

# On Non-Utilization Bounds for Arbitrary Fixed Priority Policies \*

Xue Liu and Tarek Abdelzaher  
Department of Computer Science  
University of Illinois at Urbana Champaign  
Urbana, IL 61801  
e-mail: {xueliu,zaher}@cs.uiuc.edu

## Abstract

*Prior research on schedulability bounds focused primarily on bounding utilization as a means to meet deadline constraints. Non-trivial bounds were found for a handful of scheduling policies in which utilization is directly related to the ability of the policy to meet deadlines. Examples include Rate Monotonic, Deadline Monotonic and EDF scheduling. For most other scheduling policies, however, utilization is not correlated with schedulability. For example, shortest job first can miss deadlines at an arbitrarily low utilization. This raises the question of whether or not some other non-utilization-based metric might be more indicative of schedulability in those cases. This paper answers the above question positively by extending the notion of schedulability bounds, in a uniform manner, to arbitrary priorities and non-utilization metrics. We present a simple function that generates the schedulability metric to be bounded from the definition of a (fixed-priority) scheduling policy, and derive a non-trivial schedulability bound on that metric. It is shown that the generated metrics and bounds are valid in that no deadline misses occur when these bounds are not violated. This result allows efficient real-time admission control to be performed in systems with arbitrary fixed-priority scheduling policies. As an example, we illustrate applying schedulability bounds for admission control to shortest-job-first and velocity monotonic scheduling.*

**Keywords:** Real-time scheduling, schedulability analysis, utilization bounds, aperiodic tasks.

## 1 Introduction

With the increasing complexity of computing systems and with the decreasing resource cost, a gradual drift is envisioned from seeking optimality to seeking (sufficient) sim-

plicity. Designers and engineers are increasingly more predisposed to use simple rules that are sufficient (i.e., guarantee correctness) in lieu of complex necessary and sufficient conditions that are resource optimal. This is because in a complex system, simple rules are easier to apply and are less error-prone. In real-time systems where a predominant concern is to observe task deadlines, schedulability bounds re-gain importance because of their simplicity and sufficiency.

While many exact (and rather complex) schedulability tests exist in prior literature, such as time-demand analysis and processor-demand analysis, the simplest heuristics are those that rely on utilization bounds. The choice of utilization as the metric to bound in prior literature assumes implicitly that the scheduling policy is more capable of meeting deadlines when the utilization is lower. This observation is not always true of many useful policies. For example, shortest job first (which maximizes throughput in non-real-time systems) may miss deadlines at an arbitrarily low utilization. The inability to find non-trivial utilization bounds for such policies has traditionally caused them to be discarded as irrelevant.

This paper derives schedulability results that allow one to reason about deadline satisfaction in a much broader category of scheduling policies than that for which (non-trivial) utilization bounds exist. We show that there exists a systematic way of finding the particular non-utilization-based load metric most suitable to use for a given fixed-priority scheduling policy. Finally, we derive the corresponding schedulability bounds.

For generality, we consider an aperiodic task model where a task can execute on a pipeline of multiple resources (stages). It is shown that previous utilization bounds derived for this model are just a special case of a more fundamental schedulability criterion. Utilization is a special case of a more general load metric proposed in this paper that is a function of the scheduling policy. This metric defaults to utilization in the case of deadline monotonic scheduling. A universal schedulability bound exists on this general metric

---

\*The work reported in this paper was supported in part by MURI grant N00014-01-1-0576 and NSF grants CCR-0208769 and CSR-0553419.

such that all deadlines are met as long as the bound is not violated.

Note that, results in this paper are specific to fixed priority scheduling. Dynamic-priority scheduling policies, such as EDF, are not considered. Note also that we do not explicitly address the common case of periodic tasks. However, bounds derived from our aperiodic task model do apply to the special case of periodic arrivals, since the derivations take into account the worst case scenario over all possible task arrival patterns (including periodic arrivals). Observe also that in multi-stage or distributed systems, even if tasks entering the first stage are periodic, tasks leaving that stage lose much of their periodicity arriving at the next stage in a bursty manner. An aperiodic task framework for analyzing schedulability avoids the complications of having to compute and account for bursts in the periodic model.

We believe that these results will allow schedulability bounds to be derived for a broader category of scheduling policies. The framework, for example, allows generalization of schedulability bounds to policies that might be suitable for distributed systems, such as velocity-monotonic scheduling, and policies that optimize other (non-deadline-related) performance metrics (such as shortest job first or maximum utility first).

The remainder of this paper is organized as follows. Section 2 describes the system model and outlines the main results of the authors. Section 3 presents the derivation of these results. Section 4 evaluates the derived universal bound demonstrating that it is successful in preventing deadline misses. Related work is summarized in Section 5. The paper concludes with Section 6.

## 2 System Model and Contribution

Consider a task model in which aperiodic tasks arrive at a system of multiple resources (e.g., a cluster of servers). Each task  $T_i$  has a relative end-to-end deadline  $D_i$ , which defines its maximum acceptable total latency in the system. The task requires multiple processing stages, collectively denoted by set  $G_i$ . The execution time of task  $T_i$  on stage  $j \in G_i$  is denoted  $C_{ij}$ . In this paper, we assume that a fixed-priority scheduling policy is used that assigns the same priority to the same task  $T_i$  on all resources in  $G_i$ . The priority is assigned monotonically decreasing in some function  $x(\cdot)$  of the task, such that the lower the value of  $x$  the higher the priority. We denote the value of function  $x$  for task  $T_i$  by  $x_i$ . We restrict that  $x$  be always positive. The choice of function  $x$  can be arbitrary and not restricted to execution parameters of the task. To give a few examples, in deadline-monotonic scheduling,  $x_i = KD_i$ . In shortest job first,  $x_i = K \sum_{j \in G_i} C_{ij}$ . In velocity monotonic scheduling,  $x_i = KD_i/|G_i|$ . In smallest cost-to-utility-ratio first,  $x_i = KCost_i/Utility_i$ . In a policy that sched-

ules tasks in ascending alphabetic order by user's last name,  $x_i = K(ASCII(lastname))$ . In the examples above,  $K$  is an arbitrary proportionality constant. Observe that the priority order does not depend on  $K$  and can be arbitrary and unrelated to task execution parameters.

