Admission Control for Real-Time Demand-Curve Interfaces

Farhana Dewan Nathan Fisher
Department of Computer Science
Wayne State University
Detroit, Michigan, USA
farhanad@wayne.edu fishern@cs.wayne.edu

Abstract

Server-based resource reservation protocols (e.g., periodic and bandwidth-sharing servers) have the advantage of providing temporal isolation between subsystems co-executing upon a shared processing platform. However, for many of these protocols, temporal isolation is often obtained at the price of over-provisioned reservations. Other more fine-grained approaches such as real-time calculus (RTC) permit a precise characterization of the resources required by a subsystem via demand-curve interfaces. An important, unsolved challenge for subsystems specified by demand-curve interfaces is the development of admission control algorithms. Admission control is required to ensure that the demand-curve specified by the interface is never violated. In this paper, we take an initial step towards addressing this challenge by designing an admission controller for a simple, “single-step” demand curve. Our approach utilizes and extends techniques originally proposed for admission control in the settings of aperiodic jobs upon dedicated uniprocessors and bandwidth-sharing servers. In future work, we hope to use these techniques as the basis for constructing admission controllers for complex, arbitrary demand-curve interfaces.

1. Introduction

Recent real-time and embedded systems research is increasingly trending towards open environments [9] due to the ease of portability and integration of independently-developed subsystems upon a shared platform. Such systems are mostly implemented via resource partitions [14] which ensure temporal isolation among subsystems. In subsystem-based design, resource partitions are usually specified by interfaces, which serve as an abstraction of the total resource allocation requirements for the jobs of a subsystem. One potential implementation of a resource-partition interface is via a server-based model [10, 14, 16] where access to the physical resource is provided by the servers. Such implementations have strong temporal isolation properties (e.g., one subsystem is isolated from the potentially faulty temporal behavior of another subsystem). However, server-based systems often over-provision resources for individual subsystems leading to an inefficient global allocation of the processing resource. Demand-based interfaces (e.g., using real-time calculus [8, 17] or the demand-bound interface for multiprocessors [4]) have been proposed in which subsystem demand is precisely modeled by a demand-curve interface. However, the demand-based interface has the drawback of being computationally intractable. Furthermore, there are no known “policing” protocol for ensuring that a system does not violate its demand-curve interface at runtime [4].

To address this lack of interface-policing protocols for the more precise demand-curve interface models, we propose admission controller algorithms for subsystem with aperiodic jobs which will ensure temporal isolation by admitting tasks based on a demand bound function (DBF). As a first step towards this goal, we develop such an admission controller for a simple demand-curve interface. In on going work, we hope to generalize the admission controller for arbitrary demand curve. We do not address the orthogonal issues of how to generate and compose demand-curve interfaces in this paper (e.g., see Thiele et al. [17] for a discussion of these issues).

§Contribution. We propose a simple demand-curve interface model called single-step demand interface (SSDI) and give admission control algorithms of aperiodic jobs for our model. We propose a constant-time algorithm for aperiodic jobs with monotonic absolute
**2. Related Works**

A very recent paper by Kumar et al. [11] has proposed a Demand-Bound Server (DBS) for scheduling jobs according to a demand-curve interface. The proposed server successfully achieves the goal of providing temporal isolation between subsystems specified precisely by a demand-curve. However, the approach has fundamental differences with our proposed approach. First, Kumar et al. do not provide an admission controller for DBS; thus, if a subsystem incorrectly generates workload which exceeds the specified demand curve, the over-allocation error would only become apparent when the subsystem misses a deadline. In our approach, we seek to identify jobs that exceed a subsystem’s demand-curve interface before they are admitted to the scheduler. This approach permits a subsystem designer to identify and recover from potential over-allocation errors early before a temporal violation has occurred. The second fundamental difference is that Kumar et al. assume that jobs are scheduled in FCFS order. Our general approach makes no assumptions regarding the underlying execution of admitted jobs; we only guarantee that the set of admitted jobs does not violate the demand-curve interface. Finally, the technical details for both DBS and our approach have, so far, been specified for simple demand-curves. In Kumar et al. [11], while a general mathematical model is presented for arbitrary demand curves, the server algorithm has only been specified for a simple periodic demand function. In this paper, we assume a simple, single-step approximation of the demand curve is provided. Thus, there is great opportunity in providing algorithms for both techniques for handling more complex demand curves.

