NORTHWESTERN
UNIVERSITY

Electrical Engineering and Computer Science Department

Technical Report
NWU-EECS-06-07
July 30, 2006

Putting the User in Direct Control of CPU Scheduling

Bin Lin **Peter A. Dinda**

Abstract

CPU scheduling to enforce service-level agreements (SLAs) is a problem of key importance in service-oriented systems. For services whose ultimate customers are naive end-users, it is often a significant challenge simply to determine the terms of the SLA. We propose a new approach to both SLA specification and scheduling for enforcement that is unique in that it is based around the use of *direct user input*. Our implementation of the idea is designed for virtual machine (VM)-based computing environments. In our system, a user's VM is scheduled as a periodic real-time task. The user can instantaneously manipulate his VM's schedule using a joystick. An on-screen display illustrates the current schedule's cost and indicates when the user's desired schedule is impossible due to the schedules of other VMs or resource constraints. An extensive user study of the system indicates that even a naive user is capable of using the interface to our system to find a schedule that balances cost and the comfort of his VM. Good schedules are user- and application-dependent to a large extent, illustrating the benefits of user involvement.

Effort sponsored by the National Science Foundation under Grants ANI-0093221, ANI-0301108, and EIA-0224449. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).

Keywords: Adaptive systems, CPU scheduling, human-computer interaction

Putting the User in Direct Control of CPU Scheduling

Bin Lin *Peter A. Dinda*
{*binlin,pdinda*}*@cs.northwestern.edu*
Department of Electrical Engineering and Computer Science, Northwestern University

Abstract

CPU scheduling to enforce service-level agreements (SLAs) is a problem of key importance in service-oriented systems. For services whose ultimate customers are naive end-users, it is often a significant challenge simply to determine the terms of the SLA. We propose a new approach to both SLA specification and scheduling for enforcement that is unique in that it is based around the use of *direct user input*. Our implementation of the idea is designed for virtual machine (VM)-based computing environments. In our system, a user's VM is scheduled as a periodic real-time task. The user can instantaneously manipulate his VM's schedule using a joystick. An on-screen display illustrates the current schedule's cost and indicates when the user's desired schedule is impossible due to the schedules of other VMs or resource constraints. An extensive user study of the system indicates that even a naive user is capable of using the interface to our system to find a schedule that balances cost and the comfort of his VM. Good schedules are user- and application-dependent to a large extent, illustrating the benefits of user involvement.

1 Introduction

Service-oriented systems must schedule resources such as the CPU so that the relevant service-level agreements (SLAs) are honored. Increasingly, service-oriented computing is targeting end-users by providing services that are more and more akin to those available on typical interactive desktop computers. With the push toward the desktop, we are faced with increasingly naive users and shorter duration tasks. An important challenge emerges: what is an appropriate SLA in a desktop replacement environment and how do we get it?

We are investigating a general approach to this problem that combines resource scheduling techniques that expose direct control inputs, a cost model, and appropriate user interfaces to match the control inputs and the cost model to naive users. We bring such *direct user input* into the scheduling process because it has been demonstrated that tolerance for different schedules in desktop environments is highly application- and user-dependent. Simply put, desktop users are extremely diverse in their demands.

In this paper, we focus specifically on scheduling virtual machines (VMs) that support interactive desktop users on the CPU of a provider machine. We propose, implement, and evaluate a unique new approach to CPU scheduling that is based on direct user input, even from users who know nothing about scheduling and don't want to learn. As far as we are aware, there is currently no scheduling approach that incorporates *explicit* input supplied *directly* by even naive users.

Our work takes place in the context of Virtuoso, a system for utility computing that is based on VMs interconnected with overlay networks [42, 38, 41, 21, 15, 43]. Each provider computer in the system can support multiple VMs, each of which can run a different operating system. A typical interactive user uses a VM loaded with an operating system such as Microsoft Windows and communicates with it using a thin client [37, 20] running a remote display system [35, 36]. Each user will see a slowed machine due to resources committed to other users.

All the VMs on a provider computer are scheduled as periodic real-time tasks. We summarize the design of our real-time scheduling system here, but our focus is on an interface to the system that is intended to allow even naive users to exploit it. The user of a VM can continuously adjust its schedule using an interface akin to a two-dimensional throttle (a joystick), up to the resource limits and the constraints of the other VMs. As he adjusts the schedule, an on-screen display shows the cost of the current schedule.

Through an extensive user study, we find that this interface lets all users cleanly trade off between comfort and cost, appropriately customizing their schedules without knowing anything about CPU scheduling. In

other words, our interface provides an effective means for users to express their diverse CPU needs and have the system meet them.

We strongly believe that the overall approach of incorporating direct user input into resource scheduling will extend to other resources, combinations of resources, distributed systems, and autonomic computing.

2 Related work

Systems researchers have proposed a wide range of scheduling approaches to attempt to automatically optimize both for responsiveness and utilization. Examples include the BSD Unix scheduler [28], lottery scheduling [44], weighted fair queuing and its derivatives [3], BVT [9], SRPT scheduling [2], and periodic [23, 17] and sporadic [24] hard real-time models, as well as soft real-time models for multimedia [6, 31]. In some models, user interaction is included *implicitly* and *indirectly* in scheduling decisions. For example, the Unix scheduler provides a temporary priority boost to processes that have become unblocked. Since a typical interactive process blocks mostly waiting for user input, the boost gives it the effect of responding quickly, even in a system that is experiencing high load.

Direct user input is represented as well. For example, early work [27, 8] used on-screen buttons that encapsulated code to tailor of applications UIs. Weighted fair queuing allows users to explicitly weight each of their processes, controlling the CPU share given to each. Microsoft Windows allows a user to specify the scheduling class of a process. By raising the scheduling class of a process from "Normal" to "Above Normal", he assures that Windows' fixed priority scheduler will always run his process in preference to "Normal" processes that are also ready to run. As another example, Unix systems provide the "nice" mechanism to bias Unix's dynamic priority scheduler. All of the direct user input mechanisms we are aware of, however, require that the user understand the scheduler to get good results. Indeed, with schedulers like the Windows scheduler, it is very easy for a user to live-lock the system if he doesn't know what he's doing. Ours is the first scheduling system to incorporate direct user input from even naive users.

