A Feasible Region for Meeting Aperiodic End-to-end Deadlines in Resource Pipelines *

Tarek Abdelzaher
Department of Computer Science
University of Virginia
Charlottesville, VA 22904
e-mail: zaher@cs.virginia.edu

Gautam Thaker and Patrick Lardieri
Advanced Technology Laboratories
Lockheed Martin
Cherry Hill, NJ 08002
e-mail: {gthaker, plardieri}@atl.lmco.com

Abstract

This paper generalizes the notion of utilization bounds for schedulability of aperiodic tasks to the case of distributed resource systems. In the basic model, aperiodically arriving tasks are processed by multiple stages of a resource pipeline within end-to-end deadlines. The authors consider a multi-dimensional space in which each dimension represents the instantaneous utilization of a single stage. A feasible region is derived in this space such that all tasks meet their deadlines as long as pipeline resource consumption remains within the feasible region. The feasible region is a multi-dimensional extension of the single-resource utilization bound giving rise to a bounding surface in the utilization space rather than a scalar bound. Extensions of the analysis are provided to non-independent tasks and arbitrary task graphs. We evaluate the performance of admission control using simulation, as well as demonstrate the applicability of these results to task schedulability analysis in the total ship computing environment envisioned by the US navy.

Keywords: Real-time scheduling, schedulability analysis, utilization bounds, aperiodic tasks, total ship computing environment.

1 Introduction

This paper extends the notion of utilization bounds for schedulability to the case of distributed resource systems. Tasks arrive to the system and require multiple stages of processing on multiple independent resources (e.g., CPUs). We initially focus on the case of a resource pipeline, in which each task is given by a single precedence-constrained chain of subtasks, one per stage. We derive a feasible region for the end-to-end task schedulability as a function of the instantaneous resource utilization of different pipeline stages.

Extensions of the analytic framework to non-independent tasks (i.e., those with critical sections) and arbitrary task graphs are then provided.

The study of utilization bounds, such as the Liu and Layland bound [13], is attractive in that they give rise to very simple admission control tests in real-time systems. Most utilization bounds described in previous literature are for variations of the periodic task model. Recently, the first bound for aperiodic tasks was developed by Abdelzaher and Lu [3]. The bound was extended to multiprocessors [1] as well as to non-independent tasks and arbitrary scheduling policies [2] on a single resource. The bound allows a fast admission control test to be performed and is shown not to underutilize the processor. This paper extends prior bounds for schedulability of aperiodic tasks to the case of distributed resource systems. It presents a multi-dimensional schedulability bound given by a surface in the resource utilization space, where each resource contributes one dimension. The surface reduces to a single point in the case of a single-resource system. This point is the uniprocessor aperiodic utilization bound derived in [2]. The bounding surface derived in this paper leads to an efficient on-line admission control test that is \( O(N) \), where \( N \) is the number of independent resources (CPUs) needed by the task. In particular, the complexity of admission control is independent of the number of tasks in the system, which is a great advantage in systems that expect a very high workload (e.g., thousands of concurrent tasks). The analysis presented in the paper, while geared towards aperiodic tasks, also provides sufficient (albeit pessimistic) feasibility conditions for periodic workloads, since periodic arrivals are a special case of aperiodic ones. In particular, the admission controller can reserve a fraction of utilization for periodic tasks while assigning the rest dynamically to aperiodics.

The motivation for considering aperiodic schedulability bounds is twofold. First, many applications include an increasing fraction of aperiodic tasks. Examples range from next generation military avionics and ship computing systems to current open systems that consider QoS guaran-
tees such as web and database servers. Aperiodic utilization bounds allow guarantees on timeliness to be made efficiently at run-time for accepted tasks. Second, many periodic task systems exhibit a significant amount of jitter that may reduce the minimum interarrival time of successive invocations to zero. In the absence of jitter control mechanisms, this poses challenges to traditional analysis based on a sporadic model. Instead, a schedulability theory based on an aperiodic model may allow streams of periodic tasks to be guaranteed in the presence of large jitter.

Extension of utilization bounds to the case of resource pipelines is motivated by the fact that in most current real-time systems tasks are processed on multiple stages. For example, requests on a web server must be processed by both the front-end and several tiers of back-end servers that execute the business logic and interact with database services. Similarly, sensory data from an onboard radar in a battleship must be processed on multiple stages such as tracking, target classification, threat assessment, and potentially weapon engagement, which are implemented on different machines interconnected by a LAN. Several tools exist in periodic task literature to analyze the performance of resource pipelines. Such tools typically require the introduction of intermediate (per-stage) deadlines, then performing schedulability analysis on each pipeline stage, or resort to offline response-time analysis that takes into account periods and jitter. In contrast, our end-to-end aperiodic approach to schedulability analysis is different in that it does not make task periodicity assumptions, and does not require intermediate per-stage deadlines.

