2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications

# The Multi Supply Function Abstraction for Multiprocessors

Enrico Bini, Giorgio Buttazzo, Marko Bertogna
Scuola Superiore Sant'Anna
Pisa, Italy
Email: {e.bini,g.buttazzo,m.bertogna}@sssup.it

***Abstract*—Multi-core platforms are becoming the dominant computing architecture for next generation embedded systems. Nevertheless, designing, programming, and analyzing such systems is not easy and a solid methodology is still missing.**

**In this paper, we propose two powerful abstractions to model the computing power of a parallel machine, which provide a general interface for developing and analyzing real-time applications in isolation, independently of the physical platform. The proposed abstractions can be applied on top of different types of service mechanisms, such as periodic servers, static partitions, and P-fair time partitions. In addition, we developed the schedulability analysis of a set of real-time tasks on top of a parallel machine that is compliant with the proposed abstractions.**

## I. Introduction

Multi-core architectures represent the next generation of computing devices for providing an efficient solution to the problem of increasing the processing speed with contained power dissipation. In fact, increasing the operating frequency of a single processor would cause serious heating problems and considerable power consumption [13]. Programming multi-core systems, however, is not trivial, and the research community is working to produce new theoretical results or extend the well established theory for uniprocessor systems developed in the last 30 years. The core of the difficulties in multiprocessor scheduling can be synthesized as follows: *two unit-speed processors provide less computational resource than one double-speed processor*.

One of the most useful concepts developed in the last years that needs to be extended to multiprocessors is the *Resource Reservation* paradigm [21], [1], according to which the capacity of a processor can be partitioned into a set of reservations, each equivalent to a virtual processor (VP) that provides a fraction of the available computing power. A reservation is often modelled by a pair $(Q_i, P_i)$ indicating that $Q_i$ units of time are available every period $P_i$, meaning that the virtual processor has an equivalent bandwidth $\alpha_i = Q_i/P_i$. The main advantage of this approach is for soft real-time applications with highly variable computational requirements, for which a worst-case guarantee would cause a waste of resources and degrade system efficiency. In fact, when the worst case is rare, a more optimistic reservation increases resource usage while protecting other tasks from being delayed by sporadic overruns [9]. Such a property is referred to as *temporal protection* (also called *temporal isolation* or *bandwidth isolation*).

Temporal protection has the following advantages: (i) it prevents an overrun occurring in a task to affect the temporal behavior of the other tasks, and (ii) it allows to guarantee an application allocated to a virtual machine in "isolation" (that is, independently of the other applications in the system) only based on its timing requirements and on the amount of allocated resource.

Below we discuss some works related to our approach.

### A. Related works

One of the first papers addressing resource reservations was published in 1993 by Parekh and Gallager [24], who introduced the Generalized Processor Sharing (GPS) algorithm to share a fluid resource according to a set of weights. Mercer et al. [21] proposed a more realistic approach where a resource can be allocated based on a required budget and period. Stoica et al. [28] introduced the Earliest Eligible Virtual Deadline First (EEVDF) for sharing the computing resource. Deng and Liu [10] achieved the same goal by introducing a two-level scheduler (using EDF as a global scheduler) in the context of multi-application systems. Kuo and Li [16] extended the approach to a Fixed Priority global scheduler. Kuo et al. [17] extended their previous work [16] to multiprocessors. However they made very stringent assumptions (such as no task migration and period harmonicity) that restricted the applicability of the proposed solution.

Moir and Ramamurthy [22] proposed a hierarchical approach, where a set of P-fair tasks can be scheduled within a time partition provided by another P-fair task (called "supertask") acting as a server. However, the solution often requires the weight of the supertask to be higher than the sum of the weights of the served tasks [15].

Many independent works proposed to model the service provided by a uniprocessor through a supply function. Mok, Feng, and Chen introduced the bounded-delay resource partition model [12]. Almeida et al. [2] provided timing guarantees for both synchronous and asynchronous traffic over the FTT-CAN protocol by using hierarchical scheduling. Lipari and Bini [20] derived the set of virtual processors that can feasibly schedule a given application. Shin and Lee [26] introduced the periodic resource model also deriving a utilization bound.

This work has been partially supported by the ACTORS European project under contract 216586.

1533-2306/09 $25.00 © 2009 IEEE
DOI 10.1109/RTCSA.2009.39

Easwaran et al. [11] extended this model allowing the server deadline to be different than the period.

The research on global EDF algorithms has been also very active. Funk, Goossens, and Baruah [14] derived the EDF analysis on uniform multiprocessors, later extended by Baruah and Goossens [5] to the constrained deadline model. Baker [4] proposed a method for estimating the maximum possible interference for each task. Bertogna et al. [7] proposed a very efficient test for multiprocessor systems under both EDF and fixed priority scheduling.

Shin et al. [25] proposed a multiprocessor periodic resource model to describe the computational power supplied by a parallel machine. They model a virtual multiprocessor by the triplet $\langle \Pi, \Theta, m' \rangle$ meaning that an overall budget $\Theta$ is provided by $m'$ processors every period $\Pi$. The big advantage of this interface is that it is simple and captures the most significant features of the platform. Nonetheless, it has two main drawbacks. First, the same periodicity $\Pi$ is provided to all the tasks scheduled on the same virtual multiprocessor. This can lead to a quite pessimistic interface design. In fact, the period of the interface is typically constrained by the task with the shortest period. However, tasks with longer period could be scheduled by a server with larger period, saving runtime overhead. Hence, an approach that reserves time with different periodicity is more efficient and can better capture the needs of an application composed by tasks with different periods. Second, considering the cumulative budget $\Theta$ supplied by all the processors leads to a more pessimistic analysis, than considering the contribution of each VP. This happens because the worst-case scenario in multiprocessor systems occurs when the available processors allocate resources with the maximum possible level of parallelism. Hence, the analysis must assume that the overall resource $\Theta$ is provided with a level of parallelism that is often higher than it really is.

Leontyev and Anderson [19] proposed a very simple, though effective, multiprocessor interface with a single parameter, the *bandwidth*. The authors suggest that a bandwidth requirement $w > 1$ is best allocated by an integer number $\lfloor w \rfloor$ of dedicated processors plus a fraction of $w - \lfloor w \rfloor$ allocated onto the other processors. This choice is supported by the evidence that a given amount of computing speed is better exploited on the minimum possible number of processors. However, there are some circumstances in which this approach is not best suited. In fact, the authors illustrate an example in which a set of real-time tasks is not schedulable when the suggested policy is adopted, whereas the tasks can meet their deadlines under a different bandwidth allocation strategy. Moreover, there are situations in which the proposed allocation strategy cannot be adopted, when the physical platform is already allocated to other applications, and processors may not be entirely available.