Linear-time exact admission tests for scheduling periodic and aperiodic jobs have been proposed in [1, 15]. Lipari and Buttazzo [13] proposed Bandwidth Sharing Server (BSS) algorithm which provides precise isolation between subsystems. In [12], this model has been extended to support aperiodic servers with different subsystem level schedulers. Using a similar approach, Andersson and Ekelin [3] proposed an $O(\log N)$ exact admission controller for aperiodic and periodic jobs. We extend their techniques in this paper to demand-curve interfaces.

**3. Model and Notation**

In this section, we introduce the job and interface models that characterize a subsystem.

**Job Model.** We assume that jobs can arrive aperiodically for a subsystem. Each aperiodic job $j_i$ is characterized by an arrival time $A_i$, a worst case execution requirement $E_i$, and a relative deadline $D_i$. We also denote the absolute deadline for $j_i$ as $d_i = A_i + D_i$. We completely specify a job $j_i$ by the tuple $(A_i, d_i, E_i)$. A job set $J = \{j_1, j_2, \ldots\}$ is a finite set of jobs indexed in order of increasing arrival time (i.e., for $1 \leq i < |J| : A_i \leq A_{i+1}$).

We let $J_t = \{j_1, j_2, \ldots\}$ denote job set that has arrived (and been admitted) into the subsystem by time $t > 0$. We assume that job parameters are revealed to a subsystem only upon job arrival; i.e., a subsystem does not have knowledge of future job arrivals. We call a job $j_i$ active at time $t$, if $t \in [A_i, A_i + D_i]$. Let $N$ be the maximum number of active jobs in the subsystem at any given time.

In general, we place no restriction on the parameters of jobs (except being non-negative numbers) that arrive in the system. However, as a starting point in our development of an admission controller, we first restrict ourselves to monotonic absolute deadline (MAD) [5] job arrivals. For MAD jobs, if job $j_i$ arrives before job $j_k$, then $j_i$’s absolute deadline must occur before $j_k$’s absolute deadline; more formally, $A_i \leq A_k \Rightarrow d_i \leq d_k$.

Figure 1 provides a visual depiction of a legal MAD job arrival sequence.

**Figure 1.** MAD jobs: $j_i$ has greater absolute deadline than $j_k$ for all $1 \leq \ell \leq n$.

Given a set of jobs $J$, we now describe how to accurately quantify the maximum workload over any interval [6].
Definition 1 (Demand) For any \( J \) and \( t_1, t_2 \in \mathbb{R} : 0 \leq t_1 < t_2 \), the function \( \text{demand}(J, t_1, t_2) \) represents the maximum cumulative execution requirement of all jobs in \( J \) that have both an arrival time and deadline in the interval \([t_1, t_2]\).

\[
\text{demand}(J, t_1, t_2) = \sum_{i \in J} E_i. \tag{1}
\]

Interface Model. In this paper, we consider the setting where a subsystem’s temporal interface is given by a simple, single-step, linear demand-curve.

Definition 2 (Single-Step Demand Interface (SSDI)) A single-step demand interface (SSDI), denoted by \( \Lambda = (\alpha, \Delta, \sigma) \), ensures that the total demand of the set of jobs \( J_i \) admitted by a subsystem for any \( t \geq 0 \) does not exceed the demand curve (called the demand-bound function) \( \text{dbf}(\Lambda, \cdot) \) for any subinterval of \([0, t]\). More formally, for all \( t_1, t_2 \in \mathbb{R} : 0 \leq t_1 < t_2 \leq t \),

\[
\text{demand}(J_i, t_1, t_2) \leq \text{dbf}(\Lambda, t_2 - t_1) \tag{2}
\]

where

\[
\text{dbf}(\Lambda, t) \overset{\Delta}{=} \begin{cases} 
0, & t < \Delta; \\
\sigma + \alpha(t - \Delta), & t \geq \Delta.
\end{cases} \tag{3}
\]

![Figure 2. Example of an SSDI curve.](image)

It is worthwhile to observe that the SSDI model is a generalization of the well-known bounded-delay resource model proposed by Mok et. al [14]. In the bounded-delay resource model, a resource is characterized by a pair \((\alpha, \Delta)\) where \( \Delta \) represents the “jitter” that an application executing upon the resource may experience and \( \alpha \) represents the “rate” of execution that the application is guaranteed over time. In future work, we will extend the model to permit the characterization of multiple steps of the usual demand-bound function [6].

Job/Interface Interaction. When a job is admitted into a subsystem, we must ensure that the total demand over any interval does not exceed the demand-curve specified by \( \text{dbf}(\Lambda, \cdot) \). However, by admitting a job into a subsystem, the newly-admitted job is consuming demand over an interval and is, thus, placing constraints on the subsystem’s ability to admit future jobs. We now present a function which quantifies how close the subsystem is to exceeding the SSDI curve with respect to all time intervals that end at the last admitted job’s deadline.