To assess schedulability, we propose a new abstract load metric that is a generalization of utilization. This metric is maintained for each resource separately. To express this metric, let us first define the *contribution interval*,  $b_i$ , of task  $T_i$  to be a time interval numerically equal to its priority indicator  $x_i$ . Hence,  $b_i = x_i$ . We shall now define our abstract load metric on stage  $j$  as the sum of contributions of all tasks  $T_i$ , where each task contributes to that metric an amount  $C_{ij}/x_i$  during its contribution interval starting from its arrival at stage  $j$ . More formally, at any time  $t$ , let us define the task set  $S_j(t)$  as the set of all tasks  $T_i$ , where  $T_i$  arrived at stage  $j$  no more than  $b_i$  time units ago (these are the tasks that contribute to the abstract load). The abstract load  $M_j(t)$  at stage  $j$  and time  $t$  is then defined as:

$$M_j(t) = \sum_{i|T_i \in S_j(t)} \frac{C_{ij}}{x_i} \quad (1)$$

In the following, we shall omit the time index where no ambiguity arises. The main result of the paper lies in showing that for the definition of abstract load mentioned above, a task  $T_i$  meets its deadline as long as the following schedulability condition is satisfied for all values of  $t$ :

$$\sum_{j \in G_i} \frac{M_j(1 - M_j/2)}{1 - M_j} \leq \frac{D_i}{x_i} \quad (2)$$

Observe that for a given scheduling policy there may be multiple choices of function  $x$ . For instance, for deadline monotonic scheduling, one may choose  $x_i = KD_i$ , where the choice of  $K$  is arbitrary. The definition of  $x$ , however, has implications on the contribution interval and the value of abstract load. Hence, an infinite family of valid schedulability bounds can be found for each scheduling policy depending on the choice of  $x$ . This may be thought of as a concern with the approach, since we know that schedulability depends only on priority order and should not be a function of priority values. On the other hand, it is intuitive that schedulability be a function of the load metric. Given a different load metric, we expect a different schedulability bound. Our approach can therefore be viewed as a generalization, where given a priority order, an infinite family of load metrics is proposed (as opposed to only one). In previous literature, only one bound was derived per policy, because it was semantically tied to a specific load metric; namely utilization. In contrast, the freedom in choosing  $x$  offers more flexibility in fine-tuning the abstract load metric to improve the performance of the schedulability bound.

This is not unlike the flexibility offered in choosing Lyapunov stability functions in non-linear control theory to determine the locus of stability of nonlinear systems. Different functions for the same system may yield different stability regions. In a sense, the function  $x$  is akin to a Lyapunov function for scheduling policies. It should be positive and monotonically decreasing with priority but is otherwise unrestricted. For a given scheduling policy, some functions are better than others in defining the schedulability region. The effect of the choice of  $x$  on the performance of the bound will be briefly investigated in the evaluation section. The problem of finding the best  $x$ , however remains open and we hope might invite future research.

We call the condition expressed in Inequality (2) a universal bound since it applies regardless of the definition of  $x$ , and regardless of the scheduling policy. Observe that in the special case of deadline-monotonic scheduling, when using  $x_i = D_i$ , the abstract load  $M_j$  becomes the instantaneous utilization and Inequality (2) reduces to the bound derived in earlier literature [2]. In general, the universal bound gives rise to efficient implementations of admission control algorithms that ensure that the left-hand-side does not exceed  $\min_i \{D_i/x_i\}$  for a specific task set.

### 3 The Universal Feasible Region

In this section, we derive the universal feasible region that defines end-to-end schedulability of tasks traversing multiple resources in a distributed system. While in this paper we use the terms *bound* and *feasible region* interchangeably, strictly speaking, the latter is a generalization of the former. A schedulability bound separates feasible from potentially infeasible task sets based on one value (the bound) on a uni-dimensional axis such as processor utilization. A feasible region generalizes this notion to a surface that defines a schedulable subspace within a multi-dimensional volume, where each dimension is the load on a different resource. For example, Inequality (2) defines a multi-dimensional space within which a task is schedulable. The dimensions of that space are the abstract load values of the different resources visited by the task.

Consider one arbitrary resource stage  $j$  visited by task  $T_n$  that has an end-to-end deadline  $D_n$ . We will bound the delay experienced by the task at this stage as a function of the abstract load  $M_j$  on that stage. Later, we will use this result to derive the universal multi-stage feasible region. The reasoning below is largely patterned after the proofs in [1] and [2], which were specific to utilization-based feasible regions. In this paper, we generalize our previous reasoning to arbitrary load metrics. For completeness, we present the entire proof outline with references to [1] and [2] where appropriate.

Consider the abstract load curve  $M_j(t)$  at stage  $j$ , as defined in Section 2 of this paper. An example curve is

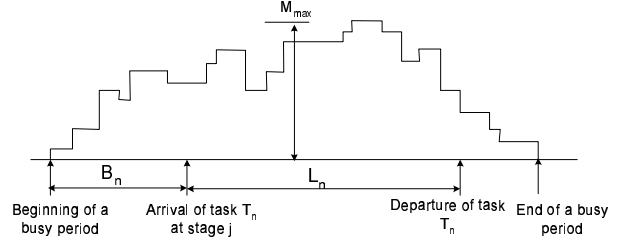


Figure 1. The abstract load curve.

shown in Figure 1. First, we prove the following useful lemma, which states an interesting invariant about the load curve:

**Lemma 1:** The area under the abstract load curve at stage  $j$  for any whole busy period is equal to the sum of the execution times of all active tasks at that stage.

