Deriving the Average-case Performance of Bandwidth-like Interfaces for Tasksets with Infinite Minimum Inter-Arrival Time, Equal Task Density, Uniformly Distributed Deadlines, and Infinite Number of Tasks

Björn Andersson
Carnegie Mellon University

Hyoseung Kim
University of California, Riverside

John Lehoczky
Carnegie Mellon University

Dionisio de Niz
Carnegie Mellon University

ABSTRACT

Many solutions for composability and compositionality rely on specifying the interface for a component using bandwidth. Some previous works specify period (P) and budget (Q) as an interface for a component. Q/P provides us with a bandwidth (the share of a processor that this component may request); P specifies the time granularity of the allocation of this processing capacity. Other works add another parameter, *deadline*, which can help to provide tighter bounds on how this processing capacity is distributed. Yet other works use the parameters α and Δ where α is the bandwidth and Δ specifies how smoothly this bandwidth is distributed. It is known [4] that such bandwidth-like interfaces carry a cost: there are tasksets that could be guaranteed to be schedulable if tasks were scheduled directly on the processor, but with bandwidth-like interfaces, it is impossible to guarantee the taskset to be schedulable. It is known that this penalty can be infinite, i.e., the use of bandwidth-like interfaces may require the use of a processor that has a speed that is k times faster, and one can show this for any k. This brings the following question: "What is the average-case performance penalty of bandwidth-like interfaces?" A previous paper [5] has partially answered this question by stating an expression on this penalty as a function of taskset parameters and then randomly generated tasksets to obtain a probability distribution of this penalty. In this paper, we answer this question analytically for the case that the taskset has tasks with infinite minimum inter-arrival time, equal task density, uniformly distributed deadlines, and the number of tasks approaches infinity. For this specific case, we derive an expression; if deadlines are uniformly distributed in [0,1], then we find that the penalty is two. We also run experiments to explore systems with these assumptions but for finite number of tasks. From these experiments, we conclude that (i) the larger the number of tasks is, the larger the penalty is, (ii) the larger the number of tasks is, the less skewed the probability distribution is, and (iii) the larger the number of tasks is, the smaller the variance of the penalty is. We are currently working on the case where deadlines follow other distributions.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design—*Real-time systems and embedded systems*; G.4 [**Mathematical Software**]: Algorithm design and analysis

General Terms

Algorithms, Performance, Theory

Keywords

Real-time, Composability, Compositionality

1. INTRODUCTION

Consider a taskset τ scheduled on a single processor with Earliest-Deadline-First (EDF). Assume tasks are arbitrary-deadline sporadic tasks, i.e., a task τ_i is characterized by T_i, D_i, and C_i, with the interpretation that τ_i generates a sequence of jobs with at least T_i time units between two consecutive arrivals of jobs of τ_i and each job of τ_i has execution time at most C_i and each job has a deadline D_i time units relative to its arrival. It is known [7] that if for all positive t it holds that

$$\sum_{\tau_i \in \tau} \max(\lfloor \frac{t-D_i}{T_i} \rfloor + 1, 0) \times C_i \leq t \quad (1)$$

then the taskset is schedulable. (A taskset is schedulable, if for each jobset that it can generate, for each schedule that EDF can generate for this jobset, it holds that all jobs meet their deadlines; note that because we assume EDF with arbitrary tie-breaking, there may be more than one valid schedule for a jobset scheduled by EDF.) The result above (in Eq. 1) is well-known and allows software practitioners to efficiently verify, before run-time, that all timing requirements will be met at run-time. This result works assuming that (i) the entire taskset is known to a single person (or schedulability analysis tool), (ii) the system does not undergo design changes without rerunning schedulability analysis, and

(iii) tasks do not use more resources than stated by their parameters.

The real-time systems research community understood these limitations of using Eq. 1 as a schedulability test and using EDF at run-time. The research community understood that it was necessary to monitor the execution of a task to see if it executed more than it was expected to, and also to monitor a task to see if it generated jobs more frequently than it was expected to. In addition, the research community understood that in system integration, it is often advantageous to describe a set of tasks with related functionality with a simpler description. Therefore, the research community created a large set of solutions to achieve this. Typically, these solutions work as follows. A component (sometimes called a server task and sometimes called a subsystem) is characterized by a bandwidth parameter and some other parameters. One or more tasks are assigned to a component; each task is assigned to exactly one component. Then, at each instant at run-time, a root scheduler (sometimes called a global scheduler) selects a component and a local scheduler in this component selects a task in this component. This selected task executes on the processor. The bandwidth of a component is characterized as a share of the processor (for example, Component 1 should use at most 20% of the processor).

One could use a run-time mechanism that guarantees that this bandwidth is allocated to each component in each time interval (even "infinitely small" time intervals). But this would require infinitely many context switches which would make such a solution impractical. Therefore, the solutions presented in the research literature use another parameter as well: server period. Some solutions ensure that in a time interval of duration at least as large as the server period, a component is allocated processing time being at least as large as its bandwidth (i.e., bandwidth multiplied by server period). Many schemes in the literature suffer from a so-called blackout; for such a scheme, the guaranteed allocation is slightly less. Some schemes have a parameter, the server deadline, which can be used to control the blackout duration. Common to these schemes, however, is that before run-time, a schedulability test is performed on the root scheduler; it takes the bandwidth and potentially other parameters of each component and determines if the root scheduler will be able to allocate enough bandwidth to each component.

