Adaptive CPU Scheduling Policies for Mixed Multimedia and Best-effort Workloads*

Melissa A. Rau
Unisys Corporation
NASA Langley Research Center
Hampton, VA
rau@dilbert.larc.nasa.gov

Evgenia Smirni
Department of Computer Science
College of William and Mary
Williamsburg, VA
esmirni@cs.wm.edu

Abstract

As multimedia applications with real-time constraints rapidly invade today's desktops, it becomes increasingly important for the operating system to provide robust resource allocation mechanisms for both multimedia and traditional best-effort workloads. We present a flexible CPU scheduling policy that adjusts the CPU proportion allocated to each application class using recent history as a feedback mechanism. The algorithm quickly adapts to varying workload conditions and compares favorably with static proportional scheduling schemes for mixed workloads.

1 Introduction

In recent years the average computer user has been experiencing dramatic changes in hardware speeds, affordable pricing of high-end computer equipment, and the ubiquity of a new type of real-time applications such as multimedia video and audio. Powered by special hardware that can effectively serve multimedia applications, a typical desktop serves workloads that differ dramatically from those that populated desktops just half a decade ago. Consequently, it is not only conventional computations that need be supported by the operating system but also multimedia applications with real-time deadlines.

The diversity of the workload that frequently populates today's desktops brings new challenges into the design of operating system schedulers. Schedulers need to provide guarantees for real time tasks by effectively meeting their deadlines so as to ensure minimization of jitter in video applications, provide lip-synchronization between audio and video, and at the same time not to starve conventional best-effort applications. Providing quality of service (QoS) guarantees in terms of different performance measures of interest in a non-predictable environment requires substantial changes in the resource allocation mechanisms of traditional operating systems.

Schedulers that deal with the performance issues outlined above can be classified as driven by deadlines or by proportionally sharing resources. Deadline driven schedulers such as earliest-deadline first (EDF) and rate monotonic [7, 6] are optimal under light load conditions but do not support well best-effort applications. The hierarchical CPU scheduler that was first proposed in [3] addresses this problem by statically partitioning the CPU bandwidth among various application classes. Different scheduling algorithms that are tailored for each specific application class are used to manage the allotted CPU bandwidth per class (e.g., EDF scheduling may be used to schedule video applications while plain time sharing may be used for best effort tasks). Lottery scheduling [12] achieves proportional partitioning of computational resources but does not provide any specific provisions for real-time jobs. Earliest Eligible Virtual Deadline First (EEVDF) focuses on real-time tasks and is also classified as a proportional share real-time algorithm. SMART [8] dynamically integrates a real-time scheduler and a conventional scheduler depending upon priorities and admission control. Resource reservations and a precomputed scheduling graph is used for scheduling real-time applications in the Rialto operating system [5]. BERT [2] effectively schedules multimedia and best-effort jobs but its implementation depends on a prediction mechanism that is tied to the Scout operating system.

The focus of this paper is on the evaluation of an adaptive CPU scheduler for mixed multimedia and best effort workloads that does not depend on a priori knowledge of the workload composition or intensity. To this end, we systematically explore the effect of *static* proportional resource sharing of a hierarchical CPU scheduler. We examine the performance of static sharing under a variety of workload

*This work was partially supported by a William & Mary Summer Research Grant.

intensities, transient load, and steady state conditions. We propose an *adaptive* scheduling scheme that uses recent history as a feedback mechanism. The proposed scheme is robust because it quickly detects changes in the workload requirements and accordingly adjusts computing resources among competing tasks. We evaluate the proposed algorithms with a discrete-event simulation. Our simulations are driven by traces of both multimedia and best effort applications that have been collected in real systems. We conclude that adaptive resource quantification is feasible and argue that the proposed scheduler can effectively handle a mix of multimedia and best-effort applications.

This paper is organized as follows. Section 2 summarizes the workload and simulation environment. The performance analysis of the static sharing policies is presented in Section 3. Section 4 outlines the adaptive scheduling algorithm and its behavior under transient and steady-state conditions. Section 5 summarizes our findings and concludes the paper.

2 Experimental Infrastructure

To evaluate the scheduling policies that we consider in this paper, we use discrete-event simulation designed with the next-event approach. Events are scheduled at specific times, while the clock advances asynchronously to the time of the next event. Event types in this system include arrivals of either multimedia or best-effort jobs, and job completions at the CPU. Service times for both multimedia and best-effort jobs are drawn from execution traces. The arrival processes for both job types are generated stochastically. All performance measures presented in this paper were obtained with a 95% confidence interval. In the following sections we describe the characteristics of the multimedia and best-effort applications, the workload mix used in the experiments, and the performance measures that the CPU policies try to optimize for each application type.

2.1 Multimedia Applications

The multimedia applications we consider consist of MPEG compressed video. The MPEG compression standard is based on the fact that within any given scene, there is one primary image, and subsequent pictures are only small variations of that image. There are three types of frames used in MPEG encoding. Intra-pictures, or I frames, represent the picture on which the scene is based and are self-contained images. Any variation within the scene is coded into predicted pictures (P frames), or bi-directional pictures (B frames). Videos are encoded using a specific pattern of these frames, and because of the variations in sizes, there is a trade-off between the number of (smaller) B frames, and the quality of the encoding.

The frame size is also affected by the level of activity within the video. Static videos like newscasts or talk shows have fewer major scene changes, and thus require smaller amounts of information in the P and B frames [9]. Action-oriented videos like sporting events that have a lot of movement within a scene require more encoded information in the form of P and B frames. Consequently, depending on the clip's action, there is significant variation in the frame-size sequence of MPEG-encoded video.

The simulations in this paper are driven by MPEG video traces that were obtained from the Scout group at the University of Arizona. A workload characterization study that describes these traces in detail is presented in [1]. We obtained several trace files of MPEG video executions. Each trace is encoded with a Group of Pictures (GOP) of size 8, with the frame pattern $IBBBPBBB$. The trace files include data for the frame type (I, P, B), number of macroblocks, size (in bytes), and number of CPU cycles to process each frame. Since the scheduler does not distinguish between frame types, we only consider the sizes of frames of videos and the required machine cycles for processing.

