Hierarchical Scheduling Framework for Virtual Clustering of Multiprocessors *

Insik Shin
MRTC, Mälardalen University
721 23 Västerás, Sweden
insik.shin@mdh.se

Arvind Easwaran, Insup Lee
University of Pennsylvania
PA, 19104, USA
{arvinde, lee}@cis.upenn.edu

Abstract

Scheduling of 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 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 processor utilization of individual clusters. Since neither partitioned nor global strategies dominate over the other, cluster-based scheduling is a natural direction for research towards achieving improved utilization bounds.

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 real-time multiprocessor scheduling theories. In general, existing real-time scheduling approaches over m 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.

*This work was partially supported by the Swedish Foundation for Strategic Research, via the strategic research centre PROGRESS. This research was also supported in part by AFOSR FA9550-07-1-0216, NSF CNS-0509327, NSF CNS-0509143, and NSF CNS-0720703.

We consider another approach using a notion of *(processor) cluster*. A cluster is a set of m' processors, where $1 \leq m' \leq m$. Under cluster-based scheduling, tasks are statically assigned to a cluster and are globally scheduled within the cluster. Cluster-based scheduling 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 [12]. A virtual cluster allows a dynamic one-to-many mapping between its m' processors and some m° out of m processors in the platform. Scheduling tasks in this virtual cluster can be viewed as scheduling them globally on m° processors in the platform with amount of concurrency at most m'. A key difference is that physical clusters share no processors in the platform, while virtual clusters can share some.

Motivating examples. We now illustrate the capabilities of cluster-based scheduling over partitioned and global scheduling using an example. Consider a sporadic task set 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 minimum separation, C worst-case execution requirement, and D 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 (EDF) [24], EDZL [13], least laxity first (LLF) [27], fp-EDF [4], and US-EDF[m/2m-1] [32] scheduling algorithms. As shown in the figure, the task set is not schedulable under any of these scheduling algorithms. Now consider cluster-based scheduling as follows: tasks τ_1, τ_2, and τ_3 are executed on a cluster $\mathcal{C}_1$ comprised of 2 processors under global LLF, and tasks τ_4, τ_5, and τ_6 are executed on another cluster $\mathcal{C}_2$ comprised of 2 processors us-

1068-3070/08 $25.00 © 2008 IEEE
DOI 10.1109/ECRTS.2008.28

IEEE computer society

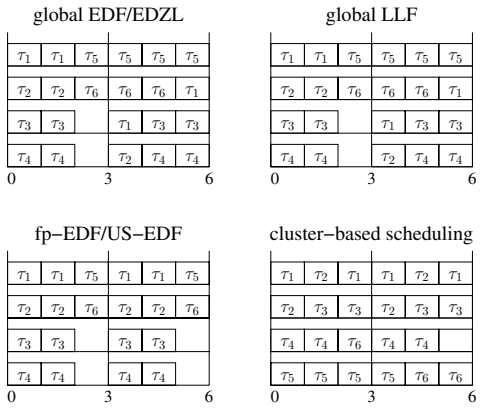


Figure 1. Motivating example

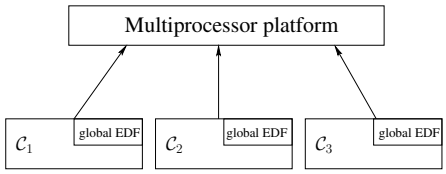


Figure 2. Example hierarchical framework

ing global EDF. 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-cluster mapping. 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 $\mathcal{C}_1$ 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 allocating a slack in the second cluster $\mathcal{C}_2$ (time interval $[5, 6]$) to the first cluster $\mathcal{C}_1$.

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 forcing only some (at most m') of the m tasks to run in parallel 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 [1].

Hierarchical scheduling. The notion of physical clustering requires intra-cluster scheduling only. This is because clusters are assigned disjoint processors, and hence tasks in different clusters cannot interrupt each others execution. However, the notion of virtual clustering inherently brings up an idea of hierarchical scheduling: inter-cluster and intra-cluster scheduling. Under inter-cluster scheduling physical processors are dynamically assigned to virtual clusters, and under intra-cluster scheduling 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 $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$, each employing global EDF scheduling strategy. If we use physical clustering, then each cluster can be separately analyzed using existing techniques for global EDF. 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 clusters $\mathcal{C}_1, \mathcal{C}_2$, and $\mathcal{C}_3$ on the multiprocessor platform. Therefore, supporting hierarchical multiprocessor scheduling is cardinal to the successful development of virtual clustering, and it is the main focus of this paper.

There have been considerable studies on hierarchical uniprocessor scheduling. Denoting a collection of tasks and a scheduler as a *component*, existing studies in uniprocessor scheduling [28, 31, 18] employed the notion of component interface, to specify resources that are required for scheduling the component's tasks. Analogously, we denote a cluster along with tasks and scheduler assigned to it as a *component* in hierarchical multiprocessor scheduling frameworks. To support inter-cluster scheduling, this paper proposes a component interface that specifies resources required for scheduling tasks in the component's cluster. Inter-cluster scheduler can then allocate processor supply to the cluster based on its component interface, and intra-cluster scheduler can then use this processor supply to schedule tasks in the cluster. Many new issues arise to adopt the notion of component interface from uniprocessor to multiprocessor scheduling frameworks. 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 concurrency constraints of tasks 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, thereby freeing the inter-cluster scheduler from such issues.