These bandwidth-like schemes have several advantages. First, they achieve isolation. This is important because it is often very difficult to find the worst-case execution time of a task. With these schemes, one can be sure that if the execution time of a job exceeds its estimated worst-case execution time, then it does not jeopardize timing guarantees of jobs in other components. Second, they allow hard and soft real-time tasks to be executed on a single processor. Third, they provide a simple interface for system integrators. In particular, the concept of bandwidth is easy to understand for laypersons. (For example: Component 1 is assigned 10% of the processor; Component 2 is assigned 70% of the processor; Component 3 is assigned 15% of the processor.) Fourth, they allow different schedulers to be used in different components (e.g., the local scheduler in Component 1 may be EDF and the local scheduler in Component 2 may be RM and the local scheduler in Component 3 may be FIFO.) Fifth, some real-time operating systems support their run-time mechanisms. Sixth, the run-time policing has low time and space complexity. Given all these advantages, it may seem that bandwidth-like schemes offer a good foundation for real-time systems.

Unfortunately, bandwidth-like schemes can waste an infinite amount of resources. It can be seen as follows: Consider a taskset τ with n tasks, $\tau_1, \tau_2, \ldots, \tau_n$ and these tasks have the parameters $T_i = \infty$, $D_i = i$, $C_i = 1$. If these tasks are scheduled directly on the processor (i.e., without components), it can be seen that the taskset is EDF-schedulable according to Eq. 1. Let us now discuss their behavior in a system with components and with a bandwidth-like scheme. Suppose that there are n components and one task in each component; specifically, task τ_i is assigned to component i. In order for the root scheduler to be schedulable, it is required that the sum of bandwidth of the components is at most 1. The bandwidth required by a component depends on the actual scheme used (the blackout duration matters and the predictability of the supply of processing time by the root scheduler matters); but for all bandwidth-like schemes, it holds that the required bandwidth of a component is at least as large as the sum of the density of the tasks in the component. This yields that component i requires at least the bandwidth C_i/D_i. Hence, component i requires at least the bandwidth $1/i$. Consequently, in order for the root scheduler to be schedulable, it must hold that $\sum_{i\in\{1..n\}} 1/i \leq 1$. It is easy to see that for $n \geq 2$, this condition is false and hence the system is not schedulable with a bandwidth-like interface. Let the processor be k times faster. Then, in order for the root scheduler to be schedulable, it must hold that $\sum_{i\in\{1..n\}} 1/i \leq k$. Letting n approach infinity yields that $\sum_{i\in\{1..n\}} 1/i$ is asympototic to $\ln n$ and hence it approaches infinity. Thus, even using a processor that is k times faster cannot guarantee that the system is schedulable with a bandwidth-like interface. We can do this reasoning for any k and hence we obtain that bandwidth-like interfaces can generate an infinite waste of resources.

This observation (that bandwidth-like interfaces can waste an infinite amount of resources) is known in the literature [3, 4]. There has also been a study [5] to determine how well bandwidth-like schemes perform in an average case as compared to a scheme that schedules tasks directly on the processor (i.e., without interfaces). The study [5] generated tasksets randomly, computed the penalty for the taskset, and generated histograms of the performance penalty. But less focus was given on understanding the result; no derivation of the results were performed.

Therefore, in this paper, we determine the average-case performance penalty of bandwidth-like schemes by deriving it analytically; we limit our scope, however, to only tasksets with infinite minimum inter-arrival time, equal task density, uniformly distributed deadlines, and infinite number of tasks.

The remainder of this paper is organized as follows. Section 2 formulates the problem in general and also for the special case where tasks have minimum inter-arrival times being infinite. Section 3 derives the performance penalty for a special case. Section 4 comments on the new results. Sec-

tion 5 conducts experiments on randomly generated tasksets and compares with our derived bound. (This is needed because the random generation of tasksets used in our derivation is different from the generation used in the previous study [5]). Section 6 presents related work. Section 7 gives conclusions.

2. FORMULATING THE PROBLEM

2.1 In general

From Eq. 1 it can be seen that for a taskset τ scheduled with EDF, a processor speed

$$\max_{t>0}(\sum_{\tau_i \in \tau} \max(\lfloor \frac{t-D_i}{T_i} \rfloor + 1, 0) \times \frac{C_i}{t}) \quad (2)$$

is sufficient to meet deadlines.

From the discussion in the previous section, it can be seen that if each task is in its own component, then with a bandwidth-like scheme, a processor speed

$$\sum_{\tau_i \in \tau} \frac{C_i}{\min(D_i, T_i)} \quad (3)$$

is necessary to meet deadlines.

