A unified analysis of global EDF and fixed-task-priority schedulability of sporadic task systems on multiprocessors
TR-060401

Theodore P. Baker*
Department of Computer Science
Florida State University
Tallahassee, FL 32306-4530
phone: 1-850-644-5452
fax: 1-800-644-0058
e-mail: baker@cs.fsu.edu

Michele Cirinei†
Scuola Superiore Sant'Anna
Pisa, Italy
e-mail:cirinei@gandalf.sssup.it

Keywords

deadline, EDF, feasibility, fixed priority, multiprocessor, real time, scheduling, sporadic, multiprocessor, fixed priority, deadline, EDF

Abstract

Fixed-task-priority (FTP) scheduling and earliest-deadline-first (EDF) scheduling policies are alike in fixing the priority of each job of a task at the time the job is released. This common feature of FTP and EDF scheduling permits a unified analysis of scheduling failures, to derive new sufficient tests for meeting all deadlines of a set of independent sporadic tasks under a global preemptive FTP or EDF scheduling policy. The performance of the new tests has been evaluated in comparison to prior schedulability tests, by simulation. The new tests are able to verify some schedulable task systems that could not be verified by prior tests, but also fail to verify some systems that can be verified by prior analysis techniques. The biggest gain appears to be for fixed-task-priority scheduling, especially with post-period deadlines.

*This material is based upon work supported in part by the National Science Foundation under Grant No. 0509131.
†Visiting at the Florida State University, supported by PARADES.

1 Introduction

Multiprocessor platforms have long been used for high performance real-time systems, starting as early as 1971 [7]. Making effective use of multiprocessors for embedded real-time applications has become more important recently, with the introduction of multi-core microprocessor chips.

Historically, the dominant approach to scheduling real-time applications on a multiprocessor has been *partitioned*; that is, to assign each task (statically) to a processor, and then apply a single-processor scheduling technique on each processor. The alternative is *global* scheduling; that is, to maintain a single queue of ready jobs and assign jobs from that queue dynamically to processors. Despite greater implementation overhead, the global approach is conceptually appealing. It is well known from queueing theory that single-queue scheduling produces better average response times than queue-per-processor scheduling [13].

This report brings together two threads of prior research on the analysis of global preemptive scheduling for general sporadic task systems on multiprocessor platforms, and at the same time unifies the treatment of FTP and EDF scheduling. One thread is the busy-interval analysis of [2, 3, 4]. The other thread is the work of Bertogna, Cirinei and Lipari [8, 9], in which they observe that if a task τ_k misses a deadline the maximum fraction of the workload of any task that can contribute to the "interference" is $1 - \lambda_k$, where $\lambda_k \stackrel{\text{def}}{=} c_i/min(d_i, T_i)$.

The empirical performance of the new unified test is evaluated in comparison to prior schedulability tests, by simulation. The new test is shown to perform sometimes better than the previously known tests, and sometimes worse, so that the combination of the new and old tests is able to cover more cases than any of the tests alone. The new test seems to be especially advantageous for fixed-task-priority scheduling.

2 Definitions

For mathematical convenience, points and durations in real time are modeled by real numbers. However, in an actual system time is not infinitely divisible. The times of event occurrences (and durations between them) cannot be determined more precisely than one *tick* of the system's most precise clock. Therefore, any time value t involved in scheduling is assumed to be a non-negative integer value and is viewed as representing the entire interval $[t, t+1) \stackrel{\text{def}}{=} \{x \in \mathbb{R} | t \leq x < t+1\}$. The notation $[a, b)$ is used as a reminder that the interval includes all of the clock tick starting at a but does not include the clock tick starting at b.

A *sporadic task set* is a collection of sporadic tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each sporadic task generates potentially infinite sequence of *jobs*, and is characterized by a triple $\tau_i = (c_i, d_i, T_i)$, where c_i is the *worst-case execution time requirement* of

each job, d_i is the *deadline* of each job relative to its release time, and T_i is the *minimum separation* between the release times of the jobs. It is assumed that $c_i \leq \min(d_i, T_i)$, since otherwise a task would be trivially infeasible.

The *utilization* of τ_i is denoted by $u_i \stackrel{\text{def}}{=} c_i/T_i$, and the *density* of τ_i is denoted by $\lambda_i \stackrel{\text{def}}{=} c_i/min(d_i, T_i)$.

A *release time sequence* for a sporadic task τ_i is a sequence of times $r_{i,1} < r_{i,2} < \cdots$ (finite or infinite) such that $r_{i,j+1} - r_{i,j} \geq T_i$, for $j = 1, 2, \ldots$. A *release time assignment* for a task set is a mapping of a valid release time sequence to each task in τ. An m-processor *schedule* for a given task system and a given release time assignment is a partial mapping of tasks and time instants to processors, such that at most one task is assigned to each processor at any time t, and a task cannot be assigned to a processor at time t unless there is at least one job of τ_k that is *backlogged* at time t. A task τ_k is said to be *backlogged* at a time t in a schedule if there is at least one job of τ_k released at or prior to t that has nonzero remaining execution time at t.

The jobs of each task are assumed to have a precedence constraint; that is, they must be executed sequentially in order of release times. This is significant because d_i can be less than or greater than T_i, and so it is possible that the release time of a job may come before the completion time of the previous job of the same task.

We are interested in global preemptive *fixed-job-priority* scheduling, in which each job has a fixed priority that is distinct from all other jobs with which it competes for execution. At any instant in time all contending (i.e., released but not completed) jobs are totally ordered by priority. The m contending jobs with highest priority are assigned to processors, or if there are fewer than m contending jobs they all are assigned to processors.

Note that this fixed-job-priority model subsumes and unifies the following two specific priority schemes:

- *Fixed-task-priority (FTP)* scheduling: Jobs are ordered by task index; that is a job of τ_i has priority higher than a job of τ_j if $i < j$.
- *Earliest-deadline-first (EDF)* scheduling: Jobs are ranked by (absolute) deadline; that is a job of τ_i released at time r_i has priority higher than a job of τ_j released at time r_j if $r_i + d_i < r_j + d_j$. Ties are assumed to be broken arbitrarily; that is, if two jobs have the same deadline, we make no specific assumption about which can preempt the other.

Our goal will be to identify characteristics of a task set that must be satisfied for a scheduling failure to occur under either of these two scheduling policies, so that by checking that a task set does not have those characteristcs we can verify that the tasks in the set will always meet their deadlines.

The prior analyses of scheduling failures reported in [3, 4] were based on obtaining upper and lower bounds on the computational demand over a time interval preceding a first missed deadline of some task τ_k. The computational demand

of an interval was defined in terms of all the computation that could be done in the interval by tasks that have higher priority than τ_k. However, as pointed out by [8], when doing this sort of analysis one does not need to consider all the work done by tasks that *can* preempt τ_k, but only the time that such tasks actually *do* preempt τ_k. That distinction is captured by the concepts of *J-preemptive work*, and *block J-preemptive time*.

For notational simplicity, the definitions of these terms and the rest of this paper assume as context a particular release time assignment and schedule, a first scheduling failure for this schedule at time t, and a specific task τ_k that misses its deadline at t. The job J of τ_k with deadline t will be called the *problem job*.

A key concept to our analysis is the amount of execution time consumed by jobs that can successfully compete against the problem job for processor time, which we call the *J-preemptive work* of an interval. A job distinct from the problem job is J-preemptive if it has higher or equal priority than the problem job J, or is a preceding job of the same task.

The contribution $W_i^J(a,b)$ of a task τ_i to the J-preemptive work in an interval $[a,b)$ is the sum of the lengths of all the subintervals of $[a,b)$ during which a J-preemptive job of τ_i is scheduled to execute. the total J-preemptive work $W^J(a,b)$ in the interval $[a,b)$ is defined to be the sum $\sum_{i=1}^{n} W_i^J(a,b)$. The *$J$-preemptive load* is defined to be the ratio $W^J(a,b)/(b-a)$.

A second key concept is the amount of clock time during which all m processors are busy executing J-preemptive work, which we call the *block J-preemptive time* of an interval. The notation $B^J(a,b)$ is used for the total block J-preemptive time of an interval $[a,b)$. The *block J-preemptive time contribution* $B_i^J(a,b)$ of a task τ_i is defined to be the length of the subintervals of $[a,b)$ in which a τ_i is one of m tasks that have J-preemptive jobs scheduled to execute. The *block J-preemptive load* is the ratio $B^J(a,b)/(b-a)$.

The following are some relationships between the above definitions:

- $B_i^J(a,b) \leq W_i^J(a,b)$. The instants at which a J-preemptive job of task τ_i is scheduled along with $m-1$ other J-preemptive jobs is a subset of all the instants when that job is scheduled to execute.
- $mB^J(a,b) \leq W^J(a,b)$. At every instant where J experiences block preemption there are m J-preemptive jobs executing. All of that execution is included in the total J-preemptive work of the interval.
- $\sum_{i=1}^{n} B_i^J(a,b) = mB^J(a,b)$. Each instant where J experiences block preemption it is due to m jobs of the tasks $\tau_1 \ldots \tau_n$, and each instant where a task τ_i contributes to the block J-preemptive time it is one of m tasks causing block preemption for J.