### B. Contribution of the paper

In this paper, we propose two abstractions for a parallel machine: (i) the Multi Supply Function (MSF) abstraction, which describes the exact amount of resource provided by the platform, and (ii) the Multi-$(\alpha, \Delta)$ (M$\alpha\Delta$) abstraction, which is much simpler to use for the programmer but introduces some waste of the available resource.

We propose a schedulability test that can be used on top of both resource abstractions for verifying the feasibility of a real-time task set under global EDF, global fixed priority (FP), and any work-conserving algorithm.

The rest of the paper is organized as follows. Section II introduces the terminology and notation. Section III shows the reference architecture. Section IV defines the multi supply function (MSF) abstraction. Section V proposes a guarantee test on top of a multiprocessing device abstracted by the multi supply function. Finally, Section VI states our conclusions and presents some future work.

## II. Terminology and notation

We model an application as a set of $n$ sporadic tasks $\Gamma = \{\tau_i\}_{i=1}^{n}$. Each task $\tau_i = (C_i, T_i, D_i)$ is characterized by a worst-case computation time $C_i$, a minimum interarrival time $T_i$ (also referred to as period), and a relative deadline $D_i$. Each task $\tau_i$ releases a sequence of jobs $\tau_{i,k}$, where each job is characterized by an arrival time $r_{i,k}$, an absolute deadline $d_{i,k}$, a computation time $c_{i,k}$. We have that $c_{i,k} \leq C_i$, $r_{i,k} \geq r_{i,k-1} + T_i$, and $d_{i,k} = r_{i,k} + D_i$. In this paper, we assume a *constrained deadline* model, where $D_i \leq T_i$. Time is continuous and time variables are represented by real numbers.

Each application $\Gamma$ is scheduled onto a virtual platform $\mathcal{V}$, modelled as a set of $m$ virtual processors (VP) $\mathcal{V} = \{\nu_j\}_{j=1}^{m}$. Each VP $\nu_j$ is characterized by a supply function $Z_j(t)$ that models the amount of time $\nu_j$ can provide. The concept of supply function is recalled in Section IV-A.

All the VPs belonging to all the virtual platforms in the system are allocated onto the physical platform $\Pi$, which consists of a set of $p$ physical processors $\Pi = \{\pi_k\}_{k=1}^{p}$. Finally, to lighten the notation, we may denote $\max\{0, x\}$ as $(x)_0$.

## III. The overall architecture

The quick evolution of hardware platforms strongly motivates the adoption of appropriate design methodologies that simplify portability of software on different architectures. This problem is even more crucial for multi-core systems, where the performance does not grow linearly with the number of cores and the efficiency of resource usage can only be achieved by tailoring the software to the specific architecture and exploiting the parallelism as much as possible. As a consequence, an embedded software developed to be highly efficient on a given multi-core platform, could be very inefficient on a new platform with a different number of cores.

To reduce the cost of porting software on different multi-core architectures, we propose to abstract the physical architecture with a set of virtual processors. In general, the system should be designed as a set of abstraction layers, each offering a specific service according to a given interface. The advantage of this approach is that one can replace a mechanism inside a layer without modifying the other layers,

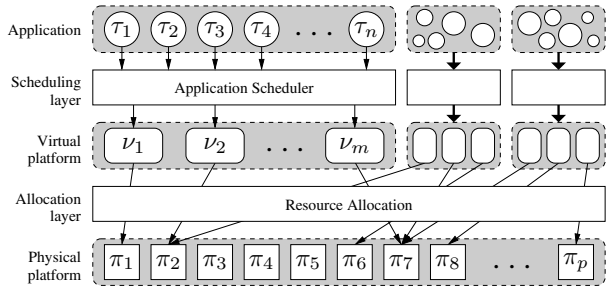


Fig. 1. Architecture overview.

as long as the new mechanism complies with the specified interface. To virtualize the multi-core platform, we use the general architecture depicted in Figure 1.

At the upper layer, the application is developed as a set of real-time tasks with deadline constraints running on a set of virtual processors. Either global or partitioned scheduling schemes can be used at this level to assign tasks to virtual processors. Each virtual processor $\nu_j$ is implemented by a server mechanism capable of providing execution time according to a given supply function. Virtual processors are then allocated to physical processors based on a different scheduling policy. In this way, a change in the hardware platform does not affect the application and the upper-layer scheduler, but only the server allocation layer.

In this paper, we focus on the virtual processor abstraction, proposing a general interface for describing a virtual processor and presenting a feasibility analysis to guarantee the application on the virtual processors using global EDF, global FP, and any work conserving scheduler.

## IV. The multi supply function abstraction

In this section, we describe a suitable abstraction for a set of $m$ VPs that allows exploiting arbitrary fractions of processing time available in the physical platform. With respect to other interfaces proposed in the literature [25], [19], our approach is more general and more precise, because it can capture arbitrary reservations.

The reason for proposing a new interface is that, in multi-application systems, some fraction of the processor can already be occupied by other applications that are not under our control; hence, we often cannot assume that each processor is fully available. Second, considering the amount of resource provided by each VP individually is more precise than dealing with the cumulative value and allows achieving tighter results.

For these reasons, we introduce the following definition to abstract a parallel machine.

*Definition 1:* A Multi Supply Function (MSF) of a set $\mathcal{V} = \{\nu_j\}_{j=1}^m$ of VPs is a set of $m$ supply functions $\{Z_{\nu_j}\}_{j=1}^m$, one for each VP, respectively.

Below, we illustrate the definition of supply function as proposed in the literature [23], [20], [26] and then we extend it to more general cases.

### A. The supply function

The supply function of a single VP represents the minimum amount of resource that the VP can provide in a given interval of time. The VP allocates time to the application during a "resource time partition" (Def. 3 in [23]) that here is extended to non-periodic partitions.

*Definition 2 (compare with Definition 3 in [23]):* A *time partition* $\mathcal{P} \subseteq \mathbb{R}$ is a countable union of non-overlapping intervals[1]

$$\mathcal{P} = \bigcup_{i \in \mathbb{N}} [a_i, b_i) \quad a_i < b_i < a_{i+1}. \tag{1}$$

Without loss of generality we set the instant when the VP is created in the system equal to 0. Hence we have $a_0 \geq 0$.

Given a partition $\mathcal{P}$, its supply function [23], [20], [26] measures the minimum amount of time that is provided by the partition in any interval.

*Definition 3 (Def. 9 in [23], Def. 1 in [20]):* Given a partition $\mathcal{P}$, we define the *supply function* $Z_{\mathcal{P}}(t)$ as the minimum amount of time provided by the partition in every interval of time of length $t \geq 0$, that is

$$Z_{\mathcal{P}}(t) = \min_{t_0 \geq 0} \int_{\mathcal{P} \cap [t_0, t_0+t]} 1 \, \mathrm{d}x. \tag{2}$$

Definition 3 requires the knowledge of the exact time partition $\mathcal{P}$ allocated by the VP to the application, which is often known only at run-time (and not at design time). In fact, the actual allocation typically depends on events (such as the contention with other VPs) that cannot be easily predicted. In the following, we extend Definition 3 by removing the need for such a knowledge.