For a taskset τ, let $\text{spdf}(\tau)$ be defined as:

$$\text{spdf}(\tau) = \frac{\sum_{\tau_i \in \tau} \frac{C_i}{\min(D_i, T_i)}}{\max_{t>0}(\sum_{\tau_i \in \tau} \max(\lfloor \frac{t-D_i}{T_i} \rfloor + 1, 0) \times \frac{C_i}{t})} \quad (4)$$

Here $\text{spdf}(\tau)$ should be read as speed-up factor. Intuitively, $\text{spdf}(\tau)$ indicates a lower bound on how much faster the processor needs to be in order for a bandwidth-like scheme to make the taskset τ schedulable.

Our goal is to compute $\text{spdf}(\tau)$ for a known probability distribution of taskset parameters.

2.2 Infinite minimum inter-arrival time

In this subsection, we consider the special case where for each task τ_i in τ, it holds that $T_i = \infty$.

For this case, $\text{spdf}(\tau)$ can be computed as:

$$\text{spdf}(\tau) = \frac{\sum_{\tau_i \in \tau} \frac{C_i}{D_i}}{\max_{t>0}(\sum_{\tau_i \in \tau} \theta(t-D_i) \times \frac{C_i}{t})} \quad (5)$$

where θ is the step function (it returns 1 if its input is non-negative and it returns 0 if its input is negative).

Look at the denominator of Eq. 5. Note that this step function only changes for those t that are equal to a D parameter. It can be seen that we only need to check those t that are equal to a D parameter. Using this observation yields:

$$\text{spdf}(\tau) = \frac{\sum_{\tau_i \in \tau} \frac{C_i}{D_i}}{\max_{\tau_j \in \tau}(\sum_{\tau_i \in \tau} \theta(D_j - D_i) \times \frac{C_i}{D_j})} \quad (6)$$

Look at the denominator of Eq. 5. Observe that we only need to include the terms where $D_j - D_i \geq 0$; the other ones are zero. With this observation, additional rewriting yields:

$$\text{spdf}(\tau) = \frac{\sum_{\tau_i \in \tau} \frac{C_i}{D_i}}{\max_{\tau_j \in \tau}(\sum_{\tau_i \in \tau \ s.t. \ D_i \leq D_j} \frac{C_i}{D_j})} \quad (7)$$

3. NEW RESULT: DERIVING THE PENALTY

Let us consider a taskset τ and let n denote the number of tasks in τ. Assume that the taskset is a constrained-deadline sporadic tasksets; i.e., for each task $\tau_i \in \tau$, it holds that $D_i \leq T_i$. Let us sort the tasks in ascending order of deadlines and let an index within parenthesis denote the index after the sorting. Clearly, it holds that:

$$i \leq j \Rightarrow D_{(i)} \leq D_{(j)} \quad (8)$$

Let $\{x..y\}$ denote the set of integers that are at least as large as x and at most y. With these notations, the right-hand side of Eq. 7 can be rewritten as:

$$\frac{\sum_{i=1}^{n} \frac{C_i}{D_i}}{\max_{j \in \{1..n\}}(\sum_{i \in \{1..j\}} \frac{C_{(i)}}{D_{(j)}})} \quad (9)$$

We assume that all tasks have the same density. From this assumption, it follows that there is a number X such that for each task τ_i, $\frac{C_i}{D_i} = X$. Using this on the above yields:

$$\frac{n}{\max_{j \in \{1..n\}}(\sum_{i \in \{1..j\}} \frac{D_{(i)}}{D_{(j)}})} \quad (10)$$

We will now discuss the expression $\frac{D_{(i)}}{D_{(j)}}$ in the denominator. A cumulative distribution function for a single random variable takes as input one parameter and outputs the probability that the random variable takes a value less than or equal to the parameter. The inverse cumulative distribution function for a single random variable takes as input a probability and outputs a value such that the cumulative probability distribution of that value is equal to the probability. Let F_D denote the cumulative distribution function of deadlines and let F_D^{-1} denote the inverse of the cumulative distribution function of deadlines.

If we take n samples from a distribution and n approaches infinity, then the ordered values of these samples will be given by the inverse cumulative distribution function. Specifically, if we take n values of deadlines and n approaches infinity, it holds that:

$$D_{(i)} = F_D^{-1}(\frac{i}{n+1}) \quad (11)$$

Applying Eq. 11 on Eq. 10 yields:

$$\frac{n}{\max_{j \in \{1..n\}}(\sum_{i \in \{1..j\}} \frac{F_D^{-1}(\frac{i}{n+1})}{F_D^{-1}(\frac{j}{n+1})})} \quad (12)$$