The concept of block J-preemptive time introduced here is an extension of the concept of interference introduced in [8], to allow for the possibility that the deadline of a task may exceed its period. Since more than one job of the problem task

τ_k may execute in the interval, we must distinguish the problem job J from the problem task τ_k. However, the difference turns out not to be very great, since any job that is preemptive of a predecessor of J in τ_k is also J-preemptive. Therefore, the block J-preemption time of an interval is always greater than or equal to the τ_k interference.

The following lemma, which is adapted from [8], is useful for limiting the contribution of each task to the block J-preemptive time in a generic time interval. The idea is that to verify that the block preemption time is greater than a certain value x, it is sufficient to consider each single contribution up to x.

Lemma 1 *If $B^J(a,b) > x$, then one of the following is true:*

$$\sum_{i=1}^{n} \min(B_i^J(a,b), x) > mx \tag{1}$$

$$\sum_{i=1}^{n} \min(B_i^J(a,b), x) = mx, \text{ and } \forall i \; B_i^J(a,b) < x \Rightarrow B_i^J(a,b) = 0 \tag{2}$$

proof: Suppose $B^J(a,b) > x$. Let $S \stackrel{\text{def}}{=} \{ \; i \; | B_i^J(a,b) > x\}$ and $\xi \stackrel{\text{def}}{=} |S|$. If $\xi > m$, clearly condition (1) holds, so we consider only the case $\xi \leq m$.

$$\sum_{i=1}^{n} \min(B_i^J(a,b), x) = \xi x + \sum_{i \notin S} B_i^J(a,b) \tag{3}$$

If $\xi = m$, then condition (1) holds, or $\sum_{i \notin S} B_i^J(a,b) = 0$, in which case, since $\forall i B_i^J(a,b) \geq 0$, condition (2) holds vacuously.

If $\xi < m$, then based on (3) and the fact that $B^J(a,b) > x$,

$$\begin{aligned} \sum_{i=1}^{n} \min(B_i^J(a,b), x) &= \xi x + mB^J(a,b) - \sum_{i \in S} B_i^J(a,b) \\ &\geq \xi x + mB^J(a,b) - \xi B^J(a,b) \\ &= \xi x + (m - \xi) B^J(a,b) > mx \end{aligned}$$

□

In the next two sections the core results of [3, 4] are recast in terms of block J-preemptive time, in order to combine the lower and upper bounds on work found in these prior publications with the result of the above Lemma 1.

3 Lower Bound

In this section we derive a lower bound on the J-preemptive work of the maximal τ_k-busy interval ending at t, the missed deadline of the problem job.

A time interval is said to be τ_k-*busy* if τ_k is backlogged (continually) at all the points in the interval. A time interval $[t-\Delta, t)$ is said to be the *maximal τ_k-busy interval* with endpoint t if the interval is τ_k-busy and there is no $\Delta' > \Delta$ such that $[t-\Delta', t)$ is τ_k-busy. Note that $t-\Delta$ surely coincides with the release time of a job of τ_k, and if $d_k \leq T_k$ then $\Delta = d_k$.

Lemma 2 (lower bound) *If t is the first missed deadline and τ_k misses its deadline at t, then there is a unique maximal τ_k-busy interval $[t-\Delta, t)$ with endpoint t, $\Delta \geq d_k$, and*

$$B^J(t-\Delta, t)/\Delta > 1 - \lambda_k$$

proof: Since t is a missed deadline of some job J of τ_k, τ_k is continually backlogged from the release time of J through t. Let $\Delta = \max\{\delta \mid [t-\delta, t) \text{ is } \tau_k\text{-busy}\}$. Such a value Δ exists and is less than or equal to t, since τ_k cannot be backlogged before time zero. Since job J of τ_k is released at time $t - d_k$ and misses its deadline at t, the interval $[t-d_k, t)$ is τ_k-busy, and so $\Delta \geq d_k$.

By the maximality of Δ, $t-\Delta$ is the release time of a job of τ_k. Moreover, since τ_k is not backlogged before $t-\Delta$ and has a deadline miss at t, only jobs of τ_k that are released in $[t-\Delta, t)$ and have deadline in $(t-\Delta, t]$ can execute in the interval. If we define j to be the number of such jobs, jc_k is the execution time necessary to complete all these jobs. By the minimum interarrival constraint, and considering that j is an integer value we have that

$$(j-1)T_k + d_k \leq \Delta \Rightarrow j \leq \left\lfloor \frac{\Delta - d_k + T_k}{T_k} \right\rfloor \tag{4}$$

Since a deadline of τ_k is missed, part of the last job cannot be executed in the interval. Let y denote the sum of the lengths of all the subintervals in which τ_k does not execute. It follows that the total execution time of τ_k in the interval is $\Delta - y$. Since the jobs of τ_k must be executed sequentially, the amount of time that τ_k does not experience interference in this interval is less than jc_k. In formulae, we obtain the following

$$\Delta - y < jc_k \Rightarrow y > \Delta - jc_k$$

For any instant at which τ_k does not execute there are m jobs of other tasks executing, and those jobs can all preempt the current (earliest-released and uncompleted) job of τ_k at that instant. That current job is either J itself or a preceding job of the same task, so by the transitivity of priority all m of the executing jobs are J-preemptive, and so contribute to $B^J(t-\Delta, t)$. Therefore,

$$B^J(t-\Delta, t) > \Delta - jc_k \tag{5}$$

The rest of the proof has two cases. Consider first the case $d_k \leq T_k$. From the definition of maximal τ_k-busy interval,

$\Delta = d_k$, and so

$$B^J(t-\Delta,t)/\Delta > 1 - u_k - u_k(T_k - d_k)/d_k = 1 - \frac{c_k}{d_k} = 1 - \lambda_k$$

Consider now the case $d_k > T_k$. In this case $\Delta \geq d_k$ and the expression on the right of (5) is decreasing with respect to Δ, so

$$B^J(t-\Delta,t)/\Delta > 1 - u_k = 1 - \lambda_k$$

□

Note that it also follows that $W^J(t-\Delta,t) > m(1-\lambda_k) + \lambda_k$, since every instant at which there are m processors executing J-preemptive tasks contributes m units of J-preemptive work.

4 Upper Bound

In this section we introduce a parameter λ to generalize λ_k in the lower bound on the block J-preemptive load of an interval preceding the first missed deadline, and then derive an upper bound on the block J-preemptive demand as a function of λ. Because we do not know how to derive an upper bound on $B^J(a,t)$ directly, we instead look at $W^J(a,t)$ and rely on the fact that $W^J(a,t) \geq mB^J(a,t)$.

The first step is to bound the portion of $W_i^J(t-\Delta,t)$ that is contributed by jobs released before the interval, which we call *carried-in* jobs. In order to do this we will consider a specific interval ending at t, obtained by starting with the τ_k-busy interval guaranteed by Lemma 2 and extending it downward as far as possible while still maintaining a lower bound on the block J-preemptive time.

For any $\lambda > 0$, an interval $[t-\Delta,t)$ is defined to be $(1-\lambda)$-*busy* if $B^J(t-\Delta,t)/\Delta > 1-\lambda$. It is a *maximal* $(1-\lambda)$-busy interval if it is $(1-\lambda)$-busy and there is no $\Delta' > \Delta$ such that $[t-\Delta',t)$ is also $(1-\lambda)$-busy.

In other words, an interval is $(1-\lambda)$-busy if the block J-preemptive load is greater than $1-\lambda$, and it is maximal $(1-\lambda)$-busy if it is the longest $(1-\lambda)$-busy interval ending at t.

Lemma 3 *If t is a first missed deadline, τ_k is a task that misses its deadline at t, and $\lambda \geq \lambda_k$ then there is a unique maximal $(1-\lambda)$-busy interval $[t-\hat{\Delta},t)$ with endpoint t and length $\hat{\Delta} \geq d_k$.*

proof: By Lemma 2, there is a value Δ for which the interval $[t-\Delta,t)$ is $(1-\lambda)$-busy. Therefore, the set of all starting points $t' \leq t-\Delta$ of $(1-\lambda)$-busy intervals $[t',t)$ is non-empty. This set must have a minimal member, since there is a start time of the system, before which no jobs arrive. Let $\hat{\Delta} = t - t'$ for this minimum value t' and the lemma is satisfied. □

Recall that throughout this paper we assume as context an arbitrary sporadic task set $\tau = \{\tau_1, \ldots, , \tau_n\}$, an arbitrary release time assignment for τ, and an EDF or FTP schedule for this release time assignment. The first missed deadline occurs at time t, and task τ_k misses its deadline at time t. The job of τ_k with deadline t is the *problem job*.