*Definition 4:* Given a VP $\nu$, we define $\mathsf{legal}(\nu)$ as the set of partitions $\mathcal{P}$ that can be allocated by $\nu$.

*Definition 5:* Given a virtual processor $\nu$, its *supply function* $Z_\nu(t)$ is the minimum amount of time provided by the virtual processor $\nu$ in every time interval of length $t \geq 0$,

$$Z_\nu(t) = \min_{\mathcal{P} \in \mathsf{legal}(\nu)} Z_{\mathcal{P}}(t). \tag{3}$$

Below we report the supply function for several well known server mechanisms.

*a) Explicit Deadline Periodic:* The Explicit Deadline Periodic (EDP) model [11], that generalizes the periodic resource model [20], [26], has the following supply function

$$Z(t) = \max\{0, t - D + Q - (k+1)(P-Q), kQ\} \tag{4}$$

with $k = \left\lfloor \frac{t-D+Q}{P} \right\rfloor$, where the VP provides $Q$ time units every period $P$ within a deadline $D$.

*b) Static partition:* When a VP $\nu$ allocates time statically according to a partition $\mathcal{P}$, then the set of legal partitions $\mathsf{legal}(\nu)$ consists of the unique element $\mathcal{P}$. In this special case of Eq. (3), the supply function $Z_\nu(t)$ can be computed as follows (Lemma 1 by Mok et al. [23]):

$$Z_\nu(t) = \min_{t_0 = 0, b_1, b_2, \ldots} \int_{\mathcal{P} \cap [t_0, t_0+t]} 1 \, \mathrm{d}x. \tag{5}$$

[1]The mathematical development does not change if $\mathcal{P}$ is any Lebesgue measurable set.

*c) P-fair time partition:* Now we investigate the implementation of a VP through a P-fair server [6], [3] with weight $w$. We think that this case is relevant, because P-fair algorithms allow full resource usage on multiprocessors to be achieved.

Holman and Anderson proposed the following lower bound of P-fair supply function (Corollary 2 in [15])

$$Z_\nu(t) \geq \lfloor w(\lfloor t \rfloor - 1) \rfloor - 1. \tag{6}$$

However, the supply function proposed in this paper (Eq. (11)) is tighter (see Figure 3).

In P-fair schedules the processing resource is allocated to the different tasks by time quanta. Without loss of generality, the length of the time quanta can be assumed unitary. Using the notation of Def. 2, it means that a P-fair partition $\mathcal{P}$ has

$$a_i \in \mathbb{Z} \qquad b_i = a_i + 1. \tag{7}$$

A time partition $\mathcal{P}$ associated to a weight $w$ is defined to be P-fair [6] when

$$\forall t \geq 0 \qquad -1 < w\,t - \int_{[0,t]\cap P} 1 \mathrm{d}x < 1. \tag{8}$$

From Equation (8), it follows [6], [3] that the $j^{\text{th}}$ time quantum (we start counting from $j = 0$ to be consistent with Def. 2) allocated in $[a_j, a_j + 1)$, must be within the interval

$$\left[ \left\lfloor \frac{j}{w} \right\rfloor, \left\lceil \frac{j+1}{w} \right\rceil \right) \tag{9}$$

denoted as the $j^{\text{th}}$ subtask window. Figure 2 shows an example of the subtask windows (represented by horizontally aligned segments) when $w = \frac{7}{17}$.

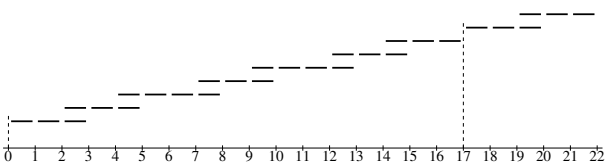


Fig. 2. An example of the subtask window.

For expressing the supply function of a P-fair server mechanism we first define the following quantity.

*Definition 6:* Let $\nu$ be a VP implemented by a P-fair server. We define $\mathsf{len}(k)$ as the length of the longest interval where at most $k$ time quanta are allocated. Formally

$$\mathsf{len}(k) = \max_{\mathcal{P}\in\mathsf{legal}(\nu), t_0\in\mathbb{N}} \left\{ h \in \mathbb{N} : \int_{[t_0,t_0+h)\cap\mathcal{P}} 1\,\mathrm{d}x \leq k \right\} \tag{10}$$

The introduction of $\mathsf{len}(k)$ allows the definition of the supply function by the following Lemma.

*Lemma 1:* The supply function of a VP $\nu$ implemented by a P-fair server whose weight is $w$, is given by:

$$Z_\nu(t) = \begin{cases} 0 & 0 \leq t \leq \mathsf{len}(0) \\ t + k - \mathsf{len}(k) & \mathsf{len}(k) \leq t \leq \mathsf{len}(k) + 1 \\ k + 1 & \mathsf{len}(k) + 1 \leq t \leq \mathsf{len}(k+1) \end{cases} \tag{11}$$

*Proof:* Since a P-fair task allocates time by integer time quanta, we have that

$$\forall t \in \mathbb{N},\ Z_\nu(t) \in \mathbb{N}. \tag{12}$$

We start by proving that

$$\forall k \in \mathbb{N} \quad Z_\nu(\mathsf{len}(k)) = k. \tag{13}$$

From Definition 6, it follows that $Z_\nu(\mathsf{len}(k)) \geq k$, because there exists a legal time partition $\mathcal{P}$ and an interval $[t_0, t_0 + \mathsf{len}(k))$ that contains at least $k$ time quanta. Nonetheless, it cannot happen that $Z_\nu(\mathsf{len}(k)) > k$, because $\mathsf{len}(k)$ is the maximum length among the intervals that contains *at most* $k$ time quanta. Hence, Eq. (13) follows.

From Equations (12) and (13) it follows that

$$\forall k \in \mathbb{N} \quad Z_\nu(\mathsf{len}(k) + 1) \in \{k, k + 1\}$$

because, for every integer step, $Z_\nu$ can either increment by one or remain constant. Nonetheless, it cannot happen that $Z_\nu(\mathsf{len}(k) + 1) = k$, because $\mathsf{len}(k)$ is the *maximum* length of an interval, where $\nu$ allocates $k$ quanta. Hence,

$$\forall k \in \mathbb{N} \quad Z_\nu(\mathsf{len}(k) + 1) = k + 1. \tag{14}$$

Since $Z_\nu$ can be either constant or increase with unitary slope, the lemma follows. ■

In the next lemma, we compute the value of $\mathsf{len}(k)$ when the weight $w$ of the VP is a rational number.