Using direct user input in the scheduling process would appear to be in the purview of human-computer interaction research and psychology. However, the work in those areas has concentrated on the impact of latency on user-perceived utility of the system [19, 10], and on user frustration with different user interfaces [18, 34]. Within the systems community, related work has examined the performance of end-user operating systems using latency as opposed to throughput [11], and suggested models for interactive user workload [4]. Recent work has also demonstrated using careful user studies that many programs can be modified by naive programmers to support limited adaptation [1]. However, there are no results on using direct user input from even naive users to guide the scheduler.

The importance and challenge of determining appropriate SLAs or quality of service (QoS) constraints has been recognized within the adaptive applications and autonomic computing communities. Adaptive application frameworks such as Odyssey [32], QuOin [46], and Amaranth [16] assume that SLAs or QoS constraints are supplied to them. Many forms for this information have been proposed, including composable utility functions [33], decision procedures [5], and aspects [26]. Our work relates in two ways. First, it presents a way of discovering appropriate SLAs from end-user input. Second, it shows that it is possible to avoid such intermediate representations, tying the end-user directly to the scheduling/optimization process.

Autonomic computing seeks to either automate the management of computer systems or simplify administrator-based management of them. Some work in this area has focused on direct interaction with the administrator, including capturing the effects of operator error [29], exposing debugging of configurations as a search process [45], adjusting cycle stealing so as to control impact on users [40], and using performance feedback to the administrator to help adjust policy [25]. As far as we are aware, however, no work in autonomic computing directly incorporates the end-user.

3 User diversity

It is important that we spend some time summarizing results from our previous paper [14] and why they motivate direct user input in resource schedulers.

Our previous paper described a double-blinded, controlled intervention study of the effects of resource contention on the comfort of the users of typical interactive desktop applications on Microsoft Windows. The resources studied were CPU bandwidth, disk bandwidth, and physical memory. The applications used were identical to those described in Section 6. 38 users, including graduate students, undergraduates, and staff at Northwestern University participated. The users were recruited and selected in a similar manner to that of Section 6, and most were not knowledgeable about resource scheduling. In the study, a user was instructed to carry out a given task (e.g., replicate a drawing using Powerpoint), and to press a "discomfort button" if she felt the computer was operating uncomfortably. While the user carried out the task, we applied randomly selected resource contention profiles for the three resources. These profiles included a blank profile (the "placebo"), ramp

functions (contention gradually increases with time), and step functions (contention abruptly increases at a point in time). Users were unaware of the specifics of the study (i.e., that we were subjecting them to differing resource contentions), and the proctors were unaware of the profile ordering for any user. The machines used were hidden, and the UI was modified so that the user could not determine the degree of contention other than the slowdown or jitter it induced in the application.

Analyzing the contexts in which users indicated discomfort resulted in a range of qualitative and quantitative conclusions. The observation that users were generally quite intolerant to jitter led to the choice of a periodic real-time scheduling model in Virtuoso, described in the next section. Users were far more sensitive to restrictions or variations in CPU bandwidth than disk or memory.

There was a large variation in the acceptable CPU contention among users, applications, and users and applications taken jointly. The CPU contention at which ≤ 5 % of users were uncomfortable differed by a factor of 17 from the least sensitive application (Word) to the most sensitive one (Quake II); equivalent tolerable average slowdowns range from 4.1 to 1.2. Because each user conducted a self-rating of her familiarity with Windows and our various applications before participating, we could analyze diversity among self-defined classes of users. Again studying the CPU contention at which ≤ 5 % of users are uncomfortable, we found significant ($p \leq 0.031$) differences in tolerable contention and slowdown ranging from 0.2 to 1.1 among the various groups defined by the self-rating. The necessarily limited amount of data available from a user study such as this forces us to draw only qualitative conclusions about the users and applications taken jointly. However, clearly, it is the user who chooses the application in a desktop environment and thus is the dominant source of variation.

The key conclusion of the study with respect to the present paper is that there is considerable diversity in the tolerance that users have for resource contention. This diversity argues for per-user tailoring of utility functions and/or SLAs, and leads to the question we address in this paper: How do we accomplish this per-user tailoring? We now show how to answer this question for the specific case of CPU scheduling.

4 Scheduling VMs in Virtuoso

Virtuoso is middleware for virtual machine-based utility computing that for a user very closely emulates the existing process of buying, configuring, and using an Intel-based computer, a process with which many users and certainly all system administrators are familiar. In Virtuoso, the user visits a web site, much like the web site of Dell or IBM or any other company that sells Intel-based computers. The site allows him to specify the hardware and software configuration of a computer and its performance requirements, and then order one or more of them. The user receives a reference to the virtual machine which he can then use to start, stop, reset, and clone the machine. The system presents the illusion that the virtual machine is right next to the user. The console display is sent back to the user's machine, the CD-ROM is proxied to the user's machine's CD-ROM, and the VM appears to be plugged into the network side-by-side with the user's machine. The user can then install additional software, including operating systems.

Virtuoso connects users with providers. A provider makes physical computers available on which users' VMs can be executed. A single physical computer can host multiple VMs.

A Virtuoso provider uses VMware GSX Server as its virtual machine monitor (VMM). GSX is a "type II" VMM [12], meaning that it executes as a process on an underlying operating system, Linux in our case. By scheduling the process, we schedule all the activity occurring inside the VM (all the applications running in a VM whose operating system is Microsoft Windows, for example) as a single unit.