In this context, the unique interval $[t - \hat{\Delta}, t)$ that is guaranteed by Lemma 3 when $\lambda \geq \lambda_k$ is called *the* $(1 - \lambda)$*-busy interval* for short, and $[t - \hat{\Delta}, t)$ always denotes that interval. The next lemma provides an upper bound on the J-preemptive work $W_i^J(t - \hat{\Delta}, t)$ done by each task τ_i in the $(1 - \lambda)$-busy interval.

Lemma 4 (upper bound) *If* $[t - \hat{\Delta}, t)$ *is the* $(1 - \lambda)$*-busy interval for task* τ_k *then for any task* τ_i*,*

$$W_i^J(t - \hat{\Delta}, t)/\hat{\Delta} \leq \beta_k^\lambda(i)$$

where

$$\beta_k^\lambda(i) \stackrel{def}{=} \begin{cases} 0 & \text{if } i \geq k \text{ and scheduling is FTP} \\ u_i(1 + \max(0, \frac{\gamma_i}{d_k})) & \text{if } u_i \leq \lambda \text{ and } (i \leq k \text{ or scheduling is EDF}) \\ u_i(1 + \max(0, \frac{d_i + \gamma_i - \lambda d_i/u_i}{d_k})) & \text{if } u_i > \lambda \text{ and } (i \leq k \text{ or scheduling is EDF}) \end{cases}$$

and

$$\gamma_i \stackrel{def}{=} \begin{cases} -d_i & \text{if } i = k \text{ and scheduling is EDF} \\ T_i - c_i & \text{if scheduling is FTP and } i < k \\ T_i - d_i & \text{if scheduling is EDF and } i < k \end{cases} \tag{6}$$

proof: The only interesting cases are those where $W_i^J(t - \hat{\Delta}, t)$ is nonzero. In those cases we can assume that there is at least one job of τ_i that has higher priority than the problem job. We will start by using reasoning similar to that of Lemma 2 to derive a lower bound on the amount of J-preemptive work that τ_i must complete before the start of $[t - \hat{\Delta}, t)$, and later use this to derive an upper bound on the amount of J-preemptive work that τ_i can contribute to $W_i^J(t - \hat{\Delta}, t)$.

Let $t - \hat{\Delta} - \phi$ be the release time of the first carried-in job of τ_i that contributes to $W_i^J(t - \hat{\Delta}, t)$, if such exists; otherwise, let $\phi = 0$. Observe that the way ϕ is chosen guarantees $\phi < d_i$.

If $\phi > 0$ the interval $[t - \hat{\Delta} - \phi, t - \hat{\Delta})$ is non-empty. Call this the *preamble* with respect to τ_i of $[t - \hat{\Delta}, t)$. Since the job of τ_i that is released at the start of the preamble has nonzero remaining execution time at the end of the preamble, there is only one job released in the preamble that can execute in the preamble. Call this job J'. Since J' contributes to $W_i^J(t - \hat{\Delta}, t)$, it has higher priority than the problem job. It follows that $W_i^J(t - \hat{\Delta} - \phi, t - \hat{\Delta})$ is equal to the amount of time that J' executes in the preamble. The only times in the preamble that J' does not execute are when m other jobs with higher priority are executing, and those must be from m distinct tasks. Moreover, J' has higher priority than the problem job, and the priority ordering is transitive, so any job that has higher priority than J' also has higher priority than the problem job. Therefore,

$$B^J(t - \hat{\Delta} - \phi, t - \hat{\Delta}) \geq \phi - W_i^J(t - \hat{\Delta} - \phi, t - \hat{\Delta}) \tag{7}$$

Also, since $[t - \hat{\Delta}, t)$ is $(1 - \lambda)$-busy,

$$B^J(t - \hat{\Delta}, t) > \hat{\Delta}(1 - \lambda) \tag{8}$$

Since, the block J-preemptive time of the concatenation of any two contiguous intervals is the sum of the block J-preemptive times of the intervals, by substitution of (7) and (8),

$$\phi - W_i^J(t - \hat{\Delta} - \phi, t - \hat{\Delta}) + \hat{\Delta}(1 - \lambda) < B^J(t - \hat{\Delta} - \phi, t) \tag{9}$$

Since $[t - \hat{\Delta}, t)$ is maximal $(1 - \lambda)$-busy and $\phi > 0$,

$$B^J(t - \hat{\Delta} - \phi, t) \leq (\hat{\Delta} + \phi)(1 - \lambda) \tag{10}$$

By transitivity, (9) and (10) imply that

$$\phi - W_i^J(t - \hat{\Delta} - \phi, t - \hat{\Delta}) + \hat{\Delta}(1 - \lambda) < (\hat{\Delta} + \phi)(1 - \lambda)$$

and so,

$$W_i^J(t - \hat{\Delta} - \phi, t - \hat{\Delta}) \geq \lambda\phi \tag{11}$$

That completes the analysis of the case $\phi > 0$. In the remaining case, where $\phi = 0$, we have $W_i^J(t-\hat{\Delta}-\phi, t-\hat{\Delta}) = 0 = \lambda\phi$, so (11) holds in all cases.

Given this lower bound on the amount of J-preemptive work that τ_i must complete before the $(1 - \lambda)$-busy interval, we can derive the following upper bound on the amount $W_i^J(t - \hat{\Delta}, t)$ of J-preemptive work that τ_i can complete within the interval.

$$W_i^J(t - \hat{\Delta}, t) = W_i^J(t - \hat{\Delta} - \phi, t) - W_i^J(t - \hat{\Delta} - \phi, t - \hat{\Delta}) \leq W_i^J(t - \hat{\Delta} - \phi, t) - \phi\lambda \tag{12}$$

The next part of the proof is to show that in each case (EDF and FTP) an expression γ_i can be defined that is independent of $\hat{\Delta}$ and ϕ and such that

$$W_i^J(t - \hat{\Delta} - \phi, t) \leq u_i(\hat{\Delta} + \phi + \gamma_i) \tag{13}$$

It is easy to see that the J-preemptive work due to τ_i in the interval $[t - \hat{\Delta} - \phi, t)$ is maximized when the jobs of τ_i are released as close together as possible, at times $t - \hat{\Delta} - \phi + jT_i$ for integers $j = 0, 1, 2, \ldots$. Let $t - \hat{\Delta} - \phi + \hat{j}T_i$ be the release time of the last job of τ_i that is J-preemptive.

We next must analyze the cases EDF and FTP separately, because the value of $\hat{j}$ depends on the job priorities.

For FTP: We are only interested in the case $i < k$, since $W_i^J(t - \hat{\Delta}, t) = 0$ if $i \geq k$. The amount of J-preemptive work done by the job of τ_i released at time $t - \hat{\Delta} - \phi + \hat{j}T_i$ is bounded by c_i and by $\hat{\Delta} + \phi - \hat{j}T_i$, so

$$W_i^J(t - \hat{\Delta} - \phi, t) \leq \hat{j}c_i + \min(c_i, \hat{\Delta} + \phi - \hat{j}T_i) \tag{14}$$

Suppose first that the term c_i is the minimum. We then have that $\hat{j} \leq (\hat{\Delta} + \phi)/T_i - u_i$, and so from (14) we obtain

$$W_i^J(t - \hat{\Delta} - \phi, t) \leq (\hat{j} + 1)c_i \leq (\frac{\hat{\Delta} + \phi}{T_i} - u_i + 1)c_i = u_i(\hat{\Delta} + \phi + (T_i - c_i)) = u_i(\hat{\Delta} + \phi + \gamma_i)$$

Suppose next that the term c_i is not the minimum. We have that $\hat{j} > (\hat{\Delta} + \phi)/T_i - u_i$, and so from (14) we obtain

$$\begin{aligned} W_i^J(t - \hat{\Delta} - \phi, t) &\leq \hat{j}c_i + \hat{\Delta} + \phi - \hat{j}T_i = \hat{\Delta} + \phi - \hat{j}(T_i - c_i) \\ &= \frac{\hat{\Delta} + \phi}{T_i}c_i + (\frac{\hat{\Delta} + \phi}{T_i} - \hat{j})(T_i - c_i) \\ &< \frac{\hat{\Delta} + \phi}{T_i}c_i + u_i(T_i - c_i) = u_i(\hat{\Delta} + \phi + \gamma_i) \end{aligned}$$

For EDF:

First, we consider the case of $i = k$. This case is special because the problem job is not preemptive of itself, but earlier jobs of τ_k are preemptive of the problem job. Since t is the deadline of the last job of τ_k, $t - \hat{\Delta} - \phi + \hat{j}T_k + d_k - T_k \leq t$, and so

$$\hat{j} \leq \frac{\hat{\Delta} + \phi - d_k - T_k}{T_k}$$

It follows that

$$W_k^J(t - \hat{\Delta} - \phi, t) \leq (\hat{j} + 1)c_i \leq \frac{\hat{\Delta} + \phi - d_k}{T_k}c_k = u_k(\hat{\Delta} + \phi + \gamma_k)$$

The remaining case is of $i \neq k$. The deadline of the last J-preemptive job of τ_i must be no later than t, *i.e.*, $\hat{j} < \frac{\hat{\Delta} + \phi - d_i}{T_i}$, and so

$$W_i^J(t - \hat{\Delta} - \phi, t) \leq (\hat{j} + 1)c_i < u_i(\hat{\Delta} + \phi + T_i - d_i) = u_i(\hat{\Delta} + \phi + \gamma_i)$$

That concludes the proof that (13) is satisfied for both FTP and EDF scheduling, so we can now substitute (13) into (12) and obtain

$$\frac{W_i^J(t - \hat{\Delta}, t)}{\hat{\Delta}} < \frac{u_i(\hat{\Delta} + \phi + \gamma_i) - \lambda\phi}{\hat{\Delta}} \tag{15}$$

Let $f(\phi, \hat{\Delta})$ be the function defined by the formula on the right side of inequality (15). That is

$$f(\phi, \hat{\Delta}) \stackrel{\text{def}}{=} u_i(1 + \frac{\gamma_i + \phi(1 - \lambda/u_i)}{\hat{\Delta}})$$

The value of $f(\phi, \hat{\Delta})$ is bounded by consideration of the following cases.