Recall that Eq. 12 is only valid for the case that n approaches infinity. We will now assume that deadlines are uniformly distributed in [DMIN,DMAX], where DMIN and DMAX are real numbers. Note that DMIN and DMAX are parameters we use for generating tasksets; there is no guarantee that in a given taskset, a task will have a deadline equal to these values. Then, we obtain that: $F_D(x) = \frac{x-\text{DMIN}}{\text{DMAX}-\text{DMIN}}$ and $F_D^{-1}(y) = \text{DMIN} + y \cdot (\text{DMAX} - \text{DMIN})$. This yields:

$$\frac{F_D^{-1}(\frac{i}{n+1})}{F_D^{-1}(\frac{j}{n+1})} = \frac{\text{DMIN} + \frac{i}{n+1} \cdot (\text{DMAX} - \text{DMIN})}{\text{DMIN} + \frac{j}{n+1} \cdot (\text{DMAX} - \text{DMIN})} \quad (13)$$

Applying Eq. 13 on Eq. 12 yields:

$$\frac{n}{\max_{j \in \{1..n\}}(\sum_{i \in \{1..j\}} \frac{\text{DMIN} + \frac{i}{n+1} \cdot (\text{DMAX} - \text{DMIN})}{\text{DMIN} + \frac{j}{n+1} \cdot (\text{DMAX} - \text{DMIN})})} \quad (14)$$

It can be seen (from reasoning in Appendix) that the max in the denominator occurs for $j = n$. This yields:

$$\frac{n}{\sum_{i\in\{1..n\}} \frac{\text{DMIN}+\frac{i}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{n}{n+1}\cdot(\text{DMAX}-\text{DMIN})}} \quad (15)$$

We can rewrite the expression as follows:

$$\frac{n\cdot(\text{DMIN}+\frac{n}{n+1}\cdot(\text{DMAX}-\text{DMIN}))}{n\cdot\text{DMIN}+\frac{\text{DMAX}-\text{DMIN}}{n+1}\cdot(\sum_{i\in\{1..n\}} i)} \quad (16)$$

We know that $\sum_{i\in\{1..n\}} i$ is $n\cdot(n+1)/2$. Applying this on the above yields:

$$\frac{n\cdot(\text{DMIN}+\frac{n}{n+1}\cdot(\text{DMAX}-\text{DMIN}))}{n\cdot\text{DMIN}+\frac{\text{DMAX}-\text{DMIN}}{n+1}\cdot(n\cdot(n+1)/2)} \quad (17)$$

Simplifying yields:

$$\frac{\text{DMIN}+\frac{n}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{\text{DMAX}-\text{DMIN}}{2}} \quad (18)$$

Recall that the above expression was derived assuming that n is infinite. Using this on the above yields:

$$\frac{\text{DMIN}+(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{\text{DMAX}-\text{DMIN}}{2}} \quad (19)$$

Rewriting yields:

$$\frac{2\cdot\text{DMAX}}{\text{DMIN}+\text{DMAX}} \quad (20)$$

4. NEW RESULT: OBSERVATIONS FROM PREVIOUS SECTION

We will now make observations from the result of the previous section (Eq. 20).

Observation 1: If we set DMIN = 0 then we get $\text{spdf}(\tau) = 2$. Hence, we can say that for this case, bandwidth-like interfaces cause a loss of a factor of two.

Observation 2: If we set DMIN = DMAX then we get $\text{spdf}(\tau) = 1$. Hence, we can say that for this case, bandwidth-like interfaces cause no loss.

Observation 3: Although we consider restrictions on taskset generation, it is still possible to experience an infinite waste (this did not happen during our experiments but it is possible). To see this consider a taskset with the following $D_i = \epsilon^{n-i}$. Applying this on Eq. 10 and letting n approach infinity and ϵ approach zero yields that the penalty is infinite. Note that in this example, all deadlines are in [0,1]. Hence, we do not need infinite deadlines in order to experience this bad performance. Thus, we have seen that the bad performance of bandwidth-like interfaces can occur even for tasksets that are more restricted (tasks with equal density and tasksets where all tasks have finite deadlines).

5. NEW RESULT: COMPARING THE DERIVED PENALTY AGAINST RANDOMLY GENERATED TASKSETS

In Section 3, we considered the case that the taskset has infinite minimum inter-arrival time, equal density of tasks, uniform deadline distribution, and infinite number of tasks. For this case, we derived the penalty of a bandwidth-like interface (Eq. 20). For the case that DMIN = 0, we saw that $\text{spdf}(\tau) = 2$. It is worthwhile to explore whether a similar result can be obtained when some of these assumptions are relaxed. We will do so in this section. We will explore the case where the assumption of infinite number of tasks is relaxed. And we will do so by generating tasksets randomly. For each taskset τ, we will compute $\text{spdf}(\tau)$ using Eq. 7. Finally, we will plot histograms. In these experiments, we assume DMIN = 0 and DMAX = 1.

The results are shown in Figure 1. It can be seen (in Figure 1a) that for the case of two tasks, the penalty is in [1,2]; the expected value of the penalty is approximately 1.4. With larger number of tasks, the variance decreases and the penalty increases too. One extreme can be seen for 100000 tasks (in Figure 1l) where the variance is very small and the penalty is around two. In order to see this more clearly, we generate the plots for 1000 tasks, 10000 tasks, and 100000 tasks with the x-axis in the range [1.80,2.20]. Results are shown in Figure 2. This allows us to see more clearly the behavior for large number of tasks.