*Proof.* From the definition of abstract load, we know that upon the arrival of task  $T_i$  on stage  $j$ , the abstract load is raised at that stage by an amount  $C_{ij}/x_i$ . This increment continues for a duration numerically equal to  $x_i$ . Task  $T_i$  therefore contributes a rectangle of area  $x_i \times C_{ij}/x_i = C_{ij}$  to the area under the abstract load curve. Since the total area under the curve is the sum of contributions of all individual tasks, this area is equal to  $\sum_i C_{ij}$ . The lemma is thus proved. ■

Let the *height* of the abstract load curve (when task  $T_n$  is on stage  $j$ ) represent the highest value of abstract load on stage  $j$  that occurs within the busy period in which  $T_n$  executes and prior to the departure of  $T_n$  from stage  $j$ . We would like to find the lowest height of the curve for which  $T_n$  experiences delay  $L_n$  on stage  $j$ . This lowest value represents a worst case condition. If abstract load never exceeds that value, the delay of  $T_n$  never exceeds  $L_n$ .

In the following discussion, without loss of generality, we assume that the busy period in which  $T_n$  executes on stage  $j$  consists only of  $T_n$  and higher priority tasks that execute while  $T_n$  is on stage  $j$ . We call it a level- $n$  busy period. Following similar derivations in earlier literature, lower priority tasks and other tasks that have no execution intervals in the busy period prior to the departure time of  $T_n$  from stage  $j$  need not be considered. This is because they increase abstract load without reducing the schedulability of  $T_n$ , and hence do not lead to the worst case condition. Let  $T_n$  arrive  $B_n$  time units after the start of a level- $n$  busy period on stage  $j$  and let it experience delay  $L_n$  on stage  $j$  prior to departure. Since enough computation time must exist on stage  $j$  to fill the busy period, by Lemma 1, the area under the abstract load curve on stage  $j$  must be equal at least to  $B_n + L_n$ .

Next, we minimize the height of the abstract load curve, subject to the constraint that the minimum area under the curve is equal to  $B_n + L_n$ . Let  $A$  be the area under the part of the curve that extends after the departure of  $T_n$  from stage  $j$ . Hence, the area under the curve in the interval from the beginning of the busy period and until the departure of  $T_n$  is  $B_n + L_n - A$ . The length of that interval is  $B_n + L_n$ . From geometry, we know that the height of a shape of a given area (in this case,  $B_n + L_n - A$ ) and base (in this case,  $B_n + L_n$ ) is minimized when that shape is a rectangle, in which case the height is simply area/base. Hence, the minimum abstract load that leads to a delay  $L_n$  for task  $T_n$  is given by the curve height:

$$M_j = \frac{B_n + L_n - A}{B_n + L_n} \quad (3)$$

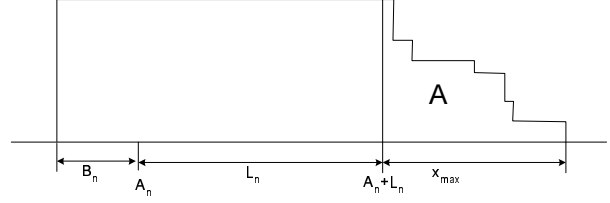
A curve that satisfies this height is shown in Figure 2. Observe that  $M_j$  is minimized with respect to  $B_n$  when the latter is equal to zero. Hence, in the worst case, task  $T_n$  arrives the beginning of a new busy period and the abstract load is given by:

$$M_j = \frac{L_n - A}{L_n} \quad (4)$$

$M_j$  is further minimized by maximizing area  $A$ . To compute the maximum value of area  $A$ , Let  $\mathcal{E}$  be the set of tasks in the level- $n$  busy period whose contribution intervals end *after* the departure time of  $T_n$  from stage  $j$ . These are the only tasks that contribute to area  $A$  (since tasks that arrive after  $T_n$ 's departure do not affect its schedulability and hence need not be considered). For each task  $E_i$  in set  $\mathcal{E}$ , let  $A_{ij}$  be the arrival time of the task to stage  $j$  and  $C_{ij}$  be its computation time on the stage. Without loss of generality, let tasks  $E_i$  in set  $\mathcal{E}$  be numbered in increasing order of their arrival times;  $E_1$  is the first to arrive and  $E_k$  is the last. In [1], it was shown that in the context of utilization bounds, area  $A$  is maximized when the following two properties hold. First, for tasks  $E_2, \dots, E_k \in \mathcal{E}$ , we must have  $A_{ij} = A_{i-1,j} + C_{i-1,j}$ . The last task,  $E_k$ , arrives such that  $A_{kj} + C_{kj}$  is equal to the departure time of  $T_n$  (i.e.,  $A_{nj} + L_n$ ). Moreover, all tasks  $E_i$  have the maximum possible contribution interval,  $x_{max}$ . Observe that since they are higher priority than  $T_n$ ,  $x_{max} < x_n$ . Lemma 2 states this result for the abstract load curve.

**Lemma 2:** In the worst case pattern when the height of the abstract load curve is minimized (which will lead to minimum abstract load bound), all tasks  $E_i$ ,  $1 \leq i \leq K$  satisfy the following properties

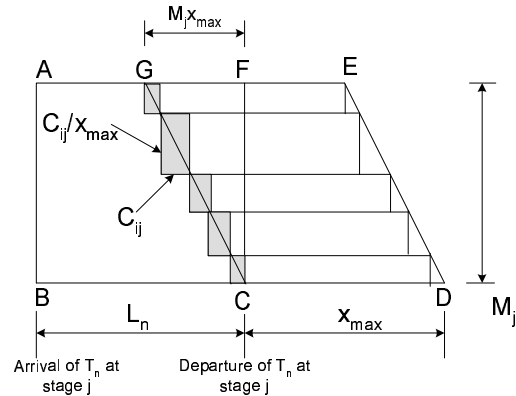
1. All tasks  $E_i \in \mathcal{E}$  are tasks of type  $T_{max}$  (i.e., with  $x_i$  equals to  $x_{max}$ ).
2. For tasks  $E_2, \dots, E_k$ , we have  $A_{ij} = A_{i-1,j} + C_{i-1,j}$ , and  $A_{kj} + C_{kj} = A_{nj} + L_n$ .