Definition 3 (Minimum Demand Difference) The minimum demand difference function \( \phi(J, \Lambda, t) \) quantifies the minimum difference between \( \text{dbf}(\Lambda, \cdot) \) and \( \text{demand}(J, \cdot, \cdot) \) for all intervals ending at some time \( t > 0 \). If the demand of \( J \) is zero for all such intervals, the function returns \( \infty \). More formally,

\[
\phi(J, \Lambda, t) \overset{\Delta}{=} \left\{ \begin{array}{ll}
\min_{t' : 0 \leq t' \leq t} \{ \psi(J, \Lambda, t', \cdot) \}, & \text{if } J \neq \emptyset; \\
\infty, & \text{if } J = \emptyset,
\end{array} \right.
\tag{4}
\]

where

\[
\psi(J, \Lambda, t_1, t_2) \overset{\Delta}{=} \text{dbf}(\Lambda, t_2 - t_1) - \mu_0^{-\infty}(\text{demand}(J, t_1, t_2))
\tag{5}
\]

and \( \mu^{-\infty}(x) \) equals \( -\infty \) if \( x \) is zero and \( x \) otherwise.

4. Important Properties of the Minimum Demand Difference Function

In this section, we prove important statements regarding the \( \phi \) function. These statements will be used to justify and prove the correctness of our proposed admission control algorithms in the subsequent sections. In our first lemma, we show that the \( \phi \) function may be utilized as an exact test of whether a job set satisfies the demand constraints of SSDI interface \( \Lambda \).

Lemma 1 For all \( t > 0 \), \( \phi(J, \Lambda, t) \geq 0 \), if and only if,

\[
\text{dbf}(\Lambda, t_2 - t_1) \geq \text{demand}(J, t_1, t_2), \quad \forall t_1, t_2 \in \mathbb{R} : 0 \leq t_1 < t_2.
\tag{6}
\]

Proof: \((\Rightarrow)\) If \( J \) is empty, the demand is zero for any choice of \( t_1 \) and \( t_2 \). Thus, since \( \text{dbf} \) is non-negative for all positive inputs, Equation 6 is trivially satisfied. If \( J \) is not empty, then \( \phi(J, \Lambda, t) \geq 0 \) for all \( t > 0 \) is equivalent to the following (by the Definition 3).

\[
\min_{t' : 0 \leq t' \leq t} \{ \psi(J, \Lambda, t', \cdot) \} \geq 0, \quad \forall t > 0 \
\]

\[
\iff \psi(J, \Lambda, t', \cdot) \geq 0, \quad \forall 0 \leq t' < t
\]

Equation 6 follows by substituting in the definition of \( \psi \) from Equation 5.

\((\Leftarrow)\) The other direction of the proof can be obtained by simply reversing the proof above.
The next two lemmas (Lemmas 2 and 3) show that for computing \( \phi \) we may restrict attention to time values corresponding to arrivals or deadlines of jobs of \( J \).

Lemma 2 The value of \( \phi(J, \Lambda, t) \) remains unchanged if we restrict in Equation 4 the values of \( t' \) considered in the min function to be from the set \( \{ A_i \mid j_i \in J \} \).

Proof: If \( J = \emptyset \), the lemma is clearly true. So, let us assume that \( J \neq \emptyset \). Let \( A_0 \) denote zero and \( A_{|J|+1} \) denote \( t \). Consider the partition of the interval \( [0, t] \) into subintervals \( [A_i, A_{i+1}] \) where \( 0 \leq i \leq |J| \). Assume that the min function in the right-hand side of Equation 4 achieves its minimum at some \( t' \notin \{ A_1, A_2, \ldots, A_{|J|} \} \). Thus, there exists some \( i : 0 \leq i \leq |J| \) such that \( t' \in (A_i, A_{i+1}) \). Now let us consider the time instant \( A_{i+1} \). By Equation 1, \( \text{demand}(J, t, t') = \text{demand}(J, A_{i+1}, A_i) \) as the set of jobs included in the summation of Equation 1 does not change for \( t' \) ranging over \( (A_i, A_{i+1}) \). Furthermore, since \( \text{dbf} \) is a non-decreasing function with respect to the interval-length argument and \( A_{i+1} > t' \), it must be that \( \text{dbf}(\Lambda, t - t') \geq \text{dbf}(\Lambda, t - A_{i+1}) \). Thus, the min function of Equation 4 also achieves the same minimum at \( A_{i+1} \in \{ A_1, A_2, \ldots, A_{|J|} \} \).