The remainder of this paper is organized as follows. Section 2 describes the proposed task model. Section 3 details the proof of the bounding utilization surface. The performance of admission control based on the aforementioned surface bound is presented in Section 4. Section 5 discusses a prospective application. Section 6 presents related work. The paper concludes with Section 7 which summarizes the results and presents avenues for future work.

2 Problem Statement and Contributions

In the bulk of this paper, we consider the basic case of a distributed processing system composed of \( N \) stages. We first analyze independent tasks that arrive aperiodically and must be processed by these stages sequentially. Each incoming task \( T_i \) is defined by its arrival time, \( A_i \), at which it arrives to the first stage, a relative end-to-end deadline \( D_i \) within which it should leave the pipeline, and a subtask computation time \( C_{ij} \) for each stage \( j, 1 \leq j \leq N \). Subtasks are arranged into a single precedence-constrained chain. At any time \( t \), we call the set of tasks that have arrived but whose deadlines have not expired, the set of current tasks, \( S(t) \). Hence, \( S(t) = \{ T_i | A_i \leq t < A_i + D_i \} \). We define the synthetic utilization, \( U_j(t) \) of stage \( j \) as \( \sum_{T_i \in S(t)} C_{ij} / D_i \), which is the sum of individual subtask utilization needs (on this stage) accrued over all current tasks. The question addressed in this paper is to find a feasible condition on schedulability that is a function \( g(U_1, \ldots, U_N) \) of resource utilizations, such that all deadlines are met as long as \( g(U_1, \ldots, U_N) \leq B \), where \( B \) is a constant. The first contribution of this paper is to prove that under deadline-monotonic scheduling all end-to-end deadlines are met if:

\[
\sum_{j=1}^{N} \frac{U_j(1 - U_j/2)}{1 - U_j} \leq 1
\]  

(1)

The result is then extended to arbitrary fixed-priority scheduling policies and to non-independent tasks. A fixed-priority scheduling policy, in the context of aperiodic tasks, is defined as one in which task priority is fixed across all pipeline stages, and is not a function of the arrival-time of the task. Hence, for example, EDF is not a fixed-priority scheduling policy, as the priority of task \( T_i \) is equal to the absolute deadline \( A_i + D_i \), which is a function of task arrival time. To extend the feasible region to an arbitrary fixed-priority scheduling policy, a parameter \( \alpha \) is defined. Intuitively, \( \alpha \) represents the degree urgency inversion in the priority assignment observed under the given scheduling policy. An urgency inversion occurs when a less urgent task (i.e., one with a longer relative deadline) is given an equal or higher priority by the scheduler compared to a more urgent task. Let us call them task \( T_{ki} \) and \( T_{lo} \) respectively. Formally, we define \( \alpha = \min_{r_{ki} \geq r_{lo}} D_{lo} / D_{ki} \), which is the minimum relative deadline ratio across all priority-sorted task pairs. In the absence of urgency inversion (e.g., for deadline monotonic scheduling), \( \alpha = 1 \). Otherwise, \( \alpha < 1 \). For example, if priorities are assigned randomly, \( \alpha = D_{least} / D_{most} \), where \( D_{least} \) and \( D_{most} \) are the minimum and maximum relative deadlines in the task set respectively. We show that for an arbitrary fixed-priority scheduling policy the feasible region is given by:

\[
\sum_{j=1}^{N} \frac{U_j(1 - U_j/2)}{1 - U_j} \leq \alpha
\]

(2)

Finally, if tasks have critical sections, the subtask of task \( T_i \) running on stage \( j \) might block for an amount \( B_{ij} \) over a lower-priority task holding a resource. This blockage can be bounded by using the priority ceiling protocol [9]. We define the worst-case blockage at stage \( j \) as \( \gamma_j = \max_i B_{ij} / D_i \). We show that the feasible region in this case is given by:

\[
\sum_{j=1}^{N} \frac{U_j(1 - U_j/2)}{1 - U_j} \leq \alpha(1 - \sum_{j=1}^{N} \gamma_j)
\]

(3)

1Also called instantaneous utilization.
In the case of a single resource, the feasible regions described above reduce to the single processor utilization bounds derived by the authors in prior publications [3, 2]. For example, observe that in the case of a single stage, the deadline-monotonic bound given in Equation (1) reduces to $U(1 - U/2)/(1 - U) \leq 1$. Solving the above equation for $U$, we get $U \leq 1/(1 + \sqrt{1/2})$, which is the single-processor utilization bound derived in [3].

Finally, while the above regions are derived for a resource pipeline, we later show how they can be extended to arbitrary directed acyclic task graphs by considering the worst case path, as will be described in the paper.

