The Aperiodic Multiprocessor Utilization Bound for Liquid Tasks *

Tarek Abdelzaher
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
e-mail:*zaher@cs.virginia.edu*

Bjorn Andersson and Jan Jonsson
Depertment of Computer Engineering
Chalmers University of Technology
412 96 Gothenburg
e-mail:{*ba,janjo*}*@ce.chalmers.se*

Vivek Sharma, Minh Nguyen
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
e-mail:{*viveks,mdn4d*}*@cs.virginia.edu*

Abstract

Real-time scheduling theory has developed powerful tools for translating conditions on aggregate system utilization into per-task schedulability guarantees. The main breakthrough has been Liu and Layland's utilization bound for schedulability of periodic tasks. In 2001 this bound was generalized by Abdelzaher and Lu to the aperiodic task case. In this paper, we further generalize the aperiodic bound to the case of multiprocessors, and present key new insights into schedulability analysis of aperiodic tasks.

We consider a special task model, called the liquid task model*, representative of high-performance servers with* aperiodic *workloads, such as network routers, web servers, proxies, and real-time databases. For this model, we derive the optimal multiprocessor utilization bound, defined on a utilization-like metric we call "synthetic utilization". This bound allows developing constant-time admission control tests that provide utilization-based absolute delay guarantees. We show that the real utilization of admitted tasks can be close to unity even when synthetic utilization is kept below the bound. Thus, our results lead to multiprocessor systems which combine constant-time admission control with high utilization while making no periodicity assumptions regarding the task arrival pattern.*

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

1 Introduction

The attainment of absolute delay guarantees in commercial applications such as e-commerce servers and real-time databases has been a topic of active research for several years. Unfortunately, state of the art servers are still best effort in nature, partly due to the complexity of real-time scheduling and schedulability analysis techniques in the absence of *a priori* load knowledge.

In addition to more complex schedulability tests, simple utilization bounds have been developed in the real-time scheduling literature which enable an admission controller to decide whether an incoming task can meet its deadline based on utilization-related metrics. While earlier bounds applied only to variants of periodic tasks, in a recent result [1], Abdelzaher and Lu generalized the approach to aperiodic workloads, hence removing all assumptions about the task arrival pattern. The authors defined in [1] a new notion of utilization that applies to the aperiodic task model and showed that in single processor systems the utilization bound for fixed priority aperiodic task scheduling is $\frac{1}{1+\sqrt{1/2}}$. They also showed that aperiodic deadline monotonic scheduling is the optimal fixed-priority policy in the sense of maximizing the utilization bound. The new bound can be used for efficient admission control in a wide category of applications that operate in unpredictable environments in which request arrival patterns are not known.

In this paper, we extend the aperiodic bound derived in [1] to multiprocessors and develop key new insights into utilization-based schedulability analysis of multiprocessor scheduling in an aperiodic environment. The result, we hope, is a first step towards a theory for constant-time utilization-based schedulability analysis of *aperiodic* workloads – a parallel to rate-monotonic analysis of periodic tasks.

The main motivation for our new aperiodic utilization-based schedulability theory is fast per-task (e.g., per-request) admission control in high-performance servers. Per-task admission control aims to ensure that no server

*The work reported in this paper was supported in part by DARPA grant N00014-01-1-0576 and NSF grants CCR-0208769 and ANI-0105873.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2002		**00-00-2002 to 00-00-2002**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
The Aperiodic Multiprocessor Utilization Bound for LIquid Tasks	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Virginia,Department of Computer Science,151 Engineer's Way,Cahrlottesville,VA,22094-4740**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**12**	

resources are wasted on tasks which eventually miss their deadlines. In the absence of admission control, the aggregate amount of resources wasted on eventually abandoned tasks may be significant even if the individual tasks were very small. While admission control may save such wasted resources, its overhead is an important practical consideration in high-performance servers. Schedulability-analysis algorithms that are polynomial in the number of received requests may impose undue overhead. Instead, our bound offers a constant-time schedulability test that enforces absolute delay guarantees simply by maintaining a synthetic utilization counter updated on task arrivals and departures. We further show that admission control tests based on synthetic utilization do not underutilize the server. Simulation results show that real utilization values close to 95% are achieved when using the synthetic utilization bound for admission control.

In high performance servers, thousands of requests are served per second. Individual requests consume a negligible amount of total server capacity. This condition (which we formalize later) is called the liquid task model. The condition is true even of single processor servers as was shown in many prior server profiling results, e.g., [17]. In the remainder of this paper, we focus on the liquid task model only, and show that the bounds achieved in that case are higher than those for arbitrarily sized tasks.

We consider a class of scheduling policies, called time-independent scheduling, which can be implemented using fixed-priority operating system scheduling support. We show that, for the liquid task model, the optimal tight synthetic utilization bound of time-independent scheduling is independent of the number of processors and equal to $\frac{1}{1+\sqrt{1/2}}$. We also show that for the liquid task model, deadline monotonic scheduling is an optimal scheduling policy.

The remainder of this paper is organized as follows. Section 2 reviews the task model and problem statement. Elements of a theory for aperiodic multiprocessor utilization-based schedulability analysis are presented in Section 3. The tight optimal utilization bound for multiprocessor scheduling of aperiodic tasks is derived in Section 4. Section 5 presents an experimental evaluation of admission control algorithms based on the derived bound. Section 6 describes related work. Finally, Section 7 presents the conclusions of the paper and avenues for future work.

2 Task Model and Problem Statement

Consider an aperiodic task model in which independent aperiodic tasks arrive at a multiprocessor-based real-time system at random without prior knowledge from the scheduler. The multiprocessor schedules the arrived tasks by arranging them in a single queue and dequeuing them in priority order as soon as a processor becomes available. In a system of m processors, up to m tasks can be executing concurrently at any given time.

Each task T_i, constitutes a *single invocation*, described by an arrival time A_i, an execution time $C_i > 0$, and a relative deadline $D_i \geq C_i$. The absolute deadline of the task is $A_i + D_i$. The tasks are to be scheduled preemptively. Upon preemption, tasks can be resumed on any processor without penalty.[1]

We are especially interested in the case were the order of task computation times is much smaller than the order of task deadlines, i.e., $\forall i, j : C_i << D_j$. We call it the *liquid task model*. The model is representative of high performance servers which handle many thousands of requests per second. For example, in the case of web servers, a typical response time (i.e., relative deadline) would be of the order of seconds to tens of seconds, whereas a typical computation time would be of the order of hundreds of microseconds to single milliseconds. The liquid task model represents the limiting case in which the relative deadlines are generally finite, yet the computation times are infinitesimal. More formally, in the liquid task model, $\forall i : C_i \to 0$, and $\forall i : C_i/D_i \to 0$. While this model is an idealization, we show that results based on this model hold well in systems exhibiting a large number of small tasks. In other words, while applying utilization bounds for liquid tasks to systems in which computation times are finite does not strictly guarantee meeting all deadlines, these bounds present a very good practical heuristic for admission control as long as rare misses can be tolerated.