Case 1: Suppose $u_i > \lambda$. It follows that f is increasing with respect to ϕ, and since $\phi < d_i$,

$$f(\phi, \hat{\Delta}) \leq u_i(1 + \frac{d_i + \gamma_i - \lambda d_i/u_i}{\hat{\Delta}})$$

Case 1.1: Suppose f is decreasing with respect to $\hat{\Delta}$. Since $\hat{\Delta} \geq d_k$,

$$f(\phi, \hat{\Delta}) \leq u_i(1 + \frac{d_i + \gamma_i - \lambda d_i/u_i}{d_k})$$

Case 1.2: Suppose f is non-decreasing with respect to $\hat{\Delta}$. Taking the limit as $\hat{\Delta} \to \infty$,

$$f(\phi, \hat{\Delta}) \leq u_i$$

In both cases 1.1 and 1.2,

$$f(\phi, \hat{\Delta}) \leq u_i(1 + \max(0, \frac{d_i - \gamma_i - \lambda/u_i}{d_k})) = \beta_k^\lambda(i)$$

Case 2: Suppose $u_i \leq \lambda$. It follows that f is non-increasing with respect to ϕ, and since $\phi > 0$,

$$f(\phi, \hat{\Delta}) \leq u_i(1 + \frac{\gamma_i}{\hat{\Delta}}) \tag{16}$$

Case 2.1: If the expression on the right in (16) is decreasing with respect to $\hat{\Delta}$, since $\hat{\Delta} \geq d_k$, it follows that

$$f(\phi, \hat{\Delta}) \leq u_i(1 + \frac{\gamma_i}{d_k})$$

Case 2.2: If the expression on the right in (16) is non-decreasing with respect to $\hat{\Delta}$, by taking the limit as $\hat{\Delta} \to \infty$, it follows that

$$f(\phi, \hat{\Delta}) \leq u_i$$

Combining the two cases 2.1 and 2.2, we have

$$\frac{W_i^J(t - \hat{\Delta}, t)}{\hat{\Delta}} \leq f(\phi, \hat{\Delta}) \leq u_i(1 + \max(0, \frac{\gamma_i}{d_k})) = \beta_k^\lambda(i)$$

□

5 Schedulability Test

Theorem 1 (BC) *Let $\tau = \{\tau_1, \ldots, \tau_n\}$ be a set of independent sporadic tasks, and let $\beta_k^\lambda(i)$ be as defined in Lemma 4. The task set τ is EDF or FTP schedulable on m processors if, for every task τ_k there exists $\lambda \geq \lambda_k$ such that one of the following criteria is satisfied*

$$\sum_{i=1}^{n} \min(\beta_k^\lambda(i), 1 - \lambda) < m(1 - \lambda) \tag{17}$$

$$\textstyle\sum_{i=1}^{n} \min(\beta_k^\lambda(i), 1 - \lambda) = m(1 - \lambda) \text{ and } \exists i \; 0 < \beta_k^\lambda(i) < 1 - \lambda_k \tag{18}$$

proof: The proof is by contradiction. Suppose there is a task set τ and a release time assignment such that for the m-processor preemptive EDF schedule some task τ_k has a first missed deadline at time t. Let $[t-\hat{\Delta}, t)$ be the $(1-\lambda)$-busy interval guaranteed by Lemma 3.

Based on the definition of maximal $(1-\lambda)$-busy interval, by application of Lemma 1 with $x = (1-\lambda)\hat{\Delta}$, one of the following must be true:

$$\sum_{i=1}^{n} \min(\frac{B_i^J(t-\hat{\Delta}, t)}{\hat{\Delta}}, 1-\lambda) > m(1-\lambda) \tag{19}$$

$$\sum_{i=1}^{n} \min(\frac{B_i^J(t-\hat{\Delta},t)}{\hat{\Delta}}, 1-\lambda) = m(1-\lambda) \text{ and } \forall i \ \frac{B_i^J(t-\hat{\Delta},t)}{\hat{\Delta}} < (1-\lambda) \Rightarrow B_i^J(t-\hat{\Delta}, t) = 0 \tag{20}$$

Combining the fact that $B_i^J(t-\hat{\Delta}, t) \leq W_i^J(t-\hat{\Delta}, t)$ and the upper bound of Lemma 4, we can substitute $\frac{B_i^J(t-\hat{\Delta},t)}{\hat{\Delta}}$ in (19) and (20), to show that one of the following must be true:

$$\sum_{i=1}^{n} \min(\beta_k^\lambda(i), 1-\lambda) > m(1-\lambda) \tag{21}$$

$$\sum_{i=1}^{n} \min(\beta_k^\lambda(i), 1-\lambda) = m(1-\lambda) \text{ and } \forall i \ \beta_k^\lambda(i) \leq (1-\lambda) \Rightarrow \beta_k^\lambda(i) = 0 \tag{22}$$

It is easy to verify that if either of the two conditions above is true then both conditions (17) and (18) are false. □

The above theorem can be used as a schedulability test by testing the conditions for each value of k. The test is of complexity $O(n^3)$ since the only values of λ that need be considered are λ_k and the points where $\beta_k^\lambda(i)$ might change from non-decreasing to decreasing with respect to λ, *i.e.*, $\lambda = u_i$, $i = 1, \ldots, n$.

The schedulability test can be tightened slightly more. Though time and page constraints do not permit a full explanation here, the basic idea is simple. Observe that there can be at most $m-1$ tasks τ_i that carry J-preemptive work into the $(1-\lambda)$-busy interval $[t-\hat{\Delta}, t)$. That is because, by the definition of $(1-\lambda)$-busy, at least one processor must be idle at time $t-\hat{\Delta}$. The analysis of Lemma 3 can be repeated under the assumption that τ_i carries in no J-preemptive work, reducing the bound on $W_i^J(t-\Delta, t)$ by $c_i - \phi\lambda$. This results in a two-part bound $\beta_k^\lambda(i) = \alpha_k^\lambda(i) + \kappa_k^\lambda(i)$, such that $\kappa_k^\lambda(i)$ is a bound on the demand from carried in work and $\alpha_k^\lambda(i)$ is a bound on the demand from work that arrives within the $(1-\lambda)$-busy interval. In the schedulability test, one can then replace the sum of $\beta_k^\lambda(i)$ by the sum of $\alpha_k^\lambda(i)$ plus the sum of the $m-1$ largest values of $\kappa_k^\lambda(i)$.

6 Prior Work

6.1 Fixed Task Priorities

Andersson, Baruah, and Jonsson [1] first showed that any system of independent periodic tasks for which deadline equals period and the utilization of every individual task is at most $m/(3m-2)$ can be scheduled successfully on m processors

using rate monotonic scheduling if the total utilization is at most $m^2/(3m-1)$. Baruah and Goossens [6] also published a similar result, that a total utilization of at least $m/3$ can be achieved if the individual task utilizations do not exceed $1/3$.

Baker [2, 4] derived sufficient conditions for general FTP schedulability for sporadic task sets with unconstrained deadlines. The schedulability test of [4], restated in the notation of Theorem 1 to aid comparison, is as follows:

Theorem 2 (BAK-FTP) *A set of sporadic tasks $\tau = \{\tau_1, \ldots, \tau_n\}$ is schedulable on m processors using global preemptive fixed-priority scheduling if, for every task τ_k there exists $\lambda \geq \lambda_k$ such that*

$$\sum_{i=1}^{n} \min(\beta_k^\lambda(i), 1) \leq m(1-\lambda) \tag{23}$$

where

$$\beta_k^\lambda(i) \stackrel{def}{=} \begin{cases} 0 & \text{if } i >= k \\ u_i(1+\max(0, \frac{\gamma_i}{d_k})) & \text{if } u_i \leq \lambda\frac{m}{m-1} \\ u_i(1+\max(0, \frac{d_i+\gamma_i-\lambda\frac{m}{m-1}d_i/u_i}{d_k})) & \text{if } u_i > \lambda\frac{m}{m-1} \end{cases}$$

and γ_i is defined as in Theorem 1.

