. Let I_i $(1 \le i \le n)$ denote the total workload in interval $[a,b]$ of jobs of τ_i that have priority at least τ_k^b. Since $\text{sbf}_\mu(b-a)$ denotes a lower bound on the processor supply available to $\mathscr{C}$ in $[a,b]$, whenever τ_k^b misses its deadline it must be true that

$$\sum_{i=1}^{n} I_i > \text{sbf}_\mu(b-a) = \text{sbf}_\mu(A_k + D_k) \tag{4}$$

This inequality can be derived from the following observations: (1) the actual processor supply available to component $\mathscr{C}$ in $[a,b]$ is at least $\text{sbf}_\mu(A_k + D_k)$ and (2) there are no t_{idle} time instants in the interval $(a,b]$, *i.e.*, all available processor supply is used by $\mathscr{C}$ to schedule tasks from $\mathscr{T}$. For $\mathscr{C}$ to be schedulable using μ, it then suffices to show that for all tasks τ_k and for all values of A_k Equation (4) is invalid.

We now derive an upper bound for each workload I_i. We separately consider the workload of τ_i in the following two interval classes: (1) time intervals in $[a,b]$ in which τ_k^b executes (intervals $[t_1,t_2], [t_3,t_4]$ and $[t_5,t_6]$ in Figure 8) and (2) the other time intervals in $[a,b]$. Let $I_{i,1}$ denote the workload of τ_i in intervals of type (1) and $I_{i,2}$ denote the workload of τ_i in intervals of type (2). We bound I_i using upper bounds for $I_{i,1}$ and $I_{i,2}$. In the dedicated resource case, only intervals of type (2) were considered when deriving the schedulability condition [Baruah, 2007]. We however consider the contiguous interval $[a,b]$, because sbf of MPR models are only defined over such contiguous time intervals.

Since the cumulative length of intervals of type (1) is at most C_k and there are at most m' processors on which $\mathscr{C}$ executes, the total workload of all the tasks in

intervals of type (1) is clearly upper bounded by $m'C_k$. Therefore, $\sum_{i=1}^{n} I_{i,1} \leq m'C_k$. To bound $I_{i,2}$ we use the workload upper bound $\mathscr{W}_i$ presented in Section 4.1. Recall that $\mathscr{W}_i(b-a)$ $(=\mathscr{W}_i(A_k+D_k))$ upper bounds the workload of all jobs of τ_i that execute in the interval $[a,b]$ and have priority higher than τ_k^b. Therefore $\mathscr{W}_i(A_k+D_k)$ also upper bounds $I_{i,2}$. Further, there is no need for $I_{i,2}$ to be larger than $A_k+D_k-C_k$, because we have already considered a total length of C_k for intervals of type (1). Also this bound can be further tightened for $i=k$, because in $I_{k,2}$ we do not consider the executions of τ_k^b. These executions are already considered for intervals of type (1). Thus we can subtract C_k from $\mathscr{W}_k(A_k+D_k)$ and $I_{k,2}$ cannot be greater than A_k.

$$I_{i,2} \leq \bar{I}_{i,2} = \min\{\mathscr{W}_i(A_k+D_k), A_k+D_k-C_k\} \text{ for all } i \neq k$$
$$I_{k,2} \leq \bar{I}_{k,2} = \min\{\mathscr{W}_k(A_k+D_k)-C_k, A_k\}$$

Now by definition of time instant a at most $m'-1$ tasks can be active, and hence have carry-in demand, at a. This follows from the fact that at least one processor is not being used by $\mathscr{C}$ at a even though that processor is available as per supply μ. Hence we only need to consider $m'-1$ largest values of CI_i when computing an upper bound for $\sum_{i=1}^{n} I_{i,2}$ using the above equations, where CI_i denotes the carry-in demand in $\mathscr{W}_i$. Let us now define the following two terms.

$$\hat{I}_{i,2} = \min\{\mathscr{W}_i(A_k+D_k)-CI_i(A_k+D_k), A_k+D_k-C_k\} \text{ for all } i \neq k$$
$$\hat{I}_{k,2} = \min\{\mathscr{W}_k(A_k+D_k)-C_k-CI_k(A_k+D_k), A_k\}$$

Let $L_{(m'-1)}$ denote a set of task indices such that if $i \in L_{(m'-1)}$, then $(\bar{I}_{i,2} - \hat{I}_{i,2})$ is one of the $m'-1$ largest values among all tasks. Then an upper bound on the worst-case resource demand in the interval $(a,b]$ can be defined as,

$$\text{DEM}(A_k+D_k, m') = m'C_k + \sum_{i=1}^{n} \hat{I}_{i,2} + \sum_{i:i \in L_{(m'-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}).$$

The following theorem gives our schedulability condition and its proof follows from the above discussions.

Theorem 1 *A component comprising of cluster $\mathscr{C}$ with m' processors and sporadic tasks $\mathscr{T} = \{\tau_1 = (T_1, C_1, D_1), \ldots, \tau_n = (T_n, C_n, D_n)\}$ is schedulable under gEDF using MPR model $\mu = \langle \Pi, \Theta, m' \rangle$, if for all tasks $\tau_k \in \mathscr{T}$ and all $A_k \geq 0$,*

$$\text{DEM}(A_k+D_k, m') \leq \text{sbf}_\mu(A_k+D_k). \tag{5}$$

In Theorem 1 if we set $\Theta = m'\Pi$, then we get the schedulability condition under dedicated resource that was proposed earlier [Baruah, 2007]. This shows that our condition is no more pessimistic than the one under dedicated resource. Although this theorem gives a schedulability test for component $\mathscr{C}$, it would be highly inefficient if we were required to check for all values of A_k. The following theorem shows that this is not the case.

Theorem 2 *If Equation* (5) *is violated for some A_k, then it must also be violated for a value satisfying the condition*

$$A_k < \frac{C_\Sigma + m' C_k - D_k \left(\frac{\Theta}{\Pi} - U_{\mathcal{T}}\right) + U + B}{\frac{\Theta}{\Pi} - U_{\mathcal{T}}},$$

where C_Σ denotes the sum of $m' - 1$ largest C_i's, $U_{\mathcal{T}} = \sum_{i=1}^n \frac{C_i}{T_i}$, $U = \sum_{i=1}^n (T_i - D_i) \frac{C_i}{T_i}$ and $B = \frac{\Theta}{\Pi} \left[2 + 2\left(\Pi - \frac{\Theta}{m'}\right)\right]$.

Proof It is easy to see that $\hat{I}_{i,2} \le \text{dbf}_{\tau_i}(A_k + D_k)$ and $\bar{I}_{i,2} \le \text{dbf}_{\tau_i}(A_k + D_k) + C_i$, where $\text{dbf}_{\tau_i}(t) = \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i$. Then the left hand side of Equation (5) is less than or equal to $C_\Sigma + m' C_k + \sum_{i=1}^n \text{dbf}_{\tau_i}(A_k + D_k)$. For this equation to be violated it must be true that

$$C_\Sigma + m' C_k + \sum_{i=1}^n \text{dbf}_{\tau_i}(A_k + D_k) > \text{sbf}_\mu(A_k + D_k)$$

$\Rightarrow$ (Using dbf_{τ_i} bound from [Baruah et al., 1990])

$$C_\Sigma + m' C_k + (A_k + D_k) U_{\mathcal{T}} + U > \text{sbf}_\mu(A_k + D_k)$$

$\Rightarrow$ (From Equation (2))

$$C_\Sigma + m' C_k + (A_k + D_k) U_{\mathcal{T}} + U > \frac{\Theta}{\Pi}(A_k + D_k) - B$$

$\Rightarrow$ (Rearranging)

$$A_k < \frac{C_\Sigma + m C_k - D_k \left(\frac{\Theta}{\Pi} - U_{\mathcal{T}}\right) + U + B}{\frac{\Theta}{\Pi} - U_{\mathcal{T}}}$$

$\square$

It can also be observed that Equation (5) only needs to be evaluated at those values of A_k for which at least one of $\hat{I}_{i,2}, \bar{I}_{i,2}$ or sbf_μ change. Therefore Theorem 1 gives a pseudo-polynomial time schedulability condition whenever utilization $U_{\mathcal{T}}$ is strictly less than the resource bandwidth $\frac{\Theta}{\Pi}$. In our techniques described later we compute minimum possible Θ and minimum required concurrency m' for a given value of Π. Since Θ appears inside floor and ceiling functions in sbf_μ, these computations may be intractable. We therefore replace sbf_μ in Theorem 1 with lsbf_μ from Equation (2) before using it to generate MPR interfaces.