2.1 Synthetic Utilization

To compute the utilization of an aperiodic workload, we use a generalized notion of Liu and Layland's *utilization factor* [13], called *synthetic utilization*, which we describe next. At any given time instant, t, let $V(t)$ be the set of aperiodic tasks that have arrived but whose deadlines have not expired, i.e., $V(t) = \{T_i | A_i \leq t < A_i + D_i\}$. The set $V(t)$ at some time t is illustrated by the shaded tasks in Figure 1. We call $V(t)$ the set of current tasks. Note that $V(t)$ is independent of the task-to-processor assignment. It is also independent of the scheduling policy used because it is independent of actual task completion times.

We define a quantity called *global multiprocessor utilization*, $U_{global}(t)$, as the utilization contributed by the current tasks at time t. It is given by the expression:

$$U_{global}(t) = \sum_{T_i \in V(t)} C_i/D_i \qquad (1)$$

The *normalized* synthetic utilization, or simply synthetic utilization, is defined as:

[1] In later work, we shall extend this model to account for task-to-processor assignment constraints, processor heterogeneity, and task migration costs.

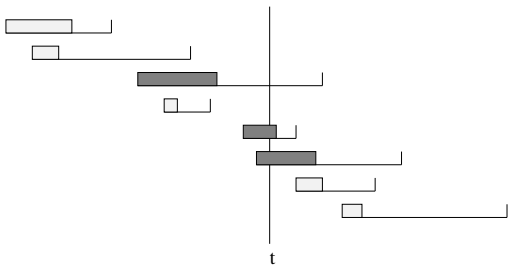


Figure 1. $V(t)$: Current tasks at t

$$U(t) = \frac{U_{global}(t)}{m} \tag{2}$$

where m is the number of processors. Note that synthetic utilization is a natural generalization of the widely-used notion of "utilization factor" central to periodic task schedulability analysis and utilization-bound literature. This generalization is not a contribution of this paper. Different variants of aperiodic task utilization definitions have already been around for many years. The above definition, however, is what we choose for the derivation of the new multiprocessor bound. It is easy to see that synthetic utilization reduces to the utilization factor when tasks are periodic. Hence, by relying on synthetic utilization, our bound can be meaningfully compared to those for periodic tasks.

2.2 Time-Independent Scheduling

A multiprocessor system queues up arrived tasks for execution in a priority order defined by its scheduling policy. Traditionally, scheduling policies are classified into fixed-priority and dynamic-priority depending on whether or not different invocations of the same task have the same priority. In the context of aperiodic tasks, this classification is inappropriate because each task has only one invocation. Instead, we classify aperiodic task scheduling policies into time-independent and non-time-independent. A scheduling policy is time-independent if the priority assigned to a task does not depend on the absolute arrival time of this task. More formally, in the context of non-partitioned multiprocessor scheduling of aperiodic tasks, we have the following definition:

Definition: *A time-independent non-partitioned multiprocessor scheduling algorithm is a function $f(\tau, t) \rightarrow P$, such that:*

1. *$f(\tau, t) \rightarrow P$ maps an infinite set of tasks τ whose arrival times are given by vector t into a finite set of values P, and*
2. *$f(\tau, t) \rightarrow P$ satisfies $f(\tau, t) = f(\tau, t')$, for any t and t'.*
3. *The multiprocessor ready queue is sorted by values P such that the ready task with the smallest value is resumed on the next available processor. Tasks with the same value of P are queued in FIFO order.*

In the liquid task model, since all $C_i \rightarrow 0$, we further assume that task priorities are independent of task computation times. For example, in a tiered-services web server, user requests are often categorized into classes on the basis of client identity, regardless of the computation time or arrival time of the request. Such prioritization meets our definition of a time-independent scheduling policy.

2.3 Statement of Contributions

In this paper, we make the following four contributions:

- We present new important insights into the general problem of deriving synthetic utilization bounds for schedulability of aperiodic tasks. The insights are based on a geometric interpretation of the problem.
- We derive a tight synthetic utilization bound for deadline-monotonic multiprocessor scheduling for the liquid task model. We show that the bound is independent of the number of processors and approaches $\frac{1}{1+\sqrt{1/2}}$ (i.e., about 58.6%).
- We prove that deadline monotonic scheduling is the optimal time-independent multiprocessor scheduling policy for liquid tasks in the sense of maximizing the synthetic utilization bound. Hence, the bound derived in this paper is optimal for liquid aperiodic tasks.
- We dispel the concern that aperiodic bounds may be overly pessimistic. We show that in the aperiodic case, by resetting the admission controller's synthetic utilization counter to zero at appropriate instants (e.g., at processor idle times) the average synthetic utilization becomes less than the average real utilization. Thus, while admission control can enforce the synthetic utilization bound of only 58.6%, real utilization can be close to unity. Consequently, using the synthetic utilization bound for admission control will *not* underutilize the system. This result has no parallel in the utilization-bound literature for periodic tasks.

3 Towards Aperiodic Schedulability Theory Based on Synthetic Utilization

In this section, we present a collection of results, insights, and a methodology for using synthetic utilization to analyze the schedulability of liquid aperiodic tasks on multiprocessors. We hope these results and insights will constitute a first step towards a general theory on constant-time schedulability analysis of aperiodic tasks. Section 3.1 demonstrates the conceptual flow of the derivation of the schedulable multiprocessor utilization bound for aperiodic tasks.

In Section 3.2 we define useful properties of "worst-case" aperiodic task arrival patterns that lead to the multiprocessor bound. Finally, Section 3.3 presents new bounds for utilization-based schedulability analysis of aperiodic tasks on multiprocessor systems. They have the property that they never overestimate the tight utilization bound. Hence, they are both simple to derive and safe to use from an admission control and schedulability perspective.

3.1 Considerations in Aperiodic Tasks

Let a *critically schedulable* task arrival pattern be one in which some task has zero slack. Conceptually, the derivation of the multiprocessor utilization bound amounts to the following steps:

1. Consider an arbitrary critically-schedulable task arrival pattern ζ and an arbitrary critically-schedulable task T_n in that arrival pattern. We call this choice, a scenario $s = (\zeta, T_n)$.
2. For each scenario $s = (\zeta, T_n)$, find the maximum utilization $U^{\zeta}_{max}(s) = \max_t U^{\zeta}(t)$ that occurs between the end of the last multiprocessor idle time (defined as an interval of time where all processors were idle) and the absolute deadline of T_n, as shown in Figure 2. By definition of $U^{\zeta}_{max}(s)$, there exists at least one point (*prior* to the deadline of T_n) in the pattern ζ where the utilization $U^{\zeta}(t) = U^{\zeta}_{max}(s)$.
3. Compute the minimum such utilization $U_{bound} = \min_s \{U^{\zeta}_{max}(s)\}$ across all possible scenarios. Thus, in *every* task arrival pattern ζ there exists at least one point prior to the deadline of *each* critically schedulable invocation where the utilization $U(t) \geq U_{bound}$.