The differences, going from Theorem 2 to Theorem 1, are:

1. The bound used in the minimum is reduced from 1 to $1-\lambda$. This improvement was the main objective of the new analysis, using $B^J(a,b)$ instead of $W^J(a,b)$ in the definition of $(1-\lambda)$-busy. This change is most likely to pay off in improved accuracy when there are tasks with large densities.

2. In the definition of $\beta_k^\lambda(i)$, $\lambda\frac{m}{m-1}$ is replaced by λ. This is a loss, since it results in a larger value of $\beta_k^\lambda(i)$. It seems to be an unavoidable down-side consequence of the change above.

Later, Bertogna *et al.* [9] looked at fixed-task-priority scheduling and derived the following test.

Theorem 3 (BCL-FTP) *A set of sporadic tasks $\tau_1,\ldots,\tau_n$ with constraint $d_i \leq T_i$ is FTP schedulable on m identical processors if for each task τ_k one of the following is true:*

$$\sum_{i \neq k} \min(\beta_k^\lambda(i), 1-\lambda_k) < m(1-\lambda_k) \tag{24}$$

$$\textstyle\sum_{i\neq k} \min(\beta_k^\lambda(i), 1-\lambda_k) = m(1-\lambda_k) \text{ and } \exists i \neq k : 0 < \beta_k^\lambda(i) \leq 1-\lambda_k \tag{25}$$

where

$$\beta_k^\lambda(i) \stackrel{def}{=} \frac{N_i c_i + \min(c_i, \max(0, d_k - N_i T_i + d_i - c_i))}{d_k}, \text{ and } N_i \stackrel{def}{=} \left\lfloor \frac{d_k - d_i}{T_i} \right\rfloor + 1$$

Because the above result relies on the assumption that $d_i \leq T_i$, it is difficult to compare against Theorem 1. Both proofs use the technique of bounding $\beta_k^\lambda(i)$ by the lower bound on block J-preemptive demand, but they define demand slightly differently and look at different intervals. The principal advantage in precision of the above result seems to be that, because it only considers a single interval of known length (the one between a missed deadline and the corresponding release of τ_k), it can use the exact value of N_i. In contrast, the proof of Theorem 1 relies on simplifications made possible by over-bounding the floor function, in order to eliminate the dependences on the interval length $\hat{\Delta}$ and the release-time offset ϕ of the first carried-in J-preemptive job.

6.2 EDF Priorities

Goossens, Funk, and Baruah [11] showed that a system of independent periodic tasks can be scheduled successfully on m processors by EDF scheduling if the total utilization is at most $m(1 - u_{\max}) + u_{\max}$, where $u_{\max}$ is the maximum utilization of any individual task.

Baker [2, 3] derived several sufficient feasibility tests for m-processor preemptive EDF scheduling of sets of periodic and sporadic tasks with arbitrary deadlines. The EDF schedulability test of [4], restated to aid comparison with Theorem 1 here, is as follows:

Theorem 4 (BAK-EDF) *A set of independent sporadic tasks $\tau_1, \ldots, \tau_n$ is EDF-schedulable on m identical processors if, for every task τ_k, there exists a positive value $\lambda \geq \lambda_k$ such that*

$$\sum_{i=1}^{n} \min(\beta_k^\lambda(i), 1) \leq m(1 - \lambda) + \lambda$$

where

$$\beta_k^\lambda(i) \stackrel{def}{=} \begin{cases} u_i(1 + \max(0, \frac{\gamma_i}{d_k})) & \text{if } u_i \leq \lambda \\ u_i(1 + \max(0, \frac{d_i + \gamma_i - \lambda d_i/u_i}{d_k})) & \text{if } u_i > \lambda \text{ and } d_i \leq T_i \\ u_i(1 + \max(0, \frac{d_i + \gamma_i}{d_k})) & \text{if } u_i > \lambda \text{ and } d_i > T_i \end{cases}$$

and $\gamma_i \stackrel{def}{=} T_i - d_i$.

The differences going from Theorem 4 to Theorem 1 are:

1. The bound used in the minimum is reduced from 1 to $1 - \lambda$, as explained for Theorem 2.
2. The lower bound is reduced by λ. This a loss that seems to be the price of using block J-preemptive demand instead of J-preemptive work in the definition of $(1 - \lambda)$-busy.

3. The upper bound $\beta_k^\lambda(i)$ is reduced by $u_i \frac{T_i}{d_k}$ for the case $i = k$. This is an improvement from using block J-preemptive demand that only partially offsets the loss above.

4. The upper bound $\beta_k^\lambda(i)$ is reduced by $u_i \frac{\lambda d_i / u_i}{d_k}$ for the case $u_i > \lambda$ and $d_i > T_i$. This is an improvement that apparently could be backed into the proof of Theorem 4.

Bertogna, Cirinei and Lipari [8] observed that the proof of the utilization bound test of [11] extends naturally to cover pre-period deadlines if the utilization u_i is replaced by c_i/d_i. As observed by Sanjoy Baruah[1], the same proof extends to the case of post-period deadlines if utilization is replaced by *density*.

Theorem 5 (GFB) *A set of sporadic tasks $\tau_1,\ldots,\tau_n$ is EDF schedulable on m identical processors if*

$$\sum_{i=1}^{n} \lambda_i \leq m(1 - \lambda_{\max}) + \lambda_{max}$$

where $\lambda_{\max} \stackrel{def}{=} \max(\lambda_i | i = 1, \ldots, n)$.

Bertogna *et al.* [8] also derived the following sufficient test for EDF schedulability of task tests with constrained deadlines.

Theorem 6 (BCL-EDF) *A set of sporadic tasks $\tau_1,\ldots,\tau_n$ with constraint $d_i \leq T_i$ is EDF schedulable on m identical processors if for each task τ_k one of the following is true:*

$$\sum_{i \neq k} \min(\beta_k^\lambda(i), 1 - \lambda_k) < m(1 - \lambda_k) \tag{26}$$

$$\textstyle\sum_{i \neq k} \min(\beta_k^\lambda(i), 1 - \lambda_k) = m(1 - \lambda_k) \textit{ and } \exists i \neq k : 0 < \beta_k^\lambda(i) \leq 1 - \lambda_k \tag{27}$$

where

$$\beta_k^\lambda(i) \stackrel{def}{=} \frac{N_i c_i + \min(c_i, \max(0, d_k - N_i T_i))}{d_k}, \textit{ and } N_i \stackrel{def}{=} \left\lfloor \frac{d_k - d_i}{T_i} \right\rfloor + 1$$

The discussion of the relationship of Theorem 1 to Theorem 3 also applies to this result. In addition, observe that here the sums exclude the kth term, while in Theorem 1 the kth term is included, resulting in a loss of accuracy in some cases.

7 Empirical Comparisons

Bertogna *et al.* [8] reported simulations on collections of pseudo-randomly generated tasks sets with a few heavy tasks, for which the BCL was able to discover significantly more schedulable task sets than the GFB and BAK tests for EDF schedulability.

[1] personal communication

To discover how the new BC schedulability test compares to the previously known tests, a series of experiments were conducted on pseudo-randomly generated sets of sporadic tasks. Due to a page limit, only a few such experiments are reported here:

BAK Baker's FTP and EDF tests from [2, 3, 4], as stated in Theorems 2 and 4.

GFB Goossens, Funk and Baruah's EDF test, extended to arbitrary deadlines by Bertogna, Cirinei and Lipari, as stated Theorem 5.

BCL Bertogna, Cirinei, and Lipari's FTP and EDF tests, as stated in Theorems 3 and 6.

BC The unified FTP and EDF test based on Theorem 1 above.

7.1 Generation of Datasets

The performance of the schedulability tests was evaluated on several datasets. Each dataset contained 1,000,000 sets of tasks. The task periods were generated pseudo-randomly with a uniform distribution between 1 and 1000. The processor utilizations (and, implicitly, the execution times) were chosen according to the following distributions, truncated to bound the utilization between 0.001 and 1.0.

1. uniform distribution, between 1/period and 1.0
2. bimodal distribution: heavy tasks uniform between 0.5 and 1; light tasks uniform between 1/period and 0.5; probability of being heavy = 1/3
3. exponential distribution with mean 0.25
4. exponential distribution with mean 0.50

The bimodal distribution was intended to bias toward cases with a few heavy tasks, similar to the experiments of Bertogna, Cirinei, and Lipari [8], on which the BCL test performed well.