Virtuoso uses the periodic real-time model as a unifying abstraction to describe the needs of diverse workloads and then schedule them. In the periodic real-time model, a task is run for $slice$ seconds every $period$ seconds. Using earliest deadline first (EDF) schedulability analysis [23], the scheduler can determine whether some set of $(period, slice)$ constraints can be met. The scheduler then simply uses dynamic priority preemptive scheduling with the deadlines of the admitted tasks as priorities.

VSched, which is described in detail in a previous paper [22], is a user-level implementation of this approach for Linux that offers soft real-time guarantees. It runs as a Linux process that schedules other Linux processes. We use it here to schedule the VMs on the host computer. VSched supports $(period, slice)$ constraints ranging from the low hundreds of microseconds (if certain kernel features are available) to days. Using this range, the needs of both highly interactive VMs and long running batch VMs can be described and accommodated.

An important design criterion for VSched is that a VM's constraints can be changed very quickly (in about a millisecond) so that an interactive user can change his VM's performance immediately.

4.1 Implementation

VSched consists of a server and a client, as shown in Figure 1. The VSched server is a daemon running on Linux that spawns the scheduling core, which executes

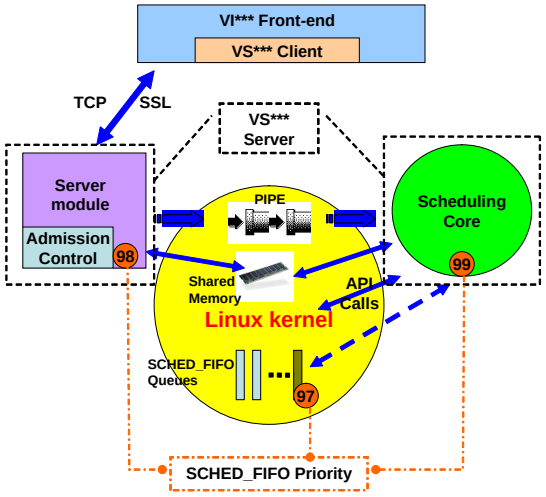


Figure 1: Structure of VSched.

the scheduling scheme described above. The VSched client communicates with the server over a TCP connection that is encrypted using SSL. Authentication is accomplished by a password exchange. The server communicates with the scheduling core through a shared memory array and signaling.

Client interface Using the VSched client, a user can connect to VSched server and request that any process be executed according to a period and slice. Virtuoso keeps track of the *pids* used by its VMs. In response to such a request, the VSched server determines whether the request is feasible. If it is, it will add the process to the array and inform the scheduling core.

Admission control VSched's admission control algorithm is based on the admissibility test of the EDF algorithm. Instead of trying to maximize the total utilization, we allow the system administrator to reserve a certain percentage of CPU time for non-real-time (SCHED_OTHER) processes. The percentage can be set by the system administrator when starting the VSched daemon.

Scheduling core The scheduling core is a modified EDF scheduler that dispatches processes in EDF order but interrupts them when they have exhausted their allocated CPU for the current period. If configured by the system administrator, VSched will stop the processes at this point, resuming them when their next period begins.

When the scheduling core receives scheduling requests from the server module, it will interrupt the current task and make an immediate scheduling decision based on the new task set. The scheduling request can be a request for scheduling a newly arrived task or for changing a task that has been previously admitted.

Employed mechanisms The three highest priorities of SCHED_FIFO, the highest priority scheduling class in Linux, are reserved for VSched use. The scheduling core is run as the highest priority SCHED_FIFO process on the system, assuring that when it becomes runnable, it immediately is given the CPU. The server module is run as SCHED_FIFO with the next highest priority so that it will immediately service new requests whenever the scheduling core is not running. The scheduling core assigns the process that it currently wants to run the third highest SCHED_FIFO and (optionally) sends it a SIG_CONT. The process that is being switched away from is assigned an ordinary SCHED_OTHER priority. If the administrator has configured hard limiting on its resource use, it is also sent a SIG_STOP, otherwise VSched operates as a work-conserving scheduler with respect to its admitted processes.

The mechanisms of VSched and scheduling in Virtuoso are described in much more detail our earlier paper and the software itself is publicly available. The UI software we describe next is available from the authors.

4.2 Performance isolation

Most variants of the Linux kernel, including the one we use in this paper, do not provide bounded ISR times and can suffer from priority inversion, it is impossible to provide hard real-time support, and VSched does not claim to. Nonetheless, as we report in our earlier paper, its soft real-time behavior is quite good, rarely missing deadlines and only then by very small amounts. We also demonstrate that VSched effectively isolates VMs, and that, despite only controlling CPU, it serves to effectively throttle I/O as an indirect effect. Physical memory isolation is provided by the VMM we use.

5 User interface

We have developed a graphical interface to allow even a naive user to easily use VSched to set an appropriate $(period, slice)$ constraint for his Windows VM. The tool indicates to the user the cost of his current schedule and allows him to directly manipulate $(period, slice)$. VSched can change the schedule of a VM in about a millisecond, allowing for very smooth control.

The holy grail for such an interface is that it be invisible or non-intrusive until the user is unhappy with performance, and then can be nearly instantly manipulated to change the schedule. We have explored several possible interfaces, including an on-screen interface (sliders), a centering joystick, a centering joystick with force

(a) GUI

(b) Force Feedback Joystick

(c) Non-centering Joystick

(d) Precision Non-centering Joystick

Figure 2: Control interface. The combination of (a) and (c) is used in our study.

feedback[1], a non-centering joystick, and a precision non-centering joystick. These interfaces are illustrated in Figure 2. We are also looking at trackballs, throttle controllers, and knob controllers.

Although we considered many different interfaces, non-centering joysticks appear to be the best option. In such a joystick, the control stalk maintains its current deflection even after the user removes his hand. The horizontal and vertical deflection are mapped into increasing $period$ (left to right) and increasing utilization ($slice/period$) (bottom to top). Note that all positions of the joystick correspond to valid schedules.