3 Deriving the Feasible Region

In this section, we first derive the basic feasible region for independent tasks arriving at a resource pipeline. We then extend this result to a more general task model that accommodates critical sections and arbitrary directed acyclic task graphs. The reader is cautioned that the proofs below have been simplified for readability and abbreviated to fit within length constraints. The goal is to highlight the main ideas of the derivation, rather than present the entire proof.

3.1 The Basic Resource Pipeline

To derive the feasible region, we consider an arbitrary task $T_n$, which arrives at the first stage of a pipeline at time $A_n$ and must depart the last stage by $A_n + D_n$. The arrival time of task $T_n$ at stage $j$ denotes the time subtask $T_{nj}$ is first entered into the ready queue. The departure of time of $T_n$ from stage $j$ denotes the time $T_{nj}$ finishes execution at that stage. Due to precedence constraints, the departure time of a task at stage $j$ is equal to the arrival time of the task at stage $j + 1$. Let the time task $T_n$ spends at stage $j$ be denoted $L_j$, which is the interval between its arrival time and departure time at stage $j$. Thus, for the task to be schedulable it must be that $\sum_j L_j \leq D_n$. To obtain the feasible region, we lower-bound the maximum synthetic utilization at each stage $j$, subject to the condition that task $T_n$ experiences delay $L_j$ at that stage. Let the resulting lower bound be $U_j$. By definition of $U_j$, in any task pattern in which $T_n$ experiences delay $L_j$ at stage $j$, the synthetic utilization of stage $j$ must equal or exceed $U_j$ at least at one point in time. Conversely, if the synthetic utilization never exceeds $U_j$ then the task delay on stage $j$ is upper-bounded by $L_j$. This is expressed as a general relation between $L_j$ and $U_j$ in the form:

$$L_j = f(U_j)$$ (4)

Finally, we invoke the observation that for task $T_n$ to be schedulable, it must be that $\sum_j L_j \leq D_n$. Substituting from Equation (4) for $L_j$ in this summation, we get the equivalent condition $\sum_j f(U_j) < D_n$. The above inequality defines the feasible region. If the inequality is satisfied, it must be that the combination of per-stage utilizations is such that the end-to-end deadline is not exceeded. In the following we present the main theorem of this paper; namely, one that defines the shape of the function $f(U_j)$ in Equation (4). We call it, the *stage delay theorem*.

**Theorem 1** (the stage delay theorem): If task $T_n$ spends time $L_j$ at stage $j$, and $U_j$ is a lower bound on the maximum synthetic utilization at that stage, then:

$$L_j = \frac{U_j(1 - U_j/2)}{1 - U_j} D_{\text{max}}$$ (5)

where $D_{\text{max}}$ is the maximum end-to-end deadline of a higher priority task.

**Proof:** To prove the theorem it is useful to visualize the synthetic utilization $U_j(t)$ at stage $j$ as a function of time due to all subtasks scheduled at that stage. We call it the *synthetic utilization curve* at stage $j$. An example curve is shown in Figure 1.

![Figure 1. The synthetic utilization curve](image)

We lower-bound the maximum height of the synthetic utilization curve at stage $j$ subject to the condition that task $T_n$ experiences delay $L_j$ at that stage. The authors have shown in [2] that the area under the synthetic utilization curve is equal to the sum of the computation times of all arrived tasks in the busy period under consideration. The reason is that each task $T_i$, upon arrival, raises synthetic utilization at stage $j$ by an amount $C_{ij}/D_i$. This increment continues for a duration $D_i$, thereby contributing a rectangle of area $C_{ij}/D_i \times D_i = C_{ij}$ to the area under the utilization curve, as shown in Figure 1. The total area under the curve is the sum of contributions of all tasks, which is $\sum_i C_{ij}$.

Hence, to lower-bound the maximum height of the curve it is enough to (i) minimize the area under the curve, and then (ii) minimize the maximum height of the curve given the minimum area. The former is achieved by noting that the minimum sum of computation times of tasks arrived at stage $j$ must be at least equal to $L_j$, or else subtask $T_n$ will finish sooner on stage $j$. Thus, it is enough to minimize the maximum height of the utilization curve subject to the assumption that the area under the curve is equal to $L_j$. We call it, the *area property*. 

3
Consider the busy period in which task $T_n$ executes on stage $j$. Without loss of generality, we assume that this busy period consists only of $T_n$ and higher priority tasks that arrive while $T_n$ is on stage $j$. The reason we can make this assumption is threefold. First, to minimize the height of the utilization curve, tasks of lower priority than $T_n$ need not be considered. Lower priority tasks only increase utilization without affecting the schedulability of $T_n$. Second, tasks that arrive after the departure of $T_n$ need not be considered since they only increase utilization without affecting the schedulability of $T_n$. Third, the authors have shown in [3] that for a given task delay, the maximum synthetic utilization is minimized when the task arrives at the beginning of a busy period. Without repeating the proof, the intuition behind it is that making the busy period longer increases utilization. Observe that adding any amount of computation time, prior to the arrival of $T_n$, increases both the area under the synthetic utilization curve and the length of the base by the same amount, bringing the synthetic utilization closer to 1.