Contributions. The contribution of this paper is fourfold. First, we introduce the notion of general hierarchical multiprocessor scheduling framework to support virtual cluster-based scheduling. Second, we present an approach to specify in a component's interface, the concurrency constraints of tasks in the component. We introduce a multiprocessor resource model based interface that not only captures task-level concurrency constraints, but also specifies total resource requirements of the component. This enables the inter-cluster scheduler to schedule clusters using their interfaces only. Third, since such interfaces represent partial resource supplies, as opposed to dedicated ones, we also extend existing schedulability conditions for global EDF mul-

tiprocessor scheduling in this direction[1]. Such extensions to schedulability conditions are essential in supporting development of component interfaces. Fourth, we consider the optimization problem of developing a component interface with an aim to minimize its total resource requirement (processor utilization). We present an efficient solution to this problem on the basis of following property of our global EDF schedulability condition: 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 can be obtained when we find the smallest number of processors that can guarantee schedulability of the component.

2. Related work

Multiprocessor scheduling. In general, studies on real-time multiprocessor scheduling can fall into two categories: *partitioned* and *global* scheduling. Many partitioning algorithms [6, 25, 29] and global scheduling algorithms [13, 3, 10, 17, 5, 14, 9] have been proposed in the past. It is well known that neither partitioning nor global scheduling algorithms dominate the other, and this motivates the more general task-processor mapping that clustering proposes.

Partitioning algorithms which allow a task to execute on at most two processors have been proposed in the past by Andersson *et. al.* [2] and Kato and Yamasaki [20]. These algorithms are special instances of cluster-based scheduling, because each task can be viewed as belonging to its own virtual cluster having at most two processors. Also, these studies focus on implicit-deadline task systems and to the best of our knowledge extensions to deadline constrained systems have not been done. In another direction, global scheduling algorithms, US-EDF[$m/2m-1$] [32] and fp-EDF [4], which give special treatment to certain high utilization tasks have also been proposed. These algorithms can also be regarded as special instances of clustering. Each high utilization task can be viewed as belonging to its own physical cluster having exactly one processor, and the low utilization tasks can be assigned to a single cluster comprised of remaining processors.

Baruah and Carpenter [8] introduced an approach that can restrict the processor migration of jobs, in order to alleviate the inflexibility of partitioned scheduling and the processor migration overhead of global scheduling. Their approach can be categorized as job-level partitioned scheduling. They showed that task-level and job-level partitioned scheduling approaches do not dominate each other. This algorithm is also a special instance of virtual clustering, where each task belongs to its own virtual cluster having one processor, and cluster-processor mapping can change with each job.

[1]Due to lack of space, we have chosen to focus on one scheduling algorithm in this paper. However, the issues are same for other schedulers, and hence techniques developed here are applicable to other schedulers as well.

Moir and Ramamurthy [26] and Anderson *et. al.* [1] presented an approach that can upper bound the amount of concurrent execution within a group of tasks. They developed their approach using a hierarchical scheduling framework under Pfair scheduling. These approaches are most related to our work, 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-cluster and inter-cluster scheduling; it allows to develop schedulability analysis techniques for virtual clustering that are easily extensible to many different scheduling algorithms. 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 ones considered in this paper. We believe this flexibility provides a powerful tool for the development of various task-cluster mapping and scheduling algorithms.

Calandrino *et. al.* [12] presented a study with a focus on determining an appropriate size for physical clusters, while our study focuses on virtual clustering.

Hierarchical scheduling. For uniprocessor platforms, there has been a growing attention to hierarchical scheduling frameworks. Since a two-level framework was introduced [16], its schedulability has been analyzed under fixed-priority [21] and EDF-based [23] scheduling. For multi-level frameworks, many resource model based component interfaces such as bounded-delay [28], periodic [22, 31], and EDP [18] have been introduced, and schedulability conditions have been derived under fixed-priority and EDF scheduling [19, 22, 31, 15, 18]. 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.

3. 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.

3.1. Task and platform models

Task model. We assume a deadline constrained sporadic task model [7]. In this model, a sporadic task τ_i is specified as (T_i, C_i, D_i), where T_i is the minimum separation, C_i is the worst-case execution time requirement, and D_i is the relative deadline. These task parameters satisfy the property $C_i \leq D_i \leq T_i$. Successive instances of τ_i are released with a minimum separation of T_i time units. We refer to each instance of a sporadic task as a *job*. Each job of τ_i must receive C_i units of processor capacity within D_i time units. We also assume that these C_i units must be supplied non-concurrently to the job.

Multiprocessor platform and scheduling strategy. In this paper we assume an identical, unit-capacity multiproces-

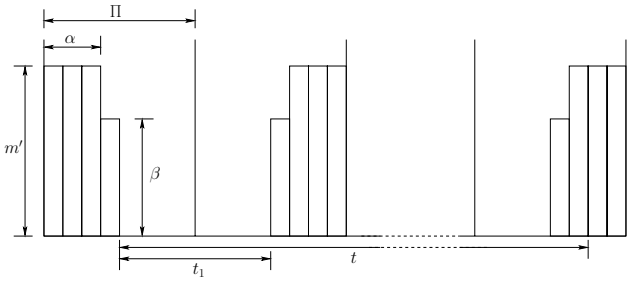


Figure 3. Schedule of Γ w.r.t $\mathrm{sbf}_\Gamma(t)$

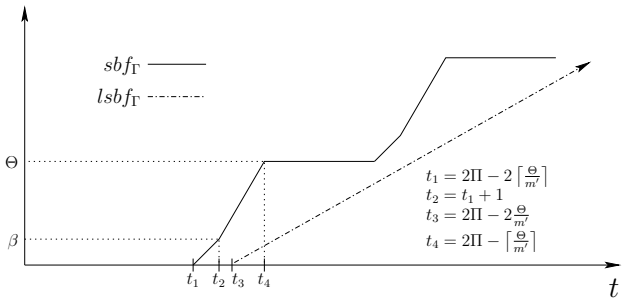


Figure 4. sbf_Γ and lsbf_Γ

sor platform. Each processor in this platform has a schedulable utilization of one. 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 [3, 10, 5]. We assume such a global scheduling strategy within each cluster, and in particular, we assume that tasks in clusters are scheduled using global EDF.