In the following, we use a low precision non-centering joystick. In particular, we modified a cheap centering joystick (a variant of Figure 2(b)) to produce a non-centering joystick (Figure 2(c)) by removing a spring. This joystick is not as precise as a precision non-centering joystick (Figure 2(d)), but using it serves two purposes. First, it demonstrates that the interface can be quite inexpensive ($<$\$10 versus $>$\$200 for the precision joystick). Second, the joystick need not offer high precision to be useful (the precision joystick has an order of magnitude more levels both vertically and horizontally).

The interface shows the cost of the current schedule (which can be changed in milliseconds). The specific cost function that is used is

$$cost = 100 \times \left(\frac{slice}{period} + \beta \times \frac{overhead}{slice} \right)$$

Where $overhead$ is the time to execute the scheduling core of VSched once. The purpose here is to capture the fact that as $slice$ declines, more time is spent in VSched and the kernel on behalf of the process. For typical user-selected schedules for interactive VMs, the influence of the overhead is minimal, and the cost is effectively the utilization of the user's VM.

[1]The idea here is to physically convey to the user when he is asking for $(period, slice)$ constraints that are impossible due to the lack of hardware resources on the machine or to conflicting constraints from other VMs.

6 User study

We conducted a user study to determine whether end-users could use our interface to find schedules for their interactive VMs that were comfortable, and to determine whether users could trade off between cost and comfort using the interface.

6.1 Particulars

The 18 users in our study consisted primarily of graduate students and undergraduates from the engineering departments at Northwestern University, and included two participants who had no CS or ECE background. None of the users were familiar with real-time scheduling concepts. We advertised for participants via flyers and email, and vetted respondents to be sure they were at least slightly familiar with the common applications we would have them use. Each user was given \$15 for participating.

The test machine was a Dell Optiplex GX270 (2 GHz P4, 512 MB, 80 GB, 17" monitor, 100 mbit Ethernet). The machine ran:

- VMware GSX Server 3.1
- VSched server running as a daemon,
- VM running Windows XP Professional, the applications (Microsoft Word 2002, Powerpoint 2002, Internet Explorer 6.0, Quake II), and our interface, and
- Logitech WingMan Attack 2 Joystick modified to be non-centering, as described earlier.

The VM was run in full-screen mode and the users were not told that they were using virtualization. The smallest slice and period possible were 1 ms, while the largest were 1 s.

6.2 Process

During the study, the user used the Windows VM for four tasks: word processing, presentation creation, web browsing, and game playing. Each task was 15 minutes long with 5 minutes for each of three sub-tasks. We asked users to answer some questions (described later) after each sub-task. We also video-taped users during the study, and the users were told that the video tape and other mechanisms would allow us to determine their degree of comfort during the study independently of the questions we were asking them. This mild use of deception, a widely used technique in psychological research [39], was employed to decrease the likelihood that study participants would lie or operate the system less aggressively than they might outside of the study.

From the user's perspective, the study looked like the following. At the beginning of each task and subtask, the joystick was recentered, corresponding to a 500 ms period with a 50% utilization. The intent was to force the user to manipulate the joystick at least once, since for all of the applications, this schedule was intolerable to us.

1. Adaptation Phase I (8 minutes): The user acclimatized himself to the performance of the Windows VM by using the applications. Questions:
   - Do you feel you are familiar with the performance of this computer? (Y/N)
   - Are you comfortable with these applications? (Y/N)
2. Adaptation Phase II (5 minutes): The user acclimatized himself to the VSched control mechanism, Figures 2(a) and (c). The user listened to MP3-encoded music using Windows Media Player and noticed how the playback behavior changed when he moved the joystick. At the beginning of this stage, the proctor told the user that moving the joystick would change the responsiveness of the computer and that, in general, moving the joystick to the upper-right would make the machine faster, while moving the joystick to the lower left would slow the machine down. However, the proctor also told them that the control was not a simple linear control, that all joystick positions are valid, and that the user should do his best to explore using the joystick control by himself. Questions:
   - Do you feel that you understand the control mechanism? (Y/N)
   - Do you feel that you can use the control mechanism? (Y/N)
3. Word processing using Microsoft Word (15 minutes): Each user typed in a non-technical document with limited formatting.
   - **Sub-task I: Comfort (5 minutes)** The user was told to try to find a joystick setting that he felt was comfortable for him. Questions:
     – Did you find that the joystick control was understandable in this application? (Y/N)
     – Were you able to find a setting that was comfortable? (Y/N)
   - **Sub-task II: Comfort and Cost (5 minutes)** The user was given a cost bar (Figure 2(a)) that showed the current cost of using the Windows VM. When the user moved the joystick, both the responsiveness of the computer and current cost change. The proctor asked the user to do their best to find a comfortable joystick setting that was of the lowest cost. Questions:
     – Did you find that the joystick control was understandable in this application? (Y/N)
     – Were you able to find a setting that was comfortable? (Y/N)
     – If yes, what's the cost?
   - **Sub-task III: Comfort and Cost With Perceived External Observation (5 minutes)** This sub-task was identical to the previous one, except that the proctor told the user that the system had mechanisms by which it could independently determine whether the user was comfortable or not and whether a comfortable setting was of the lowest possible cost. It was claimed that this analysis was achieved through measurement of the efficiency of the VM (the percentage of cycles that the user has allocated that he is actually using), analysis of their mouse, keyboard, and joystick input, and psychological analysis of the video tape. Questions:
     – Did you find that the joystick control was understandable in this application? (Y/N)
     – Were you able to find a setting that was comfortable? (Y/N)
     – If yes, what's the cost?
4. Presentation creation using Microsoft Powerpoint (15 minutes): Each user duplicated a presentation consisting of complex diagrams involving drawing and labeling from a hard copy of a sample presentation.
   - The same three sub-tasks as for word processing with the same questions following each sub-task.