Discussion. We have only focused on one intra-cluster scheduling algorithm in this paper. However our analysis technique can be easily extended to other intra-cluster scheduling algorithms. Specifically, in the schedulability condition given in Equation (5), $\text{DEM}(A_k + D_k, m')$ depends on gEDF and $\text{sbf}_\mu(A_k + D_k)$ depends on MPR model μ. Suppose there exists a function $\text{DEM}_{\text{DM}}(A_k + D_k, m')$ that can compute the workload upper bound for a task set scheduled under global DM. Then we can plug in $\text{DEM}_{\text{DM}}(A_k + D_k, m')$ into Equation (5) to derive a schedulability condition for global DM intra-cluster scheduling. In fact, such a DEM_{DM} can be obtained by extending current results over dedicated resource [Bertogna et al., 2005b].

Bertogna and Cirinei have derived an upper bound for the worst-case response time of tasks scheduled under gEDF or global DM [Bertogna and Cirinei, 2007]. They have also used this bound to improve the carry-in demand CI_i that we use in our schedulability condition. However this improvement to the carry-in demand cannot be applied in our case. Since we schedule tasks using MPR model, any response time computation depends on the processor supply in addition to task demand. Then to use the response time bounds presented in [Bertogna and Cirinei, 2007], we must extend it with sbf of MPR model. However, since we are computing the MPR model (capacity Θ and concurrency m'), its sbf is unknown and therefore the response time is not computable. One way to resolve this issue is to compute Θ and m' using binary search. However, since Θ belongs to the domain of non-negative real numbers, binary search for the minimum Θ can take a prohibitively long time.

5 Component interface generation

In this section we develop a technique to generate interface $\mu = \langle \Pi, \Theta, m' \rangle$ for a cluster $\mathscr{C}$ comprising of sporadic tasks $\mathscr{T} = \{\tau_1 = (T_1, C_1, D_1), \ldots, \tau_n = (T_n, C_n, D_n)\}$ scheduled under gEDF. For this purpose we use the schedulability condition given by Theorem 1. We assume that period Π of interface μ is specified a priori by the system designer. For instance, one can specify this period taking into account preemption overheads in the system. We then compute values for capacity Θ and number of processors m' so that resource bandwidth of the interface is minimized. Finally, we also develop a technique that transforms MPR interfaces to periodic tasks[7], in order to schedule clusters on the multiprocessor platform (inter-cluster scheduling).

5.1 Minimum bandwidth interface

It is desirable to minimize the resource bandwidth of μ when generating an interface for $\mathscr{C}$, because $\mathscr{C}$ then consumes the minimum possible processor supply. We now give a lemma which states that the resource bandwidth required to guarantee schedulability of task set $\mathscr{T}$ monotonically increases as number of processors in the cluster increases.

Lemma 2 *Consider interfaces $\mu_1 = \langle \Pi_1, \Theta_1, m'_1 \rangle$ and $\mu_2 = \langle \Pi_2, \Theta_2, m'_2 \rangle$, such that $\Pi_1 = \Pi_2$ and $m'_2 = m'_1 + 1$. Suppose these two interfaces guarantee schedulability of the same component $\mathscr{C}$ with their smallest possible resource bandwidth, respectively. Then μ_2 has a higher resource bandwidth than μ_1 does,* i.e., $\Theta_1 < \Theta_2$.

Proof We prove this lemma by contradiction. Consider $\mu'_2 = \langle \Pi_2, \Theta'_2, m'_2 \rangle$ such that $\Theta'_2 \leq \Theta_1$. Suppose μ'_2 guarantees schedulability of component $\mathscr{C}$ as per Theorem 1.

Let δ_d denote the difference in processor requirements of $\mathscr{C}$ on m'_1 and m'_2 processors for some interval length $A_k + D_k$, *i.e.*, difference in function DEM used in

---

[7] A periodic task $\tau = (T, C, D)$ is a special case of the identically defined sporadic task; T in the periodic case denotes the exact separation between successive job releases instead of minimum separation.

Theorem 1. Then

$$
\begin{aligned}
\delta_d &= \mathrm{DEM}(A_k + \mathrm{D}_k, m_2') - \mathrm{DEM}(A_k + \mathrm{D}_k, m_1') \\
&= \sum_{i:i \in L_{(m_2'-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}) - \sum_{i:i \in L_{(m_1'-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}) + (m_2' - m_1')\, \mathrm{C}_k \\
&= \sum_{i:i \in L_{(m_2'-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}) - \sum_{i:i \in L_{(m_1'-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}) + \mathrm{C}_k \\
&> 0 .
\end{aligned}
\tag{6}
$$

It is indicated by $\delta_d > 0$ that the same component has a greater upper bound on processor demand when it executes on more processors. Now let δ_s denote the difference in the linear supply bound function between μ_1 and μ_2' for interval length $A_k + \mathrm{D}_k$, *i.e.*,

$$
\begin{aligned}
\delta_s &= \mathrm{lsbf}_{\mu_2'}(A_k + \mathrm{D}_k) - \mathrm{lsbf}_{\mu_1}(A_k + \mathrm{D}_k) \\
&= \frac{\Theta_2'}{\Pi}\left(t - 2\left(\Pi + 1 - \frac{\Theta_2'}{m_2'}\right)\right) - \frac{\Theta_1}{\Pi}\left(t - 2\left(\Pi + 1 - \frac{\Theta_1}{m_1'}\right)\right) \\
&\leq \frac{\Theta_1}{\Pi}\left(t - 2\left(\Pi + 1 - \frac{\Theta_1}{m_2'}\right)\right) - \frac{\Theta_1}{\Pi}\left(t - 2\left(\Pi + 1 - \frac{\Theta_1}{m_1'}\right)\right) \\
&\leq \frac{2(\Theta_1)^2}{\Pi_1}\left(\frac{1}{m_2'} - \frac{1}{m_1'}\right) \\
&= -\frac{2(\Theta_1)^2}{m_1' m_2' \Pi_1} \\
&< 0 .
\end{aligned}
\tag{7}
$$

It is indicated by $\delta_s < 0$ that MPR models provide less processor supply with more available processors, when values of period and capacity are fixed. Thus $\delta_d > 0$ and $\delta_s < 0$ for all $A_k + \mathrm{D}_k$. Since μ_1 guarantees schedulability of component $\mathscr{C}$ using the smallest possible resource bandwidth, $\mathrm{DEM}(A_k + \mathrm{D}_k, m_1') = \mathrm{lsbf}_{\mu_1}(A_k + \mathrm{D}_k)$ for some $A_k + \mathrm{D}_k$. Then $\mathrm{DEM}(A_k + \mathrm{D}_k, m_2') > \mathrm{lsbf}_{\mu_2'}(A_k + \mathrm{D}_k)$ for that $A_k + \mathrm{D}_k$, and therefore μ_2' does not guarantee schedulability of $\mathscr{C}$ according to Theorem 1. This contradicts the assumption $\Theta_2' \leq \Theta_1$. $\qquad\square$

Lemma 2 suggests that when we generate interface μ, we should use the smallest number of processors to minimize resource bandwidth of μ. However an arbitrarily small number for m', say $m' = 1$, may result in an *infeasible* μ. Recall that a MPR model $\mu = \langle \Pi, \Theta, m' \rangle$ is defined to be feasible if and only if $\Theta \leq m'\Pi$. Therefore we find a feasible interface μ for $\mathscr{C}$ that: (1) guarantees schedulability of $\mathscr{C}$ based on Theorem 1 and (2) uses the smallest possible number of processors (m^*). We can find such m^* through search. Since bandwidth is monotonic with number of processors, a binary search can be performed to determine m^*. For this search to terminate a lower and upper bound on m^* should be known. $\lceil U_{\mathscr{T}} \rceil$ is clearly a lower bound on the number of processors necessary to schedule $\mathscr{C}$ where $U_{\mathscr{T}} = \sum_i \frac{\mathrm{C}_i}{\mathrm{T}_i}$. If the number of processors on the multiprocessor platform is known, then that number can be used

as an upper bound for m^*. Otherwise, the following lemma gives an upper bound for m^* as a function of task parameters.

