Soft Real-Time on Multiprocessors: Are Analysis-Based Schedulers Really Worth It?*

Christopher J. Kenna†, Jonathan L. Herman†, Björn B. Brandenburg†, Alex F. Mills‡, and James H. Anderson†
Departments of Computer Science† and Statistics and Operations Research‡
The University of North Carolina at Chapel Hill

Abstract

The evolution of multicore platforms has led to much recent work on multiprocessor scheduling techniques for soft real-time workloads. However, end users routinely run such workloads atop general-purpose operating systems with seemingly good results, albeit typically on over-provisioned systems. This raises the question: when, if ever, is the use of an analysis-based scheduler actually warranted? In this paper, this question is addressed via a video-decoding case study in which a scheme based on the global earliest-deadline-first (G-EDF) algorithm was compared against Linux's CFS scheduler. In this study, the G-EDF-based scheme proved to be superior under heavy workloads in terms of several timing metrics, including jitter and deadline tardiness. Prior to discussing these results, an explanation of how existing G-EDF-related scheduling theory was applied to provision the studied system is given and various "mismatches" between theoretical assumptions and practice that were faced are discussed.

1 Introduction

The advent of multicore technologies has fueled much recent work on scheduling algorithms to support *hard real-time* (HRT) and *soft real-time* (SRT) systems on multiprocessor platforms. In a HRT system, no deadlines can be missed, while in a SRT system, misses are tolerable to some extent. Current multicore designs are arguably better suited for hosting SRT rather than HRT applications. In particular, HRT workloads require provable upper bounds on worst-case execution times (WCETs), and such bounds are difficult to determine on current multicore machines. In contrast, less rigorous execution-time analysis usually suffices for SRT systems.

Motivated by these observations, a number of researchers have investigated the possibility of supporting SRT applications using multiprocessor real-time scheduling algorithms for which formal schedulability analysis exists; such work has included both research on scheduling-theoretic issues (*e.g.*, [10, 11, 15]) and prototyping efforts (*e.g.*, [1, 13]). In practice, however, SRT applications are often implemented atop general-purpose operating systems (GPOSs), which typically use heuristic-oriented schedulers that must be "tuned" using a trial-and-error approach until SRT-like performance is achieved in the common case. To the best of our knowledge, no prior investigation has examined whether an analysis-driven scheduling approach actually yields superior results on a multiprocessor in practice. In this paper, we present such an investigation. Our goal is to determine: *when, if ever, is SRT multiprocessor schedulability analysis really worth it?*

Different definitions of SRT schedulability exist, depending on how the effects of deadline misses are constrained. Throughout this paper, we assume that a task system is SRT-schedulable if *deadline tardiness* is provably bounded. This definition of SRT schedulability was first considered in [10], and a number of additional results regarding bounded tardiness have been subsequently proven (*e.g.*, [11, 15]). Global schedulers, which allow any task to execute on any processor, have considerable advantages in meeting this definition of SRT schedulability. In particular, when scheduling sporadic task systems, bounded tardiness can be ensured with no schedulability related capacity loss under a variety of global algorithms, including the well-known global earliest-deadline-first (G-EDF) algorithm, which gives higher priority to jobs (task invocations) with earlier deadlines [10]. Due to bin-packing issues, the same is not true of partitioned algorithms, which statically bind tasks to processors. These tardiness results have been extended to hold in expectation when task execution costs are determined stochastically and processing capacity is provisioned based on average-case requirements [17, 18].

In this paper, we present details of a case study that was conducted to determine whether the desirable properties of G-EDF give it any discernible advantage over a widely used GPOS scheduler. The application considered in this study is a system of multiple video decoders. We chose this application not because it is a common use case,[1] but because it has interesting real-world characteristics (*e.g.*, video frames have variable processing times and precedence dependencies) and

*Work supported by NSF grants CNS 0834270, CNS 0834132, and CNS 1016954; ARO grant W911NF-09-1-0535; AFOSR grant FA9550-09-1-0549; and AFRL grant FA8750-11-1-0033.

[1] Such a system could conceivably be applied (for example) in an airline entertainment system, in a surveillance system that must process multiple video streams, or in applications for video compositing that run on ordinary desktop computers today and operate with multiple streams of (beyond-) HD-resolution.

because real-time correctness is straightforward to define (*e.g.*, display jitter should be low). The schedulers considered in this study include several scheduling options available in Linux, and a G-EDF-based framework called EDF-HSB [4] that is a hybrid of G-EDF and partitioned fixed-priority scheduling with budget enforcement. EDF-HSB is described further in Sec. 3. In provisioning the EDF-HSB-based system, we employed analysis pertaining to restricted processor supplies [15] and stochastic execution [18] that was unavailable when EDF-HSB was first presented. Indeed, one of the key contributions of this paper is to show how these disparate analysis pieces, which target different sub-problems, can be effectively combined. This required resolving various trade-offs that arise when integrating these results.

In conducting this study, we sought to answer two basic questions concerning the practical viability of SRT analysis. First, how well does such analysis predict the behavior of the system—in other words, how does theory compare to practice? Second, for which workloads does an analysis-based SRT scheduler perform more predictably than a GPOS scheduler (either "tuned" or "un-tuned")?

Related work. Quality-of-service support for multimedia workloads on *uniprocessors* is a well-studied problem (*e.g.*, see [7, 16, 19, 20]); however, practical *multiprocessor* SRT support has received comparatively little attention to date. A number of studies have been conducted in which different real-time multiprocessor schedulers[2] were experimentally compared *against each other*, but to our knowledge, no prior study has considered GPOS schedulers as well. In most of the prior studies that have been conducted, synthetically-created workloads were used (*e.g.*, [1]). Such studies are clearly of value, as they enable a controlled range of task systems to be analyzed. However, the ultimate test of a new scheduling algorithm is whether its use actually enables better implementations of *real* applications.

One notable effort in which practical applications were considered is work by Kato *et al.*, in which several real-time scheduling algorithms were experimentally compared against each other using (like us) a video-decoder workload [13, 14]. This is one of the first research efforts in which real workloads were used in evaluating different real-time schedulers. As in this paper, SRT systems were the focus of that work, and thus it is complementary to the effort described herein. However, the two research efforts differ in several key respects. First, our primary motivation is to determine whether analysis-oriented approaches offer any benefits over heuristic-oriented schedulers used in GPOSs. Such schedulers were not considered in [13, 14]. Second, while SRT applications are the focus of [13, 14], HRT analysis was actually employed, with average-case execution times used instead of WCETs. Moreover, it is not fully explained how this analysis was applied in provisioning the tested system. In contrast, we use analysis that allows stochastic effects to be accounted for, show how this analysis can be used to provision the system (both processing rates and memory buffers), and discuss potential mismatches between analytical-derived predictions of system performance and that seen in reality.

[2]Henceforth, all references to scheduling without qualification should be taken to mean *multiprocessor* scheduling.

Results. The first major goal of this study was to assess whether prior SRT analysis is actually *useful* in provisioning systems. Virtually any real system is going to violate some assumptions in theoretical analysis, and this is certainly true of the study here. However, we found that such violations had a negligible impact on predicted system performance (*e.g.*, such violations did not cause tardiness to grow unexpectedly).