5. Browsing and research with Internet Explorer (15 minutes): Each user was assigned a news web site and asked to read the first paragraphs of the main news stories. Based on this, they searched for related material and saved it. This task involved multiple application windows.

    - The same three sub-tasks as for word processing with the same questions following each sub-task.

6. Playing Quake II (15 minutes)

    - The same three sub-tasks as for word processing with the same questions following each sub-task.

The written protocol and the form the user filled out are available from the authors by request.

As the user performed the tasks, we recorded the following information:

- Periodic measurements of system and user time,
- Periodic measurements of utilization (portion of the allotted time that was spent unblocked), and
- For each joystick event, the time stamp and the new $(period, slice)$ and $cost$.

The user was unaware of the recording process. He saw only the cost of the current schedule.

6.3 Qualitative results

Our users interacted with the system in many different ways. No classification scheme seems obvious other than the extent to which they manipulated the joystick (both the number of times and the range covered.) Recall that after moving the joystick, the user needs to return to the application to test the new schedule's effects on its performance. However, to explore the effect on cost, the user has immediate feedback in the form of the on-screen meter (Figure 2(a)). We present the traces of three users as examples of different ways of interacting with the system, ranging from focusing first on the cost to focusing primarily on the comfort of the application.

Figure 3 shows the behavior of a user primarily searching for low cost. Recall that at the beginning of each task and subtask, the joystick was recentered, corresponding to a 500 ms period with a 50% utilization. Each row of graphs in the figure represents the behavior for a single application (specifically, sub-task III in the task). Within a row, the left hand graph shows the track of the user's joystick over time, the horizontal axis being the $period$, while the vertical axis is the utilization ($\frac{slice}{period}$). The top right corner of the graph has highest performance (smallest period, highest utilization), while the bottom left has the lowest performance (largest period, smallest utilization). The right hand graph shows the cost of the schedule over time. Note that because the cost is dominated by utilization, there are flat regions. These are typically due to a user changing the period while keeping the utilization constant.

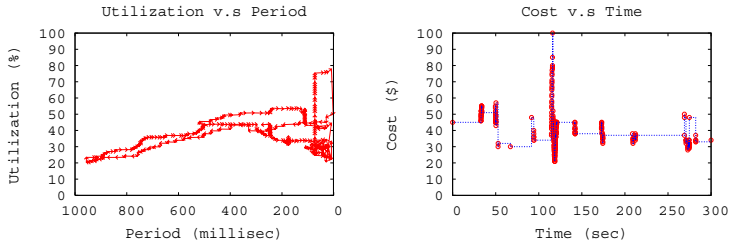
Note that the user of Figure 3 is hunting through the space, smoothly changing the joystick to look for better prices. This is particularly evident in Figure 3(g), where the user eventually discovers he can tolerate a much lower utilization (and thus cost) in the game if he uses a very small period.

The user of Figure 4 spends less time hunting for a low cost and more time finding comfortable settings. Also, we notice that unlike the previous user, this one needs high utilization for the game, even though he tried a small period as well. In general this user has optimized for a more costly machine than the previous user.

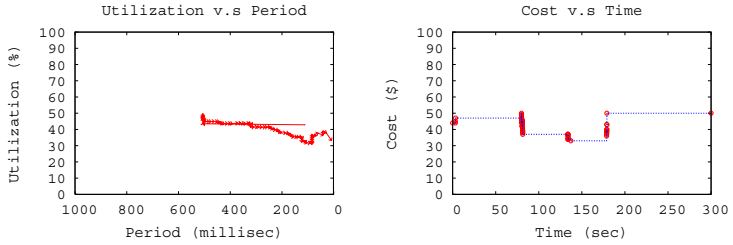
Figure 5 is a user who is optimizing for comfort. He is carefully moving the joystick, and then testing the new schedule's impact on the application for a while before moving it again. Because this takes time, only a few movements are recorded. Probably the most significant movement from the initial position (500 ms period, 50% utilization) is in the case of the game, where the user slowly shrinks the period and increases utilization until it plays comfortably for him. Notice that because the game action continues even if the user is not playing, the effects can be seen almost immediately.

For this last user, Figure 6 shows the cost and efficiency as a function of time, where the efficiency is the ratio of the actual time used by the VM (system and user time) to the amount of time it was allotted. Ideally, at the lowest cost at which the user feels comfortable, the efficiency should be very high. However, this is clearly not possible since the user cannot modify his schedule continuously and still use the application. Hence, the user will generally choose to have a certain amount of "buffering" in his schedule to accommodate bursts of computation. From the figures we can see that the less CPU intensive an application is, the lower the efficiency. Applications like Word, Powerpoint and IE don't need a continuous allocation of the CPU, but nonetheless need to be quickly scheduled when a user event occurs. A certain percentage of the unused slice serves as the "buffer" to make user feel comfortable. The user can shrink this buffer as close to the limit as he can tolerate.