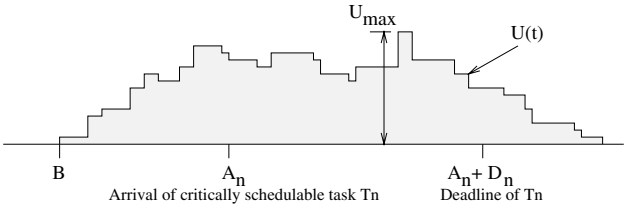


Figure 2. The Maximum Utilization

The value U_{bound}, above, is the utilization bound. If the synthetic utilization is always maintained below U_{bound}, there cannot be any critically schedulable tasks in the pattern.

It is easy to show that if $\forall t : U^{\zeta}(t) < U_{bound}$, there cannot be any unschedulable tasks either. To see that, consider an unschedulable pattern ζ_{un}. A critically schedulable pattern ζ_{cr} of lower utilization[2] is obtained by reducing the execution time of the invocations that miss their deadlines in ζ_{un} by the amount left unfinished at the instant when the deadline is reached. (It is assumed that the remainder of the task is otherwise dropped when the deadline is reached.) Note that we assumed that in the liquid task model, task priorities are independent of task computation times. Thus, the above transformation does not alter the schedule. Since reducing execution time reduces utilization, it follows that $U^{\zeta_{cr}}_{max} \leq U^{\zeta_{un}}_{max}$. However, $U_{bound} \leq U^{\zeta_{cr}}_{max}$. Thus, $U_{bound} \leq U^{\zeta_{un}}_{max}$ for any arbitrary unschedulable task pattern ζ_{un}. In other words, there are no unschedulable patterns under U_{bound}.

In the following, we derive the bound by finding a scenario, s, that minimizes synthetic utilization, i.e., for which $U^{\zeta}_{max}(s) = U_{bound}$. We call it a *worst case* scenario. The notion of a worst case scenario, defined above, plays a central role in this paper, and will be used repeatedly in the following sections. Note that the worst-case scenario might not be unique. We need to find only one such scenario to determine the bound.

3.2 Properties of the Worst-Case Scenario

To find a worst-case scenario on a multiprocessor, we first establish some properties to guide the search, summarized by Theorem 1:

Theorem 1: *There exists a worst-case scenario (ζ, T_n) with the following properties:*

- *The critically schedulable task, T_n, has the lowest priority.*
- *No tasks arrive at or after the absolute deadline of T_n.*
- *All processors are busy since the last multiprocessor idle time and until the arrival of T_n.*
- *The area under the synthetic utilization curve is equal to the sum of the execution times in the pattern normalized by the number of processors.*

Note that the theorem does not imply that *all* worst case scenarios have the above properties. It only implies that at least one such scenario does. Thus, the search for a worst case scenario needs only to consider those cases that share the properties above.

Proof: The following lemmas prove the aforementioned properties respectively.

Lemma 1: *To find a worst case scenario it is enough to consider only those patterns in which the critically schedulable task is the lowest priority task.*
Proof: Let us consider a critically schedulable aperiodic task arrival pattern. By definition, some task in this pattern must have zero slack. Let us call this task T_m. Consider the interval of time $A_m \leq t < A_m + D_m$ during which T_m is current. At any time t within that interval,

[2] By *lower utilization* we mean that $\forall t : U^{\zeta_{cr}}(t) \leq U^{\zeta_{un}}(t)$

$mU(t) = C_m/D_m + \sum_{T_i>T_m} C_i/D_i + \sum_{T_i<T_m} C_i/D_i$, where C_m/D_m is the utilization of task T_m, $\sum_{T_i>T_m} C_i/D_i$ is the utilization of higher priority tasks that are current at time t, and $\sum_{T_i<T_m} C_i/D_i$ is the utilization of lower priority tasks that are current at time t. Since lower priority tasks do not affect the schedulability of T_m, $U(t)$ is minimized when $\sum_{T_i<T_m} C_i/D_i = 0$. In other words, one can always reduce the utilization of a critically schedulable task pattern (in which task T_m has zero slack) by removing all tasks of priority lower than T_m. Thus, to arrive at a minimum utilization bound, T_m must be the lowest priority task of all those that are current in the interval $A_m \leq t < A_m + D_m$. In the following, we shall denote the lowest priority task by T_n, i.e., set $m = n$.

Lemma 2: *To find a worst case scenario, if T_n is the lowest priority critically schedulable task, it is enough to consider only those arrival patterns where no tasks arrive at or after $A_n + D_n$.*

Proof: Consider an arbitrary aperiodic invocation arrival pattern, ζ, in which T_n is critically schedulable (and, by Lemma 1, T_n is also the lowest priority task). Since any invocations that may have arrived at or after the deadline of T_n do not affect the schedulability of T_n, to find the minimum utilization bound it is enough to consider patterns ζ where no invocations arrive at or after $A_n + D_n$.

Lemma 3: *To find a worst case scenario, if T_n is the lowest priority critically schedulable task, and B is the last time instant at which all processors were idle prior to A_n, then it is enough to consider only those task patterns in which all processors are busy between B and A_n.*

Proof: Consider an arbitrary task arrival pattern ζ. Let $t < A_n$ be the last time instant at which *some* processor was idle in ζ prior to A_n. At time t, it must be that the multiprocessor ready queue was empty or else the top ready task in the queue would have been executed on the idle processor. Let $Q(t)$ be the set of tasks that were in execution at time t. Let us form an arrival pattern ζ^x which consists of all the tasks in ζ except that each task in $Q(t)$ is modified as follows: (i) its arrival time is advanced to time t (ii) its relative deadline remains the same, and (iii) its computation time is reduced to the remaining computation time of the task at time t in ζ.[3] Since the transformation does not alter the amount of execution load in the system since time t, it does not affect the schedulability of the lowest priority task T_n. In the new pattern, however, all processors are idle immediately before t, i.e., $B = t$. Furthermore, by definition of t, all processors are busy in the interval $[t, A_n)$. Finally, since the modification to the tasks in $Q(t)$ alter synthetic utilization only infinitesimally, the synthetic utilization of the new task set at all times remains either identical or infinitesimally close to that of the original task pattern. For the purposes of deriving the utilization bound, it is therefore enough to consider pattern ζ^x.

[3]Note that since the computation time is reduced while the relative deadline is kept the same, the utilization of the task is not increased by the transformation. Hence, it remains infinitesimal.