Prior to conducting this study, the assumption of greatest concern was that per-job execution times are independent, which is required in determining expected tardiness bounds. Independence assumptions (of some degree) are *essential* for tractable stochastic analysis. Measured execution times revealed that frame decode times are not independent, which is not surprising. However, EDF-HSB re-distributes spare processing capacity via slack reclaiming and background scheduling, and these techniques tend to compensate for situations in which several successive (dependent) jobs execute for significantly longer than the provisioned average case.

Our second major goal was to compare EDF-HSB to Linux's default *completely fair scheduler* (CFS). We found that EDF-HSB and CFS yield comparable results when ample idle capacity exists. This matches the experience of end users who use Linux to host various kinds of SRT workloads (including video playback) on typically powerful, mostly idle computers. However, when faced with an increasingly heavy workload, the performance of CFS in terms of jitter, deadline miss ratios, and tardiness deteriorated quickly, whereas EDF-HSB maintained a predictable quality-of-service, albeit at the expense of reduced throughput for background processes.

In a final validation experiment, we approximated EDF-HSB's design on top of Linux's "real-time" SCHED_FIFO scheduler. The tardiness analysis used to provision EDF-HSB applies to global FIFO as well. Thus, the expected tardiness bounds derived for EDF-HSB (mostly) apply to SCHED_FIFO assuming a priority assignment that (roughly) emulates EDF-HSB. As expected, this SCHED_FIFO configuration proved to be more effective than CFS and comparable to EDF-HSB. This validates that our EDF-HSB implementation is efficient and highlights that stochastic SRT analysis is of general utility. However, unlike EDF-HSB, SCHED_FIFO was unable to maintain a minimum guaranteed background throughput in our experiments.

Taken together, these results answer our question: if the system is over-provisioned and processing capacity is ample, then the online scheduling problem is easy and the choice of scheduler has little impact on observable SRT performance; however, if the system is (almost) fully utilized, then the use of an analysis-based SRT scheduler has significant benefits.

Organization. In the following sections, we present needed background (Sec. 2), describe relevant prior theoretical SRT analysis (Sec. 3) and discuss challenges that arise when applying it in practice (Sec. 4), present the results of our case-study evaluation (Sec. 5), and then conclude (Sec. 6).

2 Background

We begin by describing the video-decoding workloads considered in this case study and the hardware/software environment within which these workloads were evaluated.

Video-decoding workloads. The workloads we consider consist of a number of high-definition video streams together with a number of repeatedly but irregularly arriving best-effort (BE) jobs that represent additional background processing. All examined videos have a frame rate of 23.98 fps and the majority have resolution 1920×800 or 1920×1080. The overall goal is to achieve acceptable real-time behavior for the video streams, while ensuring that BE jobs make progress (hence minimizing their response times to the extent practicable).

Each video stream must be decoded and then displayed and thus is naturally supported by two tasks. To avoid excessive display jitter, it is desirable to view each *display* task as a HRT task. However, requiring *decode* tasks to be HRT can lead to significant capacity loss, because conservative HRT analysis techniques would then be required of all real-time tasks; thus, we view such tasks as SRT. Our case study focuses on videos with a constant frame rate. In this case, each decode and display task maps cleanly to an implicit-deadline periodic task with one job (task invocation) per frame; with a frame rate of 23.98, each such task has a period of $1/23.98 \approx 41.7$ ms.

The various frames in a single video stream require different amounts of time to decode. This means that the per-job execution times of decode tasks can vary widely. Therefore, is it reasonable not only to implement such tasks as SRT tasks, but also to provision them using average-case execution times. The alternative of using worst-case times would lead to excessive capacity loss. Provided decode tasks have bounded tardiness, we can compensate for the expected deadline misses in decoding by inserting a frame queue between each decode task and the corresponding display task. In our implementations, such queues are wait-free, so priority inversions cannot occur when accessing them.

To summarize, the problem at hand is to support a collection of HRT and SRT periodic tasks along with aperiodic BE jobs on a multiprocessor system, where the HRT tasks are provisioned on a worst-case basis, and the SRT tasks are provisioned on an average-case basis.

Hardware/software environment. The hardware platform used in the presented study has two six-core Xeon X5650 processors (running at 2.67 GHz), giving the system a total of 12 CPU cores. The memory architecture is NUMA (non-uniform memory access), with one 12 GB memory module per processor. Each processor has a 12 MB L3 cache that is shared among its six cores.

This platform was used to examine various implementations of the video-decoding workloads described above both in Linux and in LITMUS$^{\mathrm{RT}}$. LITMUS$^{\mathrm{RT}}$ is a UNC-produced extension of Linux that supports a variety of multiprocessor real-time schedulers and synchronization protocols as plugins [9, 12]. The current version of LITMUS$^{\mathrm{RT}}$ is 2011.1 and is released as a patch against Linux 2.6.36. For this case study, we designed and implemented a new scheduler plugin that implements a scheduling policy called EDF-HSB, which is described in detail in Sec. 3.

We compared EDF-HSB both to the default Linux scheduler, the *completely fair scheduler* (CFS), as well as a Linux-provided policy called SCHED_FIFO. CFS is a partitioned scheduler that attempts to fairly distribute "virtual" run time among tasks. CFS's default scheduling decisions can be biased in favor or against certain tasks via the $nice()$ system call. SCHED_FIFO schedules tasks globally based on fixed priorities, with tasks at the same priority level prioritized on a FIFO basis. While SCHED_FIFO is a more real-time-oriented policy, CFS is the default Linux scheduler and thus is much more widely used, even in settings where SRT requirements exist. Indeed, SCHED_FIFO requires root privileges, which are not commonly granted to SRT applications.

In order to conduct viable comparisons of various implementation choices, we required a predictable video player with a straightforward design. We developed a simple video player that uses the FFmpeg library as the video demuxer[3] and decoder instead of modifying a more complex player such as the VLC Media Player, whose code-base has been under development for more than ten years. FFmpeg is same library suite used by popular Linux video players such as MPlayer and VLC. Each job of the decode task is responsible for demuxing the corresponding video file, decoding an individual video frame, scaling the result, and enqueuing it into a FIFO frame queue. Each job of the display task dequeues a single decoded frame and prepares it for display.

3 Applicable SRT Scheduling Results

Having provided a general description of the workload, we now discuss prior work on SRT schedulers and associated analysis that can be leveraged to obtain an analysis-based implementation. Note that our intent here is not to provide a thorough review of all prior related work, but to instead describe those mechanisms that we used in this study.

Although the main purpose of the paper is to present the implemented case study, integrating the various theoretical results prior to implementation was non-trivial. In particular, when combining previously published tardiness analysis for SRT tasks, the resulting tardiness bound obtained depended on the decode budgets in a non-trivial way that the authors of the previously published papers could not have anticipated.

[3] A demuxer separates the video and audio streams in a video file.