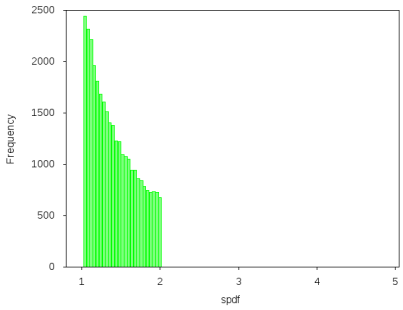
Since our proven bound is two for an infinite number of tasks, we would like to see histogram for spdf close to two with higher resolution. For this reason, we take the same data as before and plot them with the x-axis in the range [1.98,2.02]. Results are shown in Figure 3. This allows us to see even more clearly the behavior for large number of tasks.

In general, from these experiments, we conclude that (i) the larger the number of tasks is, the larger the penalty is, (ii) the larger the number of tasks is, the less skewed the probability distribution is, and (iii) the larger the number of tasks is, the smaller the variance of the penalty is.

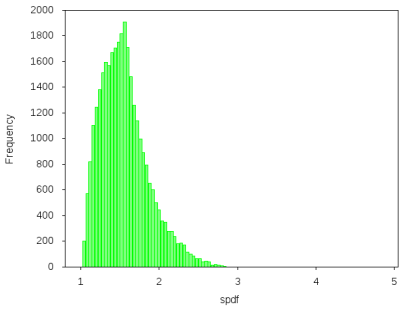
6. RELATED WORK

The literature on hierarchical scheduling, composability, and compositionality is vast; here we only survey some of the previous work. When Rate-Monotonic scheduling was developed as a comprehensive framework, it was recognized that many systems have software whose resource consumption is hard to characterize; e.g., it is hard to find its worst-case execution time or minimum inter-arrival time. For this reason, reservation-based frameworks were developed (see for example [18]). The run-time behavior of such a framework is as follows: A task may be associated with a server task and this server task is scheduled as a normal task (for example with an execution time, often called budget, and a period) and if a task is associated with a server task then it is only allowed to execute when the server task executes. In this way, if a task τ_i is in a server task and if τ_i experiences an execution overrun or arrives more often than expected then its impact on other tasks is bounded and it is bounded by the parameters of the server task. Later works created such reservation for EDF; one example of that is the constant-bandwidth server (CBS) [1] and presented a methodology for dimensioning [20].

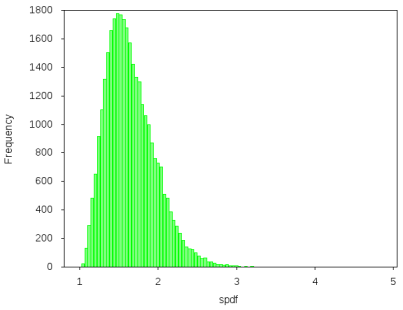
Researchers realized that reservation-based frameworks can be used to form hierarchical scheduling; some work that did so with EDF include [19, 11]. With the focus on hierarchical scheduling, Feng and Mok developed [14] a resource



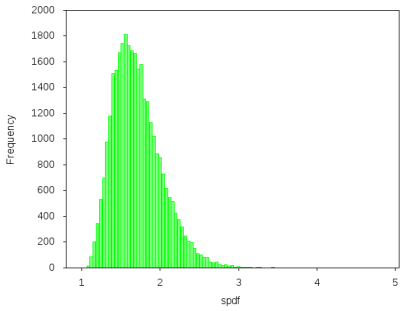
(a) Tasksets with $|\tau|$=2.



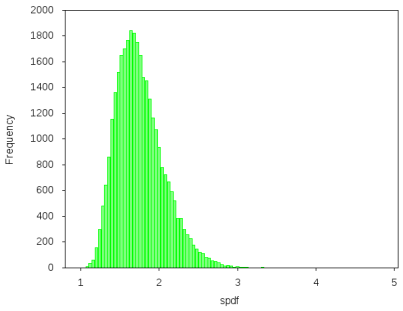
(b) Tasksets with $|\tau|$=3.



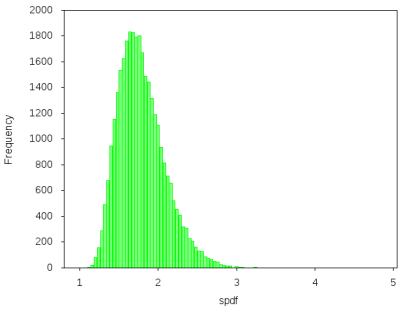
(c) Tasksets with $|\tau|$=4.



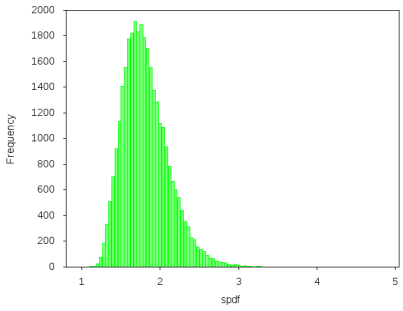
(d) Tasksets with $|\tau|$=5.