*Lemma 2:* Given a weight $w = \frac{p}{q},\ p, q \in \mathbb{N} \setminus \{0\}$, we have

$$\mathsf{len}(k) = \max_{j=0,\ldots,p-1} \left\{ \left\lceil \frac{(j + k + 2)\,q}{p} \right\rceil - \left\lfloor \frac{j\,q}{p} \right\rfloor \right\} - 2 \tag{15}$$

when $k = 0, \ldots, p - 1$. Moreover we have

$$\mathsf{len}(k + p) = \mathsf{len}(k) + q. \tag{16}$$

*Proof:* Let $\mathcal{P}$ be the critical time partition and $t_0$ be the start of the interval $[t_0, t_0 + \mathsf{len}(k))$ that originates the maximum interval length $\mathsf{len}(k)$, as defined in Def. 6. We claim that.

- $t_0$ must coincide with the end of an allocated time quantum (that we call the $j^{\text{th}}$ time quantum), otherwise it would be possible to left shift $t_0$, achieving a larger $\mathsf{len}(k)$ without increasing the amount of resource provided in $[t_0, t_0 + \mathsf{len}(k))$.
- In the critical partition $\mathcal{P}$, the $j^{\text{th}}$ time quantum must start at the beginning of the $j^{\text{th}}$ subtask window (that is, the interval of Eq. (9)), otherwise we can build another P-fair time partition that anticipates the $j^{\text{th}}$ time quantum, so achieving a larger $\mathsf{len}(k)$.

Hence, $t_0 = \left\lfloor \frac{j}{w} \right\rfloor + 1$ for some $j$, because it must be one unit after the start time of the $j^{\text{th}}$ subtask window. Since the interval $[t_0, t_0 + \mathsf{len}(k))$ must contain (at most) $k$ time quanta, by similar arguments as those exposed before, we conclude

that the end of the critical interval occurs one quantum before the end of the $j+k+1$ time window, that is

$$t_0 + \mathsf{len}(k) = \left\lceil \frac{j+k+2}{w} \right\rceil - 1.$$

Since we do not know what is the $j^{\text{th}}$ time window that originates the critical interval, we must check all of them, that is

$$\mathsf{len}(k) = \sup_{j \in \mathbb{N}} \left\{ \left\lceil \frac{j+k+2}{w} \right\rceil - \left\lfloor \frac{j}{w} \right\rfloor \right\} - 2.$$

However, if the weight is rational ($w = p/q$), then we only need to test for $j$ from 0 to $p-1$. This proves Eq. (15).

Finally, we conclude by proving Eq. (16).

$$\begin{aligned} \mathsf{len}(k+p) &= \max_{j=0,\ldots,p-1} \left\{ \left\lceil \frac{(j+k+p+2)\,q}{p} \right\rceil - \left\lfloor \frac{j\,q}{p} \right\rfloor \right\} - 2 \\ &= \max_{j=0,\ldots,p-1} \left\{ \left\lceil \frac{(j+k+2)\,q}{p} \right\rceil - \left\lfloor \frac{j\,q}{p} \right\rfloor \right\} - 2 + q \\ &= \mathsf{len}(k) + q \end{aligned}$$

as required. ∎

In Table I we evaluate the values of $\mathsf{len}(k)$, as indicated by Eq. (15), for a VP implemented by a P-fair server with weight $w = \frac{7}{17}$. Figure 3 illustrates the corresponding supply function.

| $k$ | $j$=0 | $j$=1 | $j$=2 | $j$=3 | $j$=4 | $j$=5 | $j$=6 | $\mathsf{len}(k)$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 4 | 4 | 4 | 4 | 3 | 4 | 4 |
| 1 | 6 | 6 | 7 | 6 | 6 | 6 | 6 | 7 |
| 2 | 8 | 9 | 9 | 8 | 9 | 8 | 9 | 9 |
| 3 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 |
| 4 | 13 | 13 | 14 | 13 | 14 | 13 | 14 | 14 |
| 5 | 15 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| 6 | 18 | 18 | 19 | 18 | 19 | 18 | 18 | 19 |
| 7 | 20 | 21 | 21 | 21 | 21 | 20 | 21 | 21 |
| 8 | ... | ... | ... | ... | ... | ... | ... | ... |

TABLE I
EVALUATION OF $\mathsf{len}(k)$, WHEN $w = \frac{7}{17}$.

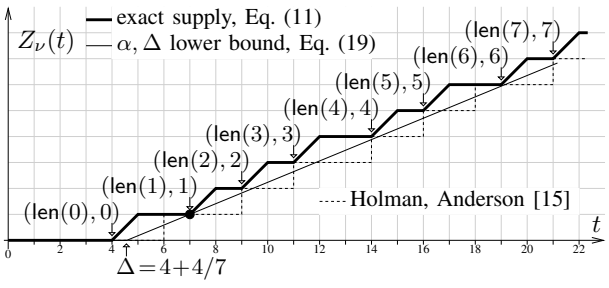


Fig. 3. The supply function for a P-fair server with weight $w = \frac{7}{17}$.

### B. The $(\alpha, \Delta)$ Virtual Processor

The supply function defined in Def. 5 represents a tight model of the service provided by a VP. As shown in Section IV-A, however, it depends on the specific server implementing the reservation and it may not be straightforward to derive. A simpler abstraction able to describe the reservation through a few parameters, independently of the specific virtual processor implementation, would be often more desirable.

Mok et al. [23] introduced the "bounded delay partition", which is described by two parameters: a bandwidth $\alpha$ and a delay $\Delta$. The bandwidth $\alpha$ measures the amount of resource that is assigned to the demanding application, whereas $\Delta$ represents the worst-case service delay. This abstraction has also the additional benefit of being common to other fields, such as networking [27], disk scheduling [8], and network calculus analysis [18]. This means that the analysis proposed here can easily be extended to a more complex system including different architecture components. The $\alpha$ and $\Delta$ parameters are formally defined below.

*Definition 7 (compare Def. 5 in [23]):* Given a VP $\nu$ with supply function $Z_\nu$, the *bandwidth* $\alpha_\nu$ of the VP is defined as

$$\alpha_\nu = \lim_{t \to \infty} \frac{Z_\nu(t)}{t}. \tag{17}$$

Indeed the bandwidth captures the most significant feature of a VP. However, two VPs with the same bandwidth can allocate time in a significantly different manner. Suppose that a VP allocates the processor for one millisecond every 10 and another one allocates the processor for one second every 10 seconds. Both the VPs have the same bandwidth (10% of the physical processor), however, the first VP is more *responsive* in the sense that an application can progress more uniformly. The $\Delta$ parameter provides a measure of the responsiveness, as proposed by Mok et al. [23].

*Definition 8 (compare Def. 14 in [23]):* Given a VP $\nu$ with supply function $Z_\nu$ and bandwidth $\alpha_\nu$, the *delay* $\Delta_\nu$ of the VP is defined as

$$\Delta_\nu = \sup_{t \geq 0} \left\{ t - \frac{Z_\nu(t)}{\alpha_\nu} \right\}. \tag{18}$$

Informally speaking, given a VP $\nu$ with bandwidth $\alpha_\nu$, the delay $\Delta_\nu$ is the minimum horizontal displacement such that the line $\alpha_\nu(t - \Delta_\nu)$ is a lower bound of $Z_\nu(t)$.