Lemma 4: *The area under the multiprocessor synthetic utilization curve, $U^\zeta(t)$, is equal to the sum of computation times of all arrived invocations normalized by the number of processors*

$$\int_0^\infty U^\zeta(t)dt = \frac{C^\zeta}{m} \tag{3}$$

Proof: To see why Equation (3) is true, observe that the integral on the left-hand-side is the area under the synthetic utilization curve, as shown in Figure 2. Each task with arrival A_i, computation time C_i, and relative deadline D_i contributes to this area a rectangle of hight $u_i = \frac{1}{m}\frac{C_i}{D_i}$, for a duration D_i. The area of this rectangle is $u_i D_i = C_i/m$. Hence, the total area under the utilization curve is the sum of the areas of the individual rectangles over all task invocations i, i.e., $\sum_i u_i D_i = \frac{1}{m}\sum_i C_i = C^\zeta/m$.

From the four lemmas above, Theorem 1 is proved. The search for the worst-case scenario can now be restricted to those defined by Theorem 1.

3.3 A Methodology for "Quick-and-Dirty" Bound Estimates

In this section, we demonstrate a simple methodology for deriving approximate bounds, using Theorem 1, that have the favorable quality of never overestimating the exact tight bound. We also derive the second theorem of aperiodic multiprocessor utilization-based schedulability analysis. We believe that the methodology described in this section will contribute beyond the scope of this paper to the general theory of utilization-based admission control of aperiodic real-time tasks. In the following we describe the basic approach.

1. Consider a critically schedulable task pattern that satisfies the properties outlined by Theorem 1. Let $Base^\zeta$ be the interval of time between the arrival of the first task and the absolute deadline of the last task in the pattern. This is the interval during which the synthetic utilization is non-zero. Let C^ζ be the sum of computation times of all arrived tasks in the pattern. By Lemma 4, the area under the synthetic utilization curve is C^ζ/m. To derive a lower bound on the maximum hight of the utilization curve, note that the maximum hight of a geometric shape of area C^ζ/m and base length $Base^\zeta$ cannot be lower than $\frac{1}{m}C^\zeta/Base^\zeta$. Thus, it must be that:

$$U_{bound} \geq \frac{1}{m} \min_{\zeta} \frac{C^{\zeta}}{Base^{\zeta}} \quad (4)$$

Figure 3 shows the limiting condition on the utilization curve in which its maximum hight is equal to the area divided by the base. The area under the curve in this case forms a perfect rectangle.

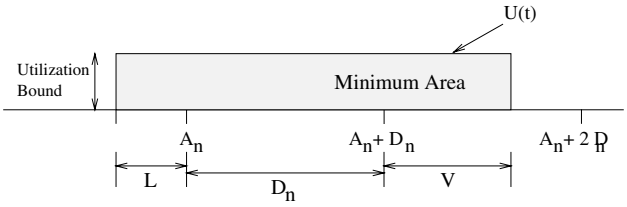


Figure 3. The Approximate Bound

A utilization bound can be derived by minimizing the right-hand-side in Equation (4), i.e., finding the largest denominator and the smallest numerator.

2. Find the minimum numerator in Equation (4). To do so, let us divide the total execution time C^{ζ} into two parts; namely, all execution time prior to A_n and all execution time since A_n. Let L be the time elapsed between the arrival of the first task in the pattern and A_n as shown in Figure 3. From Lemma 3 of Theorem 1, the total execution time prior to A_n must be mL, since the theorem states that all processors are busy during that interval. The minimum execution time needed since A_n to keep T_n critically schedulable is $C_n + m(D_n - C_n)$. The first term is the execution time of T_n. The second term is the minimum execution time needed to block T_n from all processors for the remaining time until its deadline. Hence, the minimum area $area_{min}$ under the utilization curve is the sum:

$$area_{min} = C_n + m(D_n - C_n) + mL \quad (5)$$

3. Find the maximum denominator in Equation (4). Assume the last task in the pattern has an absolute deadline V units after $A_n + D_n$. Hence, the utilization curve extends for:

$$base_{max} = D_n + L + V \quad (6)$$

4. Substituting from Equation (5) and Equation (6) into Equation (4), after simple algebraic manipulation, we get:

$$U_{bound} \geq \min \frac{1 + \frac{L}{D_n} - \frac{C_n}{D_n}(1 - 1/m)}{1 + \frac{L}{D_n} + \frac{V}{D_n}} \quad (7)$$

It can be easily seen (by inspecting the sign of dU_{bound}/dL) that the right-hand-side is minimized when $L = 0$. This happens to match the well-known observation by Liu and Layland, that the worst case occurs when the lowest priority task arrives *together* with the higher priority tasks. Substituting for $L = 0$, we get:

$$U_{bound} \geq \min \frac{1 - \frac{C_n}{D_n}(1 - 1/m)}{1 + \frac{V}{D_n}} \quad (8)$$

The above equation suggests that the bound gets lower when C_n/D_n increases. Thus, the bound for liquid tasks is higher than that for arbitrarily-sized tasks, which is a desirable property considering that we are only interested in the liquid task model.

5. In the liquid task model $C_n/D_n \to 0$. Substituting in Equation (8), we get:

$$U_{bound} \geq \min \frac{1}{1 + \frac{V}{D_n}} \quad (9)$$

From Equation (9), we state the following theorem.

Theorem 2: *In the liquid aperiodic task model, all tasks in an arrival pattern ζ are schedulable by a time-independent scheduling policy, S, (regardless of the number of processors) if the synthetic utilization $U^{\zeta}(t)$ satisfies $\forall t : U^{\zeta}(t) \leq U_{sched}$, where:*

$$U_{sched} = \frac{1}{1 + \max \frac{D_{hi}}{D_{lo}}}$$

where T_{hi} is higher priority than T_{lo} under S.

Proof: The theorem follows from minimizing the right-hand side of Equation (9), i.e., maximizing V/D_n. In this equation, D_n, by definition, is D_{lo}. Similarly, V, by definition, is the distance by which the synthetic utilization curve extends past the deadline of T_n (see Figure 3). Since, by Lemma 2, no tasks arrive after $A_n + D_n$, the maximum V is given by the relative deadline, D_{hi}, of a higher priority task that arrives immediately before the absolute deadline of T_n. Hence, $\max V/D_n = \max D_{hi}/D_{lo}$. Consequently, from Equation (9) $U_{bound} \geq \frac{1}{1+\max \frac{D_{hi}}{D_{lo}}}$. If $U^{\zeta}(t) \leq \frac{1}{1+\max \frac{D_{hi}}{D_{lo}}}$, it must be that $U^{\zeta}(t) \leq U_{bound}$. Hence all tasks are schedulable. This completes the proof.

We show the usefulness of the above theorem by deriving three simple results on multiprocessor utilization bounds for liquid aperiodic tasks.