The deadlines were chosen in two different ways:

A constrained: deadlines uniformly distributed between the execution time and the period

B unconstrained: deadlines uniformly distributed between the execution time and 4 times the period (for all but the BCL test)

Separate datasets were generated for 2, 4, and 8 processor systems, as follows. An initial set of $m+1$ tasks was generated, and tested. Then another task was generated and added to the previous set, and all the schedulability tests were run on the

new set. This process of adding tasks was repeated until the total processor utilization exceeded m. The whole procedure was then repeated, starting with a new initial set of $m + 1$ tasks.

The above procedure rules out task sets that are trivially scheduable on m processors, one per processor. The tasks sets were further screened, by dropping all task sets that could be shown to be trivially schedulable on one processor because the total density was less than or equal to one, to be infeasible on m processors because the maximum processor demand [5, 10] exceeded m. After this screening many infeasible task sets are still included in the datasets. The only necessary and sufficient test for feasibility of global EDF scheduling of n tasks on m processors known to us has worst-case execution time of the order $O(mn \cdot \Pi_{i=1}^{n} T_i c_i)$. We implemented and tested that algorithm, but running it on datasets of the size considered here was not practical. Reporting the relative performance of the efficient sufficient tests of feasibility on large numbers of tasks sets seemed more important than comparing them against perfection on a much smaller number of task sets, with smaller periods.

Histograms are used to compare the success rates of the different schedulability tests. Each bucket x corresponds to the collection of task sets that have total utilization in a range between $0.01x$ and $0.01(x + 1)$, *i.e.*, a utilization range of one percent. The lines with points show how many task sets were verifiably feasible according to each of the three tests.

All the schedulability tests described in this paper were run on these datasets. Only the results of a few of the experiments are reported here, due to space limitations.

7.2 Results for Basic EDF

Figures 1, 2, and 3 compare the performance of the new BC test against the GFB, BAK, and BCL tests on datasets with constrained deadlines and bimodal task utilizations, on which the BCL test is known to do well. Histograms for $m = 4$ and $m = 8$ are shown. As can be seen from the huge gap between the plot of the total number of task sets (N) and the verifiably schedulable task sets in Figure 1, none of the tests performed very well. To make the differences between the schedulability tests more visible, Figures 2 and 3 omit the plot of N. The BCL test (solid squares) performed best over all for the bimodal utilization distribution, as expected. For datasets less weighted toward high density tasks (not shown here) the GFB test did better. The BC test peformed better than the BAK test for the two values of m shown, but performed worse than the BAK test for $m = 2$ (not shown).

Figures 4 and 5 show the performances of the GFB, BAK, and BC tests on datasets with *un*constrained deadlines and bimodal task utilizations, for which the BCL test does not apply. For two (not shown) or four processors, the GFB test does best, and the BC test actually does worse than the original BAK test. However, for eight processors, the BC test does better than the BAK and BC tests.

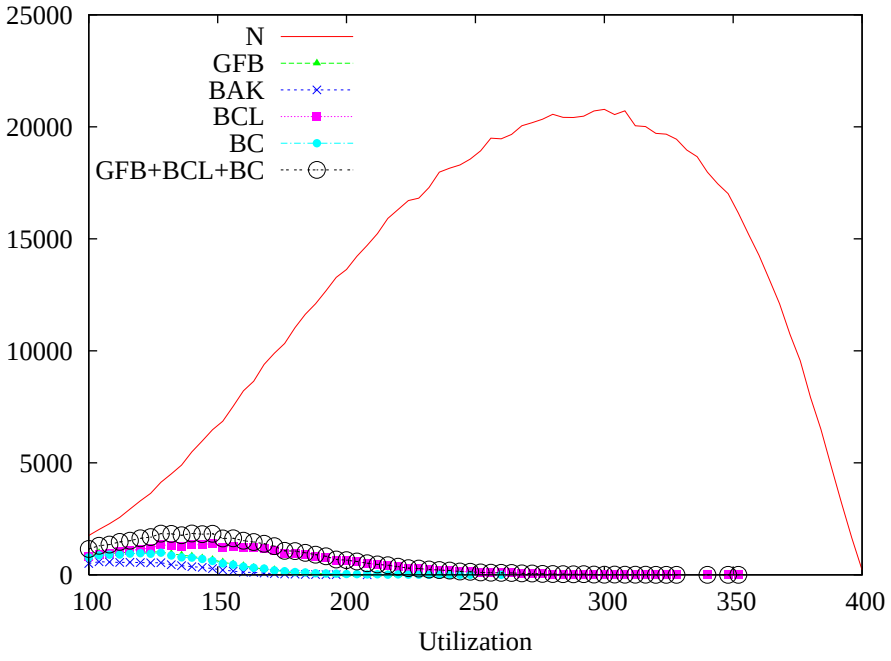

Figure 1. Histograms showing success of GFB, BCL, BAK, and BC tests for global EDF schedulability on 4 processors, of task sets with constrained deadlines and bimodal task utilizations, categorized by u_{sum}

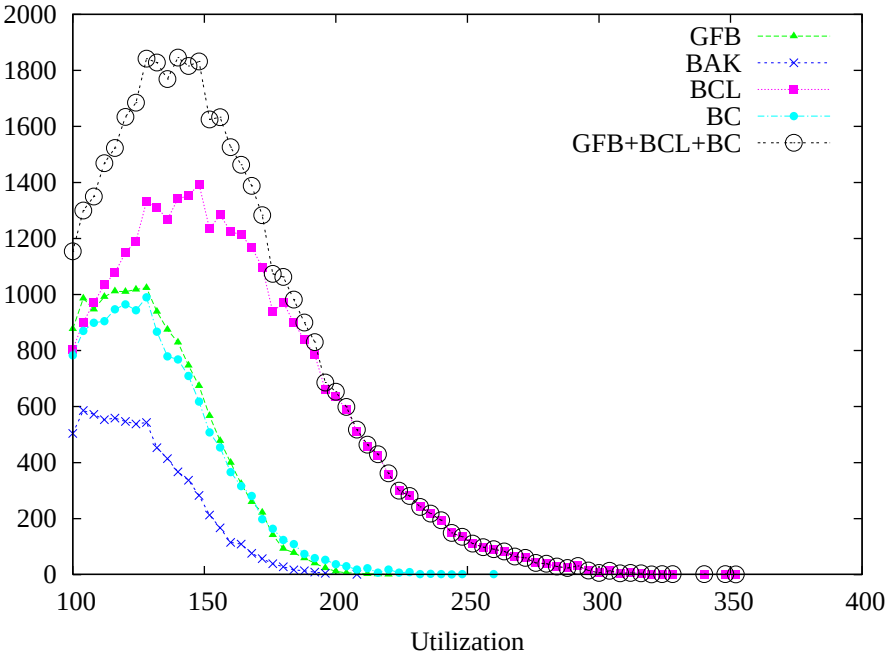

Figure 2. Same as Figure 1, but without N.