Once the bandwidth and the delay are computed, the supply function of the VP $\nu$ can be lower bounded as follows:

$$Z_\nu(t) \geq \alpha_\nu (t - \Delta_\nu)_0. \tag{19}$$

This linear lower bound of the supply function allows the definition of an abstraction of the multiprocessor that is simpler than the MSF.

*Definition 9:* The Multi-$(\alpha, \Delta)$ (M$\alpha\Delta$) abstraction of a set $\mathcal{V} = \{\nu_j\}_{j=1}^m$ of VPs, represented by the $m$ pairs $\{(\alpha_j, \Delta_j)\}_{j=1}^m$, is a special MSF defined by $\{Z_{\nu_j} : Z_{\nu_j}(t) = \alpha_j (t - \Delta_j)_0\}_{j=1}^m$.

Below we propose the computation of the $\alpha, \Delta$ parameters for some classes of servers.

*d) Explicit Deadline Periodic:* For a VP $\nu$ modeled by EDP we have [11]

$$\alpha_\nu = \frac{Q_\nu}{P_\nu} \qquad \Delta = P_\nu + D_\nu - 2Q_\nu \tag{20}$$

where $\nu$ provides $Q_\nu$ time units every period $P_\nu$ within a deadline $D_\nu$.

*e) Static partition:* The interested reader can find the computation of the $\alpha$ and $\Delta$ parameters of a static partition in the work by Feng and Mok [12].

*f) P-fair time partition:* Let $\nu$ be a VP implemented by a P-fair server with weight $w = \frac{p}{q}$. From Equation (16) it follows immediately that

$$\alpha_\nu = \lim_{t\to\infty} \frac{Z_\nu(t)}{t} = \lim_{k} \frac{k}{\mathsf{len}(k)} = \frac{p}{q} = w \tag{21}$$

For computing the delay $\Delta_\nu$ we observe (Eq. (11), Fig. 3) that the linear lower bound is constrained by the points $(\mathsf{len}(k), k)$. It follows that

$$\Delta_\nu = \sup_{k\in\mathbb{N}} \left\{ \mathsf{len}(k) - \frac{k}{w} \right\}. \tag{22}$$

In the example of Figure 3 ($w = \frac{7}{17}$) the delay results to be $\Delta = 7 - \frac{17}{7} \approx 4.571$. More in general, Figure 4 shows the delay $\Delta$ as a function of the bandwidth of the VP. From the

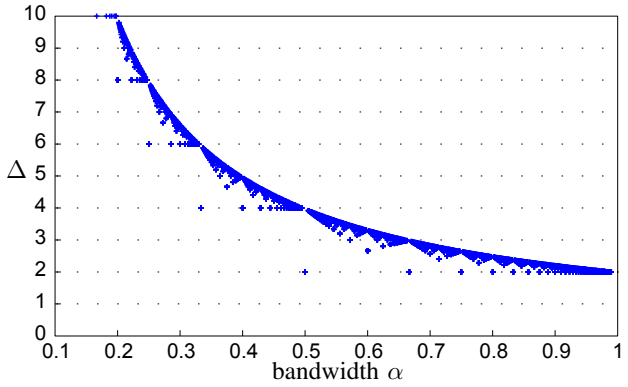


Fig. 4. Values of $\Delta$ for a P-fair VP.

graph, it can be noticed that the delay is upper bounded by a function that is inversely proportional to the bandwidth $\alpha$.

*Lemma 3:* Given a P-fair VP $\nu$ with bandwidth $w \in \mathbb{R}$, we have

$$\Delta_\nu \le \frac{2}{w}. \tag{23}$$

*Proof:* From Lemma 2 we have

$$\begin{aligned}
\mathsf{len}(k) &\le \sup_{j\in\mathbb{N}} \left\{ \left\lceil \frac{j+k+2}{w} \right\rceil - \left\lfloor \frac{j}{w} \right\rfloor \right\} - 2 \\
&\le \sup_{j\in\mathbb{N}} \left\{ \frac{j+k+2}{w} + 1 - \frac{j}{w} + 1 \right\} - 2 \\
&\le \frac{k+2}{w}
\end{aligned}$$

Hence, from Eq. (22), we have

$$\Delta_\nu = \sup_{k\in\mathbb{N}} \left\{ \mathsf{len}(k) - \frac{k}{w} \right\} \le \sup_{k\in\mathbb{N}} \left\{ \frac{k+2}{w} - \frac{k}{w} \right\} = \frac{2}{w}$$

as required. ∎

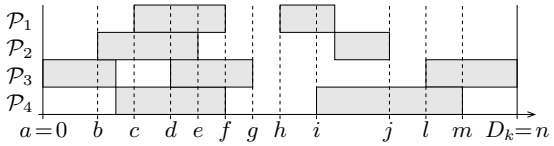


Fig. 5. Example of supply distribution.

## V. Global scheduling algorithms over a MSF

In this section, we analyze the schedulability of a task set $\Gamma = \{\tau_i\}_{i=1}^n$ on a virtual platform $\mathcal{V} = \{\nu_j\}_{j=1}^m$ abstracted by a MSF. To simplify the notation, we denote $Z_{\nu_j}$ by $Z_j$. Let $\tau_k$ be the task that we are analyzing. Without loss of generality, we set the activation of $\tau_k$'s job under analysis equal to 0. We label the VPs by decreasing value of $Z_j(D_k)$ (notice that, differently than in uniform multiprocessors [14], [5] where $Z_j(t) = \alpha_j t$, this ordering is task dependent).

First, we assume that the time partition $\mathcal{P}_j$ provided by each VP $\nu_j$ in $[0, D_k)$ is known in advance. Later, in Theorem 2, we will compute the worst-case partition $\mathcal{P}_j$ starting from the supply functions $Z_j$. For each $\mathcal{P}_j$, we define its characteristic function $S_j(t)$ as

$$S_j(t) = \begin{cases} 1 & t \in \mathcal{P}_j \\ 0 & t \notin \mathcal{P}_j \end{cases} \tag{24}$$

We introduce the subset $L_\ell \subseteq [0, D_k)$ during which the time is provided by $\ell$ VPs in parallel.

$$\forall \ell = 0, \ldots, m,\ L_\ell = \left\{ t \in [0, D_k) : \sum_{j=1}^{m} S_j(t) = \ell \right\} \tag{25}$$

To simplify the presentation, we use $L_\ell$ to denote both the set defined by the previous equation and its measure $|L_\ell|$. Whether we refer to the set or its length will be clear from the context. To lighten the notation, we do not report the dependency of the lengths $L_\ell$ on the task index $k$.