Lemma 3 *If $m' \geq \frac{\sum_{i=1}^{n} C_i}{\min_{i=1,\ldots,n}\{D_i - C_i\}} + n$, then feasible MPR model $\mu = \langle \Pi, m'\Pi, m' \rangle$ guarantees schedulability of $\mathscr{C}$ as per Theorem 1.*

Proof

$$m' \geq \frac{\sum_{i=1}^{n} C_i}{\min_{i=1,\ldots,n}\{D_i - C_i\}} + n$$

(Since $\forall k, A_k \geq 0$ in Theorem 1)

$$\Rightarrow m' \geq \frac{\sum_{i=1}^{n} C_i}{A_k + D_k - C_k} + n \qquad \forall k \text{ and } \forall A_k$$

$$\Rightarrow m'(A_k + D_k - C_k) \geq \sum_{i=1}^{n} C_i + n(A_k + D_k - C_k) \qquad \forall k \text{ and } \forall A_k \qquad (8)$$

Now consider the function $\text{DEM}(A_k + D_k, m')$ from Theorem 1.

$$\text{DEM}(A_k + D_k, m') = \sum_{i=1}^{n} \hat{I}_{i,2} + \sum_{i:i\in L_{(m'-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}) + m' C_k$$

$$\left(\text{Since each } \hat{I}_{i,2} \leq A_k + D_k - C_k \text{ and } \sum_{i:i\in L_{(m'-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2}) \leq \sum_{i=1}^{n} C_i \right)$$

$$\Rightarrow \text{DEM}(A_k + D_k, m') \leq n(A_k + D_k - C_k) + \sum_{i=1}^{n} C_i + m' C_k$$

(From Equation (8))

$$\Rightarrow \text{DEM}(A_k + D_k, m') \leq m'(A_k + D_k - C_k) + m' C_k$$

$$\Rightarrow \text{DEM}(A_k + D_k, m') \leq \text{sbf}_\mu(A_k + D_k)$$

Since this inequality holds for all k and A_k, from Theorem 1 we get that μ is guaranteed to schedule $\mathscr{C}$. $\qquad \square$

Since μ in Lemma 3 is feasible and guarantees schedulability of $\mathscr{C}$, $\frac{\sum_{i=1}^{n} C_i}{\min_{i=1}^{n}\{D_i - C_i\}} + n$ is an upper bound for m^*. Thus we generate an interface for $\mathscr{C}$ by doing a binary search for m^* in the range $\left[\lceil U_{\mathscr{T}} \rceil, \frac{\sum_{i=1}^{n} C_i}{\min_{i=1}^{n}\{D_i - C_i\}} + n \right]$. For each value of the number of processors m', we compute the smallest value of Θ that satisfies Equation (5) in Theorem 1, assuming sbf_μ is replaced with lsbf_μ. Θ ($= \Theta^*$), corresponding to the smallest value of m' ($= m^*$) that guarantees schedulability of $\mathscr{C}$ and results in a feasible interface, is then chosen as the capacity of μ. Also m^* is chosen as the number of processors in the cluster, *i.e.*, $\mu = \langle \Pi, \Theta^*, m^* \rangle$.

Algorithm complexity. To bound A_k as in Theorem 2 we must know the value of Θ. However, since Θ is being computed, we use its smallest (0) and largest ($m'\Pi$) possible values to bound A_k. For each value of $m' > U_{\mathscr{T}}$, Θ can then be computed in pseudo-polynomial time using Theorem (1), assuming sbf is replaced with lsbf.

This follows from the fact that the denominator in the bound of A_k in Theorem 2 is non-zero. The only problem case is when $m' = \lceil U_{\mathcal{T}} \rceil = U_{\mathcal{T}}$. However in this case, we now show that $\mu = \langle \Pi, m' \Pi, m' \rangle$ can schedule $\mathcal{C}$ if and only if, $m' = 1$ and $D_i \geq T_i$ for each task τ_i in $\mathcal{T}$. Clearly, if some $D_i < T_i$, then a resource bandwidth of $U_{\mathcal{T}}$ is not sufficient to guarantee schedulability. Now suppose $m' > 1$. Then the left hand side of Equation (5) is $> \sum_{i=1}^{n} \mathrm{dbf}_{\tau_i}(A_k + D_k) \geq (A_k + D_k)U_{\mathcal{T}}$, because $\hat{I}_{i,2} = \mathrm{dbf}_{\tau_i}(A_k + D_k)$, $\bar{I}_{i,2} \geq \mathrm{dbf}_{\tau_i}(A_k + D_k)$, and $m' C_k > 0$. Hence in this case $m' > U_{\mathcal{T}}$ and this is a contradiction. Therefore computing the interface for $m' = U_{\mathcal{T}}$ can be done in constant time. The number of different values of m' to be considered is polynomial in the input size, because the search interval is bounded by numbers that are polynomial in the input parameters. Therefore the entire interface generation process has pseudo-polynomial complexity.

Cluster	Task set	$\Sigma_i \frac{C_i}{T_i}$	$\Sigma_i \frac{C_i}{D_i}$
$\mathcal{C}_1$	$\{(60,5,60),(60,5,60),(60,5,60),(60,5,60),(70,5,70),(70,5,70),$ $(80,5,80),(80,5,80),(80,10,80),(90,5,90),(90,10,90),$ $(90,10,90),(100,10,100),(100,10,100),(100,10,100)\}$	1.304	1.304
$\mathcal{C}_2$	$\{(60,5,60),(100,5,100)\}$	0.1333	0.1333
$\mathcal{C}_3$	$\{(45,2,40),(45,2,45),(45,3,40),(45,3,45),(50,5,45),$ $(50,5,50),(50,5,50),(50,5,50),(70,5,60),(70,5,60),$ $(70,5,65),(70,5,65),(70,5,65),(70,5,65),(70,5,70)\}$	1.1222	1.1930

Table 1 Clusters $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$

Example 1 Consider the example virtual clustering framework shown in Figure 2. Let clusters $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$ be assigned tasks as shown in Table 1. Interfaces μ_1^*, μ_2^* and μ_3^*, for clusters $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$, are shown in Figures 9(a), 9(b) and 9(c) respectively. In the figures we have plotted the resource bandwidth of these interfaces for varying periods and m' denotes the number of processors in the cluster.

Figures 9(a) and 9(c) show that when $m' = 1$ interfaces μ_1^* and μ_3^* are not feasible; their resource bandwidths are greater than 1 for all period values. This shows that clusters $\mathcal{C}_1$ and $\mathcal{C}_3$ are not schedulable on clusters having one processor. This is as expected because the utilization of task sets in these clusters is also greater than one. However when $m' = 2$, μ_1^* and μ_3^* are feasible, *i.e.*, their respective resource bandwidths are at most two. Therefore for clusters $\mathcal{C}_1$ and $\mathcal{C}_3$, we choose MPR interfaces μ_1^* and μ_3^* with $m' = 2$. Similarly, Figure 9(b) shows that μ_2^* is a feasible
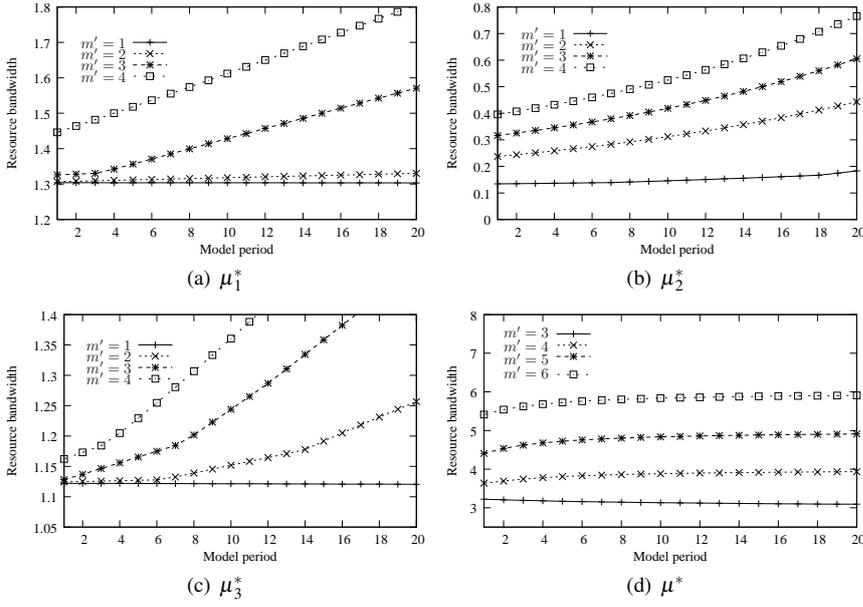
Fig. 9 MPR model based interfaces

interface for cluster $\mathscr{C}_2$ when $m' = 1$. These plots also show that resource overheads[8] incurred by our interfaces are small for the non-trivial examples presented here.

5.2 Inter-cluster scheduling

As discussed in the introduction, virtual clustering involves two-level scheduling; scheduling of tasks within each cluster (intra-cluster scheduling) and scheduling of clusters on the multiprocessor platform (inter-cluster scheduling). MPR interfaces generated in the previous section capture task-level concurrency constraints within a cluster. Hence inter-cluster scheduling need not worry about these constraints when it schedules cluster interfaces. However there is no known scheduling algorithm for MPR interfaces. Therefore we now develop a technique to transform a MPR model into periodic tasks such that processor requirements of these tasks are at least as much as those of the resource model.