Result 1: Multiprocessor Deadline-Monotonic Bound: Under deadline monotonic scheduling, any task T_{hi} that

preempts a lower priority task T_{lo} will have a relative deadline $D_{hi} < D_{lo}$. Using Theorem 2, $\max\{D_{hi}/D_{lo}\}$ approaches 1. Hence:

$$U_{sched} = \frac{1}{2} \tag{10}$$

Result 2: Differentiated Services Bound:
Consider a differentiated services framework where tasks are divided into classes such that the ratio of the deadline of class i and class $i+1$ is $\alpha < 1$. Tasks in the same class are scheduled FIFO. In this case, it is easy to see that $\max\{D_{hi}/D_{lo}\} = \alpha$. Thus:

$$U_{sched} = \frac{1}{1+\alpha}, \qquad \alpha < 1. \tag{11}$$

Result 3: Arbitrary-Priority Scheduling Bound:
Let task priorities be assigned based on some external metric, independently of task execution parameters. For example, in web servers, priorities might be assigned depending on client importance rather than any real-time properties. Let the ratio of the largest relative deadline to the smallest relative deadline be $\beta > 1$. Hence, $\max\{D_{hi}/D_{lo}\} = \beta$. This leads to the bound:

$$U_{sched} = \frac{1}{1+\beta}, \qquad \beta > 1. \tag{12}$$

As seen above, the methodology described in this section, as well as Theorem 2, present an easy way to quickly derive bounds for different cases of uniprocessor and multiprocessor scheduling. The geometric insight presented here greatly simplifies the derivation of utilization bounds. These bounds can be used for admission control as will be shown in Section 5. Interestingly, we shall show that because synthetic utilization can be reset to zero at processor idle times, the average real utilization of admitted tasks can be very high even though the synthetic utilization is clipped to the bound.

The bounds derived above are not tight because they assume that the area, $area_{min}$, under the utilization curve forms a perfect rectangle of base $base_{max}$ and uniform hight, as was shown in Figure 3. In reality, no task arrival pattern exists that generates the shown rectangular utilization curve. Instead, the synthetic utilization curve is deformed due to the fact that no tasks arrive after $A_n + D_n$ causing utilization to drop gradually. The resulting curve has a higher hight for the same area compared with the rectangle shown in Figure 3. This exact hight is derived next.

4 The Optimal Aperiodic Multiprocessor Utilization Bound

In this section, we derive the optimal multiprocessor utilization bound and policy for time-independent scheduling. The bound is new in that it considers an aperiodic task model. As before, we restrict ourselves to liquid tasks and non-partitioned multiprocessors. We show that the bound is tight in that there exists an unschedulable task pattern of utilization infinitesimally above the bound.

The proof of optimality is structured as follows. First, we derive the utilization bound for deadline monotonic scheduling. Then, we show that no other policy can achieve a higher bound, hence demonstrating optimality. The derivation of the deadline monotonic bound relies on reducing the problem to that of finding a uniprocessor bound. Since the single processor problem is already solved [1], we get the multiprocessor utilization bound. The aforementioned reduction is possible due to symmetry of processor schedules as we explain next.

Consider a worst-case scenario (ζ, T_n) described by Theorem 1, where T_n is the lowest-priority critically schedulable task. As mentioned before, in Section 3.3, in the reasoning leading to Equation (5), the minimum sum of computation times of the tasks in ζ is $C_n + m(D_n - C_n) + mL$, where L is the length of the interval between the arrival time B of the first task in ζ and the arrival time of T_n. Figure 4 depicts such a scenario. The horizontal timelines represent individual processor schedules. The black rectangles represent the execution intervals of the lowest-priority critically schedulable task, T_n. (These intervals are magnified for clarity, since in the liquid task model T_n has an infinitesimal execution time.) The lightly shaded rectangles represent the execution intervals of higher-priority tasks. The figure illustrates two important properties of this worst-case scenario:

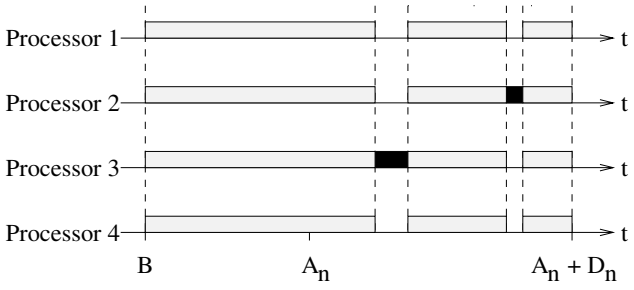


Figure 4. Symmetry in Processor Execution

- When the lowest-priority task T_n is not running, all processors are busy running higher priority tasks. To prove this property we divide the interval $[B, A_n+D_n)$ into two parts; one prior to the arrival of T_n and one since the arrival of T_n. By Lemma 3 of Theorem 1, all processors are busy prior to the arrival of T_n. Since the arrival of T_n, when T_n is not running, all processors must be busy (running higher priority tasks) because otherwise T_n would have a non-zero slack and will not be critically schedulable.

- When the lowest-priority task T_n is running on some processor, all other processors are idle. This is the condition that leads to the minimum area under the utilization curve. If any other task is allowed to run concurrently with T_n, the area under the utilization curve would increase beyond $C_n + m(D_n - C_n) + mL$, resulting in a higher bound.

Next we show that for any scenario (ζ, T_n) which satisfies the above two conditions, there exists another scenario (ζ_{symm}, T_n) of the same utilization, with task T_n still critically schedulable, and in which the schedules of higher priority tasks are identical on all processors. To obtain ζ_{symm}, Let us cut each higher priority task in ζ into m identical tasks with the same arrival time and deadline as the original task, and with a computation time $1/m$ of the original task's computation time (where m is the number of processors). The transformation does not affect task priorities in the liquid task model, since we assumed they are independent of task computation times. It is easy to see that the non-partitioned multiprocessor schedule of high priority tasks in this modified task set will be identical on each processor. This is because there are m identical copies of each single task, all arriving at the same time. In non-partitioned multiprocessor scheduling, every such set of m identical task replicas will be equally load balanced on all processors. Note that the new task pattern has the same synthetic utilization. The intervals during which processors are occupied by high priority tasks are identical on each processor (since all schedules are identical). The lowest priority task is still critically schedulable since the transformation didn't change the amount of time it is preempted.

Since the above transformation didn't increase utilization and didn't change the slack of T_n, starting with any worst-case scenario (ζ, T_n) for which Corollary 1 holds we obtain another worst case scenario (ζ_{symm}, T_n) in which the multiprocessor schedules of high priority tasks are identical. We can now express the utilization bound as the synthetic utilization of the latter scenario. Thus, $U_{bound} = U^{\zeta_{symm}} = \frac{1}{m}(C_n/D_n + mU_i^{\zeta_{symm}})$, where $U_i^{\zeta_{symm}}$ is the synthetic utilization of current tasks on processor i. Note that $U_i^{\zeta_{symm}}$ is identical for all i. In the liquid task model, $C_n/D_n \to 0$. Hence, $U_{bound} = U_i^{\zeta_{symm}}$.