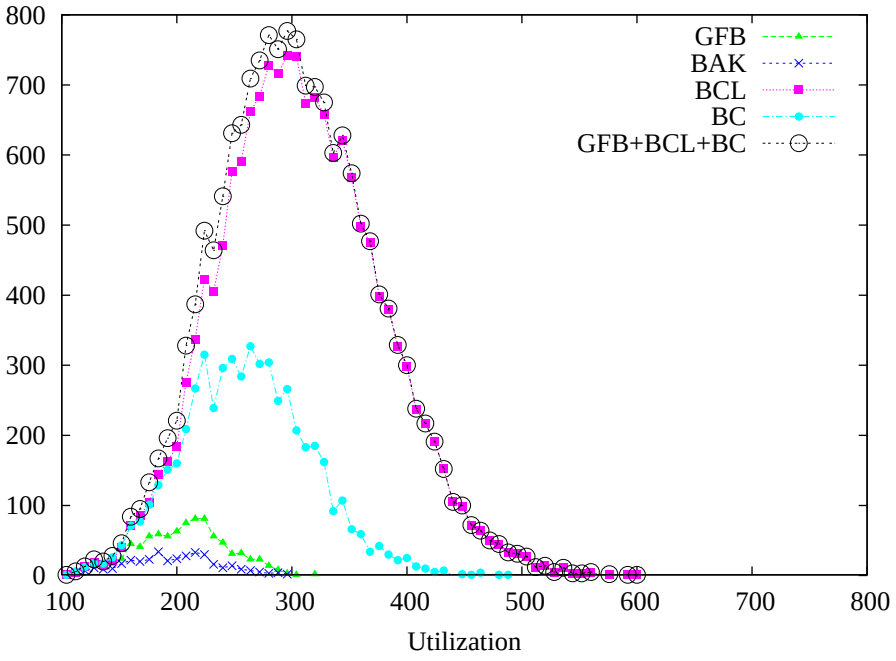

Figure 3. Same as Figure 2, but for 8 processors

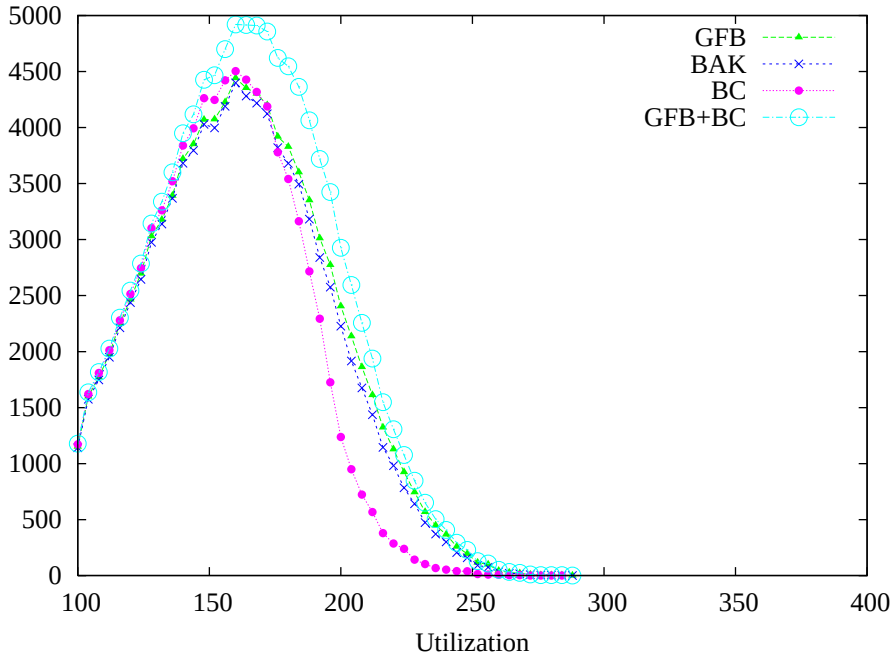

Figure 4. Histograms showing success of GFB, BAK, and BC tests for global EDF schedulability on 4 processors, of task sets with unconstrained deadlines and bimodal task utilizations, categorized by u_{sum}

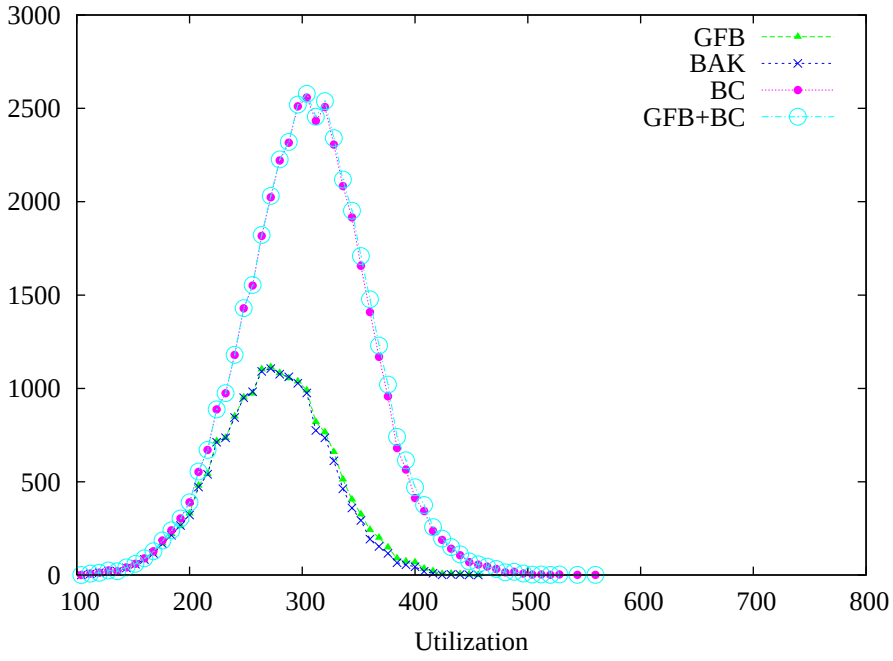

Figure 5. Same as Figure 4, but for 8 processors

From the performance on these datasets and others, it is clear that the tests are generally incomparable, and the performance of the BC test for EDF generally falls in the middle between the GFB, BAK, and BCL tests, sometimes being outdone by one and sometimes by the other. As mentioned above, the reasons for this mediocre performance can be traced back primarily to including the contribution of the task τ_k in the upper bound on J-preemptive work but not in the lower bound on block J-preemptive time.

7.3 Results for Basic Deadline-Monotonic

Figures 6 and 7 show the performances of the BAK, BCL, and BC tests for deadline-monotonic schedulability, on the same datasets as Figures 2 and 3. The BC test outperformed the BAK test, outperformed the BCL test for $m = 2$ (not shown) and $m = 4$, and did about the same as the BCL test for $m = 8$.

Figures 8 and 9 show the performances of the BAK and BC tests on the same datasets as Figure 4, with post-period deadlines, for which the BCL test does not apply. The BC test clearly outperforms the BAK test.

In general, performance of the FTP with all the available tests is better than EDF with any of the available tests, as can be seen by comparing Figure 2 to Figure 6, and Figure 4 to Figure 8. This is the strongest area of the BC test. It does better than the BAK test in nearly all cases, does better than the BCL test for most of the cases where the BCL test applies (those

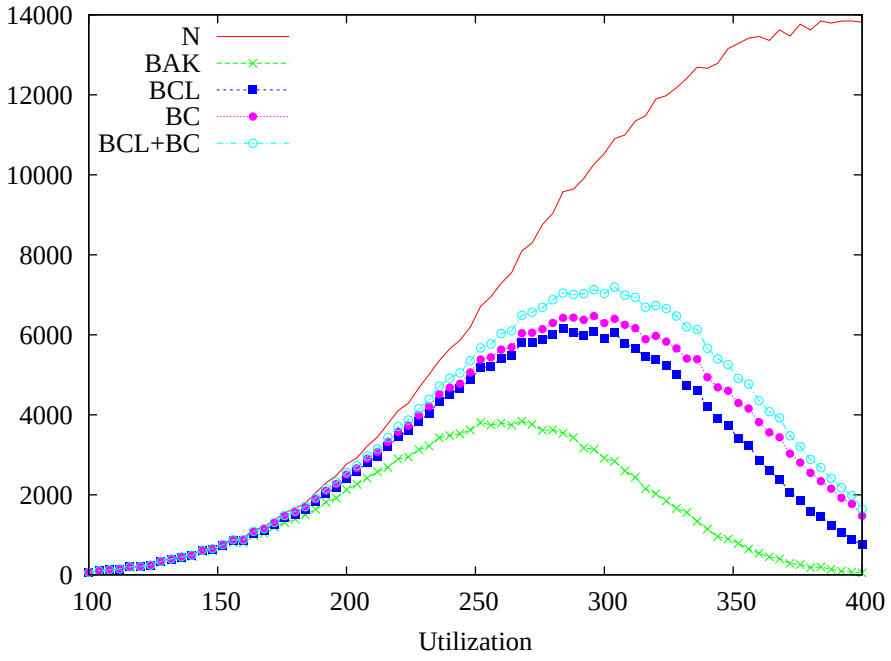

Figure 6. Histograms showing success of BAK, BCL, and BC tests for global deadline-monotonic schedulability on 4 processors, of the same task sets as Figures 2 and 3, categorized by u_{sum}