Definition 2 Consider a MPR model $\mu = \langle \Pi, \Theta^*, m^* \rangle$ and let $\alpha = \Theta^* - m^* \left\lfloor \frac{\Theta^*}{m^*} \right\rfloor$ and $k = \lfloor \alpha \rfloor$. Define the transformation from μ to a periodic task set $\mathscr{T}_\mu$ as $\mathscr{T}_\mu = \{\tau_1 = (T_1, C_1, D_1), \ldots, \tau_{m^*} = (T_{m^*}, C_{m^*}, D_{m^*})\}$, where $\tau_1 = \ldots = \tau_k = \left(\Pi, \left\lfloor \frac{\Theta^*}{m^*} \right\rfloor + 1, \Pi\right)$,

---

[8] Difference between $\max_k \max_{A_k} \frac{\mathrm{DEM}(A_k + D_k, m')}{A_k + D_k}$ and resource bandwidth of MPR interface.

$$\tau_{k+1} = \left(\Pi, \left\lfloor \frac{\Theta^*}{m^*} \right\rfloor + \alpha - k \left\lfloor \frac{\alpha}{k} \right\rfloor, \Pi \right) \text{ and }$$

$$\tau_{k+2} = \ldots = \tau_{m^*} = \left(\Pi, \left\lfloor \frac{\Theta^*}{m^*} \right\rfloor, \Pi \right).$$

In this definition it is easy to see that the total processor demand of $\mathscr{T}_\mu$ is Θ^* in every period Π. Further, we have assumed that whenever Θ^* is not an integer, processor supply from μ fully utilizes one processor before using another. For example, if $\Theta^* = 2.5$ and $m^* = 3$, then μ will provide two units of resource from two processors and the remaining 0.5 units from the third processor. The following theorem proves correctness of this transformation.

Theorem 3 *If all the deadlines of task set $\mathscr{T}_\mu$ in Definition 2 are met by some processor supply with concurrency at most m^* at any time instant, then its supply bound function is lower bounded by* sbf_μ.

Proof Since $\mathscr{T}_\mu$ has m^* tasks, it can utilize at most m^* processors at any time instant. Therefore if some processor supply provides more than m^* processors at any time instant, then we can ignore these additional processor allocations. Hence we only need to consider processor supplies with concurrency at most m^*.

Total processor demand of all the tasks in $\mathscr{T}_\mu$ is Θ^* in every period of Π time units. Then to meet all the deadlines of task set $\mathscr{T}_\mu$, any processor supply must provide at least Θ^* processor units in every period of Π time units, with amount of concurrency at most m^*. But this is exactly the definition of MPR model $\mu = \langle \Pi, \Theta^*, m^* \rangle$. Therefore the supply bound function of this processor supply is lower bounded by sbf_μ. □

Thus MPR interfaces generated in the previous section can be transformed into periodic tasks using Definition 2. Once such tasks are generated for each virtual cluster, inter-cluster scheduling can be done using existing multiprocessor algorithms like gEDF, Pfair [Baruah et al., 1996], etc.

Example 2 For MPR interfaces μ_1^*, μ_2^* and μ_3^* generated in Example 1, we select periods 6, 8, and 5 respectively, *i.e.*, interfaces $\langle 6, 8.22, 2 \rangle, \langle 8, 2.34, 1 \rangle$ and $\langle 5, 5.83, 2 \rangle$. Using Definition 2 we get task sets $\mathscr{T}_{\mu_1^*} = \{(6, 5, 6), (6, 4, 6)\}, \mathscr{T}_{\mu_2^*} = \{(8, 3, 8)\}$ and $\mathscr{T}_{\mu_3^*} = \{(5, 3, 5), (5, 3, 5)\}$. Suppose the three clusters $\mathscr{C}_1, \mathscr{C}_2$ and $\mathscr{C}_3$ (*i.e.*, task set $\{\mathscr{T}_{\mu_1^*}, \mathscr{T}_{\mu_2^*}, \mathscr{T}_{\mu_3^*}\}$) are scheduled on a multiprocessor platform using gEDF. Then the resulting MPR interface μ^* is plotted in Figure 9(d). As shown in the figure, μ^* is not feasible for $m' = 3$; its resource bandwidth is greater than 3 for all period values. However these three clusters are schedulable on a multiprocessor platform having 4 processors (in the figure μ^* is feasible when $m' = 4$).

The above example clearly illustrates the advantage of virtual clustering over physical clustering. The three components $\mathscr{C}_1$, $\mathscr{C}_2$ and $\mathscr{C}_3$, would require 5 processors under physical clustering (2 each for $\mathscr{C}_1$ and $\mathscr{C}_3$ and 1 for $\mathscr{C}_2$). On the other hand, a gEDF based virtual clustering technique can schedule these clusters using only 4 processors. Although total utilization of tasks in the three clusters is 2.56, our analysis requires 4 processors to schedule the system. This overhead is as a result of the following factors: (1) gEDF is not an optimal scheduling algorithm on multiprocessor

platforms (both for intra- and inter-cluster scheduling), (2) the schedulability conditions we use are only sufficient conditions, and (3) capturing task-level concurrency constraints in a component interface leads to some increase in processor requirements (resource overhead of abstracting a cluster into MPR interface).

6 Virtual cluster-based scheduling algorithms

In this section we propose new virtual-cluster based scheduling algorithms for implicit deadline sporadic task systems. Prior to presenting these algorithms, we eliminate resource overheads from the virtual clustering framework proposed in Section 5.

6.1 Improved virtual-clustering framework

In this section we present an inter-cluster scheduling algorithm that is optimal whenever all the MPR interfaces being scheduled under it have identical periods. We also present another transformation from MPR models to periodic tasks, which along with the optimal inter-cluster scheduler, results in an improved sbf for MPR models. These two together, eliminate the resource overheads described at the end of previous section.

McNaughton [McNaughton, 1959] presented an algorithm for scheduling real-time jobs in a given time interval on a multiprocessor platform. This algorithm can be explained as follows: Consider n jobs to be scheduled on m processors in a time interval $(t_1, t_2]$ of length t, such that no job is simultaneously scheduled on more than one processor. The job set need not be sorted in any particular order. McNaughton's algorithm schedules the ith job on the first non-empty processor, packing jobs from left to right. Suppose the $(i-1)^{st}$ job was scheduled on processor k up to time instant t_3 ($t_1 \leq t_3 \leq t_2$). Then up to $t_2 - t_3$ time units of the ith job are scheduled on processor k and the remaining time units are scheduled on processor $k+1$ starting from t_1. Figure 10 illustrates this schedule for a job set $\{J_1, \ldots, J_5\}$ on 4 processors. Note that if the total resource demand of a job is at most $t_2 - t_1$, then (1) the job is scheduled on at most two processors by McNaughton's algorithm, and (2) the job is never scheduled simultaneously on both the processors. The following theorem establishes conditions under which this algorithm can successfully schedule job sets.

Theorem 4 (Theorem 3.1 in [McNaughton, 1959]) *Let* $c_1, \ldots, c_n$ *denote the number of processor units of the n jobs that must be scheduled in the interval* $(t_1, t_2]$ *on m identical, unit-capacity processors. If* $\sum_{i=1}^{n} c_i \leq m(t_2 - t_1)$, *then a necessary and sufficient condition to guarantee schedulability of this job set is that for all i,* $c_i \leq t_2 - t_1$.

If $c_i > t_2 - t_1$ then the ith job cannot be scheduled in the interval $(t_1, t_2]$ by any scheduling algorithm, unless the job is simultaneously scheduled on more than one processor. Likewise, if $\sum_{i=1}^{n} c_i > m(t_2 - t_1)$, then also the job set cannot be scheduled by any scheduling algorithm, because the total processor demand in the interval $(t_1, t_2]$ is greater than the total available processing capacity. Hence Theorem 4 in fact

Fig. 10 Schedule of job set under McNaughton's algorithm in the interval $(t_1, t_2]$

shows that McNaughton's algorithm is optimal for scheduling job sets in a given time interval.

Consider a periodic task set $\mathscr{T} = \{(T, C_1, T), \ldots, (T, C_n, T)\}$. Tasks in $\mathscr{T}$ have identical periods and implicit deadline. Suppose we use McNaughton's algorithm in the intervals $(kT, (k+1)T]$, $k \in \mathbb{I}$, to schedule jobs of $\mathscr{T}$ on m identical, unit-capacity processors. Then for each interval $(kT, (k+1)T]$ (1) all jobs of $\mathscr{T}$ are released at the beginning of the interval (kT) and (2) all jobs of $\mathscr{T}$ have deadline at the end of the interval $((k+1)T)$. Therefore, from Theorem 4, we get that McNaughton's algorithm optimally schedules these jobs in each interval and this leads to the following direct corollary.