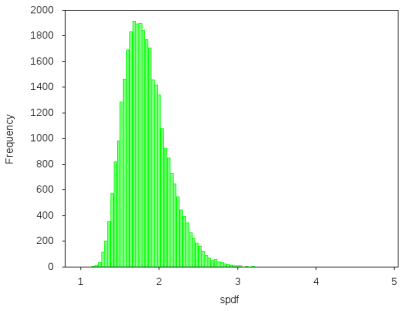
(e) Tasksets with $|\tau|$=6.



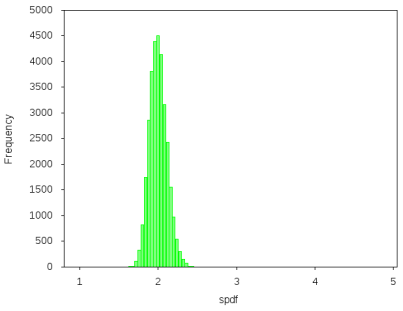
(f) Tasksets with $|\tau|$=7.



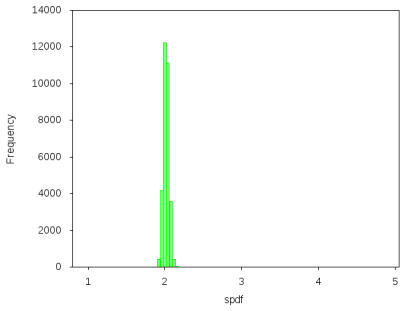
(g) Tasksets with $|\tau|$=8.



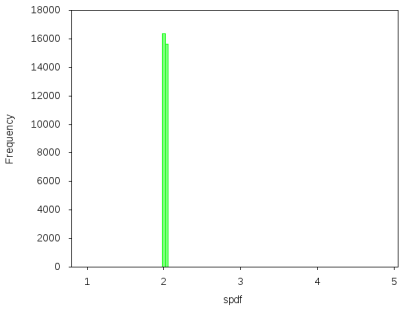
(h) Tasksets with $|\tau|$=9.



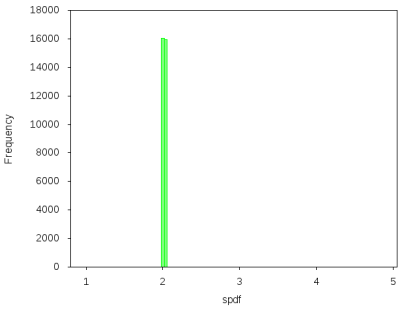
(i) Tasksets with $|\tau|$=100.



(j) Tasksets with $|\tau|$=1000.

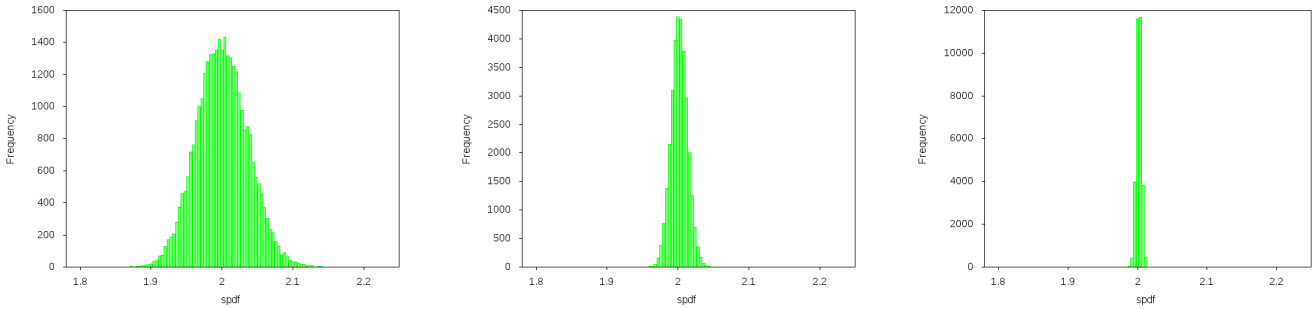


(k) Tasksets with $|\tau|$=10000.



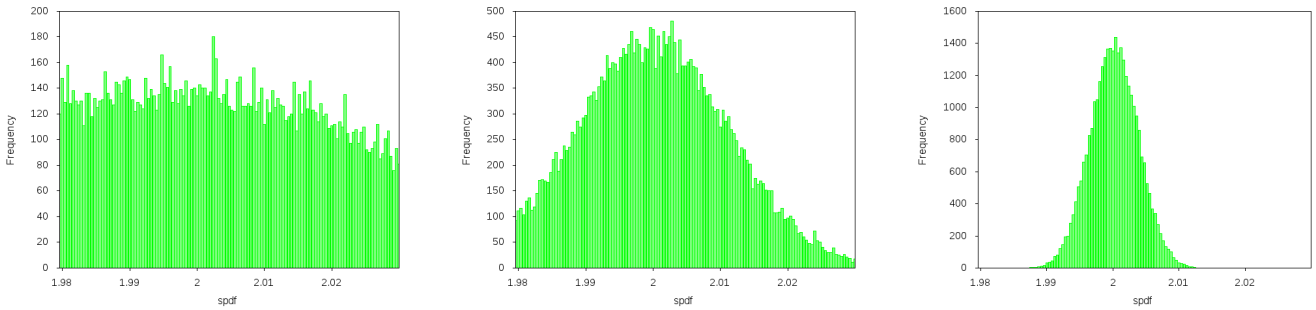
(l) Tasksets with $|\tau|$=100000.