Lemma 3 For all \( t \geq 0 \), the minimum \( \phi(J, \Lambda, t) \) occurs at \( t \) corresponding to an element of the set \( \{ d_i \mid j_i \in J \} \).

Proof Sketch: The proof is symmetric to the proof of Lemma 2.

The next lemma shows how we inductively calculate minimum demand difference \( \phi \) when a new job \( j_i \) is admitted to the system. To calculate \( \phi \) for job set \( J \cup \{ j_i \} \), we need to consider all the intervals that potentially change the value of \( \phi \) upon the arrival of \( j_i \). By Lemmas 2 and 3, these intervals are \( [A_k, d_k] \) where \( A_k \leq A_i \) and \( [A_i, d_k] \) where \( d_k > A_i \). For all the new intervals with the right endpoint at least \( d_i \), the demand will increase by \( E_i \). For intervals with right endpoints prior to \( d_i \), the demand will remain the same. The lemma below describes how to update the minimum demand difference for each of these cases.

Lemma 4 Given \( J \) and \( j_i \) such that \( A_i \geq \max_{j \in J \{ A_k \}} \), if \( \phi(J, \Lambda, t) \geq 0 \) for all \( t \in \{ d_k \}_{k \in J} \), then

\[
\phi(J \cup \{ j_i \}, \Lambda, t) =
\begin{cases}
\min \{ \text{dbf}(\Lambda, t - E_i), \phi(J, \Lambda, t - A_i) + \alpha(d_i - \text{dlast}(J, j_i)) - E_i \}, & \text{if } t = d_i; \\
\min \{ \phi(J, \Lambda, t) - E_i, \text{dbf}(\Lambda, t - A_i) - E_i \}, & \text{if } t > d_i; \\
\phi(J, \Lambda, t), & \text{if } t < d_i;
\end{cases}
\]

(7)

where \( \text{dlast}(J, j_i) = \max_{j \in J \cup \{ j_i \}, k \in J} \{ d_k \} \). (We assume that \( \text{dlast}(J, j_i) \) equals zero if \( J \) is empty.)

Proof Sketch: Case \(( t > d_i )\): By Lemma 2, \( \phi(J, \Lambda, t) \) equals \( \text{min} \{ \psi(J \cup \{ j_i \}, \Lambda, A_i) \mid j_i \in J \} \). Thus, since \( A_i \geq \max_{j \in J \cup \{ j_i \}} \{ A_k \} \) and \( d_i < t \), \( \text{demand}(J \cup \{ j_i \}, A_i, t) = \text{demand}(J, A_i, t) + E_i \) for all \( j_i \in J \). Thus, by Equation 5, \( \psi(J \cup \{ j_i \}, \Lambda, A_i, t) = \psi(J, \Lambda, A_i, t) - E_i \). There is one new interval ending at \( t \) that must be considered according to Lemma 2: \( [A_i, t] \). The value of \( \psi(J \cup \{ j_i \}, \Lambda, A_i, t) \) equals \( \text{dbf}(\Lambda, t - A_i) - E_i \) (there are no jobs other than \( j_i \) completely contained in the interval). These two observations on the value of \( \psi \) imply the second case of Equation 7.

Case \(( t < d_i )\): Unlike the previous case, the demand over \( [A_k, d_k] \) does not change under the addition of \( j_i \) as \( t < d_i \). Thus, \( \psi(J \cup \{ j_i \}, \Lambda, A_i, t) \) equals \( \psi(J, \Lambda, A_i, t) \) for all \( j_i \in J \). This implies the third case of Equation 7.

Case \(( t = d_i )\): We must consider the value of \( \psi \) over all intervals \( [A_k, d_k] \) such that \( j_i \in J \cup \{ j_i \} \). For \( [A_k, d_k] \) clearly \( \psi(J \cup \{ j_i \}, \Lambda, A_i, d_k) \) is \( \text{dbf}(\Lambda, d_k) - E_i \). For \( [A_k, d_k] \) such that \( j_i \in J \), we first observe that \( d_i - A_k \geq \Delta \). If this was not true, then \( \text{dbf}(\Lambda, d_k) \) equals zero, since \( d_k \leq d_i - A_k < \Delta \). However, since \( \phi(J, \Lambda, A_k, t) \geq 0 \), this implies that \( 0 \geq \text{demand}(J, A_k, d_k) \geq E_k \) which is a contradiction. Thus, by Equation 3, we may rewrite \( \text{dbf}(\Lambda, d_i - A_k) \) as

\[
\alpha(d_i - A_k - \Delta) + \sigma = \alpha(d_i - A_k) - \Delta + \sigma + \alpha(d_i - \text{dlast}(J, j_i)) = \text{dbf}(\Lambda, d_i - \text{dlast}(J, j_i)) + \alpha(d_i - \text{dlast}(J, j_i))
\]