Corollary 1 *Let* $\mathscr{T} = \{\tau_1 = (T_1, C_1, D_1), \ldots, \tau_n = (T_n, C_n, D_n)\}$ *denote a periodic task set to be scheduled on m identical, unit-capacity processors. If* $T_1 = \ldots = T_n = D_1 = \ldots = D_n (= T)$, *then a necessary and sufficient condition for* $\mathscr{T}$ *to be schedulable using McNaughton's algorithm is that* $\sum_{i=1}^{n} C_i \leq mT$ *and* $C_i \leq T_i$ *for each i.*

Consider the virtual clustering framework proposed in Section 5. Suppose all the MPR interfaces in this framework have identical periods. Then all the periodic tasks generated using Definition 2 also have identical periods. And from Corollary 1 we get that McNaughton's algorithm is optimal for scheduling these tasks on the physical platform, *i.e.*, the algorithm does not incur any resource overhead for inter-cluster scheduling.

Another source of resource overhead is the abstraction of a cluster into MPR interface and its transformation to a periodic task set. This overhead results from the sub-optimality of sbf of MPR models which can be explained as follows. Consider the two functions, sbf_μ and $usbf_\mu$, shown in Figure 11. The resource bandwidth used by μ is equal to the slope of line $usbf_\mu$ $(\frac{\Theta}{\Pi})$. Suppose μ is used to abstract the processor demand of cluster $\mathscr{C}$ in Theorem 1. Since sbf_μ has a non-zero x-axis intercept, the bandwidth of μ is strictly larger than the schedulability load, $\max_k \max_{A_k} \text{DEM}(A_k + D_k, m')/(A_k + D_k)$, of cluster $\mathscr{C}$. If not then, as shown in Figure 11, there exists some $A_k + D_k$ for which Theorem 1 is not satisfied. This explains the resource overhead in the abstraction of clusters to MPR interfaces. Now suppose μ is transformed into the periodic task set $\mathscr{T}_\mu$ using Definition 2. Then from Theorem 3 we get that the total processor demand of $\mathscr{T}_\mu$ is at least as much as sbf_μ. However, since sbf_μ does
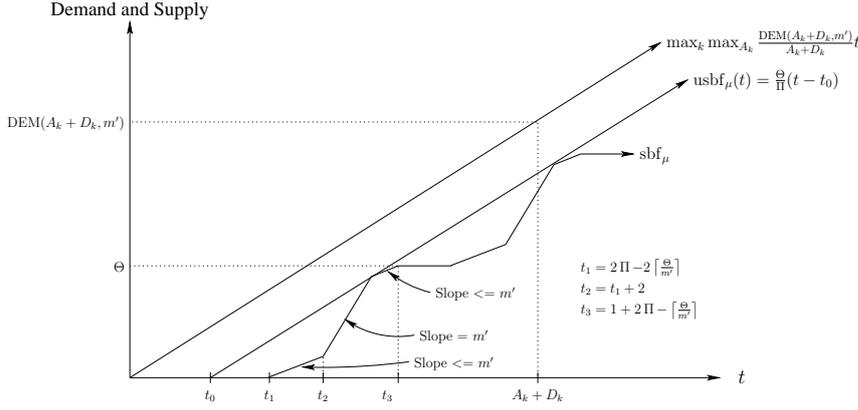
Fig. 11 Bandwidth of sbf_μ and schedulability load of cluster $\mathscr{C}$
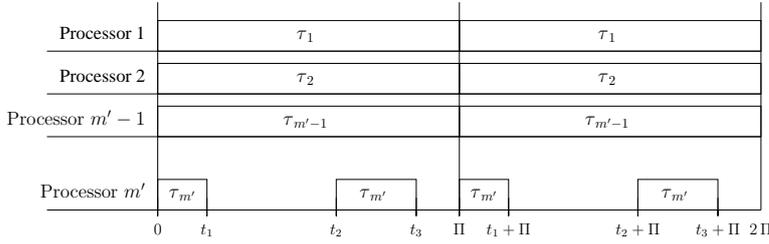

Fig. 12 McNaughton's schedule of implicit deadline periodic tasks with identical periods

not guarantee Θ resource units in an interval of length Π (see Figure 11), a processor supply with supply bound function exactly sbf_μ cannot schedule $\mathscr{T}_\mu$. This explains the resource overhead in the transformation of MPR interfaces to periodic tasks.

To eliminate the aforementioned overheads, we must modify the transformation presented in Definition 2. This is because the schedule of $\mathscr{T}_\mu$ determines the processor supply from the multiprocessor platform to μ, and this in turn determines sbf_μ. We now present a new transformation from MPR models to periodic tasks as follows.

Definition 3 Given a MPR model $\mu = \langle \Pi, \Theta^*, m^* \rangle$, we define its transformation to a periodic task set $\mathscr{T}_\mu$ as
$\mathscr{T}_\mu = \{\tau_1 = (T_1, C_1, D_1), \ldots, \tau_{m^*} = (T_{m^*}, C_{m^*}, D_{m^*})\}$, where
$\tau_1 = \ldots = \tau_{m^*-1} = (\Pi, \Pi, \Pi)$ and
$\tau_{m^*} = (\Pi, \Theta^* - (m^* - 1)\Pi, \Pi)$.

In this definition it is easy to see that the total processor demand of $\mathscr{T}_\mu$ is Θ^* in every Π time units, with concurrency at most m^*. Therefore Theorem 3 holds in this case as well, *i.e.*, if all the deadlines of task set $\mathscr{T}_\mu$ are met by some processor supply with concurrency at most m^* at any time instant, then its supply bound function is lower bounded by sbf_μ.

Now suppose a cluster is abstracted into MPR interface $\mu = \langle \Pi, \Theta, m' \rangle$, which is then transformed into task set $\mathscr{T}_\mu$ using Definition 3. Let $\mathscr{T}_\mu$ be scheduled on the
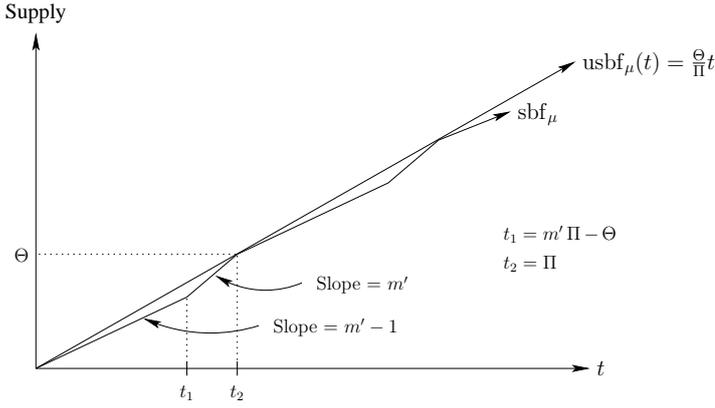
Fig. 13 Improved sbf_μ and its linear upper bound usbf_μ

multiprocessor platform using McNaughton's algorithm, along with periodic tasks that all have period and deadline Π (implicit deadline task system with identical periods). Figure 12 illustrates the McNaughton schedule for task set $\mathscr{T}_\mu$. As can be seen in the figure, tasks $\tau_1, \ldots, \tau_{m'-1}$ completely utilize $m' - 1$ processors on the platform. Further, every job of task $\tau_{m'}$ is scheduled in an identical manner within its execution window (intervals $(0, t_1]$ and $(t_2, t_3]$ relative to release time). Since this schedule of $\mathscr{T}_\mu$ is used as the processor supply for the underlying MPR interface, μ guarantees Θ processor units in any time interval of length Π, 2Θ processor units in any time interval of length 2Π, and so on. In other words, the blackout interval of sbf_μ (described in Section 2.2) reduces to zero. The resulting sbf is plotted in Figure 13 and it is given by the following equation.

$$\text{sbf}_\mu(t) = \left\lfloor \frac{t}{\Pi} \right\rfloor \Theta + \left(t - \left\lfloor \frac{t}{\Pi} \right\rfloor \Pi \right) m' - \min \left\{ t - \left\lfloor \frac{t}{\Pi} \right\rfloor \Pi, m'\Pi - \Theta \right\} \quad (9)$$

sbf_μ guarantees Θ resource units in any time interval of length Π. Then a processor supply with supply bound function equal to sbf_μ can successfully schedule task set $\mathscr{T}_\mu$. Thus we have eliminated the resource overhead that was present in the previous transformation given in Definition 2.