To minimize the height of the utilization curve (whose area is the area under the curve, hence its area is a function of its height), we find the maximum base length of that curve. Let $D_{\text{max}}$ be the maximum relative deadline of all tasks of higher priority than $T_n$ in its busy period. As argued above, such tasks arrive no later than the departure time of $T_n$ at stage $j$. Thus, their absolute deadlines are at most $D_{\text{max}}$ time units past the departure of $T_n$. Since a task contributes to synthetic utilization only between its arrival time and deadline, the synthetic utilization curve at stage $j$ extends for at most $D_{\text{max}}$ time units past the departure of $T_n$. Hence, the synthetic utilization is non-zero between its arrival time and deadline, the synthetic utilization curve at stage $j$ extends for at most $D_{\text{max}}$ time units past the departure of $T_n$. Hence, the synthetic utilization is non-zero at stage $j$ for an interval of time equal at most to $L_j + D_{\text{max}}$, which defines the maximum length of the base of the synthetic utilization curve. We now minimize the height of this curve, keeping in mind that the area under the curve is $L_j$ (by the area property) and the base is $L_j + D_{\text{max}}$.

We know from geometry that, when the base and the area under a curve are kept constant, its maximum height is minimized when the curve is flat. The area under the flat curve is a rectangle whose height is $L_j / (L_j + D_{\text{max}})$. No shape of the same area can have a lower maximum height. Unfortunately, no arrival pattern yields a perfect rectangle because no tasks arrive after the departure of $T_n$. Hence the minimum-height curve is flat only until $T_n$ after which it slopes down as current tasks reach their deadlines and their contribution to synthetic utilization is subtracted. The shape of that decline is determined by the parameters of the set of tasks whose deadlines are after the departure of $T_n$ on the processor (let us call them, set $E_i$). We apply Lemma 5 from [2], derived for the single processor case, which defines important properties of set $E_i$ that leads to the minimum utilization bound. The lemma is repeated here for convenience:

**Lemma 5** (from [2]): In the worst case pattern (when the height of the utilization curve is minimum), all tasks $E_i$, $1 \leq i \leq K$ satisfy the following properties

1. $A_{i+1} = A_i + C_i$, where $A_i$ is the arrival time and $C_i$ is the computation time for task $E_i$.

2. All tasks $E_i$ are tasks of type $T_{\text{max}}$ (i.e., with deadline $D_{\text{max}}$).

The proof of the lemma is found in [2]. The lemma states that for a single stage, the lower bound on utilization is achieved when all tasks that finish after $T_n$ on the processor (set $E_i$) have the maximum deadline $D_{\text{max}}$ and have arrival times that are separated by their respective computation times, such that $A_{i+1} = A_i + C_i$. Figure 2 plots a utilization curve satisfying the properties derived so far; namely, the area under the curve is $L_j$, its base is $L_j + D_{\text{max}}$, it is flat until the departure time of $T_n$ and it is determined by task set $E_i$ (described by Lemma 5) after that departure.

**Figure 2. The Worst Case Pattern**

Observe that the trailing edge of the curve in Figure 2 is bounded by line ED whose slope is $c_i/D_{\text{max}} = 1/D_{\text{max}}$. Since this value is independent of the computation times of the tasks, GC (and ED) are straight lines across all the tasks $E_i$. As the slope of the line GC is $1/D_{\text{max}}$, it follows that distance GF = $U D_{\text{max}}$. Let us define area A to be the area under the utilization curve in the interval that follows the departure time of $T_n$. From Figure 2, the height of the utilization curve is the height of the rectangle ABCF, which is $(E_j - A)/L_j$. This height is minimized when $A$ is maximum. Area $A$ is upper bounded by that of the trapezoid $CDEF$. Thus, in the following we replace area $A$ by the area of $CDEF$.

Let us denote areas by the names, in parenthesis, of the corresponding geometric shapes. For example, $(ABDE)$ denotes the area contained by the mentioned vertices. Let $U_j$ be the maximum synthetic utilization on stage $j$. We have the following relations:

$$\text{(ABDE)} = \text{Total Computation Time} = L_j \quad (6)$$

$$\text{(CDEF)} = U_j D_{\text{max}} \quad (7)$$
(BCGA) = \frac{1}{2}U_j(L_j + (L_j - U_jD_{max})) \quad (8)

(ABDE) = (CDGE) + (BCGA) \quad (9)

Substituting from Equations (6-8) into Equation (9) and simplifying, we get:

\[ L_j = \frac{U_j(1 - U_j/2)}{1 - U_j}D_{max} \quad (10) \]