In [1], it was proven that under deadline-monotonic scheduling, the minimum single processor utilization of a critically schedulable task pattern in which T_n is critically schedulable occurred when $C_n/D_n \to 0$, in which case the utilization bound was $\frac{1}{1+\sqrt{\frac{1}{2}}}$. Hence, the multiprocessor utilization bound for liquid tasks under deadline-monotonic scheduling is:

$$U_{bound} = \frac{1}{1 + \sqrt{\frac{1}{2}}} \tag{13}$$

The presented bound is tight in the sense that the utilization of an actual critically schedulable task pattern can be arbitrarily close to the bound. This pattern was described in [1] for a single processor. The multiprocessor pattern simply contains m identical replicas of each task in the uniprocessor pattern.

Theorem 3: *Deadline monotonic scheduling is an optimal time-independent scheduling policy in the sense of maximizing the utilization bound.*

Proof: The proof comes from two observations: (i) the multiprocessor utilization bound of deadline-monotonic scheduling derived above is the same as the *optimal* single processor bound derived in [1]. (ii) The optimal multiprocessor utilization bound cannot be higher than the optimal uniprocessor bound. This is because for each unschedulable uniprocessor pattern there exists an unschedulable multiprocessor pattern, obtained by replicating the former on each processor. From (i) and (ii) it follows that deadline monotonic scheduling achieves the maximum possible multiprocessor utilization bound.

Corollary 2: *The optimal utilization bound of time-independent scheduling of liquid aperiodic tasks on a a multiprocessor is* $\frac{1}{1+\sqrt{1/2}}$

This corollary follows directly from Theorem 3 and Equation (13). The significance of the above result lies in its suitability for run-time admission control. In particular, we show that it leads to a constant time admission control test that respects the synthetic utilization bound without underutilizing the system.

5 Experimental Evaluation

Perhaps the most significant result of this paper is not that a multiprocessor utilization bound exists for aperiodic tasks but that an admission controller based on synthetic utilization does not underutilize the system. This seems counter-intuitive at first, because Lemma 4 of Theorem 1 states that the long term integral of synthetic utilization is equal to the (normalized) sum of computation times in the task set $\int_0^\infty U^\zeta(t)dt = C^\zeta/m$. Hence, the long-term average synthetic utilization is equal to the long-term average per-processor real utilization of the task set (defined as the percentage of time a processor is utilized). A synthetic utilization bound of only 58.6% therefore does not bode well for real system utilization.

Note, however, that if tasks finish before their deadlines it is possible for computed synthetic utilization to be nonzero even when all processors are idle. An online admission controller can easily detect such situations and reset the

synthetic utilization to zero. This does not affect schedulability, because the tasks that precede the multiprocessor idle time have no effect on the schedulability of future arrivals. Hence, without loss of generality, we redefine synthetic utilization as follows:

$$U(t) = \begin{cases} \frac{1}{m} \sum_{T_i \in V(t)} C_i/D_i & \exists\ busy\ processor \\ 0 & otherwise \end{cases} \quad (14)$$

Obviously, the average redefined synthetic utilization is generally lower than the original defined by Equation (2). Hence, the average (redefined) synthetic utilization is generally lower than the average real utilization. In this section we empirically show that the real utilization is indeed quite high.

Finally, observe that in the liquid task model, it is assumed that $C_i/D_i \rightarrow 0$ for all tasks. Thus, for a finite utilization to develop, the number of admitted tasks must be very large. This is intuitively true of high performance servers which execute a large number of requests concurrently. If one of the processors is idle, however, it must be that the request queue is depleted. The only tasks in the system are those currently executing on the remaining processors. In this paper, we are interested in systems where the number of processors is finite (e.g., at most 32). The contribution of a finite number of infinitesimal tasks to synthetic utilization tends to zero. Hence, a more aggressive heuristic admission controller can be used which resets multiprocessor synthetic utilization to zero when *any* processor is idle. In other words, for the liquid task model, we can define synthetic utilization as follows:

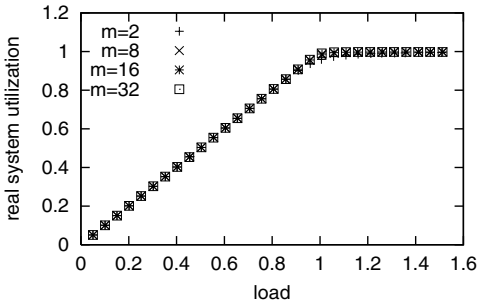
$$U(t) = \begin{cases} \frac{1}{m} \sum_{T_i \in V(t)} C_i/D_i & all\ processors\ busy \\ 0 & otherwise \end{cases} \quad (15)$$

The average synthetic utilization obtained from Equation (15) is generally even lower than the one obtained from Equation (14). Hence, keeping that utilization at 58.6% leads to even higher average real utilization, especially when the number of processors is large.

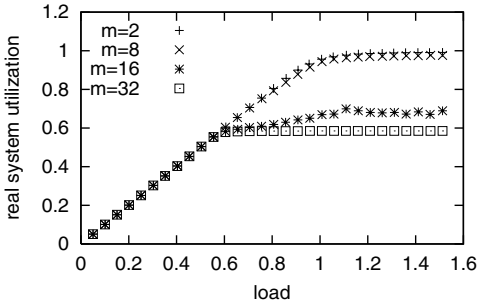
To demonstrate the effectiveness of constant-time admission control algorithms based on synthetic utilization, we generate a task arrival pattern with exponentially distributed interarrival times. The tasks are scheduled in a deadline-monotonic fashion on a multiprocessor of m CPUs. The number of CPUs is varied from 2 to 32 to explore scaling effects. The deadlines and execution times of tasks are uniformly distributed such that the average utilization of a single task is 0.5%. Hence, for most tasks, $C_i << D_j$. Tasks that miss their deadlines are dropped. The input load in our experiments is defined as $\frac{1}{m} \cdot \frac{\sum_{all\ generated\ tasks} C}{lenght\ of\ experiment}$. We vary the load by varying the expected value of the task inter-arrival times.

We consider two admission policies, reset-all-idle and reset-one-idle. They use Equation (14) and Equation (15) respectively as their definition of synthetic utilization. Both admit input tasks only if the utilization is less than 58.6%.

The resulting real utilization and rejected task ratio are shown in Figure 5 and Figure 6 respectively. The horizontal axis represents per-processor input load as a ratio to a processor's maximum capacity. Values above 1 represent overload. We can see from Figure 5-a, that reset-one-idle admission control offers a higher real system utilization of admitted tasks than reset-all-idle (Figure 5-b). When reset-one-idle is used, the average real utilization of admitted tasks increases with input load until about 100%, even though the synthetic utilization is kept below the bound by the admission controller. Hence, the multiprocessor is not underutilized. The real utilization improves slightly with the number of processors, because increasing the number of processors increases the opportunity to exercise our optimization (i.e., reset synthetic utilization).