Figure 1 illustrates the proportion of frames within each video clip as a function of the frame processing times for two selected videos, Canyon and Terminator. Note the variation in frame processing times between the two videos. Canyon is a classic example of a video clip with small scene changes that implies small I, P, and B frames. Terminator is a classic example of an action video, thus its frame processing times are significantly higher. Canyon consists of 1757 frames, which, when processed at a rate of 30 frames per second, results approximately in one minute of video. The Terminator clip at the same frame rate has 4471 frames and 2.5 minutes of video. Table 1 shows the mean size and processing time for each frame type (I, P, B) for all videos.

	Frame Size (Processing Time)		
Video	I	P	B
Canyon	2325 (0.008)	1875 (0.007)	463 (0.003)
Terminator	11756 (0.041)	7425 (0.034)	3050 (0.023)

Table 1. Mean frame size (in bytes) and mean processing time (in seconds) for I, P, B frames

2.2 Best-effort Applications

The best-effort workload is also drawn from execution traces. The traces were obtained from the University of California at Berkeley and were initially used in a modeling study of lifetime distributions of UNIX processes [4]. The service time distribution of one representative trace is presented in Figure 2.

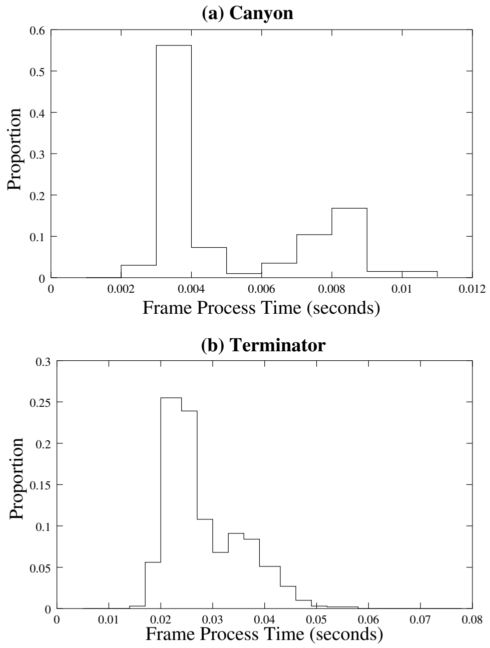

Figure 1. Frame processing time distributions for Canyon and Terminator.

This trace provides us with a multi-class workload within the best-effort class. While the bulk of the jobs are short, consisting of `ls`, `cd`, and other UNIX utilities, there is a small proportion of larger jobs such as compilations and program executions.

Since the scheduler needs to ensure that the best-effort jobs are not starved, it is important to consider the effect of job execution slowdown on user perception. We define slowdown as the ratio of the process wall clock time versus the process service time. A slowdown of 10 is going to be hardly perceived if the process service time is only a small fraction of a second. If instead the process service time is a few seconds, a slowdown of 10 will be certainly observed. In order to examine this multi-class workload in more detail, clustering analysis is performed. We used the k-Means clustering algorithm to classify jobs within a trace as "short", "medium", or "long". Table 2 illustrates the cluster centroids and the number of members for each cluster. Since the "long" class consist of a few jobs only, collecting statistics within good confidence intervals requires extremely long simulations. To shorten our simulations, we grouped the "medium" and "long" jobs into one class with 64 members and we labeled it "long".

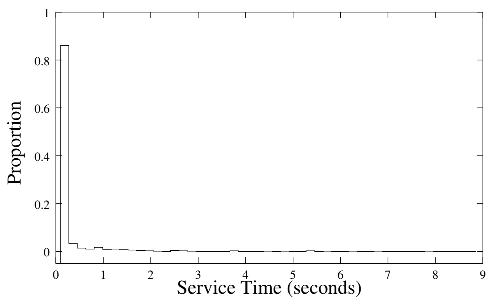

Figure 2. Service time distribution of the best-effort workload.

Best Effort Trace (Total Jobs: 776)		
	Centroid	Members
Short	0.039	712
Medium	1.322	53
Long	5.653	11

Table 2. Cluster centroids.

2.3 Workload Mix

The arrival process of the multimedia tasks is generated from an exponential distribution. In order to focus on the performance of the CPU scheduler, we assume that the frame buffer size is infinite and that the multimedia tasks arrive in time for processing. This allows us to examine the conditions under which multimedia deadlines are violated because of poor management of the CPU bandwidth, i.e., we do not consider cases where deadlines are missed because frames did not arrive in time from the network or I/O subsystem. Frames must be processed at a rate of at least 30 frames per second for the scheduler to deliver the required level of QoS to the multimedia class. In the experiments presented in this paper, we set the frame arrival rate to $\lambda_{mm} = 45$ frames per second, ensuring that frames always arrive in time for processing.

The diverse CPU demands of the Canyon and Terminator clips (see Section 2.1) allow us to examine the delivered performance under multimedia workloads with light and heavy CPU requirements. Indeed, it appears that the Canyon workload requires a mere 20% of the CPU bandwidth to meet its deadlines, while the Terminator needs almost 80% of the available CPU bandwidth for the required QoS to be delivered. We return to this point in Section 3 where the performance of the static scheduler is presented. We further combine the two videos to construct a mixed workload that imposes a transient load of multimedia jobs on the system. The mixed workload alternatively plays the Canyon and Terminator clips with a delay of 60 seconds interleaved between each video play.[1] In the remainder of the

[1] Space precludes presentation of the performance of mixed workloads with delays other than 60 seconds. It is important to note that the algorithms' behavior across a variety of mixed workloads remains qualitatively the same as the one presented here. We point the interested reader to [10]

paper, the three distinct multimedia workloads used in our simulation experiments will be referred to as Canyon (continuous play of the Canyon video), Terminator (continuous play of the Terminator clip), and Mix_60 (mixed workload with 60 seconds delay between consecutive video plays).