**Figure 2.** An example of an abstract load curve for a real pattern.

*Proof.* The proof follows the steps in [1] after replacing deadlines  $D_i$  with abstract priority function values  $x_i$ , and replacing the instantaneous utilization curve with the abstract load curve. The intuition is as follows. To minimize the height of the curve it is desired to maximize area  $A$ . Intuitively, in order to maximize  $A$ , one must maximize the contribution intervals of higher priority tasks in  $\mathcal{E}$  (so that they “protrude” as far as possible past the departure of  $T_n$ ). For the same reason, the arrival of these tasks should be packed as closely as possible to the departure time of  $T_n$ . The two conditions of the lemma follow. The result is shown in Figure 3. For the detailed proof see [1]. ■

Figure 3 plots an abstract load curve satisfying the properties derived so far; namely, the area under the curve is  $L_n$ , its base length is  $L_n + x_{max}$  (remember that  $B_n = 0$  in the worst case), it is flat until the departure time of  $T_n$  and the shape after that is determined by the worst case task pattern  $\mathcal{E}$  (described by Lemma 2).



**Figure 3.** Worst case pattern.

First, we see the trailing edge of the curve in Figure 3 is bounded by line  $ED$ . The slope of line  $ED$  is  $\frac{\text{height of step}}{\text{computation time}} = \frac{C_i/x_{max}}{C_i} = \frac{1}{x_{max}}$ . The height of the curve is  $M_j$ . Further, the length  $CD$  is equal to  $x_{max}$ . Hence, the maximum area  $A$  is given by the area of the

trapezoid CDEF:

$$A = M_j \frac{x_{max} + (x_{max} - M_j x_{max})}{2} \quad (5)$$

Substituting for  $A$  in Equation (4) and solving for  $L_n$ , we get:

$$L_n = \frac{M_j(1 - M_j/2)}{1 - M_j} x_{max} \quad (6)$$

where  $M_j < 1$  from Equation (4). When the task traverses multiple stages, in order for the end-to-end deadline to be met, we must enforce that the sum of the delays  $L_n$  incurred by task  $T_n$  on all stages be less than its end-to-end deadline  $D_n$ . From Equation 6, we thus get:

$$\sum_{j \in G_i} \frac{M_j(1 - M_j/2)}{1 - M_j} \leq \frac{D_n}{x_{max}} \quad (7)$$

This defines the universal multi-stage feasible region in terms of the priority values  $x_i$ . Since the larger the value of  $x$ , the smaller the priority, we know  $\frac{D_n}{x_{max}} > \frac{D_n}{x_n}$ . A sufficient and practical feasible region is thus:

$$\sum_{j \in G_i} \frac{M_j(1 - M_j/2)}{1 - M_j} \leq \frac{D_n}{x_n} \quad (8)$$

where  $M_j < 1$  from Equation (4). Notice that the bound in the feasible region (the right-hand-side of the above equation) is *not* the same for all tasks. For a set of tasks traversing  $N$  stages, an admission controller should ensure that:

$$\sum_{j \in G_i} \frac{M_j(1 - M_j/2)}{1 - M_j} \leq \min_i \left( \frac{D_i}{x_i} \right) \quad (9)$$

where the minimization is carried out over the set of active tasks, or over the set of tasks in the current busy period. Observe that in the latter case the minimum on the right-hand-side can be updated in constant time upon the arrival of each new task and reset when the busy period ends (i.e., a processor becomes idle). By substituting appropriate values for  $x_i$  in the above equation, we can readily get feasible regions for different scheduling policies. For example, when we select  $x_i = D_i$  for each task, the feasible region defined above reduces to:

$$\sum_{j=1}^N \frac{M_j(1 - M_j/2)}{1 - M_j} \leq 1. \quad (10)$$

This is exactly the previous result given in [2] for deadline monotonic scheduling. In this case, the abstract load  $M_j$  reduces to  $U_j = \sum_i \frac{C_{ij}}{D_i}$ , which is the instantaneous utilization. To give another example, when select  $x_i = \sum_j C_{ij}$ ,

we get

$$\sum_{j=1}^N \frac{M_j(1 - M_j/2)}{1 - M_j} \leq \min_i \left( \frac{D_i}{\sum_j C_{ij}} \right), \quad (11)$$

where  $M_j = \sum_i \frac{C_{ij}}{\sum_j C_{ij}}$ . This is the feasible region for the shortest-job-first scheduling policy.

Equation (8) is the main result of this paper. It quantifies, for the first time in real-time computing literature, a universal schedulable region that is applicable to an arbitrary scheduling policy as a function of its priority definition  $x$ . We state this result as the following theorem.

**The Universal Feasible Region Theorem:** Consider a system scheduled by a fixed priority scheduling policy, where priorities are monotonically decreasing in some function  $x$ . Let  $x_n$  be the value of  $x$  for some task  $T_n$ , which traverses  $N$  stages and has an end-to-end deadline  $D_n$ . Let the abstract load at stage  $j$  be denoted by  $M_j$ .  $T_n$  meets its end-to-end deadline as long as the following condition is true:

$$\sum_{j=1}^N \frac{M_j(1 - M_j/2)}{1 - M_j} \leq \frac{D_n}{x_n}$$

A simulation study that explores the use of universal feasible regions for admission control is given in Section 4.

## 4 Evaluation

In this section, we evaluate the performance of an admission controller based on the universal feasible region theorem derived above. For all experiments discussed in this section, we consider a system of independent tasks generated with exponentially distributed per-stage computation times. The computation times of different stages are independent. End-to-end deadlines are chosen uniformly from a range that is independent of the number of stages. The arrival process is Poisson.