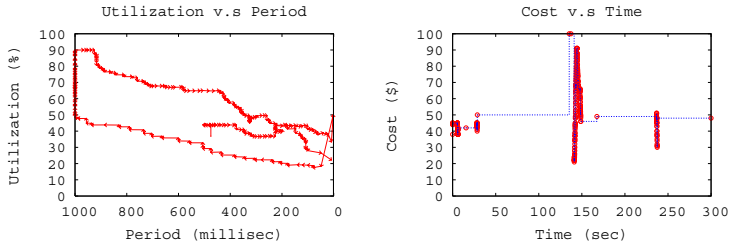
The exception is the game. Quake, like many first person shooter games, simply tries to run with as high a frame rate as possible. Whatever schedule is chosen, the efficiency is very high since all of the slice is consumed. Here, the user is indirectly controlling the frame rate and



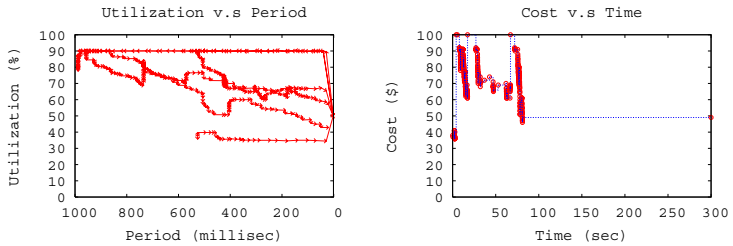
(a) Word (track) (b) Word (cost v. time)



(c) Powerpoint (track) (d) Powerpoint (cost v. time)



(e) Web (track) (f) Web (cost v. time)



(g) Game (track) (h) Game (cost v. time)

Figure 3: User A: Tracks, cost versus time.

jitter of the application.

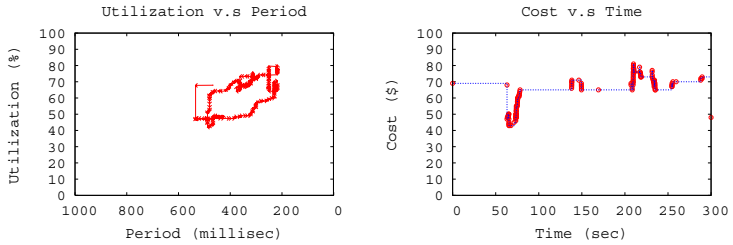
6.4 Quantitative results

Figure 7 summarizes the responses of users to the questions in our study, providing 95% confidence intervals for the proportion of users who responded in the affirmative. Notice that in all cases, the responses allow us to discount the null hypothesis, that the users are responding randomly, with $> 95\%$ confidence.

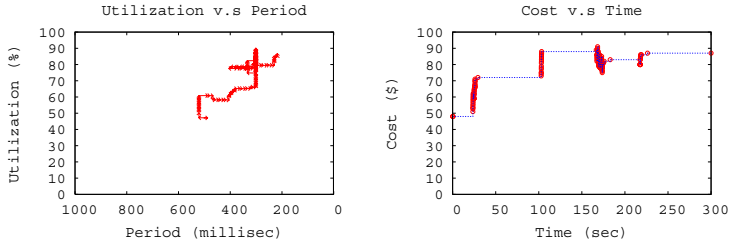
The overwhelming majority of users found that they

- understood our control mechanism,
- could use it to find a comfortable position, and
- could use to find a comfortable position that they believed was of lowest cost.

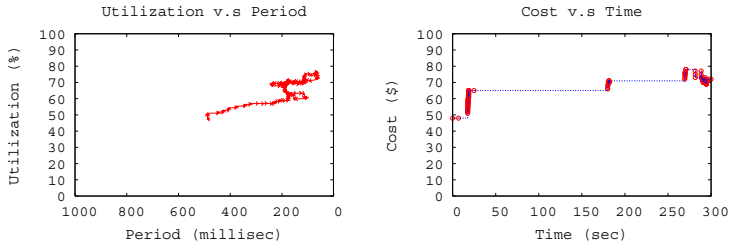
Despite the disparity among the applications and the users, there was little disparity in the users' reactions. There were only two situations where a significant fraction of the users had difficulty finding a comfortable or comfortable and low-cost setting. 28% of users had difficulty finding a comfortable setting for the web browsing task (sub-task I), while 22% had difficulty finding a comfortable, low cost setting for the first person shooter task (sub-task II). In both cases, the numbers result from one of the users answering the question unintelligibly. Furthermore, that user answered "yes" to the more restrictive corresponding question (where we are attempting to deceive him into believing we can answer the question



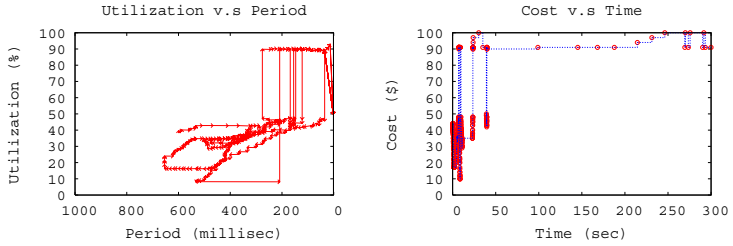
(a) Word (track) (b) Word (cost v. time)



(c) Powerpoint (track) (d) Powerpoint (cost v. time)



(e) Web (track) (f) Web (cost v. time)



(g) Game (track) (h) Game (cost v. time)

Figure 4: User B: Tracks, cost versus time.

independently).

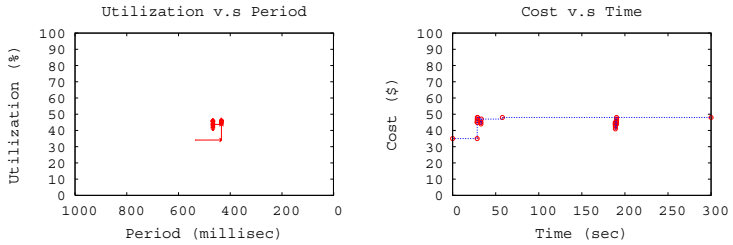
For sub-tasks II and III of each task, we had the user try to find a setting that he was comfortable with and that he believed was of minimal cost. If he felt he had found a comfortable setting, we recorded its cost. Figure 8 provides the statistics for these costs. We can see that:

- As we might expect, costs, on average increase as we look at applications with increasingly finer grain interactivity.