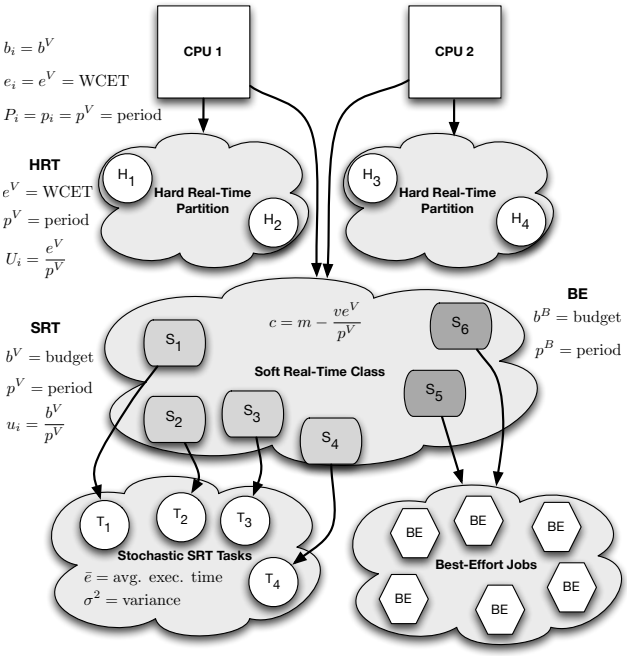


Figure 1: Example of EDF-HSB on two CPUs.

We discuss this issue further in Sec. 4 after reviewing the individual results next.

Task model. The workload in this case study can be represented by a task system $\tau = \{\tau^H, \tau^S\}$ to be scheduled on m CPUs, where τ^H is a set of periodic HRT display tasks $\{H_1, H_2, \ldots, H_{n^H}\}$, and τ^S is a set of SRT servers. The SRT servers are divided into two categories: $\{S_1, S_2, \ldots, S_{n^S}\}$, which are periodic and map directly to n^S SRT decode tasks that must be supported, and $\{S_{n^S+1}, S_{n^S+2}, \ldots, S_{n^S+m}\}$, which are sporadic and are used to schedule BE jobs. Each HRT task H_i is defined by its period P_i and its WCET e_i; its utilization is $U_i = e_i/P_i$. Each SRT server S_i is defined by its period p_i and its budget b_i; its utilization is $u_i = b_i/p_i$. We assume that there are m servers for BE work in order to give such work the opportunity to execute in parallel.

EDF-HSB. Brandenburg and Anderson introduced a framework called EDF-HSB to handle such mixed workloads on a multiprocessor, for cases where the HRT workload constitutes a relatively small fraction of the system's total utilization [4]. As illustrated in Fig. 1, EDF-HSB ensures that tasks in τ^H will never miss their deadlines by partitioning them among the m CPUs and by assigning them priorities that are higher than any SRT server. EDF-HSB ensures bounded tardiness for the SRT servers in τ^S by scheduling them using G-EDF using the processing capacity not consumed by the HRT tasks and by enforcing server budgets. (The figure also depicts stochastic SRT tasks, which were not considered in [4]—such tasks are defined below. Additionally, the figure summarizes important notation, some of which is introduced later.) Although timing guarantees cannot be made for BE work (because the properties of the BE workload are not known *a priori*), such a deployment ensures that BE work will not impact the timing guarantees for HRT and SRT tasks, yet the BE work's rate of service is lower-bounded.

EDF-HSB also allows for dynamic slack reclamation to reduce tardiness of SRT tasks and improve response times of BE jobs. This reclaiming scheme is similar to the method of M-CASH [8], which re-allocates processing capacity that becomes available when a constant-bandwidth-server completes early. In this implementation of EDF-HSB, both SRT tasks that are likely to be tardy and BE jobs can receive such capacities to improve performance.

Allowing stochastic execution times. Stochastic methods are required to properly analyze systems with average-case-provisioned components (like our decode tasks). For this purpose, we employ recent work by Mills and Anderson [18] that allows the v video decoding tasks $\{T_1, T_2, \ldots, T_v\}$ to be specified as *periodic stochastic tasks*, where task T_i has period p_i, mean execution time $\bar{e}_i$, and variance σ_i^2, and successive jobs of task T_i have independent random execution times satisfying the given mean and variance. Each such task T_i receives processor time that is allocated from the corresponding server S_i. This model has several desirable properties. First, unlike WCET, the mean and variance of video decoding times can be easily estimated from experimental data. Second, the system may be provisioned based on average execution times. That is, for task T_i to run on server S_i with bounded expected tardiness, we need only $b_i > \bar{e}_i$. The analysis for such tasks separates the deterministic tardiness, which may occur due to SRT *servers* missing deadlines, from the stochastic tardiness, which may occur due to jobs of the served SRT *tasks* running longer than their average case. This separation is essential to incorporate stochastic tasks into EDF-HSB. One potential concern, however, is the independence assumption—this is an issue we revisit later.

Schedulability analysis results. Let $c = m - \sum_{i=1}^{n^H} U_i$ be the processing capacity available to SRT servers. Suppose all m CPUs may be used to schedule the HRT tasks; let $a_{i,j} = 1$ if H_i is assigned to processor j, and 0 otherwise. The following constraints on the system parameters are required for the system to be provably schedulable:

1. $\sum_{i=1}^{n^H} a_{i,j} U_i \leq 1$ for all $j \in \{1, 2, \ldots, m\}$,
2. $\sum_{i=1}^{n^H} U_i + \sum_{i=1}^{n^S+m} u_i \leq m$,
3. $\max_{i=1}^{n^S+m} u_i < \frac{c}{2m-2}$, and
4. $\bar{e}_k < b_k$ for all $k \in \{1, 2, \ldots, v\}$.

The first two constraints are needed to ensure that individual CPUs and the entire system are not over-utilized, the third constraint is required for bounded tardiness of the SRT servers, and the fourth constraint is needed to ensure that individual SRT servers are not over-utilized.

Theorem 1 ([4]). *Under EDF-HSB, Constraint 1 implies that no HRT task misses its deadline.*

Theorem 2 ([15]). *Under EDF-HSB, Constraints 2 and 3 imply that any SRT server S_k misses its deadline by at most*

$$B_k(\tau^S, \textsf{EDF-HSB}) = b_k + \frac{B+2\sum_{j=1}^{m} y_j w_j + (m-c-1)\left(\max_{i=1}^{n^S+m} b_i\right)}{c-(m-1)\left(\max_{i=1}^{n^S+m} u_i\right)-U} \quad (1)$$

time units, where $y_j = 1 - \sum_{i=1}^{n^H} a_{i,j} U_j$, $w_j = \sum_{i=1}^{n^H} a_{i,j} e_i$, B is the sum of the $m-1$ largest values of b_i, and U is the sum of the $m-1$ largest values of u_i.

Theorem 3 ([18]). *Under any algorithm A that can schedule τ^S with tardiness bounds $B(\tau^S, A)$, Constraint 4 implies that periodic stochastic task T_k misses its deadline by no more than*

$$B_k(\tau^S, A) + \left(\frac{\sigma_k^2}{2b_k(b_k-\bar{e}_k)} + 2\right) p_k$$

time units on average.