The arrival processes of the best-effort workload are drawn from a exponential distribution. We use both stationary and non-stationary arrival processes in order to examine the scheduler's ability to quickly adapt to different loads and to study the performance under bursty arrival conditions of the best-effort class. To model a realistic work environment where the system load varies from one extreme to another, we use a non-stationary $Poisson$ process with rate $\lambda_{be}(time)$ specified as the piecewise linear spline in Figure 3. This arrival process allows us to model the system behavior under bursty conditions and transient heavy conditions. Table 3 summarizes the workloads parameters used in the simulation experiments.

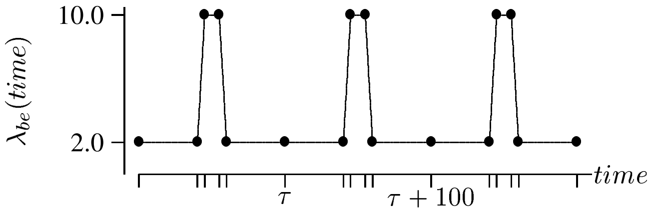

Figure 3. The non-stationary arrival process of best-effort jobs.

	MM Workload	BE Workload	λ_{be}
Exp. 1	Canyon	Stationary	$\in [0.1, 5.1]$
Exp. 2	Terminator	Stationary	$\in [0.1, 5.1]$
Exp. 3	Mix_60	Stationary	2.1
Exp. 4	Canyon	Non-Stationary	$= (2.0, 10.0)$
Exp. 5	Terminator	Non-Stationary	$= (2.0, 10.0)$
Exp. 6	Mix_60	Non-Stationary	$= (2.0, 10.0)$

Table 3. Workload parameters of the various experiments.

2.4 Performance Measures of Interest

To evaluate the delivered performance of the scheduler, we select metrics which best characterize performance for each of the application types.

- Best-effort tasks are evaluated based on their slowdown, i.e., the ratio of response time to actual service time, where response time is defined as the length of time the job spends in the system.
- Multimedia performance is evaluated by its QoS, which may be defined by several different factors. The multimedia throughput is the number of "good" frames processed per unit time, i.e., frames that finish in time to meet their deadline. Another QoS metric is the proportion of frames which miss their deadlines during execution. This is analogous to throughput, in that the two are inversely proportional to one another. Here, we choose to evaluate multimedia performance based on the proportion of missed deadlines.

In the next sections we present the performance of the static and the adaptive scheduler for the two classes of applications under the workload parameters depicted in Table 3.

3 Static Proportional Resource Sharing

3.1 Static SFQ Algorithm

The selected static scheduler that achieved proportional resource sharing is the hierarchical scheduler with Start-time Fair Queuing (SFQ) (see [3] for a detailed discussion of its performance bounds). SFQ is essentially a time-sharing algorithm that adjusts the length of the time slice (i.e., CPU bandwidth) allocated to each class of applications as a function of a predefined proportion. The bandwidth proportion that is allocated to each class is further managed by a scheduler that strives to optimize the performance metric of interest of the specific class. In our simulations EDF is used to schedule the multimedia jobs and plain time-sharing is used for best-effort tasks.

Fair allocation is achieved by factoring in the proportion given to each class and the length of the last quantum during which that class held the resource. SFQ is a work-conserving policy. If the multimedia class holds proportion p of the total weight, but has no jobs ready for execution, then the best-effort class receives the entire CPU bandwidth until more multimedia jobs arrive. In essence, the assigned CPU bandwidth is a lower bound of the effective CPU bandwidth used by the specific class. The CPU never sits idle unless there are no jobs of either class that are waiting for service.

SFQ is implemented as follows. Each class of jobs f, is assigned a start tag S_f, and a finish tag F_f. Start tags S_f are assigned according to (1), where t_f is the time stamp of the resource request. Jobs are scheduled according to the minimum start tag across all classes. In [3] ties among the start tags are broken arbitrarily. Here, we choose to break any ties in favor of the multimedia class. Once class f is allocated the resource, then F_f is updated according to (2), where l is the length of time that f held the resource, and w_f

for a detailed presentation of the effects of different delays between two video plays.

is the weight (proportion) of class f. The SFQ algorithm is shown in Figure 4.

$$S_f = \max\{t_f, F_f\} \quad (1)$$
$$F_f = S_f + \frac{l}{w_f} \quad (2)$$

```
for ever do
   processRequests();          // check classes for arrivals
   updateStartTags();          //update start tags with eqn (1)
                               //if necessary
   if ( no requests )
      updateVirtualTime();     // system is idle
   else
      f = min{S_be, S_mm}      //find class f with min start tag
      schedule(f);             //allocate CPU to class f
      updateFinishTag(f);      //update finish tag of class f with eqn (2)
```

Figure 4. Start-time Fair Queuing (SFQ) with static bandwidth partitions.

3.2 Performance of SFQ under Stationary Arrivals of Best-effort Jobs

To test the performance of the SFQ algorithm, we ran the experiments outlined in Table 3 with the following multimedia/best-effort proportion combinations: (MM,BE)= $\{(.2,.8),(.4,.6),(.6,.4),(.8,.2)\}$. Figure 5 illustrates the proportion of missed deadlines as a function of the arrival rate of the best-effort tasks under stationary best-effort arrivals and continuous playing of the Canyon and Terminator clips (i.e., experiments 1 and 2) for the multimedia/best-effort proportion combinations $\{(.4,.6),(.6,.4)\}$. See [10] for a detailed presentation of the performance of all experiments. In all cases, the Canyon workload performs flawlessly, even with a high-intensity best-effort workload. This result is a direct consequence of the static subject matter of the video and its small frames. Correspondingly, the more action-oriented Terminator achieves the required QoS level when either the best-effort workload is of very low-intensity (i.e., $\lambda_{be} < 1.6$), or when the multimedia class is statically assigned at least 80% of the CPU bandwidth. In fact, the different frame size and processing requirements of these two videos directly affect the actual proportion of the CPU obtained during execution, as well as the response time ratios (i.e., slowdowns) of the best-effort jobs.