This completes the proof. ■

For the end-to-end deadline to be met, \( \sum_j L_j \leq D_n \). Substituting for \( L_j \) from Equation (10), we get:

\[ \sum_{j=1}^{N} U_j(1 - U_j/2) \leq D_n/D_{max} \quad (11) \]

Observe that \( D_n/D_{max} \) is the ratio of the deadline of task \( D_n \) to that of a higher priority task that delays its execution. To obtain the minimum bound, this ratio must be minimized. As mentioned in Section 2, the minimum such ratio is the urgency inversion parameter \( \alpha = \min_{T_n \geq T_1} D_n/D_{hi} \) of the scheduling policy. The feasible region for such a scheduling policy is thus:

\[ \sum_{j=1}^{N} U_j(1 - U_j/2) \leq \alpha \quad (12) \]

In particular, for deadline monotonic scheduling, \( D_n/D_{max} \geq 1 \), or \( \alpha = 1 \). The feasible region for deadline monotonic scheduling is therefore:

\[ \sum_{j=1}^{N} U_j(1 - U_j/2) \leq 1 \quad (13) \]

One possible concern is whether the bound becomes increasingly pessimistic when the number of stages increases. Fortunately, this is not so. The intuitive reason is that the stage synthetic utilization \( U_j \) considers the ratio of the per-stage computation time to the end-to-end deadline. In a schedulable system, this ratio decreases as the number of stages increases. Hence, by definition, for the same degree of schedulability, \( U_j = O(1/N) \), which means that the summation on the left-hand-side of Inequality (12) and (13) does not increase with \( N \). The constraint imposed by the right-hand-side therefore does not become more severe as the number of stages increases.

3.2 Non-independent Tasks

If tasks have critical sections it is possible for task \( T_n \) to block on a lower priority task that is holding a lock, resulting in priority inversion. Priority inversion is different from urgency inversion defined in Section 2. While urgency inversion refers to a task prioritization that is inconsistent with task urgency, priority inversion refers to a task queuing scenario that is inconsistent with assigned priorities. If the priority ceiling protocol is used at each node then the maximum time, \( B_{n,j} \), that task \( T_n \) can be blocked by a lower priority task at stage \( j \) is bounded by the length of the maximum critical section of a lower priority task at that stage that uses a resource shared with \( T_n \). Hence, the delay \( L_j \) of task \( T_n \) on stage \( j \) is given by the expression in Theorem 1, plus \( B_{n,j} \). The schedulability condition becomes:

\[ \sum_{j} \left( \frac{U_j(1 - U_j/2)}{1 - U_j}D_{max} + B_{n,j} \right) < D_n \quad (14) \]

Dividing by \( D_{max} \) and rearranging, we get:

\[ \sum_{j=1}^{N} U_j(1 - U_j/2) \leq \alpha(1 - \sum_{j=1}^{N} \gamma_j) \quad (15) \]

where \( \gamma_j = \max_n B_{nj}/D_n \) is the normalized maximum blocking experienced by a task due to lower priority tasks at stage \( j \). The above expression defines a sufficient schedulability condition (or feasible region) for a resource pipeline with non-independent tasks.

3.3 Arbitrary Task Graphs

Consider a system composed of multiple independent resources. Tasks arrive to the system and require service (by an end-to-end deadline) on a subset of these resources that is specific to each task. Each task is given by a directed acyclic graph of subtasks, each allocated to a potentially different resource (e.g., see Figure 3).

![Figure 3. An Example Task Graph](image)

To compute the feasible region, we express the end-to-end delay in terms of stage delays, \( L_j \), and bound it by the relative deadline. For example, for task \( T_n \) in Figure 3 the end-to-end delay is \( L_1 + \max\{L_2, L_3\} + L_4 \leq D_n \). Substituting for \( L_j \) from Theorem 1 and rearranging, the feasible region in the given example is:

\[ \frac{U_1(1 - U_1)}{1 - U_1} + \max_{j=2,3} \left\{ \frac{U_j(1 - U_j)}{1 - U_j} \right\} + \frac{U_4(1 - U_4)}{1 - U_4} \leq \alpha \quad (16) \]

More formally, consider a task with \( M \) subtasks that form a directed acyclic graph. Let \( k_i \) be the resource (e.g., the processor) to which subtask \( i \) is assigned. Let \( L_i \) denote
the delay experienced by subtask $i$ on processor $k_i$. The following theorem defines the feasible region.