Figure 5 shows an example of time partitions and the corresponding lengths $L_\ell$. Using the labels introduced in the figure, we have: $L_0 = h-g$, $L_1 = b-a+g-f+i-h+l-j+n-m$, $L_2 = c-b+j-i+m-l$, $L_3 = d-c+f-e$, and $L_4 = e-d$. In the rest of the paper we will often use the lengths $\{L_\ell\}_{\ell=0}^m$ as an alternate representation of the set of partitions $\{\mathcal{P}_j\}_{j=1}^m$ allocated by the MSF platform.

Moreover, $W_k$ denotes the workload of jobs with higher priority interfering on $\tau_k$, and $I_k$ denotes the total duration in $[0, D_k)$ in which $\tau_k$ is ready, but cannot execute (due to either preemption or unavailability of computing resources). Bertogna et al. [7] proposed several techniques to upper bound the interfering workload $W_k$ when global EDF, global FP or a generic work-conserving (WC) scheduler is used. Below we report these upper bounds [7]. When global EDF is used, we have

$$W_k \le \overline{W}_k^{\mathsf{EDF}} = \sum_{\substack{i=1 \\ i\neq k}}^{n} \left\lfloor \frac{D_k}{T_i} \right\rfloor C_i + \min\left\{ C_i, D_k - \left\lfloor \frac{D_k}{T_i} \right\rfloor T_i \right\} \tag{26}$$

For a generic work conserving algorithm, instead, we have:

$$W_k \leq \overline{W}_k^{\mathsf{WC}} = \sum_{i=1, i\neq k}^{n} \overline{W}_{k,i} \quad (27)$$

where

$$\overline{W}_{k,i} = N_{k,i}C_i + \min\{C_i, D_k + D_i - C_i - N_{k,i}T_i\} \quad (28)$$

with $N_{k,i} = \left\lfloor \frac{D_k + D_i - C_i}{T_i} \right\rfloor$. Finally, for a global FP scheduler, we have:

$$W_k \leq \overline{W}_k^{\mathsf{FP}} = \sum_{i=1}^{k-1} \overline{W}_{k,i} \quad (29)$$

assuming that tasks are ordered by decreasing priority.

We highlight that the upper bounds on the workload can be refined by iterating the computation of the interference $I_k$ with the reduction of the workload $W_k$, as suggested by Bertogna et al. [7]. However we do not report the details here, due to space limitations.

Given the lengths $\{L_\ell\}_{\ell=0}^m$, we can compute an upper bound on the interference $I_k$ produced on a job belonging to $\tau_k$ by an interfering workload $W_k$.

*Theorem 1:* Given a window $[0, D_k)$ with MSF characterized by the lengths $\{L_\ell\}_{\ell=0}^m$, the interference $I_k$ on $\tau_k$ produced by a set of higher priority jobs with total workload $W_k$ cannot be larger than

$$I_k \leq \overline{I}_k = L_0 + \sum_{\ell=1}^{m} \min\left(L_\ell, \frac{\left(W_k - \sum_{p=0}^{\ell-1} pL_p\right)_0}{\ell}\right) \quad (30)$$

*Proof:* Given a window $[0, D_k)$ with MSF characterized by $\{L_\ell\}_{\ell=0}^m$, we first find the distribution of the interfering workload $W_k$ that maximizes the interference $I_k$ on $\tau_k$. We will prove that $I_k$ is maximized when the workload is distributed over the sets $L_\ell$ with smallest $\ell$, according to the following strategy $A$:

- start allocating the workload on the single processor available in time instants $\in L_1$;
- as long as there is remaining workload to allocate, continue distributing it over the subsequent set $L_\ell$ (with $\ell = 2, \ldots, m$), with parallelism $\ell$.
- Let $z$ be the index if the first set $L_z$ that is not entirely occupied by the interfering workload $W_k$.

Suppose, by contradiction, that a different distribution of the workload $W_k$ produces a larger interference on $\tau_k$. In this latter distribution, consider the set of instants $\in L_\ell$, $\ell = 1, \ldots, m$, where the workload $W_k$ has been allocated on strictly less than $\ell$ processors. Since there is at least one processor available, $\tau_k$ is not interfered in any such instant. Therefore, a larger $I_k$ can be produced redistributing the workload that was allocated in these instants, so that it is executed on all available processors, i.e., on $\ell$ processors in instants $\in L_\ell$. This new distribution $A'$ still produces a $I_k$ larger than $A$.

Since in $A$ the workload is distributed among all sets $L_\ell$ with $1 \leq \ell \leq z$, the larger interference produced in $A'$ must be due to workload allocations over at least one set $L_y$ with $y > z$. Let $\xi \leq L_y$ be the amount of time for which $W_k$ is allocated in $L_y$ (on all $y$ processors). There are $\xi y$ workload units that are used by $A'$ to produce $\xi$ units of interference. These $\xi y$ units were scheduled by $A$ on a lower number of processors $\leq z$. Therefore, the interference produced by $A$ allocating the above $\xi y$ units over intervals with parallelism at most $z$ is greater than $\frac{\xi y}{z} > \xi$. The same argument can be applied to any other share of $W_k$ that is being executed with parallelism $> z$, reaching a contradiction. Therefore, the largest interference is produced when the workload is distributed over the time instants $\in L_\ell$ with smallest $\ell$.

The contributions to the interference from each $L_\ell$ are therefore

- $L_\ell$ for each set $L_\ell$, with $0 \leq \ell \leq z-1$;
- $\frac{1}{\ell}\left(W_k - \sum_{\ell=0}^{z-1} \ell L_\ell\right)$ for $L_z$; and
- 0 for sets $L_\ell$, with $\ell > z$.

Eq. (30) follows summing all contributions. ∎

By replacing the workload $W_k$ of Eq. (30) by $\overline{W}_k^{\mathsf{EDF}}$, $\overline{W}_k^{\mathsf{WC}}$, and $\overline{W}_k^{\mathsf{FP}}$ (see Equations (26), (27) and (29)), we can compute the upper bounds of the interferences $\overline{I}_k^{\mathsf{EDF}}$, $\overline{I}_k^{\mathsf{WC}}$, and $\overline{I}_k^{\mathsf{FP}}$ for global EDF, a work-conserving algorithm, and global FP, respectively.

Theorem 1 assumes that the MSF platform provides time by a set of static partitions $\{\mathcal{P}_j\}_{j=1}^m$ over $[0, D_k)$. However MSF is described by the set of supply functions $\{Z_j\}_{j=1}^m$, and the partitions $\mathcal{P}_j$ actually provided to the application can be anything that complies with the supply function description. The theorem below finds the most pessimistic time partitions, i.e. the partitions such that if the task set is guaranteed on them, it is guaranteed on any time partition that can be allocated by the MSF.

*Theorem 2:* Given a MSF platform modeled by $\{Z_j\}_{j=1}^m$, if the task $\tau_k$ is feasible on the set of time partitions

$$\{\mathcal{P}_j\}_{j=1}^m = \{[D_k - Z_j(D_k), D_k)\}_{j=1}^m, \quad (31)$$

then it is feasible on any set of partitions $\{\mathcal{P}_j\}_{j=1}^m$ complying with the MSF.