Figure 6 presents the graphs of the actual CPU bandwidth consumed by each application class in experiments 1 and 2. As discussed in the previous subsection, if the queue in one class of jobs is empty, then all of the CPU bandwidth is allocated to the other class. Consequently, the statically allocated proportions are not fixed and constant, but rather provide a lower bound on the CPU bandwidth that a class receive when both classes require all of their given proportions. Figure 6 demonstrates this work-conserving aspect of the SFQ policy. When λ_{be} is low, the best-effort class is frequently idle, leaving the remaining bandwidth to be used by the multimedia class. As the arrival rate of best-effort jobs increases, we see that the actual CPU proportion gradually converges to its initially allocated proportion. The speed of the convergence to the allocation percentages depends on the video's CPU demand and the arrival rate of the competing jobs (see the behavior of Terminator in Figure 6).

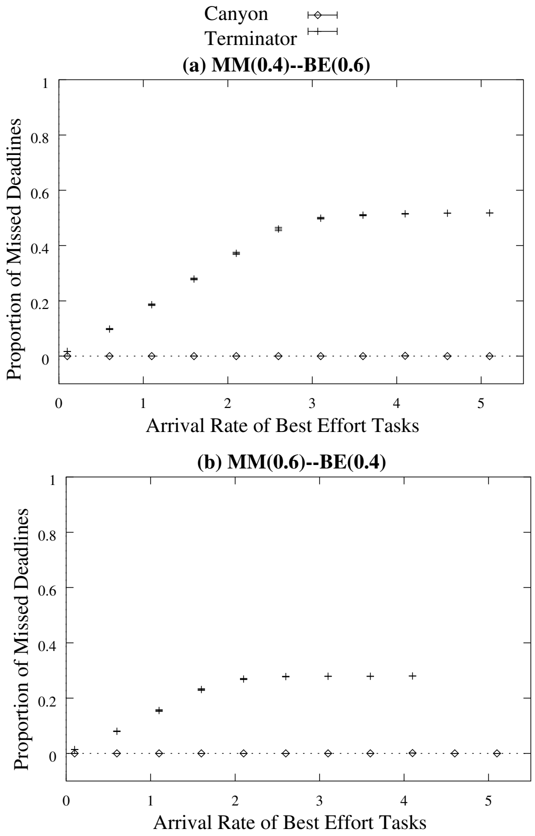

Figure 5. Proportion of missed deadlines for the multimedia workload as a function of the arrival rate of the best-effort tasks.

Figure 7 presents the slowdown experienced by the best-effort jobs in experiments 1 and 2. With a static allocation of 80% of the CPU for the best-effort class, the best-effort jobs perform equally well regardless of the intensity of the multimedia workload. However, as soon as the best-effort proportion drops to 60%, the effects of the Canyon and Terminator workloads become more clear. We see a substantial increase in slowdown with the higher-intensity Terminator workload for both classes of best-effort jobs. This trend is further magnified as the best-effort class is given smaller initial proportions. In fact, for experiment 2, in Figure 7(b) no values are plotted for $\lambda_{be} > 4.1$. The multi-class best-effort workload requires a minimum proportion of the CPU in order to keep system utilization below 1.0. As the initial proportion for best-effort decreases, $\lambda_{be} \geq \mu_{be}$, and the best-effort scheduler saturates.

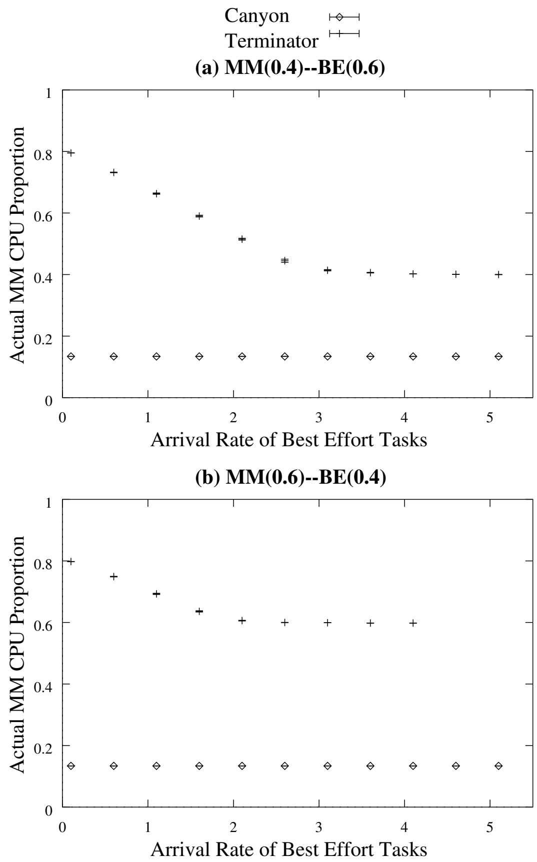

Figure 6. Proportion of the CPU bandwidth assigned to the multimedia workload as a function of the arrival rate of the best-effort tasks.

To test the performance of the policy with non-continuous playing of video but stationary best-effort arrivals we run simulations using the Mix_60 workload (experiment 3). We examine the policy performance when $\lambda_{be} = 2.1$, a moderate arrival rate of the best effort jobs. Table 4 illustrates the mean of the proportion of missed deadlines, actual CPU proportion used by the multimedia application, and the slowdown for both short and long jobs. For comparison, the data from experiments 1 and 2 are also illustrated on the table. As expected, the periodical breaks of

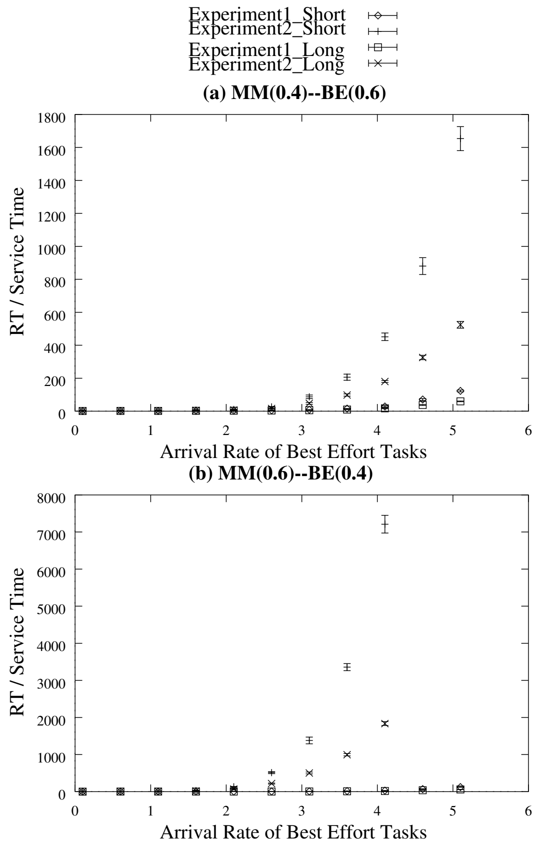

Figure 7. Slowdown of the best-effort jobs as a function of the intensity of their arrival rate.