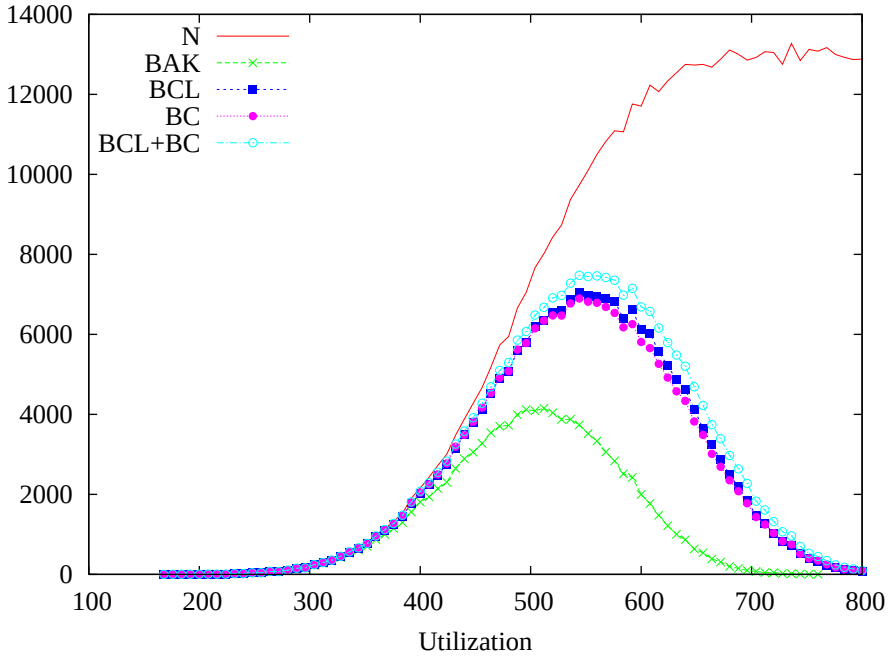

Figure 7. Same as Figure 6, but for 8 processors

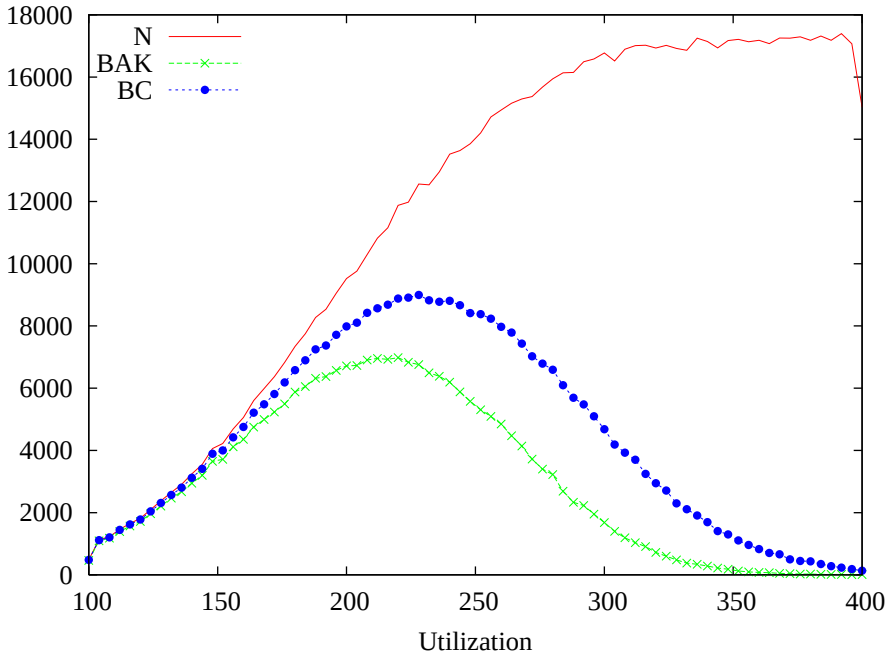

Figure 8. Histograms showing success of BAK and BC tests for global deadline-monotonic schedulability on 4 processors, for the same task sets as Figures 4 and 5, with post-period deadlines, categorized by u_{sum}

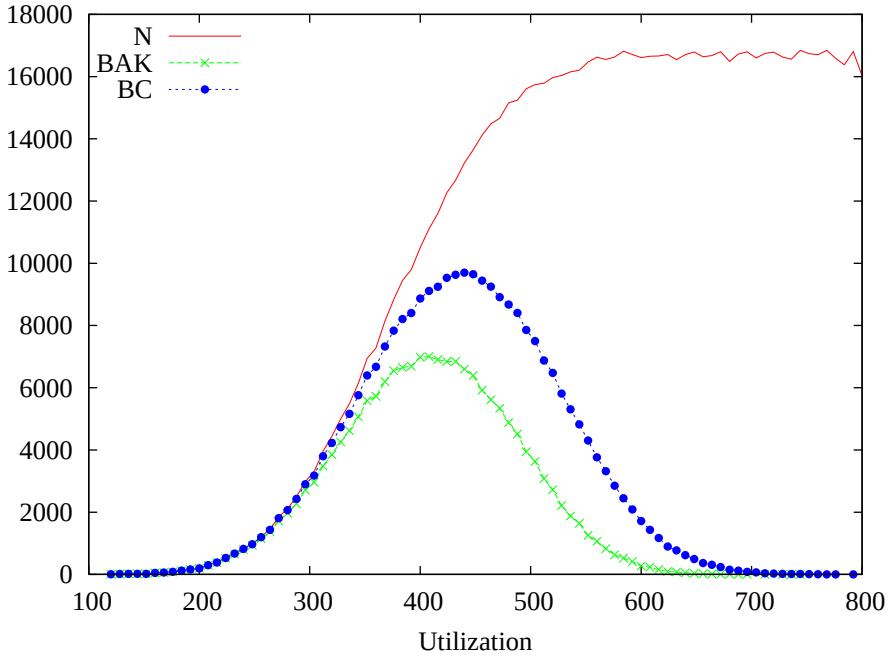

Figure 9. Same as Figure 8, but for 8 processors

with constrained deadlines), and applies to additional cases where the BCL test does not apply (those with unconstrained deadlines).

8 Conclusions and Future Work

Theorem 1 provides a unified derivation of a new sufficient test for m-processor schedulability of a set of independent sporadic tasks under preemptive global EDF and FTP policies. The derivation is based on a new metric, called block J-preemptive time. The new unified schedulability test (BC test) is more accurate for some tasks sets than previous tests based on computational "load" [2, 3, 4] and "interference" [8, 9]. However, there are also cases where it is less accurate, especially for EDF scheduling and small values of m.

The strength of the new BC test is FTP scheduling. The performance of FTP scheduling with the BC test in our experiments was significantly better, over all, than EDF scheduling with any of the known schedulability tests. The BC test also has a broader range of applicability, which extends beyond the BCL family of tests, to post-period deadlines. Such task sets are of interest because they occur in applications where input and output buffering is used to smooth out computational demand. For such task sets, with unconstrained deadlines, the BC test is able to verify schedulability of a significant number of task sets that could not be verified using the previously known algorithms.

Further improvements in schedulability tests appear to be possible. One idea for an improvement is sketched at the end of Section 5. Other improvements may be possible by retaining the floor functions in the upper and lower bounds all the way through the analysis.

In future work it would also be interesting to compare the performance of the various schedulability tests discussed here in a hybrid scheduling context. It is well known that scheduling with a pure global EDF or pure global deadline-monotonic policies is not as effective as with a hybrid policy in which up to $m - 1$ tasks with heavy resource consumption are given special high (super) priority. For example, see the following: the hybrid scheduling algorithm called RM-US[m/(3m-2)] in [1], which gives super priority to tasks with utilizations above $m/(3m-2)$; the policy called EDF-US[$m/(2m-2)$] in [14] and generalized to EDF-US[ζ] in [3], which gives super priority to tasks with utilizations above some constant threshold; the hybrid scheduling algorithm called PriD [12], which gives super priority to the k tasks ($0 \leq k < m$) with highest utilization, where the value k is chosen to be the minimum such that the remaining $n - k$ tasks satisfy a utilization-based schedulability test for $m - k$ processors; a generalization of the PriD idea to tasks with period not equal to deadline and other schedulability tests [3]; the hybrid algorithm called DM-DS[x] [9], which gives super priority to tasks with density above some threshold. All the tests for global EDF or FTP scheduling can be applied directly in such a hybrid context, by pretending each super-priority task is statically assigned to its own processor (even though the assignment is not static), and then applying the global schedulability test to the remaining tasks and processors. Such direct application seems to be more

pessimistic than necessary. Through further analysis, more precise schedulability tests for the hybrid scheduling schemes should be possible.

References

[1] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *Proc. 22nd IEEE Real-Time Systems Symposium*, pages 193–202, London, UK, Dec. 2001.

[2] T. P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proc. 24th IEEE Real-Time Systems Symposium*, pages 120–129, 2003.

[3] T. P. Baker. An analysis of EDF scheduling on a multiprocessor. *IEEE Trans. on Parallel and Distributed Systems*, 15(8):760–768, Aug. 2005.

[4] T. P. Baker. An analysis of fixed-priority scheduling on a multiprocessor. *Real Time Systems*, 2005.

[5] S. Baruah and N. Fisher. Partitioned multiprocessor scheduling of sporadic task systems. In *Proc. of the 26th IEEE Real-Time Systems Symposium*, Miami, Florida, Dec. 2005. IEEE Computer Society Press.

[6] S. Baruah and J. Goossens. Rate-monotonic scheduling on uniform multiprocessors. *IEEE Trans. Computers*, 52(7):966–970, July 2003.

[7] W. G. Bell. Department of the army, historical summary fiscal year 1971. Web, 1973.

[8] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proc. 17th Euromicro Conference on Real-Time Systems*, pages 209–218, Palma de Mallorca, Spain, July 2005.

[9] M. Bertogna, M. Cirinei, and G. Lipari. New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. In *Proc. of the 9th International Conf. on Principles of Distributed Systems*, Pisa, Italy, Dec. 2005.

[10] N. Fisher, T. P. Baker, and S. Baruah. Algorithms for determining the demand-based load of a sporadic task system. Submitted for Publication, 2006.

[11] J. Goossens and R. Devillers. Feasibility intervals for the deadline driven scheduler with arbitrary deadlines. In *Proc. 6th International Conf. Real-Time Computing Systems and Applications (RTCSA'99)*, 1999.

[12] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real Time Systems*, 25(2–3):187–205, Sept. 2003.

[13] L. Kleinrock. *Queueing Systems - Volume 2: Computer Applications*. Wiley Interscience, 1976.

[14] A. Srinivasan and S. Baruah. Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, 84:93–98, 2002.