since \( \text{dlast}(J, j_i) - A_k \geq D_k \geq \Delta \). Using the new expression for \( \text{dbf}(\Lambda, d_i - A_k) \) in \( \psi \), we obtain

\[
\psi(J \cup \{ j_i \}, \Lambda, A_i, d_k) = \text{dbf}(\Lambda, d_i - A_k) - \text{demand}(J \cup \{ j_i \}, A_k, d_k) = \text{dbf}(\Lambda, d_i - \text{dlast}(J, j_i)) + \alpha(d_i - \text{dlast}(J, j_i)) - \text{demand}(J \cup \{ j_i \}, A_k, \text{dlast}(J, j_i)) - E_i = \psi(J \cup \{ j_i \}, \Lambda, A_k, \text{dlast}(J, j_i)) + \alpha(d_i - \text{dlast}(J, j_i)) - E_i.
\]

The second to last expression is due to the fact that \( j_i \) is the only job of \( J \cup \{ j_i \} \) that arrives at \( A_k \) but has a deadline after \( \text{dlast}(J, j_i) \). Taking the minimum over all possible \( A_k \) values implies the first case of Equation 7.

5. Admission Controller for MAD Jobs

In this section, we propose a constant-time admission control algorithm for MAD job arrivals (see Section 3). Assume at time \( t \), \( n \) jobs have arrived and been admitted to the subsystem. The MAD property implies
that $A_1 + D_1 \leq A_2 + D_2 \leq \cdots \leq A_n + D_n$. Assume that a new job $j_i=(A_i,D_i,E_i)$ arrives in the system with $d_i \leq d_i$ (Figure 1). Job $j_i$ will be accepted if and only if the system can meet total demand over any interval after adding $j_i$, that is, the demand does not exceed the demand-curve specified by SSDI $\Lambda=(\alpha,\Delta,\sigma)$. In our admission controller for MAD jobs, along with checking the demand for new jobs, we keep track of minimum demand difference for future admissions. Our main observation is that we can easily calculate minimum demand difference for newly-admitted job $j_i$.

The admission control algorithm for MAD jobs is given in Figure 5. In this algorithm, at any time point we keep track of two variables: minimum demand difference $md$ and latest absolute deadline $d$ among submitted jobs in the subsystem. When job $j_i$ arrives, we calculate the DBF for the new interval from latest absolute deadline to new job’s absolute deadline (the interval will always be greater or equal 0, since the jobs follow MAD property). The demand for the new job is its execution $E_i$. From these two terms we calculate minimum demand difference ($z$) in MAD-ADMISSIONCONTROL for job $j_i$ and admit the job to the system if $z$ is greater than zero. Finally, if the $j_i$ gets through the admission controller, we update the variables $d$ and $md$.

**MAD-INITIALIZE()**
1 ▷ Let $md$ is the current minimum difference in the system and $d$ is latest absolute deadline among admitted jobs.
2 $d \leftarrow 0$.
3 $md \leftarrow \infty$.

**MAD-ADMISSIONCONTROL($j_i$)**
1 $x \leftarrow \text{dbf}(\Lambda, D_i) - E_i$
2 $y \leftarrow \alpha(d_i - d) - E_i$
3 $z \leftarrow \min\{x, md + y\}$
4 if $z \geq 0$
5 Admit $j_i$.
6 ▷ Update latest deadline, minimum difference.
7 $d \leftarrow d_i$
8 $md \leftarrow z$
9 else
10 Reject $j_i$.

\[ \text{Figure 3. Pseudo-code for admission control of MAD jobs.} \]

**Correctness.** We now use the Lemmas presented in Section 4 to show the correctness of our admission controller for MAD jobs. We first prove a lemma to show the correspondence between the variable $md$ and the minimum difference $\phi$ for newly-admitted job $j_i$.

**Lemma 5** Let $j_1,j_2,\ldots$ be a sequence of admitted jobs that arrive in the subsystem. After the $i$’th invocation of MAD-ADMISSIONCONTROL upon each job arrival, $md$ equals $\phi(j_1,j_2,\ldots,j_i,\Lambda,d_i)$. $d$ equals $d_i$, and $\phi(j_1,j_2,\ldots,j_i,\Lambda,t) \geq 0$ for all $t > 0$.

**Proof:** **Initialization:** Let us use the convention that $j_0$ is a dummy job that has deadline at time zero. Initially, $d = 0$, $md = \infty$, and the job set is empty. By Equation 4 of Definition 3, $\phi(\emptyset,\Lambda,t) = \infty$ for all $t > 0$ and the lemma initially holds.