**Theorem 2:** If $d(L_1, \ldots, L_M)$ is the expression of the end-to-end delay of a task as a function of per-stage delays, then the feasible region for this task is given by:

$$d\left(\frac{U_{k_1} (1 - U_{k_1} / 2)}{1 - U_{k_1}}, \ldots, \frac{U_{k_M} (1 - U_{k_M} / 2)}{1 - U_{k_M}} \right) + \alpha \gamma_i \leq \alpha$$

(17)

**Proof:** The proof follows trivially from the observation that for the task to be schedulable, one must satisfy $d(L_i, \ldots, L_M) \leq D_i$, where $D_i$ is the relative deadline of the task. Substituting for $L_i$ from the result of Theorem 1 (plus blocking $B_{nj}$), and rearranging, the expression of Theorem 2 is achieved.

Theorem 2 is a generalization of all previous expressions reported in this paper. It states that a feasible region can be derived for each task depending on its task graph. The approach works even when multiple subtasks are allocated to the same processor. For example, if subtasks 1 and 4 of the task in Figure 3 were to run on the same processor (say processor 1), Inequality (16) would still be the correct feasible region with the additional observation that $U_A = U_1$ is the synthetic utilization of processor 1.

### 4 Evaluation

To evaluate the performance of an admission controller based on the multidimensional feasible regions derived above, 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 grows linearly with the number of stages. All tasks in a given experiment are processed by the same number of stages. Scheduling at each stage is deadline-monotonic, which is the optimal uniprocessor fixed-priority scheduling policy for aperiodic tasks. The arrival process is Poisson.

An admission controller at the first stage allows input tasks into the system only if the system remains within the feasible region. The admission controller updates the synthetic utilization of all stages upon the arrival of a task to the first stage. The synthetic utilization is decremented at task deadlines. When a stage becomes idle, the contribution of all departed tasks to its synthetic utilization is removed, as such tasks will not affect the future schedule of this stage. Note that resetting synthetic utilization when a stage becomes idle is a very important tool that reduces the pessimism of admission control. For example, consider a single processor system in which aperiodic tasks arrive with $C_i = 1$ and $D_i = 2$. The contribution of any single task satisfies our utilization-based schedulability condition (given by Equation 13), and hence such a task can be admitted. Upon termination of each task the processor becomes idle and the synthetic utilization is reset to zero. Let a new task arrive an arbitrarily small interval of time after the termination of the previous one. Since the synthetic utilization was reset to zero, the new task will be admitted. Repeating this scenario indefinitely leads to a real processor utilization that is close to unity (almost no idle time), despite keeping the synthetic utilization below the bound. While this is obviously a contrived example, an interesting question is how well utilization-based admission control really performs for a more realistic workload. Several experiments are conducted to explore properties of the new admission control scheme as detailed in the subsections below.

#### 4.1 Effect of Pipeline Length

In the first set of experiments, we explore the effect of pipeline length on the admission controller. In particular, we test whether or not increasing the pipeline length makes the feasible region more pessimistic. In this experiment, the average stage load was kept roughly equal by generating task computation times at the different stages from the same distribution. The average total computation time across the entire pipeline was kept at about 1/100 of the average end-to-end deadline. 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 was varied from 60% to 200% of stage capacity. A family of curves was drawn showing average real stage utilization (defined as the percentage of time the processor is busy) versus input load for different pipeline lengths. Figure 4 depicts these results.

![Figure 4. Effect of Pipeline Length](image)

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, the curves
for 2, 3, and 5 stages are almost identical. The authors have experimentally verified that the curves in fact become stage-independent when the number of stages increases. Hence, increasing pipeline length has no adverse effects. From the aforementioned two observations we conclude that the bound developed in this paper is a viable tool for admission control regardless of pipeline length.

4.2 Effect of Task Resolution

In the previous experiments, task resolution (defined as the average end-to-end deadline divided by the average total computation time of all stages) was very high, leading to a liquid-like model in which the pipeline serves a very large number of very small tasks. In the following, we repeat the above experiments for a two-stage pipeline while changing task resolution. In Figure 5 Three curves are compared that differ in the overall per-stage load. The curves illustrate the average real per-stage utilization after admission control as a function of task resolution for different total load values. It can be seen that the higher the resolution the higher the fraction of accepted tasks. This is because it is easier to generate unschedulable workloads when individual tasks are larger (lower resolution).

![Figure 5. Effect of Task Resolution](image)

4.3 Effect of Load Imbalance

In the experiments above, the average load on different pipeline stages was kept balanced by choosing the subtask computation times on different stages from the same distribution. Next, we investigate the effect of load imbalance by changing the mean of that probability distribution from one stage to another to create a bottleneck stage. Figure 6 shows the results for a two-stage pipeline. The ratio of the mean computation time (which is the same as the ratio of the average workloads) across the two stages is varied. The average utilization of the bottleneck stage is shown. The midpoint of Figure 6 is the case when the load is balanced across the two stages. Moving away from this point in either direction increases the load imbalance in favor of one stage over the other. It is seen that as the load imbalance increases, the utilization of the bottleneck stage grows. This is because an imbalanced system is dominated by the bottleneck resource. Thus, as the imbalance increases, the performance of the system approaches that of a single resource system. From Figure 4 we can see that the single resource system shows a higher real utilization than a two-stage system in the presence of our admission control. Hence, the bottleneck utilization of an imbalanced two-stage pipeline increases with imbalance. Said differently, the admission controller can opportunistically increase the utilization of one stage when the other is underutilized. This is intuitive given that the end-to-end delay being bounded is the sum of the two stages. In conclusion, our admission controller automatically captures and takes advantage of any load imbalance in the system, actually improving the schedulable bottleneck stage utilization.