We allow different tasks to be processed by different stages. Suppose there are a total of  $N$  stages in the system. Our load generator uses a parameter  $p(j)$  to represent the probability that a task will be processed by stage. To balance load among all machines, we can select the same mean task computation times at all stages and let  $p(j) = p$  (i.e., the same) for all stages. For example when  $p = 1$ , every task will be processed by all stages. When  $p = 0.5$  and the number of total stages is  $N = 10$ , the mean number of stages a task will use is  $N \times p = 5$ . We let the stages be always traversed in the same order.

Scheduling at each stage is based on a priority which is monotonically decreasing with  $x$ . The priority is kept constant for all execution stages of a task. As we have discussed

above, by selecting different functions for  $x$ , we can express arbitrary static scheduling policies. In this section, we focus on Deadline Monotonic Scheduling (DMS), Shortest Job First Scheduling (SJF) and Velocity Monotonic Scheduling (VMS) as examples.

When a task  $T_i$  arrives at its first stage, an admission controller allows this incoming task into the system only if the system remains within the feasible region. The abstract load at each stage will be updated upon task arrival and decreased upon expiration of its contribution interval. This is because the abstract load due to a task is defined to persist until the end of the task’s contribution interval (rather than until task departure). The rationale for this definition is so that the abstract load is only a function of the task arrival pattern and not the task schedule (which alters task departure times). Since abstract load is independent of the schedule, it is possible to do apples-to-apples comparisons of the properties of different scheduling policies at the same abstract load. When a stage becomes idle, the contributions of all departed tasks to its abstract load are removed. This is because a departed task will not affect the future schedule of this stage. Resetting the abstract load as a stage becomes idle is a very important tool that reduces the pessimism of admission control.

In the following, we study the performance of the admission controller based on universal feasible regions, applied to different scheduling policies. Several experiments are conducted to explore properties of the new admission control scheme. The results are presented and discussed in the subsections below. No deadline misses were observed in any of the experiments, which verifies the correctness of the derived bounds.

#### 4.1 Infinite Family of Bounds Under A Specific Scheduling Policy

As mentioned in Section 2, unlike previous results, where a specific bound or feasible region is derived for a specific scheduling policy, this paper describes an infinite family of bounds for the same scheduling policy. This can be achieved by changing the function  $x$ , while keeping it monotonically decreasing in priorities.

In the following, we use a simple transformation by scaling  $x_i$  by a constant positive number  $K$ . By changing the value of  $K$ , we get an infinite family of bounds under a specific scheduling policy. We choose Deadline Monotonic Scheduling (DMS) as the underlying scheduling policy in this subsection, as it is widely studied and is easy to implement in most operating systems. Hence,  $x_i = KD_i$ .

In the first set of experiments, we explore the effect of  $K$  on the admission controller. In particular, we test whether or not changing the value of  $K$  affects the feasible region. In this experiment, the number of stages is

$N = 10$  and the probability that a task selects a stage is  $p = 0.5$ . The average stage load is kept roughly equal by generating task computation times at different stages from the same distribution with the same mean. The average end-to-end deadline was kept at about 50 times the total computation time across the entire pipeline. This is consistent with high-performance servers where individual request execution times are much smaller than response-time requirements, allowing hundreds of requests to be handled concurrently. The input load is varied from 40% to 200% of the single stage capacity. That is to say, the sum of computation times of all tasks generated over the duration of the experiment is varied from 40% to 200% of the length of the experiment. Curves are drawn to show the average real stage utilization due to admitted tasks (defined as the percentage of time a processor is busy, averaged over all processors) versus input load for different values of  $K$ . Figure 4 depicts the results with  $K = 0.1$ ,  $K = 1.0$  and  $K = 10.0$ .

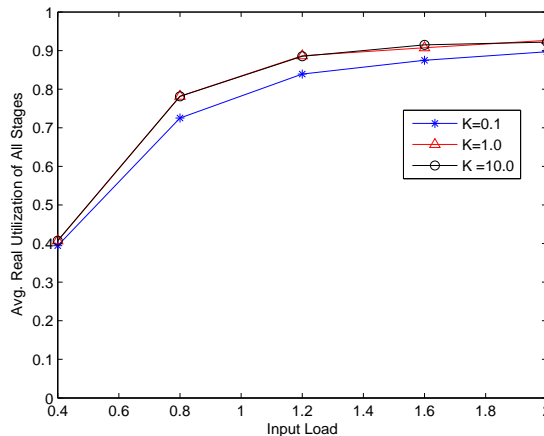
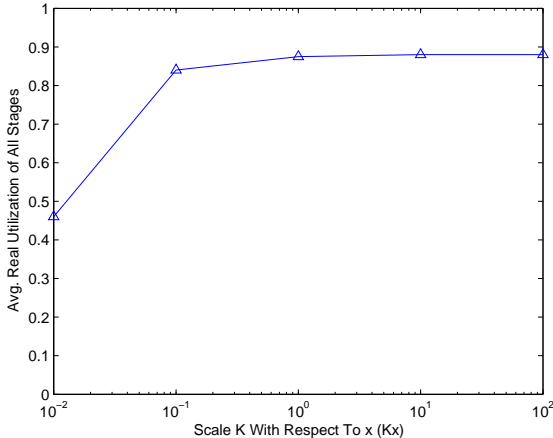


Figure 4. Effect of different bounds under different input workloads (DMS).

Two important observations follow from Figure 4. First, the resource utilization is reasonably high after admission control. For example, when the input load is 100% of stage capacity, the average stage utilization after admission control is more than 80%. This is a very good schedulable utilization for fixed-priority scheduling. Second, different values of  $K$  have different effects on the resource utilization. For example, when  $K = 0.1$ , the average utilization is less than that when  $K = 1.0$  and  $K = 10.0$  under all input load.