Now consider the schedulability condition for cluster $\mathscr{C}$ given by Equation (5) in Theorem 1. This equation needs to be evaluated for all values of A_k up to the bound given in Theorem 2 and for all tasks τ_k in cluster $\mathscr{C}$. In this equation it is easy to see that $\text{DEM}(A_k + D_k, m')$ increases by at most $m' - 1$ for every unit increase in A_k, as long as $A_k + 1 + D_k$ does not coincide with the release or deadline of some task in cluster $\mathscr{C}$. In other words, $\text{DEM}(A_k + 1 + D_k, m') \leq \text{DEM}(A_k + D_k, m') + m' - 1$, whenever $A_k + 1 + D_k$ is not equal to $l\text{T}_i$ or $l\text{T}_i + D_i$ for any l and i (denoted as property *bounded increase*). This is because over such unit increases in A_k, $m'C_k$ and each $\hat{I}_{i,2}$ remain constant and $\sum_{i:i \in L_{(m'-1)}} (\bar{I}_{i,2} - \hat{I}_{i,2})$ increases by at most $m' - 1$. However sbf_μ increases by at least $m' - 1$ over each unit time interval (see Figure 13). Therefore to generate interface μ, it is sufficient to evaluate Equation (5) at only

those values of A_k for which $A_k + D_k$ is equal to $l\,T_i$ or $l\,T_i + D_i$ for some l and i. Now suppose period Π of μ is equal to the GCD (greatest common divisor) of the periods and deadlines of all the tasks in cluster $\mathscr{C}$. Then all the required evaluations of Equation (5) will occur at time instants t for which $\mathrm{sbf}_\mu(t) = \mathrm{usbf}_\mu(t) = \frac{\Theta}{\Pi}t$ (see Figure 13). In other words, the right hand side of Equation (5) can be replaced with $\frac{\Theta}{\Pi}t$. This means that the resource bandwidth of the resulting interface μ ($\frac{\Theta}{\Pi}$) will be equal to the schedulability load, $\max_k \max_{A_k} \frac{\mathrm{DEM}(A_k + D_k, m')}{A_k + D_k}$, of cluster $\mathscr{C}$. Thus we have eliminated the resource overhead that was previously present in the cluster abstraction process.

We now summarize the contributions of this section. The following theorem, which is a direct consequence of the above discussions, states the fundamental result of this section. This theorem states that our improved virtual-clustering framework does not incur any resource overheads in transforming MPR interfaces to periodic tasks or in scheduling the transformed tasks on the multiprocessor platform.

Theorem 5 *Consider MPR interfaces $\mu_1 = \langle \Pi, \Theta_1, m'_1 \rangle, \ldots, \mu_p = \langle \Pi, \Theta_p, m'_p \rangle$. Suppose they are transformed to periodic tasks using Definition 3. McNaughton's algorithm can successfully schedule the transformed tasks on m identical, unit-capacity processors if and only if,*

$$\sum_{i=1}^{p} \frac{\Theta_i}{\Pi} \leq m$$

Suppose (1) we want to schedule a constrained deadline sporadic task set $\mathscr{T}$ using virtual clusters on m identical, unit-capacity processors, (2) task-cluster mapping is given, and (3) each intra-cluster scheduler is such that the corresponding schedulability condition satisfies *bounded increase* property described above (*e.g.*, gEDF). Let (1) each virtual cluster be abstracted into an MPR interface whose period Π is equal to the GCD of the periods and deadlines of all the tasks in $\mathscr{T}$, (2) these interfaces be transformed into periodic tasks using Definition 3, and (3) these periodic tasks be scheduled on the multiprocessor platform using McNaughton's algorithm. Then, in addition to the results stated in Theorem 5, the resource bandwidth of each MPR interface will be equal to the schedulability load of the corresponding cluster.

6.2 Virtual clustering of implicit deadline task systems

In this section we propose two virtual cluster-based scheduling algorithms for implicit deadline sporadic task sets. We consider the problem of scheduling an implicit deadline sporadic task set $\mathscr{T} = \{\tau_1 = (T_1, C_1, T_1), \ldots, \tau_n = (T_n, C_n, T_n)\}$ on m identical, unit-capacity processors. We first present a new virtual-clustering technique that is optimal like the well known Pfair algorithm [Baruah et al., 1996], but unlike Pfair, has a non-trivial bound on the number of preemptions. The second technique extends the well known algorithm US-EDF$\{m/(2m-1)\}$ [Srinivasan and Baruah, 2002] with virtual clusters. We show that the presently known processor utilization bound of US-EDF$\{m/(2m-1)\}$ can be improved by using virtual clusters.

6.2.1 VC-IDT scheduling algorithm

In VC-IDT (*Virtual Clustering - Implicit Deadline Tasks*) scheduling algorithm we consider a trivial task-processor mapping that assigns each task $\tau_i \in \mathscr{T}$ to its own virtual cluster $\mathscr{C}_i$ having one processor. Since each cluster has only one processor, we assume that each cluster uses EDF for intra-cluster scheduling[9]. Each cluster $\mathscr{C}_i$ is abstracted into a MPR interface $\mu_i = \langle \Pi, \Theta_i, 1 \rangle$, where Π is equal to the GCD of $T_1, \ldots, T_n$ and $\Theta_i / \Pi = C_i / T_i$. Further, each interface μ_i is transformed into periodic tasks using Definition 3 and the resulting task set is scheduled on the multiprocessor platform using McNaughton's algorithm. The following theorem proves that VC-IDT is an optimal algorithm for scheduling implicit deadline sporadic task systems on identical, unit-capacity multiprocessor platforms.

Theorem 6 *Consider sporadic tasks* $\mathscr{T} = \{ \tau_1 = (T_1, C_1, T_1), \ldots, \tau_n = (T_n, C_n, T_n) \}$. *A necessary and sufficient condition to guarantee that* $\mathscr{T}$ *is schedulable on m identical, unit-capacity processors using VC-IDT algorithm is*

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \le m \tag{10}$$

Proof In VC-IDT each virtual cluster $\mathscr{C}_i$, comprising of task τ_i, is abstracted to interface $\mu_i = \langle \Pi, \Theta_i, 1 \rangle$, where Π is equal to the GCD of $T_1, \ldots, T_n$ and $\frac{\Theta_i}{\Pi} = \frac{C_i}{T_i}$. The interface set $\mu_1, \ldots, \mu_n$, all having identical periods, are then transformed to periodic tasks using Definition 3 and scheduled on the platform using McNaughton's algorithm. Therefore, from Theorem 5, we get that this interface set is schedulable on the multiprocessor platform if and only if,

$$\sum_{i=1}^{n} \frac{\Theta_i}{\Pi} \le m$$

$$\Rightarrow \sum_{i=1}^{n} \frac{C_i}{T_i} \le m$$

To prove this theorem we then need to show that for each i, interface μ_i can schedule cluster $\mathscr{C}_i$. $\mathscr{C}_i$ comprises of sporadic task τ_i and uses EDF scheduler. Therefore any processor supply that can guarantee C_i processor units in all time intervals of length T_i can be used to schedule τ_i. But from the sbf of model μ_i (Equation (9)), it is easy to see that μ_i guarantees C_i processor units in any time interval of length T_i. This proves the theorem. □

Equation (10) is known to be a necessary and sufficient feasibility condition for scheduling implicit deadline sporadic task systems on *m* identical, unit-capacity processors [Srinivasan and Anderson, 2006]. Hence VC-IDT is an optimal scheduling algorithm for this problem domain. The other known optimal schedulers for this problem, to the best of our knowledge, are the PD[2] Pfair/ERfair algorithm [Srinivasan and Anderson, 2006] and the task-splitting algorithm [Andersson and Bletsas, 2008].

---

[9] Since each cluster also has only one task, any work conserving algorithm can be used for intra-cluster scheduling.