3.2. Multiprocessor resource model

A *resource model* is a model for specifying the characteristics of a resource supply. When these models represent component interfaces, they specify resource requirements of the component. Periodic [31], EDP [18], bounded delay [19], etc, are examples of resource models that have been extensively used for analysis of hierarchical uniprocessor frameworks. These resource models can also be used as component interfaces in hierarchical multiprocessor frameworks. One way to achieve this is to consider m' identical resource models as a component interface, where m' is number of processors allocated to the component's cluster. However, this interface is restrictive because each processor contributes same amount of resource as any other processor in the cluster, to satisfy the component's resource requirements. It is desirable to be more flexible, in that interfaces should be able to represent the collective processor requirement of components, without fixing the contribution of each processor a priori. Apart from increased flexibility, such interfaces can also improve processor utilization.

We now introduce *multiprocessor resource* models that specify the characteristics of resource supply provided to a cluster by an identical unit-capacity multiprocessor platform. These resource models do not fix the contribution of each processor in the cluster a priori, and hence are suitable candidates for component interfaces. A *multiprocessor periodic resource* (MPR) model $\Gamma = \langle \Pi, \Theta, m' \rangle$ specifies that a unit-capacity, identical multiprocessor platform collectively provides Θ units of resource in every Π time units to a cluster comprising of m' processors, *i.e.*, Θ resource units are provided with concurrency at most m'. It is then easy to see 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 resource supply that the model provides in a given time interval. Specifically, $\mathrm{sbf}_R(t)$ is equal to the minimum amount of resource that model R provides in any time interval of duration t. In uniprocessor frameworks, sbf is used in schedulability conditions to generate resource model based component interfaces. Extending this approach to hierarchical multiprocessor frameworks, 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 $\Gamma = \langle \Pi, \Theta, m' \rangle$, where we let $\alpha = \lfloor \frac{\Theta}{m'} \rfloor$ and $\beta = \Theta - m'\alpha$. The schedule for Γ that generates this minimum supply in a time interval of duration t is as shown in Figure 3. As can be seen, length of the largest time interval with no supply (duration t_1 in the figure) is equal to $2\Pi - 2\left\lceil \frac{\Theta}{m'} \right\rceil$. Thereafter, Γ is guaranteed to provide Θ units of resource in every Π time units. sbf_Γ is then given by the following equation where $k = \left\lfloor \frac{t - \left(\Pi - \left\lceil \frac{\Theta}{m'} \right\rceil\right)}{\Pi} \right\rfloor$ and $I = t - 2\Pi + \left\lceil \frac{\Theta}{m'} \right\rceil$. This function is also plotted in Figure 4.

$$\mathrm{sbf}_\Gamma(t) = \begin{cases} k\Theta + \max\left\{0, \left[I - k\Pi\right] m' + \Theta\right\} & t \geq \Pi - \left\lceil \frac{\Theta}{m'} \right\rceil \\ 0 & \text{Otherwise} \end{cases} \tag{1}$$

Note by setting $m' = 1$ in Equation (1), we get the sbf of periodic resource model $\langle \Pi, \Theta \rangle$ [31], indicating that MPR models generalize periodic resource models. In uniprocessor frameworks, although schedulability conditions with sbf have been derived, a linear approximation (lower bound) of sbf is often used to improve the efficiency of interface generation process. Hence, in anticipation, we present following linear lower bound for the sbf of MPR model Γ. This function is also plotted in Figure 4.

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

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 guar-

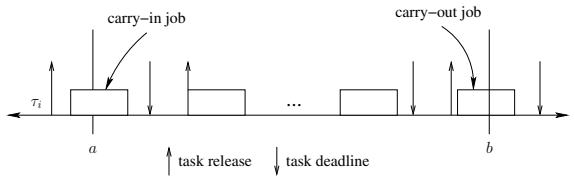


Figure 5. Workload of task τ_i in $[a, b]$

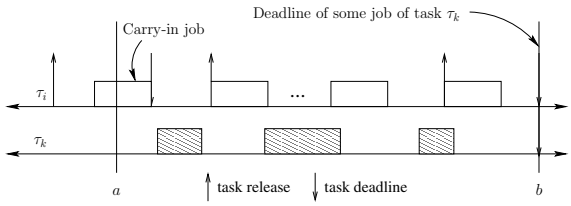


Figure 6. Dispatch pattern for $\mathcal{W}_i(b-a)$

antees a minimum total resource 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.

4. Component schedulability condition

In this section, we develop a schedulability condition for components in hierarchical multiprocessor frameworks, such that this condition accommodates the notion of a partitioned resource supply. Specifically, we extend existing global EDF 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 $\mathcal{C}$ and n sporadic tasks $\tau = \{\tau_1 = (T_1, C_1, D_1), \ldots, \tau_n = (T_n, C_n, D_n)\}$ scheduled under global EDF. To keep the presentation simple, we use notation $\mathcal{C}$ to refer to the component as well. We now develop a schedulability condition for $\mathcal{C}$ assuming it is scheduled using MPR model $\Gamma = \langle \Pi, \Theta, m' \rangle$, where m' denotes number of processors in the cluster. This condition uses the real-time processor demand of task set τ in $\mathcal{C}$. Existing work [10] has developed an upper bound for this demand which we can use. Only upper bounds are known for this demand, because no notion of worst-case arrival sequence (analogous to the synchronous arrival sequence in uniprocessors) is known for multiprocessors [5]. Hence, we first summarize this existing upper bound for component demand 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 τ is scheduled under $\mathcal{C}$'s 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 requirement 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 requirement until b.