To further study the effect of  $K$  on system utilization, we vary  $K$  from 0.01 to 100 for an input workload of 120% of the single stage capacity. Figure 5 depicts the result as a semi-log plot. As we can see, as  $K$  increases, the effective real-utilization of the system also increases, until

around  $K = 1$ , after which the utilization turns flat. This seems related to the optimality of pure deadline monotonic scheduling. It will be interesting to study at what value of  $K$  gives the highest utilization for general task patterns under a specific scheduling policy. This is left as a future research topic.

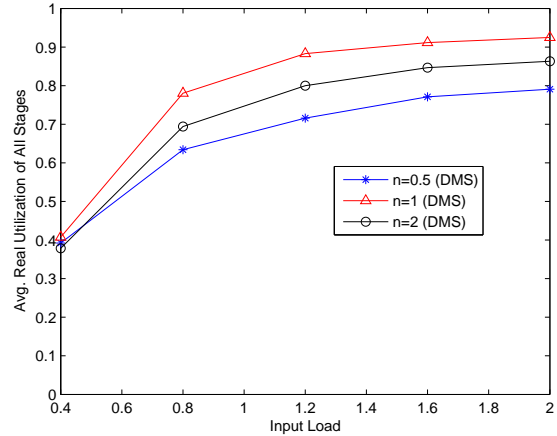


**Figure 5. Effect of scaling factor  $K$  on average pipeline utilization (DMS).**

It is also interesting to study whether non-linear functions that preserve monotonicity with priority (e.g.,  $x_i = KD_i^n$ ) might yield better bounds. For example, Figure 6 compares three choices of  $x_i$  that are valid for the DMS policy in that they are monotonically decreasing with DMS priority. These three are  $x_i = D_i^n$  for  $n = 0.5, 1$ , and  $2$ . It is shown that the exponent does have a clear effect on the schedulable region. In a way, the choice of  $x$  for finding the schedulable region of a scheduling policy is akin to the choice of a Lyapunov function for finding the stability region of a nonlinear control system. The question of finding the largest region becomes interesting. A contribution of this paper lies in providing a general framework for deriving new bounds that make it possible to eventually answer such questions. Interestingly enough, Figure 6 shows that the best results of the functions tested are obtained for  $n = 1$ . While we do not explain the above results analytically in this paper, we believe that the existence of a family of bounds can lead to future optimality results (within the family) that might further improve current schedulability bounds.

## 4.2 Feasible Region for Velocity Monotonic Scheduling

In addition to optimizing feasible regions for existing scheduling policies, universal feasible regions allow us to



**Figure 6. Effect of scaling factor  $n$  on average pipeline utilization (DMS).**

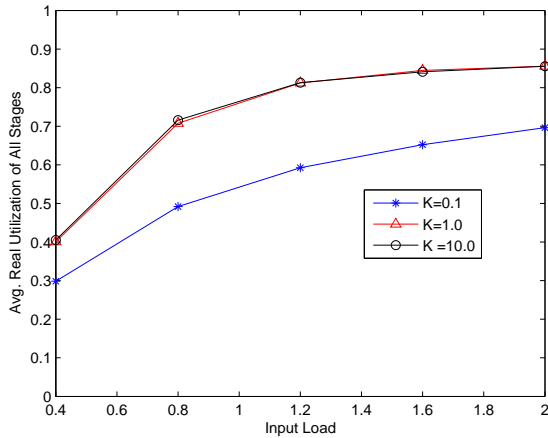
find bounds easily for new categories of scheduling policies for which utilization-bounds may or may not be a good match. In this subsection, we study the performance of admission control for Velocity Monotonic Scheduling (VMS). In VMS, priorities are set proportionally to the ratio of the end-to-end deadline to the number of stages traversed. Priority is higher when the deadline is lower and when the number of stages to be traversed is more. The intuition is that if two tasks have the same end-to-end deadline but a different number of stages to traverse, the one with more stages should be given a higher priority because it must move “faster”; hence the term, “velocity-monotonic”. Different from DMS, we set  $x_i = KD_i/l_i$ . The abstract load here is calculated as  $\sum_i \frac{C_{ij}}{KD_i/l_i}$ , where  $l_i = |G_i|$  is the number of stages task  $T_i$  will be processed on. From Equation (8), the bound is  $\frac{D_j}{D_i/l_i} = Kl_i$ . Figure 7 shows the effect of load and choice of  $K$  on the average stage utilization under VMS.

We see a similar effect of the scaling parameter  $K$  on the average utilization with DMS.

## 4.3 SJF Versus DMS and VMS

The proposed universal feasible region can also be applied to scheduling policies which used to be thought of as “non-real-time”. The Shortest Job First (SJF) scheduling policy is one of them. Figure 8 shows the average pipeline utilizations when admission control is enforced via the feasible region of DMS, VMS and SJF. The scaling parameter under each policy is chosen to be  $K = 1$ .

Two observations follow from Figure 8. First, we see that SJF’s performance is lower than those of DMS and VMS.



**Figure 7. Effect of different bounds under different input workloads (VMS)**

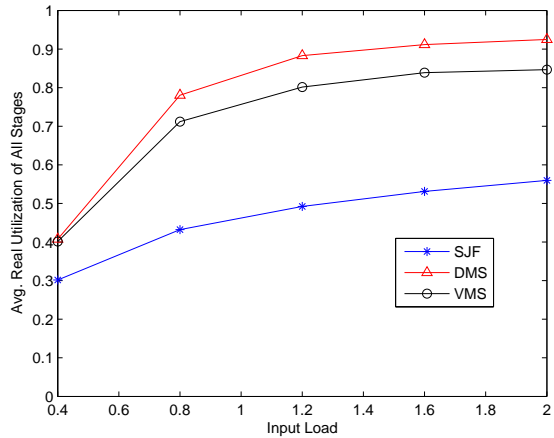
This is because SJF priorities are not correlated with task deadlines. Our admission controller under SJF does prevent missed deadlines. Note that the real-utilization of admitted tasks using our SJF bound is not zero. In other words, using a load metric different from utilization helped. It is well known that the utilization bound for SJF is zero. This is because SJF can miss deadlines for an arbitrarily low utilization. The abstract load metric used for admission control here is thus a better discriminator (than utilization) between schedulable and unschedulable workloads.