60 seconds where no multimedia jobs are present allow the system to partially recover by quickly reducing the length of waiting queue for best-effort tasks. The performance of the best-effort jobs dramatically improves (especially in comparison to the more video-intense Terminator workload). Across all performance measures, the performance of the Mix_60 workload is consistently better than that of Canyon and Terminator.

3.3 Performance of SFQ under Non-stationary Arrivals of Best-effort Jobs

To further examine the performance of the SFQ policy under non-stable workload conditions, we increase the diversity of the workload by drawing the best-effort arrival times from a a non-stationary $Poisson$ arrival process that is specified as the piecewise linear spline of Figure 3. We effectively induce a total of ten peak "bursty" arrival periods throughout the duration of the simulation. Table 4 illustrates the performance measures for experiments 4 to 6. The non-stationary process has no effect on the missed deadlines for Canyon. The slowdown of both short and long jobs becomes higher than the one shown in Table 4 and this is a

experiment	Assigned Proportions (MM/BE)	MM Workload	Proportion of Missed Deadlines	Actual MM CPU Proportion	Slowdown Short Jobs	Slowdown Long Jobs
1	$0.2/0.8$	Canyon	0.001 ± 0.000	0.134 ± 0.000	3.131 ± 0.058	2.618 ± 0.067
	$0.4/0.6$	Canyon	0.000 ± 0.000	0.134 ± 0.000	3.126 ± 0.076	2.554 ± 0.083
	$0.6/0.4$	Canyon	0.000 ± 0.000	0.134 ± 0.000	3.138 ± 0.060	2.557 ± 0.070
	$0.8/0.2$	Canyon	0.000 ± 0.000	0.134 ± 0.000	3.188 ± 0.078	2.605 ± 0.080
2	$0.2/0.8$	Terminator	0.366 ± 0.004	0.519 ± 0.003	4.488 ± 0.086	3.072 ± 0.091
	$0.4/0.6$	Terminator	0.339 ± 0.003	0.542 ± 0.002	9.213 ± 0.309	6.451 ± 0.212
	$0.6/0.4$	Terminator	0.261 ± 0.002	0.615 ± 0.001	63.856 ± 2.895	35.697 ± 1.773
	$0.8/0.2$	Terminator	0.048 ± 0.000	0.784 ± 0.000	529.345 ± 13.417	136.503 ± 7.237
3	$0.2/0.8$	mix-60	0.200 ± 0.010	0.287 ± 0.005	3.360 ± 0.088	2.464 ± 0.119
	$0.4/0.6$	mix-60	0.184 ± 0.012	0.295 ± 0.006	4.588 ± 0.232	3.481 ± 0.233
	$0.6/0.4$	mix-60	0.148 ± 0.005	0.317 ± 0.002	10.143 ± 0.714	6.336 ± 0.415
	$0.8/0.2$	mix-60	0.032 ± 0.001	0.377 ± 0.000	46.682 ± 5.060	14.705 ± 1.264
4	$0.2/0.8$	Canyon	0.001 ± 0.000	0.134 ± 0.000	16.070 ± 0.594	8.263 ± 0.279
	$0.4/0.6$	Canyon	0.000 ± 0.000	0.134 ± 0.000	16.147 ± 0.596	8.267 ± 0.279
	$0.6/0.4$	Canyon	0.000 ± 0.000	0.134 ± 0.000	16.154 ± 0.596	8.267 ± 0.279
	$0.8/0.2$	Canyon	0.000 ± 0.000	0.134 ± 0.000	16.159 ± 0.596	8.267 ± 0.279
5	$0.2/0.8$	Terminator	0.632 ± 0.006	0.306 ± 0.005	20.811 ± 0.785	10.499 ± 0.414
	$0.4/0.6$	Terminator	0.496 ± 0.001	0.416 ± 0.001	94.923 ± 3.052	47.716 ± 1.318
	$0.6/0.4$	Terminator	0.277 ± 0.000	0.603 ± 0.000	401.352 ± 6.450	135.066 ± 4.898
	$0.8/0.2$	Terminator	0.048 ± 0.000	0.784 ± 0.000	1552.623 ± 17.720	N/A
6	$0.2/0.8$	mix-60	0.449 ± 0.014	0.158 ± 0.007	16.212 ± 0.988	8.138 ± 0.554
	$0.4/0.6$	mix-60	0.334 ± 0.008	0.218 ± 0.004	28.735 ± 1.849	13.922 ± 1.078
	$0.6/0.4$	mix-60	0.194 ± 0.001	0.295 ± 0.000	75.515 ± 5.570	30.452 ± 2.179
	$0.8/0.2$	mix-60	0.034 ± 0.001	0.377 ± 0.000	239.733 ± 7.929	47.187 ± 2.730

Table 4. Performance of SFQ with stationary and non-stationary best-effort arrivals.

result of the increase in the arrival intensity of best-effort jobs. For Terminator (experiment 5), the performance for both classes degrades severely in comparison to experiment 2. Finally, for Mix_60 (experiment 6), the periodic breaks of 60 seconds allow performance to improve with respect to experiment 5.