Corollary 1. *Under EDF-HSB, Constraints 1–4 imply that no HRT task misses its deadline, no periodic stochastic task scheduled on a SRT server misses its deadline by more than*

$$B_k(\tau^S, \textsf{EDF-HSB}) + \left(\frac{\sigma_k^2}{2b_k(b_k-\bar{e}_k)} + 2\right) p_k \quad (2)$$

time units on average, and BE work may achieve a throughput of at least $\sum_{i=n^S+1}^{n^S+m} \frac{b_i}{p_i}$ per time unit in the long run.

Applying these results to the case-study system. We can further simplify the theoretical results described in Theorems 2–3 by exploiting the structure of the workload in our case study. Because we are considering a set of v videos, all of which have the same frame rate, we know that $n^H = n^S = v$, and $e_i = e^V$, $b_i = b^V$, and $P_i = p_i = p^V$ for all $i \in \{1, 2, \ldots, v\}$ (see Fig. 1). Hence, we know that the capacity available for SRT servers is $c = m - ve^V/p^V$. Furthermore, because we do not have any *a priori* knowledge of the type of BE work, we assume that $b_i = b^B$ and $p_i = p^B$ for all $i \in \{v+1, v+2, \ldots, v+m\}$. Because we are not concerned with providing maximum BE response time guarantees (our goal is to guarantee a minimum throughput), we choose budgets such that $b^V/p^V > b^B/p^B$. Finally, to ensure that the real-time workload is not trivially schedulable, we assume that $v \geq m$. These assumptions simplify $B_k(\tau^S, \textsf{EDF-HSB})$ to

$$b^V + \frac{(3m+ve^V/p^V-2)b^V + 2\sum_{j=1}^{m} y_j w_j}{m-ve^V/p^V-(2m-2)b^V/p^V}.$$

Note that because we may *choose* $\{b_1, b_2, \ldots, b_{v+m}\}$, Constraints 2–4 are satisfied if there exists *any* feasible set of SRT server budgets. The *largest-feasible-budget* heuristic attempts to pick budgets that are as large as possible while satisfying Constraints 2–4. This can be done by first choosing the desired BE budget b^B, then calculating the largest possible video budget satisfying both Constraints 2 and 3, and finally checking that Constraint 4 is satisfied. In other words,

$$b^V = \min\left\{\frac{mp^V-ve^V}{2m-2} - \epsilon, \frac{mp^V}{v}\left(1 - \frac{b^B}{p^B}\right) - e^V\right\}, \quad (3)$$

where $\epsilon > 0$ (because Constraint 3 is a strict inequality). We then check whether $b^V > \bar{e}^V$ (Constraint 4). If so, then we have a feasible set of budgets; if not, we must decrease the amount of allocation to BE work b^B and try again. This heuristic budgets as much capacity as possible for video decoding (under a given level of BE throughput); other heuristics may be desirable for different applications.

Example. Consider an application with four CPUs, five videos, each with a period of 40 ms (equivalent to 25 fps), and 25% of the system reserved for BE work. Suppose that each video frame requires 4 ms to display in the worst case and that decode times have a mean of 15 ms and a standard deviation of 5 ms. This example can be specified for EDF-HSB as the system $\tau = \{H_1, \ldots, H_5, S_1, \ldots, S_9\}$ with parameters $m = 4$, $v = 5$, $p^V = 40$, $e^V = 4$, and $\bar{e}^V = 15$. We first must statically assign HRT tasks to CPUs to satisfy Constraint 1. One way to do this is to bind H_1 and H_5 to CPU 1, H_2 to CPU 2, H_3 to CPU 3, and H_4 to CPU 4. Hence, a total utilization of 3.5 (consisting of 0.8 on CPU 1 and 0.9 on each other CPU) is available for SRT servers, which will be scheduled globally. We next specify parameters of the BE servers $\{S_6, \ldots, S_9\}$ to ensure that sufficient capacity is reserved for BE work. By setting $u_i = 0.25$ for $i \in \{6, \ldots, 9\}$, the capacity reserved for BE jobs will be 1.0, or 25% of the system.

Now we have a total utilization of 2.5 available to allocate to $\{S_1, \ldots, S_5\}$ by setting the decode budget b^V using the largest-possible-budget heuristic. According to (3), we set $b^V = \min\{23.33 - \epsilon, 20\} = 20$. We check that Constraint 4 is satisfied ($\bar{e}^V = 15 < 20$), and hence conclude that the choice of $b^V = 20$ is feasible. Because the second term in the min is smaller, the overall system utilization (Constraint 2) is the constraining factor, rather than the utilization cap required for bounded tardiness (Constraint 3). From Corollary 1, the theory states that no job of the display task will miss its deadline and predicts that video decoding jobs will be tardy no more than 508.8 ms, on average.

4 Implementation Details

Our scheduler, EDF-HSB, was implemented as a plugin within LITMUS$^{\mathrm{RT}}$. The design of EDF-HSB borrowed heavily from prior work on global real-time scheduler plugins for LITMUS$^{\mathrm{RT}}$ [5]. The video player was developed as a standalone application whose constituent tasks could be run as either real-time (*i.e.*, LITMUS$^{\mathrm{RT}}$) or regular Linux tasks. Our code is available on the project homepage [12].

Interrupt handling. The case-study application has many sources of interrupts, including disks and timers that fire due to asynchronous scheduling events, such as a server exhausting its budget. The default behavior in Linux is to distribute interrupts across all CPUs. Unfortunately, existing techniques for accounting for interrupts in schedulability analysis are rather pessimistic [6]. To avoid undue pessimism, and because it is easy to redirect certain interrupts away from the

scheduling CPUs, we used *dedicated interrupt handling* [21], wherein a single CPU is reserved for processing device- and timer-related interrupts. When this CPU needs to inform another CPU of a scheduling-related event, it does so by means of an inter-processor interrupt (IPI). Such interrupts are less disruptive from an analysis perspective because they coincide with scheduling-related activities, the overhead of which must be accounted for anyway. The interrupt-handling CPU is not used to execute real-time tasks or BE jobs, but could be used for other processing, such as system monitoring, instrumentation, and frame display.

Background processing. It is possible that a job is available to execute on an idle CPU but all unused servers have depleted budgets. To avoid wasting processing capacity, we allow such a job to execute "in the background" without consuming budget. This policy is equivalent to firm resource reservations in a resource kernel [20]. As with slack reclamation, background processing can only improve the response time of SRT jobs, and has no adverse effect on the analysis and provisioning performed earlier.

HRT task assignments. Under EDF-HSB, we chose to evenly distribute HRT tasks across CPUs. This was done to lessen display jitter (a more thorough discussion of jitter is given later). Since a HRT job can only be delayed by other HRT jobs on the same CPU, minimizing the maximum number of HRT tasks per CPU lessens such delays.

4.1 Provisioning

We now discuss how the theoretical results discussed earlier were applied to provision the studied system.

Computing WCETs and server budgets. We henceforth omit the i subscript in $\bar{e}_i$ and σ_i^2 because all SRT decode tasks are treated identically. In determining the SRT server budget b^V, Constraint 4 requires $\bar{e} < b^V$, so average-case execution times for the SRT decode tasks are required. As EDF-HSB requires budget enforcement, and budgets cannot be determined without task execution costs, we determined costs by running a video decoder workload under G-EDF. Specifically, we determined estimates of $\bar{e}$, σ^2, and the WCET e^V for the HRT display tasks in this way.