Surprisingly, we also notice from Figure 8 that the average real-utilization under VMS is lower than that under DMS. This is because tasks that traverse a smaller number of stages are unduly penalized priority-wise under VMS, hence limiting the number of higher-priority tasks that can be admitted. This suggests that perhaps a different combination of deadlines and distance (i.e., a combination other than their ratio) might be more appropriate for setting priorities of tasks in distributed systems.

#### 4.4 Practical Implementation Issues

As we have discussed in Section 3, when a new task arrives, the admission controller needs to find the minimum value of  $\frac{D_i}{x_i}$  over all tasks that have been admitted but not expired. Unfortunately, this is not a constant time check. We can approximate this check by constant time algorithms that are more practical in real-world applications. We compare the performance of these approximations in this subsection. Specifically, we consider the following three alternative admission control bound calculation algorithms:

1. Admission control based on  $\min_G \{\frac{D_i}{x_i}\}$ , where  $G$  is



**Figure 8. Comparison of average pipeline utilizations under different input workloads for DMS, VMS and SJF.**

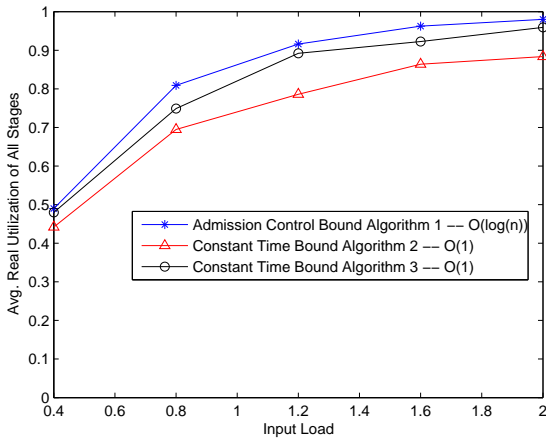
the set of all tasks that have been admitted but not expired.

2. Admission control based on  $\min_H \{\frac{D_i}{x_i}\}$ , where  $H$  is the set of all tasks have been admitted (could have departed already).
3. Admission control based on  $\min_P \{\frac{D_i}{x_i}\}$ , where  $P$  is the set of all tasks admitted since the last idle time of the stage (i.e., those tasks in the latest busy period).

Using data structures such as a heap, admission controller 1 is an  $O(\log(n))$  complexity algorithm, where  $n = |G|$ . Admission controller 2 is a constant time algorithm  $O(1)$ . It decreases the minimum bound value whenever the newly admitted task has a lower value of  $\frac{D_i}{x_i}$ , which is a constant-time operation. It does not change the bound when tasks depart. Admission controller 3 is also  $O(1)$ . It operates like 2, but resets the computed minimum whenever the processor becomes idle. (In contrast, a straightforward implementation of an exact admission controller would be linear in the number of tasks since the laxities of all tasks in the queue behind an inserted new arrival will be updated.)

Figure 9 shows the performance under the three admission control bound calculation algorithms for a single processor system. As before, the tasks are independently generated with exponentially distributed computation times. The scheduling policy is based on generic values of  $x_i$ , which are uniformly selected from  $[10, \dots, 100]$ . The arrival process is Poisson. We see that algorithm 1 gives the highest utilization under all input loads, while algorithm 2 gives the lowest utilization among the three alternatives. Algorithm 3 turns out to be a very good approximation.

Hence, admission control can be implemented efficiently in constant time.



**Figure 9. Comparison of different admission control bound calculation algorithms for single stage pipeline.**

The evaluation above illustrated several important points. First, universal feasible regions derived in this paper may improve the known schedulability bounds of existing scheduling policies by eventually finding better functions  $x$  that increase the recognized feasible regions. Second, it is possible to quickly derive schedulability bounds for new scheduling policies or those traditionally called “non-real-time”. A systematic way is presented to construct such new bounds, as opposed to deriving each from scratch using one’s creative imagination. Third, it is possible to implement admission controllers efficiently to guarantee satisfaction of the derived bounds. We hope that the mathematical tools developed in this paper will improve our understanding of schedulability and lead to better insights into feasible regions of scheduling policies in real-time systems.

## 5 Related Work

Utilization bounds for schedulability represent one of the simplest approaches for admission control in real-time systems. While not resource optimal, they offer the advantage of simplicity and computational efficiency. A plethora of bounds were proposed in real-time computing literature since the seminal work of Liu and Layland [11]. Most of these bounds were proposed for periodic task sets. For example, Kuo and Mok [10] improved the original Lui and Layland bound by collapsing harmonic tasks into one chain. They proved that the bound is a function of the number of harmonic chains and not the number of individual tasks.

The bound was further improved by considering information specific to the task set such as the actual values of task periods [7]. The authors of [14] refined this approach by introducing a design-time technique for computing a run-time bound when only periods (but not the execution times) are known *a priori*. The exact task execution times are plugged-in at run-time when the tasks arrive to yield that actual bound. A fault-tolerance extension was presented in [13], whose authors consider the overhead of failure recovery from a single fault. A utilization bound for a modified rate-monotonic algorithm which allows deferred deadlines was considered in [15]. An interesting extension was presented in [12], motivated by multimedia applications where frames have periodically recurrent but different sizes. Hence, the authors of [12] introduced a multi-frame periodic model, allowing a task to have one of multiple recurrent execution times. Improvements of the basic multiframe schedulability test were proposed in [8]. A hyperbolic bound was presented by Bini and Buttazzo in [5]. In their first-of-a-kind analysis, it was shown that a larger feasible region can be found if one bounds the product and not the sum of individual task utilizations.