*Proof:* As explained in the proof of Theorem 1, the largest interference of a workload $W_k$ on a task $\tau_k$ is produced distributing the workload over the sets $L_\ell$ with the smallest level of parallelism $\ell$.

We prove this theorem by transforming any other set of partitions $\{\mathcal{P}'_j\}_{j=1}^m$ into the one of Eq. (31) without decreasing the associated interference bound. First we shift rightward all the intervals of each partition, then we reduce the amount of resource of the partition to $Z_j(D_k)$ (Figure 6 represents these two steps). First, we shift rightwards one or more particular intervals in $\mathcal{P}'_j$. For each partition $\mathcal{P}'_j$, we are interested in chunks $[a, b)$ of continuous supply, i.e., $[a, b) \subseteq \mathcal{P}'_j$, $\lim_{t\to a^-} S_j(t) = \lim_{t\to b^+} S_j(t) = 0$, $S_j(t) = 1\ \forall t \in [a, b)$. While shifting rightwards a supply chunk, there are three possible situations, each one causing a different effect on the lengths $L'_0, \ldots, L'_m$.

1) Shifts that increase the supply parallelism. A shift of this kind decreases $L'_x$ and $L'_{y<x}$, and increases $L'_{x+1}$ and $L'_{y-1}$ by the same amount. This happens, for instance,

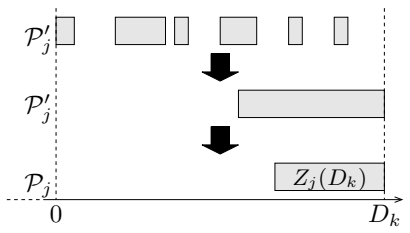


Fig. 6. The transformations of a partition.

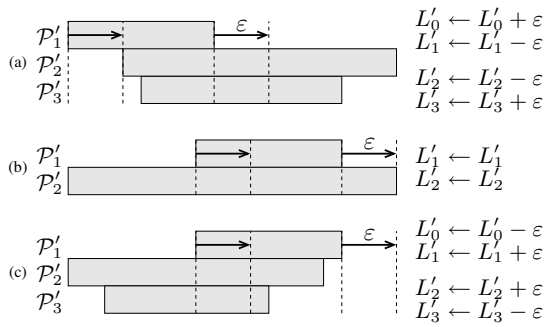


Fig. 7. Typologies of right shifts.

with the shift in Figure 7(a), where shifting rightwards the chunk of $\mathcal{P}'_1$ by $\varepsilon$ causes an increase in $L_3$ and $L_0$, and a decrease in $L_2$ and $L_1$.

2) Shifts that do not vary any $L'_j$ (Figure 7(b)).
3) Shifts that decrease the supply parallelism. A shift of this kind decreases $L'_x$ and $L'_{y<x}$, and increases $L'_{x-1}$ and $L'_{y+1}$ by the same amount (Figure 7(c)).

The upper bound on the interference (Equation (30)) might increase in the first case, is left unchanged in the second case, and might decrease in the third case. For instance, consider the first situation: there is an increase in the length with smallest index $(y-1)$ and with largest index $(x+1)$, and a decrease in the lengths with "central" indexes $L'_y$ and $L'_x$. Let $L'_z$ be the first set for which $W_k - \sum_{p=0}^{z-1} pL'_p < zL'_z$ holds. If $z < y-1$ or $z > x+1$ the interference does not change. If $y-1 \leq z \leq x+1$, the resulting interference is larger than the interference computed with the original values $L'_{y-1}, L'_y, L'_x, L'_{x+1}$ before the shift.

Hence we apply to $\{\mathcal{P}'_j\}_{j=1}^m$ only transformation of the first two kinds. We proceed as follows.

1) We start from the leftmost chunk among all the partitions. Let $a$ be the start of this chunk, and $b$ its end.
2) We shift the chunk rightwards until some of the following conditions occurs.
   a) $a$ reaches the beginning of a chunk of a different partition. In this case, we continue shifting the other chunk together with the first one, forming a block of chunks with identical start time $a$, but different end times $b_1$ and $b_2$.
   b) $b$ reaches the beginning of a chunk on the same partition. We merge both chunks.
   c) $b$ reaches the end of the window (at $D_k$), we continue shifting the next chunk if any.

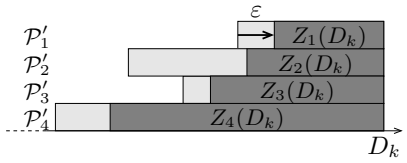


Fig. 8. Reducing the allocated time.

If the above operations are correctly performed, each move will increase the supply parallelism, avoiding situations as the one in Figure 7(c). At the end of the procedure we have transformed any set of partitions $\{\mathcal{P}'_j\}_{j=1}^m$ into the following set

$$\{[D_k - Q_j, D_k)\}_{j=1}^m, \tag{32}$$

with $Q_j \geq Z_j(D_k)$. However the interference experienced on the partition of Eq. (32) cannot exceed the interference on the partition of Eq. (31). In fact, for any partition $\mathcal{P}'_j$ of Eq. (32), if we reduce the supplied resource by an amount $\varepsilon$ then for one length $L'_x$ decreasing by $\varepsilon$, there is a length $L'_{x-1}$ increasing by $\varepsilon$, as shown in Figure 8. Hence this transformation cannot reduce the interference. Since this transformation leads to the set of partitions of Eq. (31), the Theorem is proved. ■

Finally, the following theorem provides a sufficient schedulability condition.

*Theorem 3:* A task set $\Gamma = \{\tau_i\}_{i=1}^n$ is schedulable by the algorithm ALG on a MSF platform modeled by $\{Z_j\}_{j=1}^m$, if

$$\forall k = 1, \ldots, n \qquad C_k + \overline{I}_k^{\mathsf{ALG}} \leq D_k \tag{33}$$

where the $\overline{I}_k^{\mathsf{ALG}}$ is computed from Eq. (30), assuming the lengths $\{L_\ell\}_{\ell=0}^m$ equal to

$$\begin{aligned} L_0 &= D_k - Z_1(D_k) \\ L_\ell &= Z_\ell(D_k) - Z_{\ell+1}(D_k) \\ L_m &= Z_m(D_k). \end{aligned} \tag{34}$$

*Proof:* The schedulability condition simply checks if the relative deadline of each task $\tau_k$ is large enough to accommodate the worst-case computation time of $\tau_k$ together with the interference $I_k$ imposed by other tasks. The amount of interference follows from Theorem 1, upper bounding the workload $W_k$ using Equations (26), (27), or (29).