**Induction:** Assume the lemma continues to hold after $i-1$ invocations. By Lemma 4, we may calculate $\phi(j_1,j_2,\ldots,j_{i-1},j_i,\Lambda,d_i)$ by

\[
\min \left\{ \text{dbf}(\Lambda, D_i) - E_i, \phi(j_1,j_2,\ldots,j_{i-1},\Lambda,d_{i-1}) + \alpha(d_i - d_{i-1}) - E_i \right\}.
\]

As $md \geq 0$, $\phi(j_1,j_2,\ldots,j_{i-1},j_i,\Lambda,d_{i-1})$ and $\phi(j_1,j_2,\ldots,j_{i-1},\Lambda,t)$ are non-negative for all $t > 0$ at the start of the $i$’th invocation and Lines 1, 2, and 3 update $md$ according to the above expression and $md$ satisfies the lemma. The variable $d$ is set to $d_i$. The only value that $\phi$ changes for is $t = d_i$, (according to Lemma 4 and the fact that there are no jobs with later deadline than $j_i$). Since Line 4 checks that $md \geq 0$, this implies $\phi(j_1,j_2,\ldots,j_{i-1},j_i,\Lambda,d_{i-1}) \geq 0$ for all $t > 0$. Thus, the lemma continues to hold after the invocation MAD-ADMISSIONCONTROL($j_i$).

We may now formally show that our admission controller is an exact test for determining whether an admitted job will violate the demand-curve constraints of SSDI $\Lambda$.

**Theorem 1** MAD-ADMISSIONCONTROL($j_i$) will admit job $j_i$ to a subsystem specified by $\Lambda$, if and only if,

\[
\text{demand}(J_{A_i} \cup \{j_i\}, t_1, t_2) \leq \text{dbf}(\Lambda, t_2 - t_1), \forall 0 \leq t_1 \leq t_2.
\]

(8)

where $J_{A_i}$ is the set of admitted jobs upon $j_i$’s arrival.

**Proof Sketch:** ($\Rightarrow$) We prove this direction by contrapositive. Thus, assume that $j_i$ is rejected by MAD-ADMISSIONCONTROL($j_i$). By Lemma 5, the variable $md$ corresponds to $\phi(J_{A_i},\Lambda,d_{J_{A_i}})$ and $\phi(J_{A_i},\Lambda,t)$ for all $t > 0$ at the beginning of the invocation of MAD-ADMISSIONCONTROL($j_i$). Thus, by Lemma 4, $\phi(J_{A_i} \cup \{j_i\},\Lambda,d_i)$ equals

\[
\min \left\{ \text{dbf}(\Lambda, D_i) - E_i, \phi(J_{A_i},\Lambda,d_{J_{A_i}}) + \alpha(d_i - d_{J_{A_i}}) - E_i \right\}.
\]
Since $j_i$ is rejected, it must have failed the condition of Line 4. This implies the above expression is less than zero. For either value of the minimum, it may easily be shown that this implies the negation of Equation 8 of the lemma.

($(\Rightarrow)$) This direction follows immediately from Lemma 5. $lacksquare$

**Complexity.** Clearly, MAD-ADMISSIONCONTROL has $O(1)$ time complexity. Each time a new job is admitted to the system, the minimum demand difference is updated for the arriving job only. We do not need to update the minimum demand difference for other jobs as they are unaffected. However, as we will see in the next section, this observation no longer holds for general job arrivals.

6. General Admission Controller

![Aperiodic jobs: $j_i$ has smaller absolute deadline than $j_{n-1}$.](image)

In this section we relax the constraint of the MAD property for the aperiodic jobs in the system; that is, jobs may arrive in the system at any order of deadline (Figure 4). For each new job arrival, the admission controller must check the demand of all preceding and succeeding jobs considering the execution requirement of the new job. For admission control purposes, a data structure containing the admitted jobs needs to be maintained. The data structure we propose for this paper is a variant of the sorted data structures proposed by Andersson and Ekelin [3] and Lipari and Baruah [12]. For this report, in the interest of space, we present the linked-list implementation that requires $O(N)$ time for determining whether a job should be admitted where $N$ is an upper bound on the number of active jobs in the subsystem at any given time. However, as we will briefly discuss later, this data structure can be enhanced using the techniques of Andersson and Ekelin to reduce the time complexity for admission control to $O(\log N)$.

In Figure 6, we propose an admission control algorithm for general aperiodic jobs (i.e., no constraint on arrivals/deadlines).