Aperiodic tasks have usually been considered as an exception to periodic task systems. Several algorithms were proposed to convert aperiodics into some periodically scheduled form. For example, the slack server [17], the sporadic server [6] and the deferrable server [16] present different techniques for providing periodically replenished service budgets for aperiodic tasks.

More recently, researchers have devoted efforts to investigating aperiodic schedulability bounds independently of periodic tasks. The bound in [3] was the first to describe a constant time utilization-based schedulability test for a set of aperiodic tasks on a uniprocessor under fixed-priority scheduling. The bound was later extended to account for non-independent tasks with resource requirements [1]. A significant generalization was the extension of the bound from a uni-dimensional quantity to a multi-dimensional region for end-to-end schedulability of tasks traversing multiple stages of a pipeline [2]. Each dimension is the utilization of one of the traversed resources. In [4], Andersson proposed an exact admission controller algorithm for EDF systems subjected to aperiodic tasks.

An interesting direction, inspired by network calculus, was pioneered by [9], who proposed a framework for deriving feasible regions for arbitrary real time systems. Their framework uses the notion of *workload rate* to measure the resource demand within a period of time that is proportional to the deadline of the task. The framework is independent of the scheduling policy employed. It was shown to reduce to bounds derived in earlier literature for periodic tasks.

In contrast to the work mentioned above, this paper provides a general framework for deriving schedulabil-

ity bounds that are customized to different fixed-priority scheduling policies. There are three main contributions to this work. First, a general approach is suggested for choosing appropriate load metrics for purposes of deriving bounds for fixed-priority scheduling policies. Second, it is shown that an infinite family of bounds can be derived for any one scheduling policy. Finally, a general approach is described for deriving bounds on the new load metrics. Hence, efficient schedulability conditions can be found in a uniform manner. We believe that the aforementioned advances will improve understanding of schedulability and lead to efficient admission control policies for real-time systems.

## 6 Conclusions

This paper derives a general expression for a non-utilization-based schedulability bound. The bound applies to a generalized abstract load metric that is defined for a given system as a function of its particular scheduling policy. Hence, the result is a universal bound that is customized to any fixed-priority scheduling policy simply by choosing the corresponding load metric as described in this paper. We believe that our result will facilitate the generation of simple schedulability criteria for a number of current and future scheduling policies for which no utilization bounds exist. This includes scheduling policies that are better optimized for distributed systems where classical deadline monotonic and EDF scheduling are not optimal. While current results assume fixed-priority scheduling, an interesting challenge for future work is to develop a parallel result for dynamic priority systems. Also, while current results address aperiodic tasks, it is interesting to come up with analogous theory that specifically targets periodic execution. Aperiodic bounds, of course, are applicable to periodic tasks since they simply do not make any assumptions on task arrival times. However, it is expected that bounds derived specifically for periodic task systems will be tighter for that particular case. This will be the focus of the authors' future work.

## References

- [1] T. Abdelzaher and V. Sharma. A synthetic utilization bound for aperiodic tasks with resource requirements. In *15th Euromicro Conference on Real-Time Systems*, Porto, Portugal, July 2003.
- [2] T. Abdelzaher, G. Thaker, and P. Lardieri. A feasible region for meeting aperiodic end-to-end deadlines in resource pipelines. In *International Conference on Distributed Computing Systems*, Tokyo, Japan, March 2004.
- [3] T. F. Abdelzaher and C. Lu. Schedulability analysis and utilization bounds for highly scalable real-time services. In *IEEE Real-Time Technology and Applications Symposium*, Taipei, Taiwan, June 2001.
- [4] B. Andersson and C. Ekelin. Exact admission-control for integrated aperiodic and periodic tasks. In *Real-Time and Embedded Technology and Applications Symposium*, San Francisco, California, March 2005.
- [5] E. Bini, G. Buttazzo, and G. Buttazzo. A hyperbolic bound for the rate monotonic algorithm. In *13th Euromicro Conference on Real-Time Systems*, Delft, Netherlands, June 2001.
- [6] M. Caccamo and L. Sha. Aperiodic servers with resource constraints. In *IEEE Real-Time Systems Symposium*, London, England, December 2001.
- [7] X. Chen and P. Mohapatra. Lifetime behavior and its impact on web caching. In *IEEE Workshop on Internet Applications*, 1999.
- [8] C.-C. Han. A better polynomial-time schedulability test for real-time multiframe tasks. In *19th IEEE Real-Time Systems Symposium*, pages 104–113, Madrid, Spain, December 1998.
- [9] J.-C. L. Jianjia Wu and W. Zhao. On schedulability bounds of static priority schedulers. In *Real-Time and Embedded Technology and Applications Symposium*, San Francisco, California, March 2005.
- [10] T. W. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *IEEE Real-Time Systems Symposium*, December 1991.
- [11] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. of ACM*, 20(1):46–61, 1973.
- [12] A. K. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, October 1997.
- [13] M. Pandya and M. Malek. Minimum achievable utilization for fault-tolerant processing of periodic tasks. *IEEE Transactions on Computers*, 47(10):1102–1112, October 1998.
- [14] D.-W. Park, S. Natarajan, and A. Kanevsky. Fixed-priority scheduling of real-time systems using utilization bounds. *Journal of Systems and Software*, 33(1):57–63, April 1996.
- [15] W. K. Shih, J. Liu, and C. L. Liu. Modified rate-monotonic algorithm for scheduling periodic jobs with deferred deadlines. *IEEE Transactions on Software Engineering*, 19(12):1171–1179, December 1993.
- [16] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, January 1995.
- [17] S. R. Thuel and J. P. Lehoczky. Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In *Real-Time Systems Symposium*, pages 22–33, San Juan, Puerto Rico, December 1994.