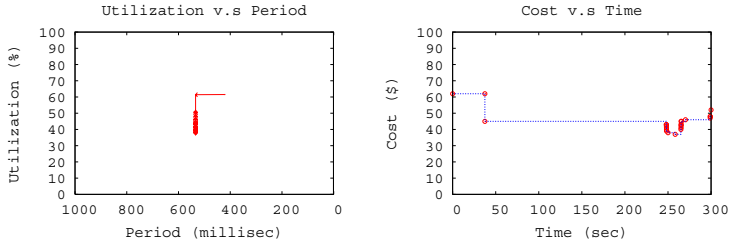
- There is tremendous variation in acceptable cost among the users. The standard deviation is nearly half of the average cost. The maximum cost is as much as five times the minimum cost. This should not be surprising given the wide variation among users found in a previous study of resource borrowing (Section 3).

- Nonetheless, almost all users were able to find a setting that gave them comfortable performance.

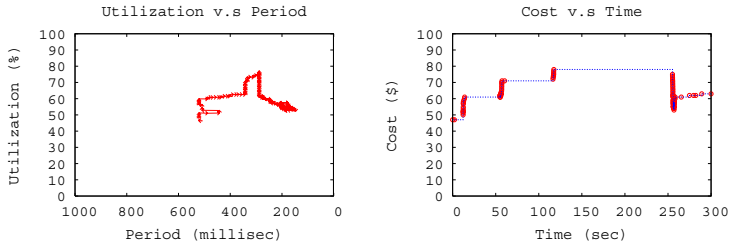
Figure 9(a) shows the statistics, aggregated across the users, on the duration from the beginning of sub-tasks II and III of each task to the time that the lowest cost was first encountered. For example, to compute the "Average" statistic for "Word III", we examined each user's trace to find the time from the beginning of sub-task III of the Word task to the time when the user's reported lowest comfortable cost was first found. We then averaged these times. The other statistics are computed similarly. Figure 9(b) shows identical statistics for the duration to the last time the lowest cost occurred. Note that one of



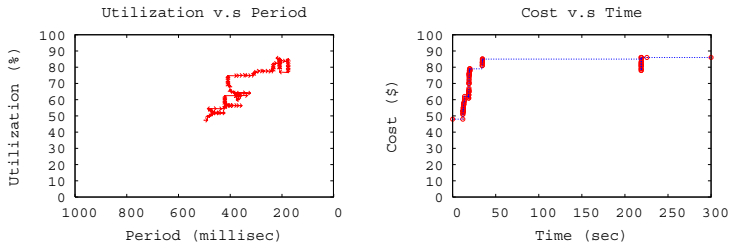
(a) Word (track) (b) Word (cost v. time)



(c) Powerpoint (track) (d) Powerpoint (cost v. time)



(e) Web (track) (f) Web (cost v. time)



(g) Game (track) (h) Game (cost v. time)

Figure 5: User C: Tracks, cost versus time.

durations slightly exceeds 300 s due to no movement of the joystick at the end of a sub-task that was terminated in slightly more than 5 minutes.

We can see that the median time for the user to find the setting of lowest cost that is comfortable for him is in the range from 25-60 seconds. Notice that this time includes use of the application. The user does a quick manipulation of the joystick and then tries to use the application for a short while with the new setting. Recall that these times are for the first 10 minutes that the user has been introduced to the combination of the application and scheduling interface. One would expect that the times would decline as familiarity increases.

The upshot of the study described in this section is that we have identified a *practical* mechanism by which user input can be incorporated into the CPU scheduling process for Virtuoso. Using that input, the system can adapt the schedule of the user's VM to fit the user and the application he is currently running, the side effect of which is that the system can run more interactive users simultaneously, or allocate more time for long-running batch VMs. The user can quickly guide the system to a schedule that simultaneously optimizes both for his comfort in using an application and for low cost.

7 Issues in extension

Having demonstrated the feasibility and utility of direct user feedback as applied to CPU scheduling for interactive VMs, we now describe issues in extending our ideas

Task	Sub-task	Question	Yes	No	NA	Yes/Total	95% CT
Acclim.	Adaptation I	Do you feel you are familiar with the performance of this computer?	18	0	0	**1**	(1,1)
		Are you comfortable with these applications?	17	1	0	**0.94**	(0.84, 1.05)
	Adaptation II	Do you feel that you understand the control mechanism?	18	0	0	**1.00**	(1,1)
		Do you feel that you can use the control mechanism?	18	0	0	**1.00**	(1,1)
Word	I Comfort	Did you find that the joystick control was understandable in this task?	17	1	0	**0.94**	(0.84, 1.05)
		Were you able to find a setting that was comfortable?	18	0	0	**1.00**	(1,1)
	II Comfort+Cost	Did you find that the joystick control was understandable in this task?	17	1	0	**0.94**	(0.84, 1.05)
		Were you able to find a setting that was comfortable?	18	0	0	**1.00**	(1,1)
	III Comfort+Cost+Ext	Did you find that the joystick control was understandable in this task?	18	0	0	**1.00**	(1,1)
		Were you able to find a setting that was comfortable?	18	0	0	**1.00**	(1,1)
Powerpoint	I Comfort	Did you find that the joystick control was understandable in this task?	16	2	0	**0.89**	(0.74, 1.03)
		Were you able to find a setting that was comfortable?	18	0	0	**1.00**	(1,1)
	II Comfort+Cost	Did you find that the joystick control was understandable in this task?	17	1	0	**0.94**	(0.84, 1.05)
		Were you able to find a setting that was comfortable?	17	1	0	**0.94**	(0.84, 1.05)
	III Comfort+Cost+Ext	Did you find that the joystick control was understandable in this task?	16	1	0	**0.89**	(0.74, 1.03)
		Were you able to find a setting that was comfortable?	17	1	0	**0.94**	(0.70, 1.08)
Web	I Comfort	Did you find that the joystick control was understandable in this task?	16	2	0	**0.89**	(0.74, 1.03)
		Were you able to find a setting that was comfortable?	13	4	1	**0.72**	(0.52, 0.93)
	II Comfort+Cost	Did you find that the joystick control was understandable in this task?	17	1	0	**0.94**	(0.84, 1.05)
		Were you able to find a setting that was comfortable?	16	2	0	**0.89**	(0.74, 1.03)
	III Comfort+Cost+Ext	Did you find that the joystick control was understandable in this task?	17	1	0	**0.94**	(0.84, 1.05)
		Were you able to find a setting that was comfortable?	16	1	1	**0.89**	(0.74, 1.03)
Game	I Comfort	Did you find that the joystick control was understandable in this task?	18	0	0	**1.00**	(1, 1)
		Were you able to find a setting that was comfortable?	16	2	0	**0.89**	(0.74, 1.03)
	II Comfort+Cost	Did you find that the joystick control was understandable in this task?	17	1	0	**0.94**	(0.84, 1.05)
		Were you able to find a setting that was comfortable?	14	3	1	**0.78**	(0.59, 0.97)
	III Comfort+Cost+Ext	Did you find that the joystick control was understandable in this task?	17	1	0	**0.94**	(0.84, 1.05)
		Were you able to find a setting that was comfortable?	16	2	0	**0.89**	(0.74, 1.03)