It would be unreasonable to expect the system to be reprovisioned each time a user runs a new video. Instead, the system should be provisioned once by considering a "typical" workload scenario. For this purpose, we used a "training set" of 11 videos selected randomly from the overall set of 58 videos that were used in this study (see Sec. 5). We chose 11 videos to ensure that each CPU is busy without overloading the system. For each video, we played a random 60 second interval. This resulted in over 15,800 samples for each job type. We calculated $\bar{e}$ and σ^2 across all decode jobs and computed e^V by maximizing across all display jobs. The corresponding obtained values are $\bar{e} = 14.49$ ms, $\sigma^2 = 5.19^2$, and $e^V = 4$ ms. Using the stated values for $\bar{e}$ and σ^2, b^V can be determined using (3). As noted earlier, (3) attempts to maximize b^V without violating Constraints 2 and 3; thus, the value of b^V is necessarily workload-dependent. Further details regarding the calculated b^V values are given below.

It is worth emphasizing that the expected tardiness bound given in Corollary 1 is based upon both a deterministic bound for the SRT servers and a term that depends on $\bar{e}$ and σ^2. The deterministic tardiness analysis is quite conservative and thus should compensate for any optimism in the above approach for computing average execution costs.

Overheads. EDF-HSB performance is affected by several sources of system overheads, including *scheduling overheads*, which are incurred during task selection, *context-switching overheads*, caused by migrations and preemptions, and *cache-related* overheads, which are incurred when a job resumes on a CPU and suffers from compulsory cache misses. If these overheads are not accounted for when a task system is analyzed, then the calculated tardiness bounds and server budgets may be invalid. Our provisioning step implicitly accounts for these overheads by ensuring that received IPIs are charged against the execution time of the currently-running job, and by ensuring that tasks have no particular CPU affinity (so that migration-related overheads are not under-estimated). The overheads are included in the estimates of $\bar{e}$ from the training set, which was discussed previously.

Queue sizes. The size of each decode-display FIFO queue depends on the decode task's tardiness, which by (2) depends on the characteristics of the overall workload. We reserved 10% of the overall system's capacity for BE work and then calculated SRT decode server budgets b^V for $v \in \{11, 16, 17, \ldots, 22\}$ using (3). When $v > 22$, average execution times exceed the maximum possible server budget and Constraint 4 is violated. b^V and σ^2 were then used to calculate the expected tardiness bound given by (2), which we denote as T. The required frame queue size is then

$$\max\left(1, \lceil T/p^V \rceil\right). \tag{4}$$

Using the average tardiness to determine the queue capacity was sufficient for the video application because the distribution of decode tardiness had a heavy right tail (i.e., very large values were very unlikely, and hence most values fell below the average). If an exact probability of exceeding the queue capacity is desired, Corollary 1 of [18] may be used to determine the size of the queue. Table 1 lists the resulting budgets, bounds, and queue sizes. These queue sizes are well in line with those used by video players in use today; the xine player, for example, uses an output queue of 15 frames by default.

An interesting problem arose when determining the values given in Table 1. As the budget increases and the expression in Constraint 3 approaches equality, the deterministic bound approaches infinity; on the other hand, as the budget decreases and the expression in Constraint 4 approaches equality, the stochastic part of the bound approaches infinity. This phenomenon is illustrated in Fig. 2. The heuristic presented by (3)

v	b^V	tardiness bound (2)	queue size (4)
11	19.94	838.57	21
16	18.94	910.81	22
17	18.74	925.04	23
18	18.74	1223.78	30
19	17.73	580.76	14
20	16.65	399.94	10
21	15.66	339.29	9
22	14.77	406.55	10

Table 1: Calculated budgets (in ms), tardiness bounds (in ms), and queue sizes (number of frames) for SRT decode tasks with 11 and 16–22 videos playing simultaneously.

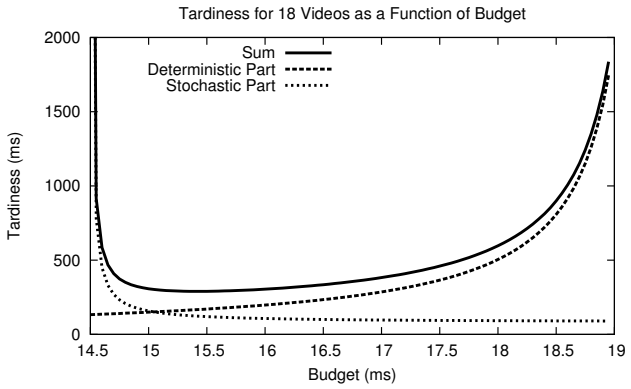


Figure 2: Deterministic part and stochastic parts of tardiness bound for $v = 18$ as a function of budget.

assumes that budgets should be as large as possible. However, in the unusual case where the two terms in the minimum of (3) are almost equal, the tardiness bound increases significantly. This phenomenon occurred at $v = 18$ in Table 1. When $v \leq 18$, the budget is determined by the first term in (3). As v increases to 18, b^V becomes close to the theoretical limit imposed by Constraint 3. We compensated by increasing ϵ, thereby reducing tardiness and frame queue sizes to a manageable level. In the case of $v = 18$, we increased ϵ until b^V decreased from 18.94 to 18.74 in order to obtain a smaller tardiness bound. In Fig. 2, this pushes the expected tardiness from the large values on the right side of the graph towards the smaller values in the middle. This was a surprising situation that could not have been predicted by either the deterministic or stochastic analysis alone.

While the above discussion of queue provisioning was directed at EDF-HSB, all of the tardiness analysis used in this provisioning is applicable if global FIFO scheduling is used instead of G-EDF in the EDF-HSB framework [15]. In the SCHED_FIFO implementation considered in Sec. 5, HRT tasks are prioritized over SRT tasks as in EDF-HSB. However, unlike EDF-HSB, it does not enforce budgets for HRT tasks or provide a minimum guarantee for BE work. Still, this implementation is quite close to being a true FIFO-variant of EDF-HSB, so we applied the same queue provisioning to it. As for the CFS variants considered in Sec. 5, analytical techniques for determining queue sizes are lacking, so we simply applied the same queue provisioning to them as well.

4.2 Theory vs. Practice

Every body of theory has its limitations, and there are several cases where the system we study clearly violates assumptions made in the theoretical results presented above.

HRT display tasks. Because we used *measured* rather than *analytically-derived* WCETs in provisioning the HRT display tasks, it is possible that such a task could execute for longer than its WCET. However, over the several million job executions that occurred in the evaluation presented in Sec. 5, this was rarely observed. Further, a frame queue may become empty when the tardiness of the corresponding decode task exceeds the expected bound, but the stochastic analysis described earlier ensures that the probability of this is low.

Independence of decode jobs. We analyzed the decode tasks as if successive decode jobs are independent. In reality, we observed that this assumption is violated. To ameliorate such violations, the stochastic tardiness work in [18] allows windows of successive jobs of the same task to be treated as a single entity in the analysis, at the expense of a higher expected tardiness bound (since the analyzed entities are larger). We observed that as the number of video frames per time window increases, the correlation of successive per-window execution times decreases, to the point where using a window of at least 100 video frames resulted in no significant correlation. However, in the case study, we chose not to combine jobs, because we found that background scheduling and slack-reclaiming were very effective in allowing tardy jobs to "catch up" before they negatively impacted system performance.