![Figure 6. Effect of Pipeline Imbalance](image)

4.4 Approximate Admission Control

All the previous experiments used exact admission control which requires knowledge of task computation times upon task arrival. Often this knowledge is unavailable. Instead, the operator might only know the mean computation time. Intuitively, if task resolution is high (i.e., there is a large number of small tasks), laws of large numbers make it a good approximation to use the average computation time instead of the actual one. In this section, we quantify the performance of such an approximation in terms of the resulting miss ratio. Figure 7 shows the miss ratio of admitted tasks versus task resolution when the admission control policy uses the average computation time in the feasible region expressions. Two curves are shown at two different input loads. The results are for a balanced two-stage pipeline. It can be seen that no tasks miss their deadlines as long as task resolution is high. As the resolution is decreased, some very small fraction of tasks might incur deadline misses. The results are very encouraging especially for soft real-time systems. They imply that knowledge of exact computation time is not essential in practice as long as task resolution is high, and occasional misses can be tolerated. The former condition is true of most server systems that are designed to
handle a large number of clients concurrently. The latter condition is met in the presence of soft real-time tasks.

The consoles display tactical track data (periodic, soft real time), navigation data (periodic, soft real time), time synchronization data (sporadic, soft real time), 2-4 voice communication channels (aperiodic, soft real time), 1 video channel (periodic, soft real time), and critical alerts (aperiodic, hard real time). Consoles also originate service requests (aperiodic, soft or hard real time), voice communications (aperiodic, soft real time), and a video stream (periodic, soft real time). In the back end computing room, many of the tactical applications are implemented in a data flow architecture consisting of multiple subtasks that may or may not be colocated on the same processor. A primary flow is sensor data processing. The processing software consists of 4-6 subtasks with possible branching and rejoining. The vast majority of this data is periodic and soft real time. However, if a series of sensor reports meet certain threat criteria, an urgent self-defense mode can be enabled. Further processing of that target becomes an urgent aperiodic task with a hard real-time deadline to launch a countermeasure. Dynamic changes in semantic importance can also happen in other systems, for example navigation control; the latency that can be tolerated in command execution when navigating in the open ocean is different then when evading a torpedo.

Cost considerations preclude a priori reservation of resources for the simultaneous occurrence of all urgent aperiodics. Instead, aperiodic schedulability analysis can be used to determine whether the current workload is feasible. In this scheme, a fraction of synthetic utilization on each stage is reserved for critical periodic and aperiodic tasks. On stage $j$, this fraction is equal to $U_{j,res} = \sum_i C_{ij}/D_i$, over all critical tasks $T_i$ that need stage $j$. For schedulability, the reserved values, $U_{j,res}$, must satisfy the inequality of Theorem 2. At run-time, the arrival of other aperiodic tasks augments the synthetic utilization of affected stages by the tasks’ respective contributions. They can be admitted as long as the system remains in the feasible region.

If an important incoming aperiodic task causes the system to move outside the feasible region, the condition is detected at admission control. Less important load in the system can be immediately shed in reverse order of semantic importance until the system returns into the feasible region and admits the new arrival. This architecture decouples the scheduling priority inside the system (which can be some optimal policy such as deadline-monotonic) from the semantic priority of tasks, which determines which tasks should be shed at overload. In the absence of an admission controller, one would have had to assign task scheduling priorities inside the system according to their semantic importance such that the right tasks are shed by the scheduler at overload. Such a semantic priority assignment is generally suboptimal from a schedulability perspective. Using our analysis, therefore, can both improve schedulability and
allow a priori pre-certification of different combinations of periodic and aperiodic task arrival scenarios.

An example is shown in Table 1. The numbers shown are notional and chosen for illustrative purposes only. The table represents at a high level the type of tasks a shipboard computing system might execute in a battle scenario. In this example, the system tracks multiple targets by running multiple instances of a soft real-time Target Tracking task, each at a period of 1s. The first stage of the task performs track updates (1 ms/track). The second stage is a distributor that runs periodically, packaging track data for subsequent display (consumes 2 ms per display independent of the number of tracks). The display takes 20 ms to present all data. Concurrently, a threat assessment task (Weapon Detection) may be launched aperiodically to assess selected tracks. This is a hard real-time task that must finish within 500 ms. The task performs tracking, weapon engagement planning, and display. Should this task make an engagement decision, the Weapon Targeting task is launched. This is a periodic hard-real-time task that runs at a 50 ms period. It performs tracking, weapon distribution and weapon release. In addition, the system processes a periodic reconnaissance video stream from a UAV.