**Algorithm Description.** We now give an informal description of the workings of our algorithm. A formal proof of correctness of the algorithm can be derived in a similar manner to the MAD admission control algorithm and mostly follows from Lemma 4. However, in the interest of space constraints, we defer a formal proof for a future extended version of this paper. The linked-list data structure stores, for each admitted, active job $j_i$ in the subsystem, the attributes $(A_i, d_i, E_i)$ and two additional variables local minimum demand difference $lmd_i$ and successive minimum demand difference $smd_i$. The variable $lmd_i$ keeps track of the minimum demand difference at time $d_i$. The variable $smd_i$ stores the minimum demand difference for the jobs with deadline after $d_i$ (i.e., the $lmd_i$ for the succeeding jobs of $j_i$ in the sorted list). The data structure is initialized by INITIALIZE$(\cdot)$ to contain two dummy nodes: a leftmost job $j_0 = (0, 0, 0)$ and a rightmost job $j_m = (\infty, \infty, 0)$. The $lmd$ and $smd$ parameters for these nodes are set to $\infty$ (Figure 6(d)).

When a new job $j_i$ arrives in the system, UPONJOBARRIVAL$(j_i)$ finds the appropriate insertion position (according to the $d_i$ parameter) in the data

---

**INITIALIZE$(\cdot)$**
1 $\triangleright$ Insert two dummy nodes into list ordered in ascending $d_i$.
2 Insert $j_0$ with $d_0 = 0$, $lmd_0 = \infty$, and $smd_0 = \infty$.
3 Insert $j_m$ with $d_m = \infty$, $lmd_m = \infty$, and $smd_m = \infty$.

**UPONJOBARRIVAL$(j_i)$**
1 $\triangleright$ Lookup the potential position for $j_i$ in list.
2 Find $j_i$’s position between jobs $j_k$ and $j_{k+1}$.
3 $x \leftarrow dbf(A, D_k) - E_i$
4 $lmd_i \leftarrow \min\{lmd_k + \alpha(d_i - d_k) - E_i, x\}$
5 $smd_i \leftarrow \min\{lmd_{k+1}, smd_{k+1}, dbf(A, d_{k+1} - A_i)\} - E_i$
6 if $lmd_i \geq 0$ and $smd_i \geq 0$
7 Admit $j_i$.
8 Insert $j_i$ with parameters $d_i$, $lmd_i$ and $smd_i$ into list.
9 Set $smd_i \leftarrow \min\{smd_i, smd, lmd\}$ for all nodes $j_i(\neq j_0)$ preceding $j_i$ in the list.
10 Set $lmd_i \leftarrow \min\{lmd_i, dbf(A, d_i - A_i)\} - E_i$
11 else
12 Reject $j_i$.

**UPONJOBDUELINE$(j_i)$**
1 Update $j_0$ with $d_0 = d_i$ and $lmd_0 = lmd_i$.
2 Delete $j_i$ from list.
structure in $O(N)$ time. Assume it is between $j_k$ and $j_{k+1}$ (i.e., $d_k \leq d_i \leq d_{k+1}$). The variable $lmd_i$ for job $j_i$ is calculated from preceding job's local minimum demand difference ($lmd_k$) and the demand and demand interface difference for this job (similar to $md$ of ADMISSIONCONTROL). The variable $smd_i$ is calculated by taking minimum of the succeeding job's ($j_{k+1}$) local minimum demand difference ($lmd_{k+1}$) and successive minimum demand difference ($smd_{k+1}$). The new job $j_i$ is admitted if both $lmd_i$ and $smd_i$ are greater than 0. Intuitively, the condition $lmd_i \geq 0$ determines that if $j_i$ is added, then its deadline will be met by the demand interface and the condition $smd_i \geq 0$ verifies that after adding $j_i$, all the succeeding jobs will also meet their deadline. In this way, the algorithm ensures the demand of the system is met by the SSDI curve.

When job $j_i$ is admitted and inserted into the subsystem (and the linked list), the demand for the succeeding jobs is increased by the amount of its execution $E_i$. Thus, the local minimum difference ($lmd$) and succeeding minimum difference ($smd$) values for jobs succeeding $j_i$ in the sorted list must also be reduced by $E_i$. The demand for preceding jobs of $j_i$ is not increased, since they have deadlines before $d_i$, which implies that the $lmd$ values for the jobs preceding $j_i$ in the sorted list will remain unchanged after inserting $j_i$. However, the successive minimum difference for such preceding jobs should be updated, if this is less than $smd_i$ or $lmd_i$, to reflect the change in minimum difference of successive jobs after inserting $j_i$. The updates to both the preceding and succeeding jobs requires $O(N)$. When the deadline of a job is elapsed, it can be removed from the data structure in $O(1)$ time according to UPONJOBDEADLINE($j_i$).