Workload bound for τ_i under global EDF. 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 component $\mathcal{C}$ in all intervals. However, since no such technique is known (apart from task set simulation), we use an upper bound for this workload that has been obtained by Bertogna *et. al.* [10]. 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 task τ_k misses its deadline. In the schedulability condition which 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 instant b are assumed to be released as late as possible. Also, the job of τ_i that is released before time instant 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 $\mathcal{W}_i(t)$ denote this workload bound for τ_i in a time interval of length $t(= b - a)$. Also, let $\mathcal{CI}_i(t)$ denote the carry-in demand generated by the execution pattern shown in Figure 6. Then,

$$\mathcal{W}_i(t) = N_i(t)C_i + \mathcal{CI}_i(t), \tag{3}$$

where $N_i(t) = \left\lfloor \frac{t+(T_i-D_i)}{T_i} \right\rfloor$ and $\mathcal{CI}_i(t) = \min\{C_i, \max\{0, t - N_i(t)T_i\}\}$. In the following section we develop a schedulability condition for $\mathcal{C}$ using this workload bound.

4.2. Schedulability condition

We now present a schedulability condition for component $\mathcal{C}$ when it is scheduled using MPR model $\Gamma = \langle \Pi, \Theta, m' \rangle$. For this purpose, we extend an existing condition that checks schedulability of $\mathcal{C}$ on a dedicated resource [5], with the notion of partitioned resource Γ.

When task τ_k is scheduled on a dedicated resource comprised of m' unit-capacity processors, Baruah [5] identifies the values for time instants a and b (discussed in the previous section) that must be checked for schedulability of τ_k. In particular, he lets b correspond to the deadline of some job of task τ_k (henceforth denoted as job τ_k^b), assumes τ_k^b misses its deadline, and then specifies different values of a that need to be considered. Figure 7 gives one such time instant a considered by Baruah. It corresponds to a point in

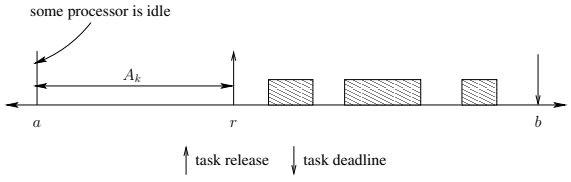


Figure 7. Example time instant a under dedicated resource

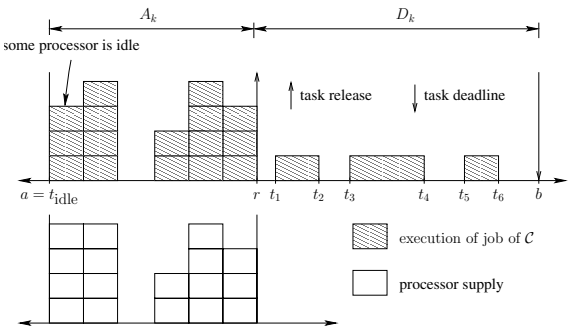


Figure 8. Example time instant t_{idle}

time such that at least one of the m' processors is idle at that time instant. Furthermore, this time instant must be prior to the release of job τ_k^b (r in the figure). Also, no processor can be 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, because at most $m' - 1$ processors are executing jobs at a. Baruah uses this observation to come up with a schedulability condition in the dedicated resource case. Intuitively, this technique aims to derive a condition on the total higher priority workload in interval $[a, b]$ that guarantees a deadline miss for τ_k^b. In the following discussion we extend this notion of time instant a to the case when τ_k is scheduled using MPR model Γ, and then present our schedulability condition.

When task τ_k is scheduled using MPR model Γ, 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 Γ. Figure 8 illustrates one such time instant. Then, to check schedulability of task τ_k we consider all time instants a such that: (1) a is t_{idle}, (2) $a \leq r$ where r denotes the release time of job τ_k^b, and (3) no time instant in the interval $(a, r]$ is t_{idle}. The time instant illustrated in Figure 8 also satisfies these properties.

To derive schedulability condition for component $\mathcal{C}$, we consider all intervals $[a, b]$ as explained above for each task $\tau_k \in \tau$, and derive condition under which a deadline miss occurs for job τ_k^b. If τ_k^b misses its deadline, then it must be true that the total workload in interval $[a, b]$ of jobs of all tasks with priority at least τ_k^b is strictly greater than the total processor supply available to $\mathcal{C}$ in that interval. Let for each i, I_i denote the total workload in interval $[a, b]$ of all jobs of task τ_i that have priority at least τ_k^b. Then, since $\text{sbf}_\Gamma(b - a)$ denotes a lower bound on the processor supply available to $\mathcal{C}$ in interval $[a, b]$, whenever job τ_k^b misses its deadline it must be true that

$$\sum_{i=1}^{n} I_i > \text{sbf}_\Gamma(b - a).$$

Let A_k denote the length of interval $[a, r]$. Then, as shown in Figure 8, $A_k + D_k$ denotes the total length of interval $[a, b]$ and whenever job τ_k^b misses its deadline it must be true that

$$\sum_{i=1}^{n} I_i > \text{sbf}_\Gamma(A_k + D_k). \tag{4}$$

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

We now derive an upper bound for each workload I_i. We separately consider the workload of task τ_i in the following two interval classes: (1) time intervals in $[a, b]$ in which job 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 task 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 $I_{i,2}$. Under dedicated resource, Baruah [5] only considered intervals of type (2) when deriving the schedulability condition. However, sbf of MPR resource models are only defined over contiguous time intervals. Hence, under partitioned resource we are required to consider the contiguous time interval $A_k + D_k$ obtained from the union of intervals of types (1) and (2).