Self-suspensions. All of the analysis outlined in Sec. 3 is based on task models in which tasks do not self-suspend. However, in the studied system, such suspensions can occur when tasks block waiting on some shared resource such as a disk. In our experimental setup, this interference was negligible. This was partly due to the fact that we stored video files in an in-memory filesystem to avoid lengthy disk-related I/O suspensions. In a production system, this could be avoided by means of a solid state drive, flash memory, or a simple prefetching algorithm. Our video player supports multiple display backends. Each display job copies a frame to a (virtual) frame buffer. The target memory could be provided by X11, a physical frame buffer, or some other (application-specific) display system. The results presented in Sec. 5 were obtained by using a "null-device" backend, where each video frame was copied into a virtual frame buffer, but not physically rendered. This simulates the case where each video is rendered to a separate display device.

Implications for case study. The point of this study is *not* to check whether the theory presented in Sec. 3 is correct (this is not in question, as each result was already proven). Rather, the point of the study is to determine whether applications of the theory are *useful*. It is clear that the system does not exactly meet every assumption of the theory. However, it is unlikely that *any* real task system will meet every assumption while also being non-trivial to schedule. Hence, for the theory to be useful, it must be robust to small mismatches in assumptions. Our implementation of the case study aims to demonstrate this very point. We implemented the system as if the assumptions were met. We hypothesized that extra tardiness due to assumption violations would be small in magnitude compared to the pessimism known to exist in some of the theoretical results [18]. Moreover, by design, slack reclaiming and background scheduling are designed to mitigate much of the tardiness present in the system (some of which may be due to assumption violations).

5 Evaluation

We instrumented our video player to record relative jitter and deadline tardiness of display jobs in the same manner as Feather-Trace [3]. *Jitter* quantifies the smoothness of video playback by measuring deviation from the desired frame playback rate. If frame i is displayed at time t_i and t_0 is 0, then its *relative jitter* is $|t_i - (t_{i-1} + p^V)|$ and its *absolute jitter* is $|t_i - ip^V|$. A job that is tardy by more than p^V time units increases the absolute jitter of the subsequent job of the same task but not its relative jitter. From a viewer's perspective, a single pause in the video stream followed by correct playback is a single error, and should not affect jitter measurements of future video frames. Therefore, we report the relative jitter instead of absolute jitter. We do not report the number of frames displayed per second (FPS), because the FPS measurement conveys no measure of video "smoothness." For example, a video player that shows 23 frames in 0.01 seconds followed by a 24^{th} frame for 0.99 seconds still achieves 24 FPS in that second, but the video stream will appear choppy to the user. On the other hand, relative jitter captures the smoothness of playback. For the BE jobs, we report throughput (TPUT). We do not report jitter or tardiness of SRT decode tasks because excessive tardiness in the decode tasks manifests as deadline misses in the display tasks.

Our experiments evaluate the video playback quality and performance of the BE jobs under EDF-HSB, EDF-HSB without slack reclaiming and background scheduling (EDF-HSB-NS), Linux's CFS scheduler, Linux's CFS scheduler where BE work is run at a $nice$ level of 5, and Linux's SCHED_FIFO scheduler where three decreasing task priorities are assigned to the sets τ^H, $\{S_1, \ldots, S_{n^S}\}$, and $\{S_{n^S+1}, \ldots, S_{n^S+m}\}$ in that order. All video players used the same queue size under each scheduler for the appropriate value of v (see Table 1). BE jobs were modeled by BE generators. Each generator produced job arrivals according to a Poisson arrival process with rate $\lambda = 0.01$. BE jobs had exponentially distributed execution times with mean 10 ms. The execution times were limited to the interval $[2, 100]$ and interarrival times to a maximum of 200 ms. Therefore, the utilization of all BE jobs created by a single generator was approximately 0.1 before truncation. By varying the number of BE generators, we simulated different levels of background work. We reserved 10% utilization for BE work by allocating $m = 11$ global servers for scheduling BE tasks, each with budget 5 ms and period 50 ms.

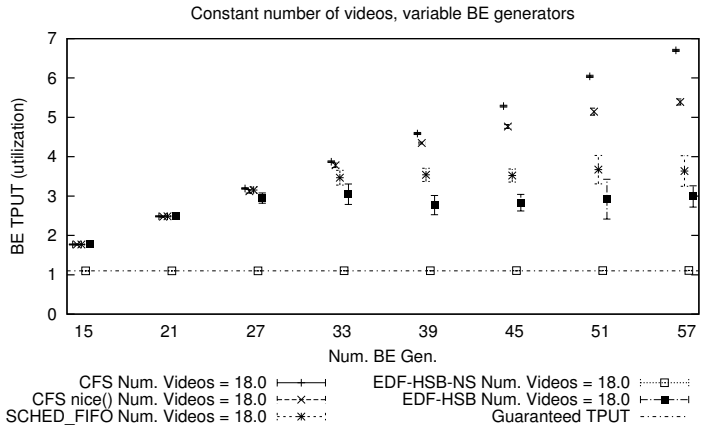
Using a pool of 58 MPEG4 movie trailers, we evaluated four experimental configurations. In the first two, the number of videos v was fixed at 11 and 18 and the number of BE generators was varied, while in the second two, the number of BE generators was fixed at 18 and 36 and v was varied. For each experimental replication, we played a randomly chosen 60 second interval of each video simultaneously. For the fixed-v experiments, each replication used a randomly selected subset of videos from the pool to determine how performance was affected as additional BE work entered the system. For experiments using a fixed number of BE generators, we drew a random subset of videos for each value of v to observe how performance was affected as additional SRT tasks were added to the system.

For each setup, the entire experiment, including the random selection of videos and playback intervals, was replicated six times. To account for variation due to random sampling, we calculated 95% confidence intervals for the mean of each metric. Non-overlapping 95% confidence intervals imply that the difference between schedulers is statistically significant. A total of 1,260 task set and scheduler combinations were evaluated, requiring more than 24 hours to execute all experiments. In total, over 1.8 million frames were decoded and more than 13 GB of benchmark data was collected for analysis.

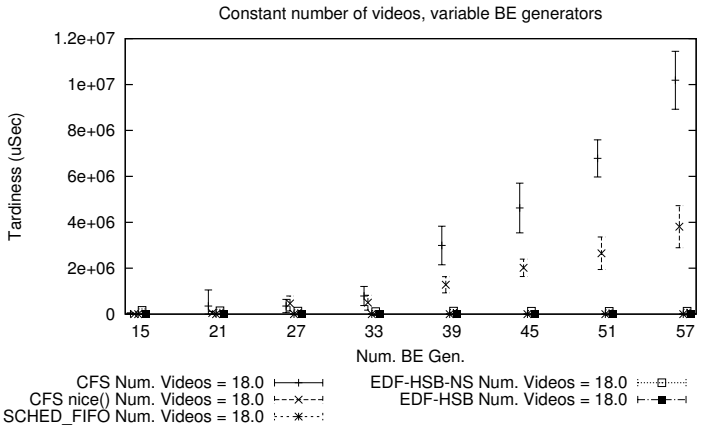
5.1 Results

Insets (a)–(d) and (f) in Fig. 3 depict jitter, tardiness, and throughput as a function of the number of video streams or BE generators. In these graphs, each point corresponds to an integral x-coordinate; however, we diffuse points over each x-axis coordinate in order to avoid clutter. Inset (e) shows tardiness as a function of time for one task set. In the following, we discuss the trends apparent in our data.