In addition to looking at means over the entire simulation, we take a closer look at the effects of the bursty arrival process by plotting the proportion of missed deadlines and the slowdown of the best-effort jobs in the successive time intervals between each knot pair of Figure 3. Figure 8 illustrates the measures of interest for the two job classes as a function of simulated time for experiment 6. It is easy to observe the moments when the best-effort arrival peaks occur. Furthermore, the oscillating behavior clearly demonstrates how much a bursty arrival process can affect system performance. We observe that for the case where the multimedia class misses the fewest deadlines is the same case in which the best-effort response time ratio is worst. Similarly, when the multimedia performance is worst, the slowdown ratio reduces. Overall, the increased variation within the best-effort workload contributes to substantial system-wide degradation of all performance metrics. Although the SFQ algorithm has the capability to adjust proportions when one class is empty, it is not able to make effective adjustments based on the current workload. In this case, the loss in performance is a result of the statically specified proportions, which are hard to optimize for dynamically changing workloads.

Based on results from the experiments presented in this section, we conclude that in order for the static proportional resource allocation algorithm to be effective the multimedia workload and the intensity of the best-effort arrivals must be known *a priori*. The problem is further exacerbated with bursty workload arrivals. The solution to this problem is to develop an algorithm which can dynamically adapt to a changing or diverse workload based on knowledge of limited past performance history.

4 Adaptive Proportional Resource Sharing

4.1 Adaptive SFQ (A-SFQ) Algorithm

In order to dynamically change the proportions of CPU bandwidth allocated to each class, we opt to continuously monitor the system's performance history. We keep one performance index per application class. This performance index is updated after each job's completion throughout the simulation. Here, we present an adaptive algorithm that uses the per class performance indices to adjust the CPU proportion allocated to each class as a function of the current system state.

The main idea of the adaptive algorithm is the following. At each time interval the current value of the performance metrics is computed. SFQ calls the function `Adjust()` before calling the function `processRequests()` (see

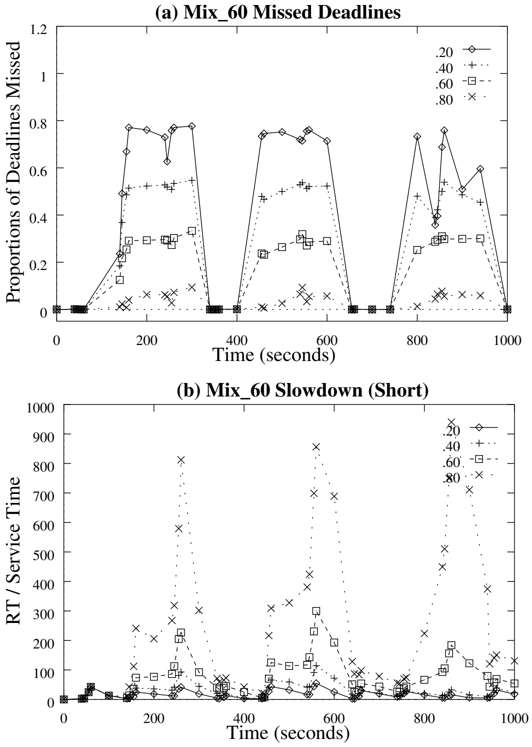

Figure 8. Performance of the SFQ scheduler as a function of simulation time for non-stationary best effort arrivals (experiment 6).

Figure 4). `Adjust()` reallocates the CPU bandwidth based on a comparison of the performance indices with threshold values that are pre-specified by the user. For the two application classes that we consider in this study, the user needs to specify two threshold values Th_{mm} and Th_{be} for the multimedia and best-effort classes respectively. Th_{mm} represents the proportion of missed deadlines that the user is willing to tolerate. Th_{be} represents the maximum slowdown that is acceptable for best-effort tasks.

The algorithm re-adjusts the allocated CPU proportions in multiples of "chunks" of 5% of the available bandwidth.[2] If Th_{mm} is smaller than the current percentage of missed deadlines, then the number of extra "chunks" required to meet the desirable performance level is defined as the ratio of percentage of missed deadlines over Th_{mm}. Assume that the proportion of the missed deadlines is 0.10 and that Th_{mm} is set to 0.02. Then, 5 additional bandwidth "chunks" are given to multimedia jobs (i.e., the multimedia proportion is increased by 0.25 of the total available bandwidth and the best-effort proportion is decreased by the same factor). Similarly, if the Th_{be} is smaller than the number of missed deadlines, then extra bandwidth "chunks" are allotted to the best-effort class. If both classes are exceeding their threshold values in the current time interval, i.e., the system operates under very heavy load and cannot meet the required levels of QoS for either class, we opt to give priority (and the extra proportion) to the multimedia class because of its soft real-time requirements. In general, we tend to quickly allocate bandwidth to a class if we observe that it suffers from the current allocation. If however a class is allocated higher bandwidth than its predefined one and does not suffer from performance loss, then the extra "chunks" are gradually given to the other class in an attempt to restore the original proportions given to each class. The readjustment occurs in single 5% "chunks". This is to ensure that once good performance is obtained, it is not immediately lost again because of some workload fluctuation.

[2] A small "chunk" allows for fine grain allocation of the available bandwidth. The user may change the "chunk" size so as to adapt more quickly or slowly to workload changes.

The algorithm for the `Adjust()` function is described in Figure 9. p_{mm} represents the proportion of the CPU bandwidth allocated to the multimedia class and p_{be} represents the proportion of the CPU bandwidth allocated to the best-effort class. The algorithm can be trivially extended to accommodate a larger number of job classes.