Certification requires that the system be able to process Weapon Detection concurrently with Weapon Targeting and with UAV video streaming, while being able to track current targets. We break this question into two. First, are the Weapon Detection, Weapon Targeting and UAV video tasks schedulable concurrently? Second, if capacity is set aside permanently for these tasks, how many instances of the target tracking task can be admitted dynamically at runtime? To answer the first question, we compute the synthetic utilization of each of the three stages that needs to be reserved for the three tasks mentioned above. In the third stage we assume that different tasks have different consoles. Thus, we do not add their utilizations, but take the largest one (UAV video). The synthetic utilization of the three stages due to Weapon Detection, Weapon Targeting and UAV video is thus 0.4, 0.25, and 0.1, respectively. Substituting in Equation (13), we get 0.93, which is lower than 1. Hence, the task set is schedulable by its end-to-end deadlines. To guarantee these critical tasks, we reserve a synthetic utilization of 0.4, 0.25, and 0.1 on the respective stages. This means that the synthetic utilization counters are initialized with these values. At run time, track updates arrive dynamically and must be admitted using an online admission controller that checks schedulability using Equation (13). Since task period is 1 s, we allow each arrival to wait up to 200 ms at the admission controller if it cannot be admitted immediately. If the waiting period expires the task is rejected. The number of Target Tracking tasks was increased gradually until rejections were observed. Simulation results show that the system was able to process up to 550 different tracks concurrently, along with the reserved tasks. Stage 1 was the bottleneck with a utilization of approximately 95%. The reason for this high efficiency is because we reset synthetic utilization to its initial value when the system is idle. A number of Tracking tasks is accepted at this point until the limit of the feasible region is reached. Others will wait for up to 200 ms, but during this time the system is likely to have an idle moment during which synthetic utilization is reset again and more tracks are accepted. Consequently, the system operates virtually at capacity. The example illustrates that the application of the proposed analysis in practice is not only possible but also advantageous.

### 6 Related Work

This paper presents the first extension of the basic utilization bound to a multi-dimensional feasible region. In their prior work, the authors derived for the first time a utilization bound for the aperiodic task model [3]. The initial result presented the optimal uniprocessor bound for aperiodic tasks. In [1] they extend it to non-partitioned multiprocessor scheduling in the special case of the liquid task model, which assumes a workload composed of infinitely many infinitesimal tasks. In [2] they presented a generalized aperiodic bound that considers a non-liquid task model with resource requirements. This paper is an extension of that bound to systems with multiple independent resources.

Several other notable extensions to the original Liu and Layland bound [13] were reported in previous literature. These extensions were confined to variations of the periodic task model. For example, in [10], it is shown that if the periodic bound is a function of only the number of harmonic chains and not the total number of tasks. In [17, 7], the bound is further improved by considering the actual values of task periods. A utilization bound for a modified rate-monotonic algorithm which allows deferred deadlines is considered in [19]. A hyperbolic bound for large periodic task sets under the rate monotonic scheduling policy was proposed in [4]. It was shown to be less pessimistic than the Liu and Layland bound. Multiprocessor bounds appeared in [16, 15, 14].

Extensive work has focused on schedulability analysis...
for aperiodic servers in hard real time systems. The priority inversion problem due to blocking was first mentioned in [11]. In [18], Sha et al. proposed modifications to schedulability tests for using utilization bounds derived by Liu and Layland [13] to guarantee schedulability of periodic tasks using the priority ceiling protocols by the rate-monotonic algorithm. Work by Ghazalie and Baker [8] take into account resource constraints in developing a schedulability test using Liu and Layland’s utilization bound. Other approaches [5]. [12] and [6] extend the work for aperiodic servers. The approach in this paper incorporates, for the first time, the effects of blocking into a generalised expression of feasible regions for distributed processing systems and tasks described by general directed acyclic graphs.

7 Conclusions

This paper presented the first synthetic utilization bound for an arbitrary scheduling policy and an aperiodic task model that applies to resource pipelines. We show that the bound can be applied to a mix of periodic and aperiodic tasks, and can account for blocking over critical sections. The analysis is extended to arbitrary task graphs. Admission control based on synthetic utilization leads to high server utilization. We show that the bound is extended to arbitrary scheduling policy and an aperiodic task model. This paper presented the first synthetic utilization bound for aperiodic servers in hard real time systems. The authors hope that this paper may be the first step towards deeper insights and a more general understanding of feasible regions for scheduling hard-real-time systems. Developing such a comprehensive understanding is a topic of future work for the authors.

References