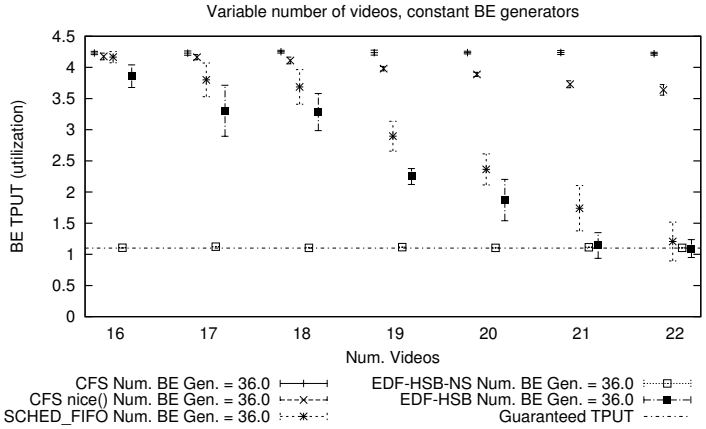
Video playback. Average-case provisioning, as compared to WCET provisioning, was useful in reducing the queue size without causing display task deadline misses before the system approached its full capacity. Because EDF-HSB isolates the BE jobs from the rest of the system, higher numbers of BE generators do not negatively impact the video tardiness in inset (b). CFS, on the other hand, did not keep BE work from competing with the video tasks, as evidenced by the high BE TPUT in inset (a). $nice$ mitigates this problem somewhat, but a higher $nice$ value does not preclude a BE job from executing



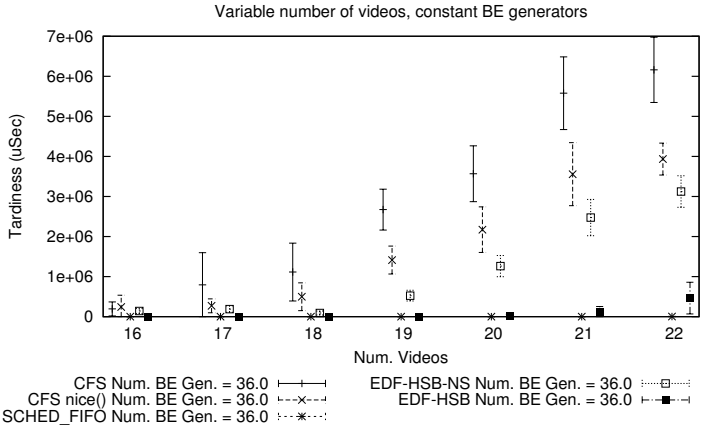
(a) BE TPUT, $v = 18$ with varying number of BE generators.



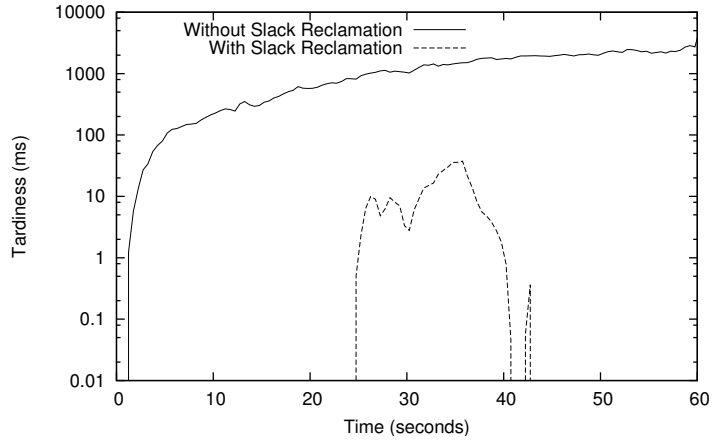
(b) Tardiness, $v = 18$ with varying number of BE generators.



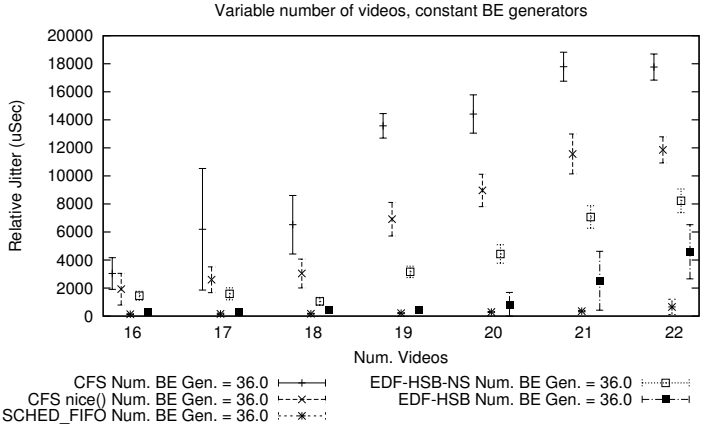
(c) BE TPUT, 36 BE generators with varying number of videos.



(d) Tardiness, 36 BE generators with varying number of videos.



(e) Tardiness over time, $v = 20$ with 36 BE generators.



(f) Relative Jitter, 36 BE generators with varying number of videos.

Figure 3: Recorded tardiness, jitter, and BE TPUT. **(a)** BE TPUT, **(b)** display job tardiness when playing 18 videos simultaneously with a variable number of BE generators. **(c)** BE TPUT, **(d)** display job tardiness, **(f)** relative jitter for a varying number of videos with 36 BE generators. **(e)** tardiness over time for $v = 20$ with 36 BE generators (note the log scale). Tardiness was averaged in 0.5 second intervals over the 20 videos in the replication.

in lieu of an eligible video task; it is merely a hint.

BE throughput. EDF-HSB allocates only processing capacity not reserved for video tasks to BE jobs, while CFS makes no guarantees about the capacity available to any particular type of task. As a result, BE TPUT was lower under EDF-HSB than under CFS. In inset (a), where $v = 18$, EDF-HSB reaches its maximum BE TPUT when the number of BE generators exceeds 27. Under CFS, BE TPUT continues to increase linearly as BE work is added to the system for the number of BE generators that we examined. When BE work is run using $nice$, the BE throughput is sub-linear for higher numbers of BE generators. Although CFS has higher BE TPUT than EDF-HSB in insets (a) and (c), it comes at the expense of video task performance in insets (b) and (d), respectively. Therefore, it can hardly be considered a benefit.