For any job $\tau_{k,i}$, from the definition of supply function, we can say that every VP $\nu_j$ provides an amount of resource $Q_j \geq Z_j(D_k)$ that is distributed by some unknown partition over the interval $[r_{k,i}, r_{k,i} + D_k)$. Thanks to Theorem 2, if $\tau_{k,i}$ is schedulable on a set of partitions that allocate $Z_j(D_k)$ at the end of the interval $[0, D_k)$, then it is schedulable on any set of partitions that allocate $Q_j \geq Z_j(D_k)$ in any way.

Since the lengths of Eq. (34) are derived assuming this time partition (see also Figure 8), the theorem follows. ■

## VI. Conclusions

In this paper we proposed to abstract a parallel machine as a set of virtual processors, each implemented through a resource reservation mechanism described by a supply function.

The proposed approach is useful to design parallel real-time applications independently of the actual platform and of the specific server used to implement the reservation.

A sufficient schedulability test has also been presented to guarantee the feasibility of real-time applications under global EDF, global fixed priority algorithms and generic work-conserving schedulers.

In the future we plan to implement the proposed abstraction on existing open source operating systems. Moreover, from the theoretical point of view, we plan to tighten the results by relaxing some pessimistic assumptions that we had to introduce to simplify the analysis.

*g) Acknowledgements:* The authors wish to thank Sanjoy Baruah for his insightful comments. We also like to thank the anonymous reviewers of a previously submitted version for their detailed comments, which helped to improve the quality of this paper.

## REFERENCES

[1] Luca Abeni and Giorgio Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27(2):123–167, July 2004.

[2] Luís Almeida, Paulo Pedreiras, and José Alberto G. Fonseca. The FTT-CAN protocol: Why and how. *IEEE Transaction on Industrial Electronics*, 49(6):1189–1201, December 2002.

[3] James H. Anderson and Anand Srinivasan. Early-release fair scheduling. In *Proceedings of the* $12^{th}$ *Euromicro Conference on Real-Time Systems*, pages 35–43, Stockholm, Sweden, June 2000.

[4] Theodore P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proceedings of the* $24^{th}$ *IEEE Real-Time Systems Symposium*, pages 120–129, Cancun, Mexico, December 2003.

[5] Sanjoy Baruah and Joël Goossens. The EDF scheduling of sporadic task systems on uniform multiprocessors. In *Proceedings of the* $29^{th}$ *Real-Time Systems Symposium, 2008*, pages 367–374, Barcelona, Spain, December 2008.

[6] Sanjoy K. Baruah, Neil K. Cohen, Greg Plaxton, and Donald A. Varvel. Proportionate progress: a notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, June 1996.

[7] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Transactions on Parallel and Distributed Systems*, 2008.

[8] John Bruno, José Brustoloni, Eran Gabber, Banu Özden, and Abraham Silberschatz. Disk scheduling with quality of service guarantees. In *IEEE International Conference on Multimedia Computing and Systems*, volume 2, pages 400–405, Firenze, Italy, July 1999.

[9] Giorgio C. Buttazzo, Giuseppe Lipari, Luca Abeni, and Marco Caccamo. *Soft Real-Time Systems: Predictability vs. Efficiency*. Springer, 2005.

[10] Zhong Deng and Jane win-shih Liu. Scheduling real-time applications in Open environment. In *Proceedings of the* $18^{th}$ *IEEE Real-Time Systems Symposium*, pages 308–319, San Francisco, CA, U.S.A., December 1997.

[11] Arvind Easwaran, Madhukar Anand, and Insup Lee. Compositional analysis framework using EDP resource models. In *Proceedings of the* $28^{th}$ *IEEE International Real-Time Systems Symposium*, pages 129–138, Tucson, AZ, USA, 2007.

[12] Xiang Feng and Aloysius K. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the* $23^{rd}$ *IEEE Real-Time Systems Symposium*, pages 26–35, Austin, TX, U.S.A., December 2002.

[13] Laurie J. Flynn. Intel halts development of 2 new microprocessors. *The New York Times*, May 2004.

[14] Shelby Funk, Joël Goossens, and Sanjoy Baruah. On-line scheduling on uniform multiprocessors. In *Proceedings of the* $22^{nd}$ *IEEE Real-Time Systems Symposium*, pages 183–192, London, United Kingdom, December 2001.

[15] Philip Holman and James H. Anderson. Group-based pfair scheduling. *Real-Time Systems*, 32(1–2):125–168, February 2006.

[16] Tei-Wei Kuo and Ching-Hui Li. Fixed-priority-driven open environment for real-time applications. In *Proceedings of the* $20^{th}$ *IEEE Real-Time Systems Symposium*, pages 256–267, Phoenix, AZ, U.S.A., December 1999.

[17] Tei-Wei Kuo, K. Lin, and Y. Wang. An open real-time environment for parallel and distributed systems. In *Proceedings of the* $20^{th}$ *International Conference on Distributed Computing Systems*, pages 206–213, Taipei, Taiwan, April 2000.

[18] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus*, volume 2050 of *Lecture Notes in Computer Science*. Springer, 2001.

[19] Hennadiy Leontyev and James H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In *Proceedings of the* $20^{th}$ *Euromicro Conference on Real-Time Systems*, pages 191–200, Prague, Czech Republic, July 2008.

[20] Giuseppe Lipari and Enrico Bini. Resource partitioning among real-time applications. In *Proceedings of the* $15^{th}$ *Euromicro Conference on Real-Time Systems*, pages 151–158, Porto, Portugal, July 2003.

[21] Clifford W. Mercer, Stefan Savage, and Hydeyuki Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 90–99, Boston, MA, U.S.A., May 1994.

[22] M. Moir and S. Ramamurthy. Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In *Proceedings of the* $20^{th}$ *IEEE Real-Time Systems Symposium*, pages 294–303, Phoenix, AZ, U.S.A., December 1999.

[23] Aloysius K. Mok, Xiang Feng, and Deji Chen. Resource partition for real-time systems. In *Proceedings of the* $7^{th}$ *IEEE Real-Time Technology and Applications Symposium*, pages 75–84, Taipei, Taiwan, May 2001.

[24] Abday K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[25] Insik Shin, Arvind Easwaran, and Insup Lee. Hierarchical scheduling framework for virtual clustering multiprocessors. In *Proceedings of the* $20^{th}$ *Euromicro Conference on Real-Time Systems*, pages 181–190, Prague, Czech Republic, July 2008.

[26] Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the* $24^{th}$ *Real-Time Systems Symposium*, pages 2–13, Cancun, Mexico, December 2003.

[27] Dimitrios Stiliadis and Anujan Varma. Latency-rate servers: A general model for analysis of traffic scheduling algorithms. *IEE/ACM Transactions on Networking*, 6(5):611–624, October 1998.

[28] Ion Stoica, Hussein Abdel-Wahab, Kevin Jeffay, Sanjoy K. Baruah, Johannes E. Gehrke, and Charles Gregory Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceeding of the* $17^{th}$ *IEEE Real Time System Symposium*, pages 288–299, Washington, DC, U.S.A., December 1996.