Figure 1: New experimental results: Histograms for spdf for randomly-generated tasksets for the case that T=∞.



(a) Tasksets with $|\tau|$=1000. (b) Tasksets with $|\tau|$=10000. (c) Tasksets with $|\tau|$=100000.

Figure 2: New experimental results: Histograms for spdf for randomly-generated tasksets for the case that T=∞.



(a) Tasksets with $|\tau|$=1000. (b) Tasksets with $|\tau|$=10000. (c) Tasksets with $|\tau|$=100000.

Figure 3: New experimental results: Histograms for spdf for randomly-generated tasksets for the case that T=∞.

model which states that a root scheduler supplies, in a time interval of duration t, at least $(t - \Delta^k) * \alpha^k$ units of execution to component k. (A similar formulation has been used in real-time calculus [22].) Shin and Lee developed [21] another model where a component k is characterized by its period and execution time and with this, presented a so-called supply-bound function; this work could be used for components that use fixed-priority as a local scheduler or components that use EDF as a local scheduler. Using an explicit deadline in the interface can help reducing the amount of time for which the component is not supplied processing time [13].

Later works have focused on resource sharing, specifically the question, if a task τ_{lock} executes within a critical section and its current budget has reached zero, what should the scheduler do? There may be other tasks that will request this critical section and these requests can be granted only if the task that currently holds that critical section (τ_{lock}) has released it. In order for the task (τ_{lock}) to release the critical section, it must be able to execute and in order for this task (τ_{lock}) to execute, it must have a current budget greater than zero. Different solutions for this have been developed; see for example [10, 9, 8]. Clearly, server parameters must be selected in order to ensure schedulability; this has been the focus of [2].

Recently, compositional scheduling theories have been implemented in real-time virtualization [23, 16, 15], where a two-level hierarchical scheduling structure is typically used. In real-time virtualization, tasks are scheduled by the local scheduler within a virtual machine, and the resource demand of each virtual machine is presented to the root scheduler of a hypervisor by an interface.

7. CONCLUSIONS

Bandwidth-like schemes can cause a performance penalty. It was known that for certain tasksets, this performance penalty is infinite and an average-case performance evaluation had been performed on randomly-generated tasksets. But no derivation of the average-case performance was available. Therefore, in this paper, we derived the average-case performance for the case that the taskset has tasks with infinite minimum inter-arrival time, equal task density, uniformly distributed deadlines, and the number of tasks approaches infinity.

We believe this result is interesting because it follows the spirit of research in the real-time systems research community in two ways. First, the paper [17] offered a derivation of the performance of Rate-Monotonic for random tasksets (88%); in this paper, we perform a derivation of the performance of bandwidth-like interfaces. Second, the paper [12] identified "Dhall's effect" showing that in global multiprocessor scheduling an infinite amount of resources can be wasted; in this paper, we also study a system where an infinite amount of resources can be wasted.

The results in this paper studied the case that task density

is uniformly distributed. We are currently investigating generalizations of our derivation to other distributions as well.

Acknowledgment

Copyright 2017 ACM. All Rights Reserved. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. DM17-0007

8. REFERENCES

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *RTSS*, 1988.

[2] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *EMSOFT*, 2004.

[3] B. Andersson. A pseudo-medium-wide 8-competitive interface for two-level compositional real-time scheduling of constrained-deadline sporadic tasks on a uniprocessor. In *CRTS*, 2009.

[4] B. Andersson. A preliminary idea for an 8-competitive, log2 DMAX + log2 log2 (1/U) asymptotic-space, interface generation algorithm for two-level hierarchical scheduling of constrained-deadline sporadic tasks on a uniprocessor. In *CRTS*, 2010.

[5] B. Andersson. Evaluating the average-case performance penalty of bandwidth-like interfaces. In *CRTS*, 2015.

[6] B. Andersson, H. Kim, J. Lehoczky, and D. de Niz. Deriving the average-case performance of bandwidth-like interfaces for tasksets with infinite minimum inter-arrival time, equal task density, uniformly distributed deadlines, and infinite number of tasks. In *CRTS*, 2016.

[7] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. In *Real-Time Systems*, 1990.

[8] M. Behnam, T. Nolte, M. Asberg, and R. J. Bril. Overrun and skipping in hierarchically scheduled real-time systems. In *RTCSA*, 2009.

[9] M. Behnam, I. Shin, T. Nolte, and M. Nolin. SIRAP: a synchronization protocol for hierarchical resource sharing in real-time open systems. In *EMSOFT*, 2007.