(a) Admission control: reset-one-idle



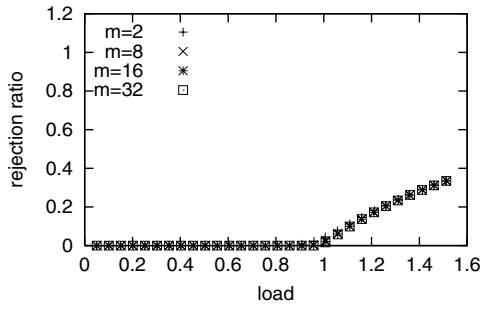
(b) Admission control: reset-all-idle

Figure 5. The Real Utilization

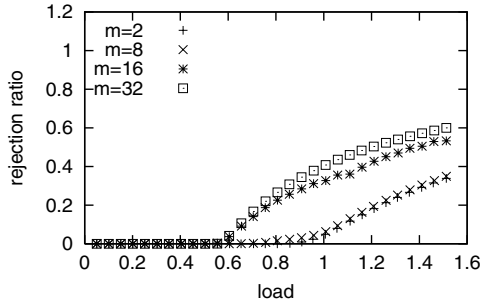
When reset-all-idle (Figure 5-b) is used for admission control, real utilization of admitted tasks is similar to the case of reset-one-idle for a small number of processors. When the number of processors is large (e.g., 16 or above) the real utilization saturates at a low value which is close to the synthetic utilization bound. This is expected because when the number of processor grows it becomes progres-

sively more difficult to reset synthetic utilization. Unlike the case with reset-one-idle, in reset-all-idle, utilization is reset only when *all* processors are idle.

Figure 6-a and Figure 6-b show the percentage of rejected tasks in the previous set of experiments for the reset-one-idle and reset-all-idle admission control policies respectively. As expected, the former policy rejects fewer tasks.



(a) Admission control: reset-one-idle



(b) Admission control: reset-all-idle

Figure 6. The Rejected Tasks

Finally, we note that reset-all-idle does not miss any deadlines (not shown in figures). In contrast, since reset-one-idle only approximates synthetic utilization, it is successful in preventing deadline misses only when the approximation is good. The approximation breaks down when the product of average task utilization and number of processors becomes non-negligible. Hence, as the number of processors increases, deadline misses occur among admitted tasks. Hence, this policy is more appropriate for soft-real-time applications. Figure 7 shows the number of missed deadlines of admitted tasks when reset-one-idle is used. It can be seen that the percentage of missed deadlines is very small. Hence, the heuristic offers a great compromise between efficiency and strictness of real-time guarantees.

To test our scheme in a real application, we implemented reset-one-idle on an Apache web server which uses a FIFO policy to serve all requests. We generated a workload in which the minimum deadline is 2 seconds and the maximum

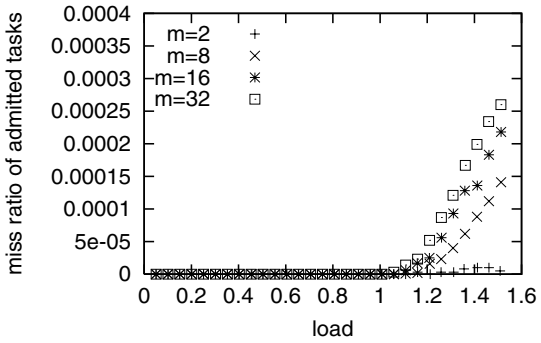


Figure 7. Miss Ratio of Admitted Tasks

deadline is 4 seconds. Hence, according to Equation (12), the bound is $1/3$. Since Apache is a multiprocess server, we implemented a shared memory data structure to keep track of synthetic utilization. The computation time of requests was approximately derived from the requested URLs using a formula $A + Bx$ where A and B are constants, and x is the requested file size.

Figure 8 compares the deadline miss rate of served requests with and without admission control as the request rate was increased. The top (dark) curve shows that when the server capacity is exceeded at approximately $200 req/s$, the number of requests that miss their deadlines increases linearly with load. When the admission control scheme is used these excess requests are rejected. Consequently, most admitted requests meet their deadlines. The bottom (light) curve in Figure 8 shows the deadline miss rate in the presence of utilization-based admission control. It can be seen that the number of missed deadlines is significantly reduced. Some misses do occur, however, despite admission control. We attribute that to inaccurate estimation of request computation times, as well as a non-zero request rejection cost which is not accounted for in the derivation of the utilization bound. This cost is incurred in the kernel primarily due to protocol stack processing which occurs before a request can be rejected.

In summary we have demonstrated that very efficient admission control policies can be implemented, based on our utilization bound, that work for aperiodic task arrival patterns. These policies keep processor utilization high despite enforcing the synthetic utilization bound. In real-life applications, our admission control significantly reduces the number of deadline misses. Future work must be done to address extensions of the bound that account for rejection cost, blocking, and uncertainty in execution times. We hope that this paper might catalyze and facilitate such future research.

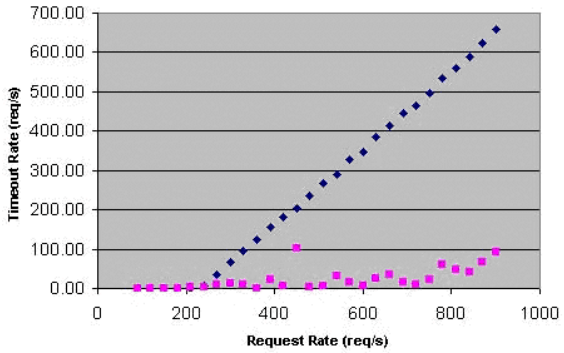


Figure 8. Testing an Apache Server

6 Related Work

The study of utilization bounds for schedulability of real-time tasks has been an active research topic in real-time computing since the publication of the first such bound by Liu and Layland [13], in 1973. Utilization bounds are the most efficient form of schedulability analysis in terms of computational complexity. Unfortunately, to date, this valuable tool was confined only to task models with periodicity constraints. The development of bounds derived for aperiodic tasks will enable applying this efficient analysis to a myriad of new applications in which workloads are random and do not have periodic behavior on short scales.