Slack reclaiming and background scheduling. We include a comparison of EDF-HSB with and without slack reclaiming and background scheduling to verify that these techniques effectively compensate for situations where successive job executions exceed the calculated average case, as well as increase BE TPUT where spare processor capacity exists. Without these techniques, BE TPUT is consistently 1.1 and tardiness increases linearly once the number of videos exceeds 18 in inset (d). In contrast, background scheduling and slack reclaiming allow EDF-HSB to increase its BE TPUT by allocating all processing capacity not devoted to video tasks to BE work, which can be seen in insets (a) and (c). Similarly, in insets (d) and (e), donating spare capacity to tardy decoding jobs decreases the tardiness accumulated in the corresponding display tasks by two orders of magnitude.

SCHED_FIFO. Our provisioning scheme was applied to both EDF-HSB and SCHED_FIFO and resulted in similar performance, with a few key differences. EDF-HSB has marginally lower BE TPUT than SCHED_FIFO in some cases, likely due to differences in implementation efficiency and overheads related to more frequent migrations (such as those caused by budget enforcement and slack reclaiming). Regarding implementation efficiency, SCHED_FIFO is a production scheduler whose implementation has been honed over many years. Another difference is that SCHED_FIFO did not enforce budgets. In contrast, EDF-HSB was able to guarantee a minimum BE TPUT, while BE TPUT under SCHED_FIFO dipped well below the 1.1 utilization mark in some instances (*e.g.*, as low as 0.56 in one case).

Over-utilization. The tardiness graphs in insets (b) and (d) show EDF-HSB as having zero tardiness for most data points. Average tardiness under EDF-HSB increased markedly in the heavy-utilization cases $v \in \{21, 22\}$ in (d). We attribute this to two factors. First, display tasks occasionally exceeded their provisioned budget for these large values of v, which results in over-utilization. Second, for $v = 22$, some sample video sets actually resulted in a system that was over-utilized based on the provisioned decode budget. EDF-HSB provides temporal isolation, *i.e.*, SRT tasks that did not exceed their budget were never tardy.

6 Conclusion

For work in the real-time systems community on analysis-based SRT multiprocessor schedulers to be taken seriously by OS developers, it is important to have evidence that such schedulers actually offer some practical benefits in comparison to heuristic-oriented alternatives. This paper has provided such evidence. Specifically, on heavily-loaded systems, analysis-based schedulers proved to be superior to Linux's default CFS scheduler in terms of jitter, deadline miss ratios, and tardiness. This study also demonstrated the value of stochastic tardiness analysis in reasoning about SRT systems, as an alternative to the common practice of simply applying HRT analysis using average-case execution costs and hoping that the system works. In particular, average-case stochastic analysis, together with slack reclaiming, avoids pessimistic worst-case provisioning of highly variable tasks such as video decoders without risking unbounded tardiness. SCHED_FIFO also proved to be a capable scheduler for the considered workload, but cannot provide guaranteed lower-bounds on BE TPUT. Indeed, because SCHED_FIFO has a corresponding real-time analysis, this bolsters our conclusion that analysis-based approaches have merit.

Several interesting avenues exist for further work. First, we have not attempted to identify the *best* analysis-based policy to implement. Continued experimental work is needed to resolve this, including comparisons to semi-partitioned approaches [13, 14], investigation of slack reclaiming heuristics, and investigation of trade-offs between BE TPUT guarantees and HRT display tardiness. Second, it would be of value to consider SRT workloads other than video. Third, our study dealt primarily with the allocation of CPU resources. Our system could be expanded to provision external resources such as video devices and disk drives. Finally, in the case-study system we considered, the workload is static. It would be interesting to consider other workloads in which applications are launched and terminated dynamically. Ample evidence already exists to suggest that the "bounded-tardiness" definition of SRT would be promising to consider in such settings [2].

References

[1] A. Bastoni, B. Brandenburg, and J. Anderson. An empirical comparison of global, partitioned, and clustered multiprocessor real-time schedulers. In *Proceedings of the 31st IEEE Real-Time Systems Symposium*, pages 14–24, 2010.

[2] A. Block. *Multiprocessor Adaptive Real-Time Systems*. PhD thesis, The University of North Carolina at Chapel Hill, 2008.

[3] B. Brandenburg and J. Anderson. Feather-trace: A light-weight event tracing toolkit. In *Proceedings of the Third International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pages 19–28, 2007.

[4] B. Brandenburg and J. Anderson. Integrating hard/soft real-time tasks and best-effort jobs on multiprocessors. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems*, pages 61–70, 2007.

[5] B. Brandenburg and J. Anderson. On the implementation of global real-time schedulers. In *Proceedings of the 30th IEEE Real-Time Systems Symposium*, pages 214–224, 2009.

[6] B. Brandenburg, H. Leontyev, and J. Anderson. An overview of interrupt accounting techniques for multiprocessor real-time systems. *Journal of Systems Architecture*, in press, 2010.

[7] S. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*, pages 396–407, 2003.

[8] M. Caccamo, G. Buttazzo, and L. Sha. Capacity sharing for overrun control. In *Proceedings of the 21st IEEE Real-Time Systems Symposium*, pages 295–304, 2000.

[9] J. Calandrino, H. Leontyev, A. Block, U. Devi, and J. Anderson. LITMUS$^{\mathrm{RT}}$: A testbed for empirically comparing real-time multiprocessor schedulers. In *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 111–123, 2006.

[10] U. Devi and J. Anderson. Tardiness bounds for global EDF scheduling on a multiprocessor. In *Proceedings of the 26th IEEE Real-Time Systems Symposium*, pages 330–341, 2005.

[11] J. Erickson, U. Devi, and S. Baruah. Improved tardiness bounds for global EDF. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems*, pages 14–23, 2010.

[12] UNC Real-Time Group. LITMUS$^{\mathrm{RT}}$ homepage. `http://www.cs.unc.edu/~anderson/litmus-rt`.

[13] S. Kato, Y. Ishikawa, and R. Rajkumar. CPU scheduling and memory management for interactive real-time applications. *Real-Time Systems*, 47(5):454 – 488, 2011.

[14] S. Kato, R. Rajkumar, and Y. Ishikawa. AIRS: Supporting interactive real-time applications on multicore platforms. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems*, pages 47–56, 2010.

[15] H. Leontyev and J. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. *Real-Time Systems*, 44(1):26–71, 2010.

[16] Xin Liu and Steve Goddard. Supporting dynamic QoS in Linux. In *Prodeedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 246–254, 2004.

[17] A. Mills and J. Anderson. A stochastic framework for multiprocessor soft real-time scheduling. In *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 311–320, 2010.

[18] A. Mills and J. Anderson. A multiprocessor server-based scheduler for soft real-time tasks with stochastic execution demand. In *Proceedings of the 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 207–217, 2011.

[19] L. Palopoli, T. Cucinotta, L. Marzario, and G. Lipari. AQuoSA—adaptive quality of service architecture. *Software: Practice and Experience*, 39(1):1–31, 2009.

[20] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. Resource kernels: A resource-centric approach to real-time and multimedia systems. In K. Jeffay and H. Zhang, editors, *Readings in multimedia computing and networking*, pages 476–490. Morgan Kaufmann, 2001.

[21] J. Stankovic and K. Ramamritham. The spring kernel: A new paradigm for real-time systems. *IEEE Software*, 8(3):62–72, 1991.