`if(` $missed\_deadlines > Th_{mm}$ `)` // MM performance is bad
  $p_{mm} += 0.05 \cdot \left\lfloor \frac{missed\_deadlines}{Th_{mm}} \right\rfloor$
  $p_{be} = 1.0 - p_{mm}$
  $extra\_chunks += \left\lfloor \frac{missed\_deadlines}{Th_{mm}} \right\rfloor$
`else if(` $slowdown > Th_{be}$ `)` // BE performance is bad
  $p_{be} += 0.05 \cdot \left\lfloor \frac{slowdown}{Th_{be}} \right\rfloor$
  $p_{mm} = 1.0 - p_{be}$
  $extra\_chunks -= \left\lfloor \frac{slowdown}{Th_{be}} \right\rfloor$
`else` // both perform below threshold
  `if(` $extra\_chunks > 0$ `)` // MM has extra proportion
    $p_{mm} -= 0.05$
    $p_{be} = 1.0 - p_{mm}$
    $extra\_chunks -= 1$
  `else if(` $extra\_chunks < 0$ `)` // BE has extra proportion
    $p_{be} -= 0.05$
    $p_{mm} = 1.0 - p_{be}$
    $extra\_chunks += 1$

Figure 9. `Adjust()`**: recomputes the allocated bandwidth to each application class.**

To examine the performance of the adaptive algorithm, we executed experiments 1 through 6. For all experiments, the "chunk" size was set to 5% of the total CPU bandwidth, Th_{be} was set to 100, and Th_{mm} was set to 0.05. The results of the experiments are outlined in the following sections.

4.2 Performance of A-SFQ under Stationary arrivals of Best-effort Jobs

Table 5 presents the performance measures for experiments 1 to 3. Recall that in these experiments the interarrival times of the best-effort jobs are exponentially distributed. A-SFQ behaves almost identically to SFQ with the Canyon workload (see Table 4). The multimedia demand is so low that the work-conserving behavior of SFQ is enough for the system to balance the CPU proportions among the two application classes. With the high demand Terminator video, A-SFQ improves the performance of the multimedia class but it worsens the performance of the best-effort class. This is a direct consequence of the fact that A-SFQ favors the multimedia workload in high load situations. In Mix_60, the A-SFQ proves superior to SFQ. Both missed deadlines and slowdown remain consistently below the thresholds set by the user. Note that across all workloads and regardless of the assigned MM/BE proportions, the A-SFQ algorithm achieves to "correct" the initial bandwidth partitioning and balance the available bandwidth between multimedia and best-effort jobs.

4.3 Performance under Non-stationary Arrivals of Best-effort Jobs

A good balance becomes difficult to reach when we change the best-effort workload by inducing a non-stationary arrival process. This increases the diversity of the workload, and thus effective scheduling of the jobs is more challenging. A-SFQ manages to reach acceptable levels of QoS for the multimedia class (compare Tables 4 and 5). The slowdown of the best-effort class increases, but this is a direct outcome of the higher arrival intensity of the best-effort jobs. In general, with bursty best-effort arrivals, the best-effort class requires proportions which vary greatly over time. This fluctuation between high and low workload intensity requires that the proportions of the classes be continuously adjusted. To reach a level of balance with this type of workload, neither class of jobs starves, but neither can have peak performance either. This is clearly demonstrated in Figure 10. The adaptive algorithm has improved the multimedia performance when its initial allocated proportion is too small, and slightly degraded the multimedia performance when the allocated proportion is too large for the best-effort class to perform well. In contrast to the behavior observed in Figure 8, A-SFQ manages to quickly adapt to the workload demands and is therefore insensitive to the initial proportion allocation.

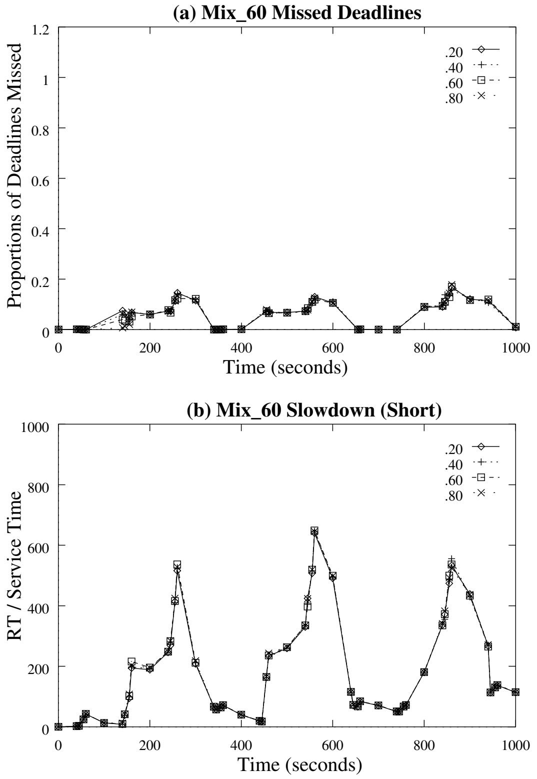

Figure 10. Performance of the A-SFQ scheduler as a function of simulation time for non-stationary best effort arrivals (experiment 6).

5 Conclusions

We examined static Start-Time Fair Queuing (SFQ), a hierarchical proportional algorithm for scheduling the CPU among applications with different performance requirements. With SFQ, the user is required to statically partition CPU bandwidth assigned to each class. Different scheduling algorithms that are tailored for each specific application class manage the allocated CPU bandwidth per class.

We investigated the delivered performance of the SFQ algorithm under a variety of workload service demands and bursty arrival conditions. Our conclusion is that determining the ideal bandwidth proportion to be allocated to each application class is a challenging problem. This is further exacerbated by possible variations in the workload arrival and service processes.

To deal with this problem, we propose an extension on the SFQ method that we call adaptive Start-Time Fair Queuing (A-SFQ). The A-SFQ algorithm continuously monitors the performance of the application classes and quickly adjusts the assigned proportions per class so as to ensure that the QoS levels set by the user are met for each class. Even in workloads that exhibit significant variability, A-SFQ quickly adjusts the allocated proportions in order for the required levels of QoS to be met. A-SFQ is shown to be a practical and effective for scheduling mixed multimedia and best-effort workloads.