Since the cumulative length of intervals of type (1) is C_k and there are m' processors on which $\mathcal{C}$ executes, the total workload of all tasks in intervals of type (1) is clearly upper bounded by $m'C_k$, i.e., $\sum_{i=1}^{n} I_{i,1} \leq m'C_k$. To bound $I_{i,2}$ we use the workload upper bound $\mathcal{W}_i$ presented in Section 4.1. Observe that $\mathcal{W}_i(b - a)$ $(= \mathcal{W}_i(A_k + D_k))$ upper bounds the workload of all jobs of τ_i that execute in interval $[a, b]$ and have priority higher than τ_k^b. Therefore, $\mathcal{W}_i(A_k + D_k)$ also upper bounds $I_{i,2}$. Furthermore, since the total cumulative length of intervals of type (2) is $A_k + D_k - C_k$, no $I_{i,2}$ can be larger than this length. Also, when $i = k$, this bound can be further tightened because in $I_{k,2}$ we do not consider the execution units of job τ_k^b (these execution units are considered for intervals of type (1)). Thus, we can subtract C_k from $\mathcal{W}_k(A_k + D_k)$ and further $I_{k,2}$ cannot be greater than A_k.

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

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 component $\mathcal{C}$ at time instant a, even though that processor is available for $\mathcal{C}$ as per supply Γ. Hence we only need to consider $m'-1$ largest values of $\mathcal{CI}_i$ when computing an upper bound for $\sum_{i=1}^{n} I_{i,2}$ using above equations, where $\mathcal{CI}_i$ denotes the carry-in demand in $\mathcal{W}_i$. The following theorem then gives our schedulability condition and its proof follows from the above discussion.

Theorem 1 *Let us define*

$$\hat{I}_{i,2} = \min\{\mathcal{W}_i(A_k + D_k) - \mathcal{CI}_i(A_k + D_k), A_k + D_k - C_k\}$$
for all $i \neq k$,

$$\hat{I}_{k,2} = \min\{\mathcal{W}_k(A_k + D_k) - C_k - \mathcal{CI}_k(A_k + D_k), A_k\}, \text{ and}$$

$$\mathrm{dem}(A_k + D_k, m') = \sum_{i=1}^{n} \hat{I}_{i,2} + \sum_{\substack{(m'-1) \\ \text{largest}}} (\bar{I}_{i,2} - \hat{I}_{i,2}) + m'C_k.$$

A component comprising of cluster $\mathcal{C}$ with m' processors and n sporadic tasks $\tau = \{\tau_1 = (T_1, C_1, D_1), \ldots, \tau_n = (T_n, C_n, D_n)\}$ is schedulable under global EDF using MPR model $\Gamma = \langle \Pi, \Theta, m' \rangle$ if for all tasks $\tau_k \in \tau$ and all $A_k \geq 0$,

$$\mathrm{dem}(A_k + D_k, m') \leq \mathrm{sbf}_\Gamma(A_k + D_k). \tag{5}$$

In Theorem 1, if we set $\Theta = m'\Pi$ then we get the schedulability condition for dedicated resource proposed by Baruah [5]. This shows that our schedulability condition is no more pessimistic than the one proposed by Baruah. Although this theorem gives a schedulability test for component $\mathcal{C}$, 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[2].

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

$$A_k < \frac{C_\Sigma + m'C_k - D_k\left(\frac{\Theta}{\Pi} - U(\tau)\right) + U + B}{\frac{\Theta}{\Pi} - U(\tau)},$$

where C_Σ denotes the sum of $m'-1$ largest C_i's, $U(\tau) = \sum_{i=1}^{n} \frac{C_i}{T_i}$, $U = \sum_{i=1}^{n}(T_i - D_i)\frac{C_i}{T_i}$ and $B = \Theta\left(2 - \frac{2\Theta}{m'\Pi}\right)$.

It is also easy to show 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_Γ change. Therefore, Theorem 1 gives a pseudo-polynomial time schedulability condition whenever utilization ($U(\tau)$) of task set τ is strictly less than utilization ($\frac{\Theta}{\Pi}$) of MPR model Γ.

Discussion. Due to space limit, we only focus on one intra-cluster scheduling algorithm, global EDF. However, our analysis technique can be easily extended to other intra-cluster scheduling algorithms. Specifically, in the schedulability condition given in Equation (5), $\mathrm{dem}(A_k + D_k, m')$ depends on global EDF, and $\mathrm{sbf}_\Gamma(A_k + D_k)$ depends on MPR model Γ. Suppose there exists a function $\mathrm{dem}_{\mathsf{DM}}(A_k + D_k, m')$ that can compute an upper bound for resource demand of a task set under global DM scheduling. Then we can plug in $\mathrm{dem}_{\mathsf{DM}}(A_k + D_k, m')$ into Equation (5) to derive a schedulability condition for global DM intra-cluster scheduling. In fact, such $\mathrm{dem}_{\mathsf{DM}}(A_k + D_k, m')$ can be obtained by extending current results over dedicated processors [11].

Bertogna and Cirinei [9] have derived an upper bound for the worst-case response time of tasks scheduled under global EDF or global DM. They have also used this bound to improve the carry-in demand $\mathcal{CI}_i$ that we use in our schedulability condition. However, this improvement to the carry-in demand cannot be directly applied in our case. Since we schedule tasks using MPR resource model, any response time computation depends on the resource supply as well, in addition to task demand. Then, to use the response time bounds presented in [9], we must extend it with sbf of MPR resource model. However, since we are generating the MPR resource model, its sbf is unknown and hence the response time is not computable. One way to resolve this issue is to compute the resource model using binary search for Θ. However, since Θ is a real number, this solution can have prohibitively long running time.