PD^2 algorithm is known to incur a high number of preemptions in order to guarantee P-fairness/ER-fairness, because fairness is a stricter requirement than deadline satisfaction. It can potentially incur m preemptions in every time unit, which is the maximum possible on this multiprocessor platform. In contrast, the number of preemptions incurred by VC-IDT has a non-trivial upper bound which can be explained as follows. When interfaces $\mu_1, \ldots, \mu_n$ are scheduled using McNaughton's algorithm (after being transformed into periodic tasks), there are at most $m-1$ of them that use more than one processor. Each such interface μ_i is preempted once in every Π time units and this may result in a preemption in the execution of task τ_i. Each of the other $n - (m-1)$ tasks may also experience preemption once in every Π time units, because the execution requirements of a job of this task cannot be entirely satisfied by a single job of the corresponding interface. The entire sporadic task set will thus incur at most n preemptions in every Π time units. Therefore when Π, the GCD of task periods, is very small VC-IDT does not offer any advantage over PD^2 algorithm. This can happen for instance even if two task periods are co-prime (the GCD in this case is one). However, in real-world systems, it has been observed that task periods are typically harmonic to (multiples of) each other. For example, harmonic task periods can be found in avionics real-time applications; see ARINC-653 standards [Eng, 2006] and sample avionics workloads in the appendix of this technical report [Easwaran et al., 2009]. In this case, the GCD of task periods is equal to the smallest task period (typically a few milliseconds as indicated by the workloads in [Easwaran et al., 2009]), and then VC-IDT incurs far fewer preemptions than Pfair/ERfair algorithms. It is worth noting that although the BoundaryFair algorithm [Zhu et al., 2003] incurs fewer preemptions than VC-IDT, it is only optimal for scheduling periodic (not sporadic) task systems.

The task splitting algorithm proposed by Andersson and Bletsas [Andersson and Bletsas, 2008] has also been shown to be optimal for implicit deadline sporadic task systems (see Theorem 3 in [Andersson and Bletsas, 2008]). Suppose $jobs(t)$ denotes the maximum number of jobs that will be released by the task system in any time interval of length t. Then this algorithm is known to incur at most $\frac{3mt}{\text{GCD}} + 2m + jobs(t)$ number of preemptions, where GCD denotes the greatest common divisor of task periods (derived from Theorems 2 and 3 in [Andersson and Bletsas, 2008]). In contrast, VC-IDT algorithm incurs at most $\frac{nt}{\text{GCD}}$ number of preemptions. Clearly, our algorithm outperforms the task splitting approach whenever $n < 3m$. When $n > 3m$, either algorithm can incur fewer preemptions depending on the value of GCD and the relation between task periods. The runtime complexity of the dispatcher under task splitting is the same as that of partitioned EDF (roughly logarithmic in the number of tasks for every scheduling decision). In contrast, under VC-IDT, the entire interface schedule based on McNaughton's algorithm can be generated and stored offline for intervals of length GCD. Therefore at runtime the tasks can be scheduled in constant time. This vastly improved runtime complexity at the expense of increased storage requirements is particularly useful in embedded systems, where cheaper ROM and Flash memory is still preferred over the more expensive RAM (for instance, MICAz, the sensor node from crossbow, has 512k of Flash memory whereas only 4k of RAM [mic]). Finally, a practical limitation of the task splitting approach is that they do not provide any error isolation mechanism, *i.e.*, a task that executes for more than its stated worst-

case execution time can cause other tasks in the system to miss deadlines. In contrast, VC-IDT provides automatic error isolation, because a mis-behaving task will never get more processor share than already provided by its MPR interface.

6.2.2 Virtual clustering for US–EDF$\{m/(2m-1)\}$

US-EDF$\{m/(2m-1)\}$, proposed by Srinivasan and Baruah [Srinivasan and Baruah, 2002], is a global scheduling algorithm for implicit deadline sporadic task systems. Under this algorithm each task with utilization ($\frac{C}{T}$) greater than $\frac{m}{2m-1}$ is given the highest priority, and the remaining tasks are scheduled based on gEDF. It has been shown that this algorithm has a processor utilization bound of $\frac{m^2}{2m-1}$, *i.e.*, any sporadic task set with total utilization ($\sum_i \frac{C_i}{T_i}$) at most $\frac{m^2}{2m-1}$ can be scheduled by US-EDF$\{m/(2m-1)\}$ on m identical, unit-capacity processors [Srinivasan and Baruah, 2002].

Now consider the following virtual cluster-based US-EDF$\{m/(2m-1)\}$ scheduling algorithm. Let each task with utilization greater than $\frac{m}{2m-1}$ be assigned to its own virtual cluster having one processor and using EDF (denoted as *high utilization cluster*), and all the remaining tasks be assigned to a single cluster using gEDF (denoted as *low utilization cluster*). Each cluster is abstracted to a MPR interface such that period Π of each interface is equal to the GCD of $T_1, \ldots, T_n$. Each high utilization cluster is abstracted to interface $\langle \Pi, \Theta, 1 \rangle$, where $\frac{\Theta}{\Pi}$ is equal to the utilization of task in the cluster (Theorem 6 proves correctness of this abstraction). The low utilization cluster is abstracted to interface $\mu_{low} = \langle \Pi, \Theta', m' \rangle$, where Θ' and m' are generated using techniques in Sections 5 and 6.1. Finally, these interfaces are transformed to periodic tasks using Definition 3 and the resulting task set is scheduled on the multiprocessor platform using McNaughton's algorithm.

We now derive a utilization bound for the virtual cluster-based US-EDF$\{m/(2m-1)\}$ algorithm described above. Suppose α denotes the total utilization of all the high utilization tasks, *i.e.*, the total resource bandwidth of all the MPR interfaces that represent high utilization clusters is α. Since all the interfaces that we generate have identical periods, from Theorem 5 we get that the maximum resource bandwidth available for μ_{low} is $m - \alpha$. This means that $\frac{\Theta'}{\Pi} \leq m - \alpha$ and $\alpha \leq m$ are necessary and sufficient conditions to guarantee schedulability of task set $\mathscr{T}$ under virtual cluster-based US-EDF$\{m/(2m-1)\}$.

Suppose $\alpha > m - 1$. Then $m - \alpha < 1$ and $m' \leq 1$. The last inequality can be explained as follows. $m' = \left\lceil \frac{\Theta'}{\Pi} \right\rceil$ because m' is the smallest number of processors upon which the low utilization cluster is schedulable. Then $\frac{\Theta'}{\Pi} \leq m - \alpha < 1$ implies $m' \leq 1$. In this case the low utilization cluster is scheduled on a uniprocessor platform and gEDF reduces to EDF, an optimal uniprocessor scheduler with utilization bound $m - \alpha$. Therefore virtual cluster-based US-EDF$\{m/(2m-1)\}$ is optimal whenever $\alpha > m - 1$, *i.e.*, it can successfully schedule task set $\mathscr{T}$ if $\sum_{i=1}^n \frac{C_i}{T_i} \leq m$.

Now suppose $\alpha \leq m - 1$. To derive the utilization bound in this case, we use a utilization bound of gEDF that was developed by Goossens *et al.* [Goossens et al., 2003]. As per this bound μ_{low} can support a low utilization cluster whose total task

utilization is upper bounded by $\left(\frac{\Theta'}{\Pi} - \left(\frac{\Theta'}{\Pi} - 1 \right) U_{max} \right)$, where U_{max} is the maximum utilization of any task in the cluster. Therefore, in this case, the utilization bound of virtual cluster-based US-EDF$\{m/(2m-1)\}$ is

$$\alpha + \left(\frac{\Theta'}{\Pi} - \left(\frac{\Theta'}{\Pi} - 1 \right) U_{max} \right)$$
$$= \alpha + (m - \alpha - (m - \alpha - 1) U_{max})$$

Since $m - \alpha \geq 1$, the bound in the above equation is minimized when U_{max} is maximized. Substituting $U_{max} = \frac{m}{2m-1}$ (largest utilization of any task in the low utilization cluster), we get a utilization bound of

$$\alpha + \left(m - \alpha \left(1 - \frac{m}{2m-1} \right) + \frac{m}{2m-1} \right)$$
$$= \frac{\alpha(2m-1) + (m-\alpha)(m-1) + m}{2m-1}$$
$$\geq \frac{\alpha(2m-1) + (m-\alpha)(m-1) + m}{2m-1}$$
$$= \frac{m^2 + \alpha m}{2m-1}$$

Thus the processor utilization bound of virtual cluster-based US-EDF$\{m/(2m-1)\}$ is $\min \left\{ m, \frac{m^2 + \alpha m}{2m-1} \right\}$. It is easy to see that whenever $\alpha > 0$, this bound is greater than the presently known utilization bound of $\frac{m^2}{2m-1}$ for US-EDF$\{m/(2m-1)\}$. This shows that virtual clustering, unlike the earlier US-EDF$\{m/(2m-1)\}$ algorithm, allows one to use the leftover processing capacity from high utilization clusters for scheduling tasks in the low utilization cluster. It also shows that the improvement in utilization bound is achievable even when clusters are scheduled on the platform using non-trivial abstractions such as MPR models. This gain however comes at a cost; since Π is equal to the GCD of task periods, the resulting schedule can potentially incur more preemptions when compared to the original algorithm.