[10] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *RTSS*, 2006.

[11] Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. In *RTSS*, 1997.

[12] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, February 1978.

[13] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using EDP resource models. In *RTSS*, 2007.

[14] X. Feng and A. K. Mok. A model of hierarchical real-time virtual resources. In *RTSS*, 2002.

[15] H. Kim, S. Wang, and R. Rajkumar. vMPCP: A synchronization framework for multi-core virtual machines. In *RTSS*, 2014.

[16] J. Lee et al. Realizing compositional scheduling through virtualization. In *RTAS*, 2012.

[17] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *RTSS*, 1989.

[18] J. P. Lehoczky, L. Sha, and J. K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *RTSS*, 1987.

[19] G. Lipari and S. K. Baruah. A hierarchical extension to the constant bandwidth server framework. In *RTAS*, 2001.

[20] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *ECRTS*, 2003.

[21] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *RTSS*, 2003.

[22] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *ISCAS*, 2000.

[23] S. Xi et al. RT-Xen: Towards real-time hypervisor scheduling in Xen. In *EMSOFT*, 2011.

APPENDIX

Proving a result about the maximum index of a sum

Consider

$$\frac{n}{\max_{j\in\{1..n\}}(\sum_{i\in\{1..j\}}\frac{\text{DMIN}+\frac{i}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{j}{n+1}\cdot(\text{DMAX}-\text{DMIN})})} \tag{21}$$

We will now prove that the max in the denominator occurs for $j = n$. We will do so by proving that for each $j \in \{1..n-1\}$, it holds that:

$$\sum_{i\in\{1..j+1\}}\frac{\text{DMIN}+\frac{i}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{j+1}{n+1}\cdot(\text{DMAX}-\text{DMIN})} \geq \sum_{i\in\{1..j\}}\frac{\text{DMIN}+\frac{i}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{j}{n+1}\cdot(\text{DMAX}-\text{DMIN})} \tag{22}$$

The k^{th} last term in the left-hand side of Eq. 22 is:

$$\frac{\text{DMIN}+\frac{j+1+(1-k)}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{j+1}{n+1}\cdot(\text{DMAX}-\text{DMIN})} \tag{23}$$

and the k^{th} last term in the right-hand side of Eq. 22 is:

$$\frac{\text{DMIN}+\frac{j+1-k}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{j}{n+1}\cdot(\text{DMAX}-\text{DMIN})} \tag{24}$$

We would like to show that the k^{th} last term in the left-hand side of Eq. 22 is at least as large as the k^{th} last term in the left-hand side of Eq. 22. Hence, we would like to show that for each k, Eq. 23 is at least as large as Eq. 24. That is, we would like to show that:

$$\frac{\text{DMIN}+\frac{j+1+(1-k)}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{j+1}{n+1}\cdot(\text{DMAX}-\text{DMIN})} \geq \frac{\text{DMIN}+\frac{j+(1-k)}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{j}{n+1}\cdot(\text{DMAX}-\text{DMIN})} \tag{25}$$

Equivalently, we would like to show that:

$$1+\frac{\frac{1-k}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{j+1}{n+1}\cdot(\text{DMAX}-\text{DMIN})} \geq 1+\frac{\frac{1-k}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{j}{n+1}\cdot(\text{DMAX}-\text{DMIN})} \tag{26}$$

Equivalently, we would like to show that:

$$1 - \frac{\frac{k-1}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{j+1}{n+1}\cdot(\text{DMAX}-\text{DMIN})} \geq 1 - \frac{\frac{k-1}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{j}{n+1}\cdot(\text{DMAX}-\text{DMIN})} \tag{27}$$

Equivalently, we would like to show that:

$$\frac{\frac{k-1}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{j+1}{n+1}\cdot(\text{DMAX}-\text{DMIN})} \leq \frac{\frac{k-1}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{j}{n+1}\cdot(\text{DMAX}-\text{DMIN})} \tag{28}$$

Note that the left-hand side differs from the right-hand side only in that in the left-hand side, in the denominator, there is $j+1$ rather than j. Hence, the left-hand has larger denominator than the right-hand side. Hence Eq. 28 is true. Since we have equivalent expressions above, it follows that Eq. 25 is true. Thus, it holds that the k^{th} last term in the left-hand side of Eq. 22 is at least as large as the k^{th} last term in the left-hand side of Eq. 22. From this, it follows that the sum of the j last terms in the left-hand side of Eq. 22 is at least as large as the the sum of the j last terms in the right-hand side of Eq. 22. Note that the left-hand side has one more term and it is non-negative. From this, it follows that Eq. 22 is true. For this reason, it holds that for

$$\frac{n}{\max_{j\in\{1..n\}}(\sum_{i\in\{1..j\}}\frac{\text{DMIN}+\frac{i}{n+1}\cdot(\text{DMAX}-\text{DMIN})}{\text{DMIN}+\frac{j}{n+1}\cdot(\text{DMAX}-\text{DMIN})})} \tag{29}$$

that the max in the denominator occurs for $j = n$.