5. Component interface generation

In this section we develop a technique to generate MPR interface $\Gamma = \langle \Pi, \Theta, m' \rangle$, for a cluster $\mathcal{C}$ comprising of n sporadic tasks $\tau = \{\tau_1 = (T_1, C_1, D_1), \ldots, \tau_n = (T_n, C_n, D_n)\}$ scheduled under global EDF. For this purpose, we use the schedulability condition given by Theorem 1. We assume that period Π of interface Γ is specified a priori by system designer. For instance, one can specify this period taking into account preemption overheads. We then compute values for capacity Θ and number of processors m' in cluster $\mathcal{C}$, such that processor utilization of the interface is minimized. Finally, we also develop a technique that transforms MPR interfaces to tasks in order to support inter-cluster scheduling, *i.e.*, to schedule clusters on the multiprocessor platform.

5.1. Minimum utilization interface

For MPR model Γ its utilization is defined as $\frac{\Theta}{\Pi}$. It is desirable to minimize this quantity when generating component interfaces, because components then consume minimum amount of resource from the multiprocessor platform. We now give a lemma which states that interface utilization required to guarantee schedulability of task set τ monotonically increases as number of processors (m') in the cluster increases.

Lemma 3 *Consider two interfaces $\Gamma_1 = \langle \Pi_1, \Theta_1, m_1 \rangle$ and $\Gamma_2 = \langle \Pi_2, \Theta_2, m_2 \rangle$ such that $\Pi_1 = \Pi_2$ and $m_1 < m_2$.*

[2]Due to space limit, we do not provide proofs of theorems and lemmas in this paper. Please refer to our technical report [30] for the proofs.

Suppose these two interfaces guarantee schedulability of the same component $\mathcal{C}$ with their smallest possible interface utilization, respectively. Then, Γ_2 has a higher processor utilization than Γ_1 does, i.e., $\Theta_1 < \Theta_2$.

Lemma 3 suggests that when we generate interface Γ for $\mathcal{C}$, we use the smallest number of processors ($m' = m^*$) in order to minimize interface utilization of Γ. However, an arbitrarily small number for m' (say $m' = 1$) may result in an *infeasible* Γ. Recall that a MPR model $\Gamma = \langle \Pi, \Theta, m' \rangle$ is defined to be feasible if and only if $\Theta \leq m'\Pi$. Therefore, we find a feasible interface Γ for $\mathcal{C}$ that (1) guarantees schedulability of $\mathcal{C}$ based on Theorem 1, and (2) uses the smallest number of processors (m^*). We can find such m^* through search. Since interface utilization is monotonic with number of processors, a binary search can be performed to determine m^*. For this search to terminate, both a lower and upper bound on m^* should be known. $\lceil U(\tau) \rceil$ is clearly a lower bound on the number of processors necessary to schedule $\mathcal{C}$, where $U(\tau)$ denotes total utilization of task set τ. Therefore, $m^* \geq \lceil U(\tau) \rceil$. If the number of processors in 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 parameters of task set τ.

Lemma 4 *If $m' \geq \frac{\sum_{i=1}^{n} C_i}{\min_{i=1}^{n}\{D_i - C_i\}} + n$, then feasible MPR model $\Gamma = \langle \Pi, m'\Pi, m' \rangle$ is guaranteed to schedule $\mathcal{C}$.*

Since Γ in Lemma 4 is feasible and guarantees schedulability of $\mathcal{C}$, $\frac{\sum_{i=1}^{n} C_i}{\min_{i=1}^{n}\{D_i - C_i\}} + n$ is an upper bound for m^*. Thus, we generate an interface for $\mathcal{C}$ by doing a binary search for m^* in the range $[\lceil U(\tau) \rceil, \frac{\sum_{i=1}^{n} C_i}{\min_{i=1}^{n}\{D_i - C_i\}} + n]$. For each value of number of processors m', we compute the smallest value of Θ that satisfies Equation (5) in Theorem 1. However, Θ appears inside floor and ceiling functions in sbf_Γ, and hence these computations may be intractable. Therefore, we replace sbf_Γ in this equation with lsbf_Γ given in Equation (2). Θ (Θ^*) corresponding to the smallest value of m' (m^*) that guarantees schedulability of $\mathcal{C}$ and results in a feasible interface is then chosen as the capacity of Γ. Also, m^* is chosen as the value for number of processors in the cluster, *i.e.*, $\Gamma = \langle \Pi, \Theta^*, m^* \rangle$.

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 $m' > U(\tau)$, 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 $m' = \lceil U(\tau) \rceil = U(\tau)$. However, in this case it can be shown that $\Gamma = \langle \Pi, m'\Pi, m' \rangle$ can schedule $\mathcal{C}$ if and only if $m' = 1$ and all tasks in τ have implicit deadlines. Therefore, computing the interface for this value of m' can be done in constant time. The number of different values of m' to be considered (*i.e.*, in binary search for m^*) is polynomial in input size, because the search interval is bounded by numbers that are polynomial in input parameters. Therefore, the entire interface generation process has pseudo-polynomial complexity.