Example. In Figure 6, we show the operation of algorithm UPONJOBARRIVAL using the job arrival scenario given in Figure 6(a). The single-step demand interface for the subsystem is $\Lambda = (\alpha, \Delta, \sigma) = (0.5, 5, 2)$ (dbf is shown in Figure 6(b)). The next figure (c) shows the node structure of the linked list at which the jobs are stored in the order of their deadline. We initialize the list with dummy-nodes $j_0$ and $j_\infty$ (Figure 6(d)) and insert them to the list ($j_0$ is the leftmost node and $j_\infty$ is the rightmost node) in the order of deadline. We assume that $\Lambda$ is implicit in the dbf calculations below.

Upon the arrival of $j_1 = (0, 20, 5)$, it is inserted to the list between $j_0$ and $j_\infty$. With $lmd_1 = \min\{dbf(20 - 0) - 5, \infty + 0.5 \ast (20 - 0) - 5\} = 4.5$, and $smd_1 = \min\{\infty, \infty - 0\} - 5 = \infty$ (Figure 6(e)). The next job $j_2 = (5, 30, 7)$ is inserted to the right of $j_1$ since its absolute deadline $d_2 = 30 > d_1 = 20$. For this job, $lmd_2 = \min\{dbf(30 - 5) - 7, lmd_1 + 0.5 \ast (30 - 20) - 7\} = \min\{5, 4.5 + 5 - 7\} = 2.5$, and $smd_2 = \infty$. After inserting $j_2$, all the preceding nodes (except the dummy node $j_0$) are updated; thus, $smd_1$ is set to $\min\{\infty, 2.5, \infty\} = 2.5$ (Figure 6(f)). The next job is $j_3 = (8, 15, 2)$, which should be inserted before $j_1$ in the current list. We calculate $lmd_3 = \min\{dbf(15 - 8) - 2, \infty + 0.5 \ast (15 - 0) - 2\} = 1$, and $smd_3 = \min\{4.5, 2.5, dbf(20 - 8)\} - 2 = 0.5$. Since both of these values are greater than 0, we insert $j_3$, and update succeeding nodes ($j_1$ and $j_2$) by setting $lmd_1 = \min\{4.5, dbf(20 - 8)\} - 2 = 2.5$, $smd_1 = \min\{2.5, dbf(30 - 12)\} - 2 = 0.5$, $lmd_2 = 0.5$, and $smd_2 = \infty$ (Figure 6(g)). Similarly, job $j_4$ and $j_5$ are inserted. The algorithm deletes a job when its deadline has elapsed and updates $d_0$ of $j_0$ with the deleted job’s deadline. Figure 6(h) shows the condition of the list after deleting jobs $j_1,j_2$ and inserting jobs $j_4,j_5$.

Complexity. The time complexity of the algorithm depends on the data structure used to store jobs. As we have mentioned for our specified linked-list description, the lookup and update operations will take $O(N)$ time. If we store the jobs in a self-balancing binary tree (e.g. AVL tree [2] or red-black tree [7]), the lookup
and insert operations require $O(\log N)$ time. However, the update operations followed by the insertion of new node may take $O(N)$ time to update preceding or succeeding job’s minimum differences ($lmd$ and $smd$ values). We can reduce the complexity of update operations by applying the techniques used in [3], in which admitted jobs are kept in an AVL tree and a lazy update is performed by keeping update “notes” for a subtree, the nodes of which have similar updates. The details on how to apply and merge update notes to the data structure will be nearly identical to [3] and will be provided in an extended version of this paper.

7. Conclusion

Our research goal is to obtain an efficient admission controller for arbitrary demand-curve interfaces. Current, demand-bound servers permit temporal isolation for a given demand-curve interface, but do not provide admission control within a subsystem to ensure that jobs that violate the demand-curve interface are not admitted (and thus potentially causing a timing violation within a subsystem). Towards this goal, we have taken an initial step by proposing a simple demand interface model called the single-step demand interface (SSDI) and deriving efficient admission control algorithms for a subsystem with this demand-based interface. We provide a constant-time admission controller for jobs with MAD arrivals and generalize it for general aperiodic jobs. Furthermore, we also outline how admission control for SSDI interfaces may be done in $O(\log N)$ for general job arrivals. In our ongoing research, we are attempting to extend these initial results to more complex, multi-step demand interfaces. Such admission controllers would be invaluable for policing systems designed according to demand/supply-curve interfaces such as the real-time calculus models.

References