7 Conclusions

In this paper we have considered the idea of cluster-based scheduling on multiprocessor platforms as an alternative to existing partitioned and global scheduling strategies. Cluster-based scheduling can be viewed as a two-level scheduling strategy. Tasks in a cluster are globally scheduled within the cluster (intra-cluster scheduling) and clusters are then scheduled on the multiprocessor platform (inter-cluster scheduling). We have further classified clustering into physical (one-to-one) and virtual (many-to-many), depending on the mapping between clusters and processors on the platform. Virtual clustering is more general and less sensitive to task-processor mappings than physical clustering.

Towards supporting virtual cluster-based scheduling, we have developed techniques for hierarchical scheduling in this paper. Resource requirements and concurrency constraints of tasks within each cluster are first abstracted into MPR interfaces. These interfaces are then transformed into periodic tasks which are used for inter-cluster scheduling. We have also developed an efficient technique to minimize processor utilization of individual clusters under gEDF. Finally, we developed a new optimal scheduling algorithm for implicit deadline sporadic task systems, and also illustrated the power of general task-processor mappings by virtualizing US-EDF$\{m/(2m-1)\}$ algorithm.

We only focused on gEDF for intra-cluster and McNaughton's for inter-cluster scheduling. However, our approach of isolating the inter-cluster scheduler from task-level concurrency constraints is general, and can be adopted to other scheduling algorithms as well. Moreover, this generality also means that our technique enables clusters with different intra-cluster schedulers to be scheduled on the same platform. It would be interesting to generalize this framework by including other intra and inter-cluster scheduling algorithms, with an aim to solve some open problems in multiprocessor scheduling.

References

MICAz. http://www.xbow.com/Products/productdetails.aspx?sid=164.

Luis Almeida and Paulo Pedreiras. Scheduling within temporal partitions: Response-time analysis and server design. In *Proceedings of ACM & IEEE International Conference on Embedded Software*, pages 95–103, 2004.

James Anderson, John Calandrino, and UmaMaheswari Devi. Real-time scheduling on multicore platforms. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 179–190, 2006.

James H. Anderson and Anand Srinivasan. Early-release fair scheduling. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 35–43, 2000.

Björn Andersson and Konstantinos Bletsas. Sporadic multiprocessor scheduling with few preemptions. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 243–252, 2008.

Björn Andersson and Eduardo Tovar. Multiprocessor scheduling with few preemptions. In *Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 322–334, 2006.

Björn Andersson, Sanjoy Baruah, and Jan Jonsson. Static-priority scheduling on multiprocessors. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 193–202, 2001.

Björn Andersson, Konstantinos Bletsas, and Sanjoy K. Baruah. Scheduling arbitrary-deadline sporadic tasks on multiprocessors. In *Proceedings of IEEE Real-Time Systems Symposium*, 2008.

Theodore Baker. An analysis of EDF schedulability on a multiprocessor. *IEEE Transactions on Parallel Distributed Systems*, 16(8):760–768, 2005a.

Theodore Baker. An analysis of fixed-priority schedulability on a multiprocessor. *Real-Time Systems*, 32(1–2):49–71, 2006.

Theodore P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 120–129, 2003.

Theodore P. Baker. Comparison of empirical success rates of global vs. partitioned fixed-priority EDF scheduling for hard real-time. Technical Report TR–050601, Department of Computer Science, Florida State University, Tallahassee, 2005b.

S. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

Sanjoy Baruah. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *IEEE Transactions on Computers*, 53(6):781–784, 2004.

Sanjoy Baruah. Techniques for multiprocessor global schedulability analysis. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 119–128, 2007.

Sanjoy Baruah and Nathan Fisher. The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems. *IEEE Transactions on Computers*, 55(7):918–923, 2006.

Sanjoy Baruah, Aloysius Mok, and Louis Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 182–190, 1990.

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

Sanjoy K. Baruah and Theodore Baker. Global EDF schedulability analysis of arbitrary sporadic task systems. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 3–12, 2008b.

Sanjoy K. Baruah and John Carpenter. Multiprocessor fixed-priority scheduling with restricted interprocessor migrations. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 195–202, 2003.

Sanjoy K. Baruah and Nathan Fisher. Global deadline-monotonic scheduling of arbitrary-deadline sporadic task systems. In *International Conference on Principles of Distributed Systems*, pages 204–216, 2007.

Marko Bertogna and Michele Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 149–160, 2007.

Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 209–218, 2005a.

Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. In *Proceedings of International Conference on Principles of Distributed Systems*, pages 306–321, 2005b.

John M. Calandrino, James H. Anderson, and Dan P. Baumberger. A hybrid real-time scheduling approach for large-scale multicore platforms. In *Proceedings of*

Euromicro Conference on Real-Time Systems, pages 247–258, 2007.

Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 101–110, 2006.

S Cho, S-K Lee, S Ahn, and K-J Lin. Efficient real-time scheduling algorithms for multiprocessor systems. *IEICE Transactions on Communications*, E85–B(12): 2859–2867, 2002.

Michele Cirinei and Theodore P. Baker. EDZL scheduling analysis. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 9–18, 2007.

Robert Davis and Alan Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 389–398, 2005.

Z. Deng and Jane Liu. Scheduling real-time applications in an open environment. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 308–319, 1997.

Arvind Easwaran, Madhukar Anand, and Insup Lee. Compositional analysis framework using EDP resource models. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 129–138, 2007.

Arvind Easwaran, Insup Lee, Oleg Sokolsky, and Steve Vestal. A compositional framework for avionics (ARINC-653) systems. Technical Report MS–CIS–09–04, University of Pennsylvania, 2009. Available at http://repository.upenn.edu/cis_reports/898/.

ARINC specification 653-2, part I. Engineering Standards for Avionics and Cabin Systems (AEEC), 2006.

Xiang Feng and Aloysius Mok. A model of hierarchical real-time virtual resources. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 26–35, 2002.

Nathan Fisher, Sanjoy Baruah, and Theodore P. Baker. The partitioned scheduling of sporadic tasks according to static-priorities. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 118–127, 2006.

Kenji Funaoka, Shinpei Kato, and Nobuyuki Yamasaki. Work-conserving optimal real-time scheduling on multiprocessors. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 13–22, 2008.

Joel Goossens, Shelby Funk, and Sanjoy Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, 25(2–3):187–205, 2003.

Philip Holman and James H. Anderson. Guaranteeing Pfair supertasks by reweighting. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 203–212, 2001.

Shinpei Kato and Nobuyuki Yamasaki. Real-time scheduling with task splitting on multiprocessors. In *Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 441–450, 2007.

Tei-Wei Kuo and Ching-Hui Li. A fixed-priority-driven open environment for real-time applications. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 256–267, 1999.

Hennadiy Leontyev and James H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 191–200, 2008.

J. Y.-T Leung. A new algorithm for scheduling periodic, real-time tasks. *Algorithmica*, 4:209–219, 1989.

Giuseppe Lipari and Enrico Bini. Resource partitioning among real-time applications. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 151–158, 2003.

Giuseppe Lipari, John Carpenter, and Sanjoy Baruah. A framework for achieving inter-application isolation in multiprogrammed hard-real-time environments. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 217–226, 2000.

C. L. Liu. Scheduling algorithms for multiprocessors in a hard-real-time environment. Technical report, JPL Space Programs Summary 37–60, vol. II, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 1969.

J. M. López, J. L. Díaz, and D. F. García. Minimum and maximum utilization bounds for multiprocessor RM scheduling. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 67–75, 2001.

Robert McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959.

Mark Moir and Srikanth Ramamurthy. Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 294–303, 1999.

Aloysius Mok, Xiang Feng, and Deji Chen. Resource partition for real-time systems. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, pages 75–84, 2001.

Dong-Ik Oh and Theodore Baker. Utilization bounds for n-processor rate monotone scheduling with static processor assignment. *Real-Time Systems*, 15(2):183–192, 1998.

Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 2–13, 2003.

Insik Shin and Insup Lee. Compositional real-time scheduling framework. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 57–67, 2004.

Insik Shin and Insup Lee. Compositional real-time scheduling framework with periodic model. *ACM Transactions on Embedded Computing Systems*, 7(3), 2008.

Insik Shin, Arvind Easwaran, and Insup Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 189–190, 2008.

Anand Srinivasan and James H. Anderson. Optimal rate-based scheduling on multiprocessors. *Journal of Computer and System Sciences*, 72(6):1094–1117, 2006.

Anand Srinivasan and Sanjoy Baruah. Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, 84(2):93–98, 2002.

Dakai Zhu, Daniel Mossé, and Rami Melhem. Multiple-resource periodic scheduling problem: how much fairness is necessary? In *Proceedings of IEEE Real-Time Systems Symposium*, pages 142–153, 2003.