Example 1 *Consider the example hierarchical scheduling 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 Γ_1^*, Γ_2^*, and Γ_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 utilization of these interfaces for varying period values, and m' denotes number of processors in the cluster.*

Figures 9(a) and 9(c) show that when $m' = 1$, interfaces Γ_1^* and Γ_3^* are not feasible, since their interface utilizations are greater than 1 for all period values. However, when $m' = 2$, Γ_1^* and Γ_3^* are feasible, *i.e.*, their respective utilizations are at most two. Therefore, for clusters $\mathcal{C}_1$ and $\mathcal{C}_3$ we choose MPR interfaces Γ_1^* and Γ_3^* with $m' = 2$. Similarly, Figure 9(b) shows that Γ_2^* is a feasible interface for cluster $\mathcal{C}_2$ when $m' = 1$. These plots also show that the overhead incurred by our interfaces is small for the non-trivial examples presented here.

5.2. Inter-cluster scheduling

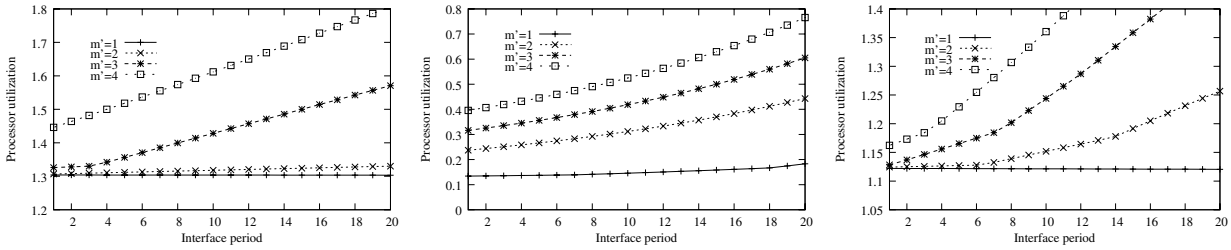
As discussed in the introduction, virtual clustering involves two-level scheduling; scheduling of tasks within each cluster (intra-cluster scheduling), and scheduling of clusters on the multiprocessor platform (inter-cluster scheduling). MPR interfaces generated in the previous section capture task-level concurrency constraints within a cluster. Hence inter-cluster scheduling need not worry about these constraints when it schedules clusters using their interfaces. However, there is no known scheduling algorithm for MPR interfaces. Therefore, we now develop a technique to transform a MPR interface into periodic tasks, such that resource requirements of these tasks are at least as much as those of the interface. Note that a periodic task is similar to a sporadic task, except that T in the tuple (T, C, D) now denotes exact separation between release times of successive jobs, instead of minimum separation.

Definition 1 *Given a MPR interface $\Gamma = \langle \Pi, \Theta^*, m^* \rangle$, let $\psi = \Theta^* - m^* \left\lfloor \frac{\Theta^*}{m^*} \right\rfloor$ and $k = \lfloor \psi \rfloor$. Then, we transform Γ into the periodic task set $\tau_\Gamma = \tau_1, \ldots, \tau_{m^*}$, where $\tau_1 = \ldots = \tau_k = \left(\Pi, \left\lfloor \frac{\Theta^*}{m^*} \right\rfloor + 1, \Pi\right)$, $\tau_{k+1} = \left(\Pi, \left\lfloor \frac{\Theta^*}{m^*} \right\rfloor + \psi - k \left\lfloor \frac{\psi}{k} \right\rfloor, \Pi\right)$, 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 resource demand of task set τ_Γ is Θ^*. Further, we have assumed that whenever Θ^* is not an integer, the resource supply from Γ

Cluster	Task set	$\sum_i \frac{C_i}{T_i}$	$\sum_i \frac{C_i}{D_i}$
$\mathcal{C}_1$	$\{(60,5,60),(60,5,60),(60,5,60),(60,5,60),(70,5,70),(70,5,70),(80,5,80),(80,5,80),$ $(80,10,80),(90,5,90),(90,10,90),(90,10,90),(100,10,100),(100,10,100),(100,10,100)\}$	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 $\mathcal{C}_3$



(a) Γ_1^* (b) Γ_2^* (c) Γ_3^*

Figure 9. MPR model based interfaces

fully utilizes one processor before moving to the next processor. For example, if $\Theta^* = 2.5$ and $m^* = 3$, then Γ will provide two units of resource from two processors and the remaining 0.5 units of resource from the third processor. Thus, MPR interfaces generated in the previous section can be transformed into periodic tasks using Definition 1. Once such tasks are generated for each virtual cluster, inter-cluster scheduling can be done using existing multiprocessor scheduling algorithms like global EDF, Pfair, etc. To prove correctness of this transformation, we must show that the following two properties hold for task set τ_Γ: (1) the amount of processor supply necessary to meet all the deadlines of tasks in τ_Γ is at least as much as sbf_Γ, and (2) the amount of concurrency in the processor supply used by τ_Γ is at most m^*, *i.e.*, at any time instant at most m^* units of processor supply is used by τ_Γ. These properties follow from the facts that (1) total amount of processor supply used by τ_Γ is Θ^* in every successive Π time units, and (2) amount of concurrency achievable is at most m^* because τ_Γ has exactly m^* tasks. By definition, $\mathrm{sbf}_\Gamma(t)$ represents the minimum resource supply during t when a processor supply of Θ^* is provided in every Π time units, with amount of concurrency at most m^*.