The basic utilization bound [13] states that a set of n independent periodic tasks on a uniprocessor will meet their deadlines if utilization is kept below $n(2^{1/n} - 1)$, which converges to 69% as n increases. The bound presents a sufficient (but not necessary) schedulability condition. The logic of the original derivation of this result has been fine-tuned in [5]. In [8], it is shown that if the values of task periods form K harmonic chains, where $K < n$, then the bound can be increased to $K(2^{1/K} - 1)$. In [3], the bound is further improved by considering more information about the task set such as the actual values of task periods. In [19] a design-time technique is described for computing the run-time bound when only periods (but not the execution times) are known *a priori*. The exact task execution times are plugged-in when the tasks arrive. In [7] task admissibility is improved using a polynomial-time admission control algorithm instead of the utilization bound. The exact characterization of the ability of rate-monotonic scheduling to meet deadlines of periodic tasks is presented in [10]. This characterization derives both sufficient and necessary conditions for schedulability of periodic tasks. A fault-tolerance extension was presented in [18], whose authors consider the overhead of failure recovery from a single fault. The authors prove that each task is recoverable under rate-monotonic scheduling (i.e., the backup replica will complete by the original deadline) if the utilization of the original task set is less than 0.5. This is less pessimistic than the trivial bound of $0.69/2$ (which trivially allows for any task to execute twice). A utilization bound for a modified rate-monotonic algorithm which allows deferred deadlines is considered in [20].

The basic utilization-based schedulability test has also been extended to the multiframe periodic task model in which successive invocations of a task alternate among multiple frames with different execution times [15]. Improvements of the basic multiframe schedulability test are proposed in [6].

Utilization bounds for multiprocessors have received recent attention. The authors of [16] derive the worst case achievable utilization of a critically schedulable task set using rate-monotonic scheduling on a partitioned multiprocessor. The authors of [14] derive a multiprocessor utilization bound for EDF scheduling. Their bound is shown to be $0.5(n + 1)$, where n is the number of processors. A less pessimistic (but more computationally involved) schedulability test is proposed in [9].

The above research efforts generally share in common the assumption that the task set is known *a priori* and that tasks are periodic (or have a minimum interarrival time). Aperiodic tasks were handled in previous literature in one of two ways. The first approach requires creation of a high-priority periodic server task for servicing aperiodic requests. Examples include the sporadic server [21], the deferrable server [23], and their variations [12]. The approach bounds the total load imposed on the system by aperiodic tasks allowing critical periodic tasks to meet their deadlines. It usually assumes that aperiodic tasks are soft, and attempts to improve their responsiveness rather that guarantee their deadlines. The second approach typically relies on algorithms for joint scheduling of both hard periodic and aperiodic tasks. It uses a polynomial acceptance test upon the arrival of each aperiodic task to determine whether or not it can meet its deadline. Examples include, aperiodic response-time minimization [11], slack maximization [4], slack stealing [24], the reservation-based (RB) algorithm [2], and the guarantee routines introduced most notably in the Spring kernel [22]. The utilization bound described in this paper is the first constant-time test that enables us to efficiently determine the schedulability of aperiodic workloads on multiprocessors.

7 Conclusions

In this paper, we derived, for the first time, the optimal utilization bound for the schedulability of aperiodic tasks on a multiprocessor under time-independent scheduling. The bound results in an $O(1)$ admission test of incoming tasks, which is faster than the polynomial tests proposed in earlier literature. We also showed that deadline monotonic scheduling is an optimal policy in the sense of maximizing

the schedulable utilization bound. This result contributes towards an aperiodic deadline monotonic scheduling theory — an analog of rate monotonic scheduling theory for the case aperiodic tasks. Such a theory may prove to be of significant importance to many real-time applications such as real-time database transactions, online trading servers, and guaranteed-delay packet scheduling algorithms. In such applications aperiodic arrivals have deadline requirements and their schedulability must be maintained. More importantly, by making a distinction between synthetic and measured utilization, we can show that the bound can be enforced on the former without considerably affecting the latter. Hence, unlike the case with periodic tasks, aperiodic utilization-based admission control does not underutilize the processor.

Acknowledgments

The authors would like to thank Shelby Funk, Deji Chen, Sanjoy Baruah, and Al Mok for their insightful comments on earlier manuscripts of this paper which revealed flaws in the original derivation and suggested several improvements. Special thanks goes to Lui Sha for commenting on an earlier version of the manuscript, and to P.R. Kumar for his insightful comments on the difference between real and synthetic utilization.

References

[1] T. Abdelzaher and C. Lu. Schedulability analysis and utilization bounds for highly scalable real-time services. In *Real-Time Technology and Applications Symposium*, TaiPei, Taiwan, June 2001.

[2] Y.-C. Chang and K. G. Shin. A reservation-based algorithm for scheduling both periodic and aperiodic real-time tasks. *IEEE Transactions on Computers*, 44(12):1405–1419, December 1995.

[3] D. Chen, A. K. Mok, and T.-W. Kuo. Utilization bound re-visited. In *Sixth International Conference on Real-Time Computing Systems and Applications*, pages 295–302, Hong Kong, China, December 1999.

[4] R. Davis and A. Burns. Optimal priority assignment for aperiodic tasks with firm deadlines in fixed priority pre-emptive systems. *Information Processing Letters*, 53(5):249–254, March 1995.

[5] R. Devillers and J. Goossens. Liu and layland's schedulability test revisited. *Information Processing Letters*, 73(5):157–161, March 2000.

[6] 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.

[7] C.-C. Han and H.-Y. Tyan. A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms. In *18th IEEE Real-Time Systems Symposium*, pages 36–45, San Francisco, CA, December 1997.

[8] T. W. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *IEEE Real-Time Systems Symposium*, December 1991.

[9] S. Lauzac, R. Melhem, and D. Mosse. An efficient rms admission control and its application to multiprocessor scheduling. In *First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, Orlando, FL, March 1998.

[10] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Real Time Systems Symposium*, Santa Monica, CA, December 1989.

[11] J. Lehoczky and S. R. Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Real-Time Systems Symposium*, pages 110–123, Phoenix, AZ, December 1992.

[12] T.-H. Lin and W. Tarng. Scheduling periodic and aperiodic tasks in hard real-time computing systems. *Performance Evaluation Review*, 19(1), May 1991.

[13] 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.

[14] J. M. Lopez, M. Garcia, J. L. Diaz, and D. F. Garcia. Worst-case utilization bound for edf scheduling on real-time multiprocessor systems. In *12th Euromicro Conference on Real-Time Systems*, pages 25–33, Stockholm, Sweden, June 2000.

[15] A. K. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, October 1997.

[16] D.-I. Oh and T. P. B. TP. Utilization bounds for n-processor rate monotone scheduling with static processor assignment. *Real-Time Systems*, 15(2), 1998.

[17] V. S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable web server. In *Proceedings of the 1999 USENIX Annual Technical Conference*, Monterey, CA, June 1999.

[18] 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.

[19] 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.

[20] 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.

[21] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1):27–60, June 1989.

[22] J. A. Stankovic and K. Ramamritham. The Spring Kernel: A new paradigm for real-time systems. *IEEE Software*, pages 62–72, May 1991.

[23] 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.

[24] 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.