References

[1] A. Bavier, B. Montz, and L.L. Peterson, "Predicting MPEG Execution Times", in *Proceedings of SIGMET-*

Experiment	Assigned Proportions (MM/BE)	MM Workload	Proportion of Missed Deadlines	Actual MM CPU Proportion	Slowdown Short Jobs	Slowdown Long Jobs
1	$0.2/0.8$	Canyon	0.000 ± 0.000	0.134 ± 0.000	3.133 ± 0.055	2.602 ± 0.062
	$0.4/0.6$	Canyon	0.000 ± 0.000	0.134 ± 0.000	3.180 ± 0.068	2.612 ± 0.066
	$0.6/0.4$	Canyon	0.000 ± 0.000	0.134 ± 0.000	3.145 ± 0.064	2.554 ± 0.066
	$0.8/0.2$	Canyon	0.000 ± 0.000	0.134 ± 0.000	3.170 ± 0.051	2.582 ± 0.070
2	$0.2/0.8$	Terminator	0.109 ± 0.001	0.730 ± 0.001	316.133 ± 4.783	144.297 ± 4.471
	$0.4/0.6$	Terminator	0.108 ± 0.002	0.731 ± 0.002	316.539 ± 6.327	148.994 ± 3.976
	$0.6/0.4$	Terminator	0.108 ± 0.001	0.731 ± 0.001	317.709 ± 6.139	151.356 ± 4.045
	$0.8/0.2$	Terminator	0.106 ± 0.002	0.733 ± 0.001	317.597 ± 5.262	151.862 ± 4.285
3	$0.2/0.8$	mix-60	0.028 ± 0.001	0.377 ± 0.000	47.660 ± 3.168	14.419 ± 0.586
	$0.4/0.6$	mix-60	0.027 ± 0.002	0.378 ± 0.001	47.040 ± 3.049	14.569 ± 0.821
	$0.6/0.4$	mix-60	0.026 ± 0.002	0.378 ± 0.001	49.458 ± 3.704	14.917 ± 0.776
	$0.8/0.2$	mix-60	0.024 ± 0.002	0.379 ± 0.001	50.201 ± 3.239	15.183 ± 0.730
4	$0.2/0.8$	Canyon	0.000 ± 0.000	0.134 ± 0.000	16.112 ± 0.594	8.265 ± 0.279
	$0.4/0.6$	Canyon	0.000 ± 0.000	0.134 ± 0.000	16.149 ± 0.595	8.267 ± 0.279
	$0.6/0.4$	Canyon	0.000 ± 0.000	0.134 ± 0.000	16.155 ± 0.596	8.267 ± 0.279
	$0.8/0.2$	Canyon	0.000 ± 0.000	0.134 ± 0.000	16.159 ± 0.596	8.267 ± 0.279
5	$0.2/0.8$	Terminator	0.188 ± 0.002	0.666 ± 0.001	636.176 ± 5.483	200.437 ± 4.897
	$0.4/0.6$	Terminator	0.187 ± 0.002	0.667 ± 0.001	637.713 ± 5.373	200.016 ± 4.361
	$0.6/0.4$	Terminator	0.185 ± 0.002	0.668 ± 0.002	640.947 ± 7.043	202.352 ± 6.347
	$0.8/0.2$	Terminator	0.184 ± 0.002	0.668 ± 0.002	640.360 ± 5.291	203.485 ± 4.873
6	$0.2/0.8$	mix-60	0.062 ± 0.003	0.361 ± 0.001	180.929 ± 7.065	46.854 ± 3.031
	$0.4/0.6$	mix-60	0.062 ± 0.002	0.361 ± 0.001	182.154 ± 7.505	47.179 ± 2.968
	$0.6/0.4$	mix-60	0.061 ± 0.003	0.362 ± 0.001	183.240 ± 7.186	47.433 ± 2.922
	$0.8/0.2$	mix-60	0.060 ± 0.003	0.362 ± 0.001	184.269 ± 7.079	47.472 ± 2.826

Table 5. Performance of A-SFQ under stationary and non-stationary best-effort arrivals.

RICS/PERFORMANCE'98, Madison, WI, pp. 131-140, June 1998.

[2] A. Bavier, L.L. Peterson, and D. Moseberger, “BERT: A Scheduler for Best Effort and Realtime Tasks”, Technical Report, Department of Computer Science, Princeton University.

[3] P. Goyal, X. Guo, and H.M. Vin, “A Hierarchical CPU Scheduler for Multimedia Operating Systems”, in *Proceedings of the Second Symposium on Operating Systems Design and Implementation (OSDI '96)*, Seattle, WA. pp. 107-122, Oct. 1996.

[4] M. Harchol-Balter and A.B. Downey, “Exploiting Process Lifetime Distributions for Dynamic Load Balancing”, in *Proceedings of SIGMETRICS'96*, Philadelphia, PA, pp. May 1996.

[5] M.B. Jones, D. Rosu, M.-C. Rosu, “CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities”, in *Proceedings of the Sixteenth Symposium on Operating System Principles*, St. Malo, France, pp. 198-211, Oct. 1997.

[6] J. Lehoczky, L. Sha, Y. Ding, “The Rate Monotonic Scheduling Algorithm: Exact Characteristics and Average Case Behavior”, in *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 166-171, Dec. 1989.

[7] C.L. Liu and J.W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment”, *JACM*, 20(1), pp.46-61, Jan. 1973.

[8] J. Nieh and M. Lam, “The Design, Implementation, and Evaluation of SMART: A Scheduler for Multimedia Applications”, in *Proceedings of the SixtheenthSymposium on Operating System Principles*, St. Malo, France, pp. 184-197, Oct. 1997.

[9] P. Manzoni and G. Serazzi, “Workload Models of VBR Video Traffic and their Use in Resource Allocation Policies”, Technical Report, Polytecnico di Milano, Italy, 1997.

[10] Melissa A. Rau, “Adaptive CPU Sxcheduling Policies for Mixed Multimedia and Best-effort Workloads”, *710 Project Report*, Department of Computer Science, College of William and Mary, Williamsburg, VA, May 1999.

[11] I. Stoica, H. Abdel-Wahab, and K. Jeffay, “A Proportional Share Resource Allocation Algorithm for Real Time, Time-Shared Systems”, in *Proceedings of Real Real Time Systems Symposium*, December 1996.

[12] C.A. Waldspurger and W.E. Weihl, “Lottery Scheduling: Flexible Proportional-Share Resource Management”, in *Proceedings of the First Symposium on Operating Systems Design and Implementation (OSDI '94)*, Monterey, CA, pp. 1-11, Nov. 1994.