Example 2 *For MPR interfaces Γ_1^*, Γ_2^*, and Γ_3^* generated in Example 1, we select period values $6, 8$, and 5, respectively. The corresponding MPR interfaces are $\langle 6, 8.22, 2\rangle, \langle 8, 2.34, 1\rangle$, and $\langle 5, 5.83, 2\rangle$, respectively. Using Definition 1 we get the periodic task sets $\tau_{\Gamma_1^*} = \{(6,5,6),(6,4,6)\}, \tau_{\Gamma_2^*} = \{(8,3,8)\}$, and $\tau_{\Gamma_3^*} = \{(5,3,5),(5,3,5)\}$. Suppose the three clusters $\mathcal{C}_1, \mathcal{C}_2$, and $\mathcal{C}_3$ (*i.e.*, task set $\{\tau_{\Gamma_1^*}, \tau_{\Gamma_2^*}, \tau_{\Gamma_3^*}\}$) are scheduled on a multiprocessor platform using global EDF. Then the resulting MPR interface Γ^* is plotted in Figure 10.*

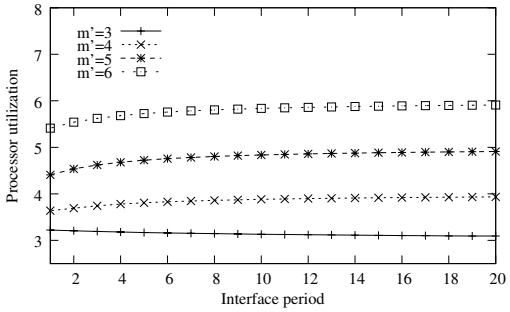


Figure 10. MPR interface Γ^*

As can be seen from Figure 10, the three clusters in Example 2 are schedulable on a multiprocessor platform having 4 processors (Γ^* is feasible when $m' = 4$). This example clearly shows the advantage of virtual clustering over physical clustering. The three components would require 5 processors under physical clustering (2 for $\mathcal{C}_1$ and $\mathcal{C}_3$ each, and 1 for $\mathcal{C}_2$). On the other hand, a global EDF 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) global EDF is not an optimal scheduling algorithm on multiprocessor platforms, (2) the schedulability conditions we use are only sufficient conditions, and (3) capturing internal concurrency constraints of a component in its interface leads to some increase in resource requirements.

6. Conclusions

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 abstracted into MPR interfaces, and 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.

In this paper, we only focused on global EDF for both intra-cluster and inter-cluster scheduling. However, our approach of isolating the inter-cluster scheduler from concurrency constraints of tasks within clusters 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. We plan to generalize our framework by including other intra-cluster and inter-cluster scheduling algorithms.

Orthogonally, although we have focused on MPR model based component interfaces, we can also consider a more general explicit deadline resource model for multiprocessors, similar to the EDP model for uniprocessor platforms [18]. Techniques developed in this paper can be extended to support component interfaces based on such models.

Splitting a given task set into processor clusters is a fundamental problem that must be addressed if cluster-based scheduling has to succeed. It is essential to develop clustering algorithms that result in good utilization and load bounds, and we are currently working in this direction.

References

[1] J. H. Anderson, J. M. Calandrino, and U. C. Devi. Real-time scheduling on multicore platforms. In *RTAS*, 2006.

[2] B. Andersson and E. Tovar. Multiprocessor scheduling with few preemptions. In *RTCSA*, 2006.

[3] T. P. Baker. Multiprocessor edf and deadline monotonic schedulability analysis. In *RTSS*, 2003.

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

[5] S. Baruah. Techniques for multiprocessor global schedulability analysis. In *RTSS*, 2007.

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

[7] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proc. of IEEE Real-Time Systems Symposium*, 1990.

[8] S. K. Baruah and J. Carpenter. Multiprocessor fixed-priority scheduling with restricted interprocessor migrations. In *ECRTS*, 2003.

[9] M. Bertogna and M. Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *Proc. of IEEE Real-Time Systems Symposium*, 2007.

[10] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of edf on multiprocessor platforms. In *ECRTS*, 2005.

[11] M. Bertogna, M. Cirinei, and G. Lipari. New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. In *OPODIS*, 2005.

[12] J. M. Calandrino, J. H. Anderson, and D. P. Baumberger. A hybrid real-time scheduling approach for large-scale multicore platforms. In *ECRTS*, 2007.

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

[14] M. Cirinei and T. P. Baker. Edzl scheduling analysis. In *ECRTS*, pages 9–18, 2007.

[15] R. I. Davis and A. Burns. Hierarchical fixed priority preemptive scheduling. In *RTSS*, 2005.

[16] Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. In *RTSS*, December 1997.

[17] U. C. Devi and J. H. Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. In *RTSS*, 2005.

[18] A. Easwaran, M. Anand, and I. Lee. Optimal compositional analysis using explicit deadline periodic resource models. In *RTSS*, 2007.

[19] X. A. Feng and A. K. Mok. A model of hierarchical real-time virtual resources. In *RTSS*, 2002.

[20] S. Kato and N. Yamasaki. Real-time scheduling with task splitting on multiprocessors. In *RTCSA*, 2007.

[21] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. In *RTSS*, 1999.

[22] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *ECRTS*, July 2003.

[23] G. Lipari, J. Carpenter, and S. Baruah. A framework for achieving inter-application isolation in multiprogrammed hard-real-time environments. In *RTSS*, 2000.

[24] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46 – 61, 1973.

[25] J. M. López, J. L. Díaz, and D. F. García. Minimum and maximum utilization bounds for multiprocessor rm scheduling. In *ECRTS*, 2001.

[26] M. Moir and S. Ramamurthy. Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In *RTSS*, 1999.

[27] A. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. Technical report, Ph.D. dissetation, MIT, 1983.

[28] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *RTAS*, 2001.

[29] D.-I. Oh and T. P. Baker. Utilization bounds for n-processor rate monotonescheduling with static processor assignment. *Real-Time Syst.*, 15(2):183–192, 1998.

[30] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. Technical report, 2008. Available at http://www.seas.upenn.edu/∼arvinde/ecrts08-tr.pdf.

[31] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *RTSS*, 2003.

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