Figure 7: Summary of user responses in study.

to other domains. Our thinking is also informed by two additional domains. First, we have applied our approach to power management on laptop computers, demonstrating CPU power reductions of $> 20\%$. Second, we have been working on the general adaptation problem exposed within the Virtuoso system [43, 42, 15].

Virtuoso provides many adaptation and resource reservation mechanisms to improve the performance of existing, unmodified applications running in communicating VMs. These include VM migration and overlay topology configuration/forwarding rules [41], lightpath setup in optical networks [21], and local scheduling of VMs [22]. Additionally, Virtuoso can observe the network and host traffic of the VMs to probe the underlying network [15] and the application [13] for their communication topology and other resource demands [42]. Using this information to engage the adaptation and reservation mechanisms to increase an application's performance is a challenging, NP-hard optimization problem [43], one that is often difficult even to pose well. Applying our concept of direct human input to it poses the following challenges.

Frequency of input: In each of our domains, we have noted that more frequent user input leads to better performance (lower cost, lower power consumption, high application throughput, among other metrics). However, it is obvious that there must be limits to this frequency. Control algorithms that make use of direct user input must be able to work when the input is infrequent and/or aperiodic. In our view, sensible low-frequency input will take one of three forms:

1. Evaluations of a current configuration or SLA, resulting in user-specific utility functions.

2. Specifications of configurations or SLAs.

3. Directions for search within a space of configurations or SLAs.

The work described here takes the later two forms.

Interface: Careful user interface design and evaluation are critical to success, especially when targeting naive users. We have generally found that having a very simple, tactile interface separate from the "main" user interface of the application or OS, is preferable because it clearly demarcates "system" control from "application" control in the user's mind, can be completely ignored when not needed, and is easier to explain. Designing an adequate process for acquiring exploiting input of form 1 is far easier than for forms 2 and 3. We next describe specific issues related to the latter forms.

Task	Sub-task	Question	Avg	Std	Min	Max	Med	Mod
Word	II Comfort+Cost	Did you find that the joystick control was understandable in this task?						
		Were you able to find a setting that was comfortable?						
		If yes, what's the cost?	46.0	**20.4**	19	86	40.5	40
	III Comfort+Cost+Ext	Did you find that the joystick control was understandable in this task?						
		Were you able to find a setting that was comfortable?						
		If yes, what's the cost?	48.4	**20.7**	19	84	48	19
Powerpoint	II Comfort+Cost	Did you find that the joystick control was understandable in this task?						
		Were you able to find a setting that was comfortable?						
		If yes, what's the cost?	52.4	**19.5**	20	91	45	62
	III Comfort+Cost+Ext	Did you find that the joystick control was understandable in this task?						
		Were you able to find a setting that was comfortable?						
		If yes, what's the cost?	52.3	**19.2**	18	87	50	38
Web	II Comfort+Cost	Did you find that the joystick control was understandable in this task?						
		Were you able to find a setting that was comfortable?						
		If yes, what's the cost?	49.6	**22.7**	15	90	47	41
	III Comfort+Cost+Ext	Did you find that the joystick control was understandable in this task?						
		Were you able to find a setting that was comfortable?						
		If yes, what's the cost?	50.2	**23.3**	16	87	50	28
Game	II Comfort+Cost	Did you find that the joystick control was understandable in this task?						
		Were you able to find a setting that was comfortable?						
		If yes, what's the cost?	78.8	**14.1**	50	93	84.5	90
	III Comfort+Cost+Ext	Did you find that the joystick control was understandable in this task?						
		Were you able to find a setting that was comfortable?						
		If yes, what's the cost?	76.5	**14.9**	49	91	81	81

Figure 8: Statistics of the lowest costs reported by users in study.

Mechanism transition: In our system, changing a VM's schedule is virtually instantaneous, if the schedule is feasible on the physical host it is currently running on. If the desired schedule is not feasible, we must indicate this to the user and use a different mechanism (e.g., migrate his VM to a different host) to satisfy him. While very fast VM migration techniques now exist [7, 30], they still take much longer than changing a schedule, and have a much higher resource cost. How can we represent these time and resource costs to the user?

Categorical dimensions: A configuration or SLA can be thought of as a point within a multidimensional space. If a dimension is categorical (for example, a VM can be mapped to one of several choices, or a overlay link can be added or not), it is difficult to present it using an easily understood external interface.

Dimensionality: In the present work, we expose the schedule directly to the user. This is easy to do because its two dimensions map directly to the two dimensions of the joystick, and both dimensions are continuous. As we add resources, the number of dimensions grows and makes a simple mapping impossible. Of course, there are many examples of using low dimensional input devices to explore high dimensional spaces. A large part of the problem is how to visualize the current configuration/SLA and its neighborhood.

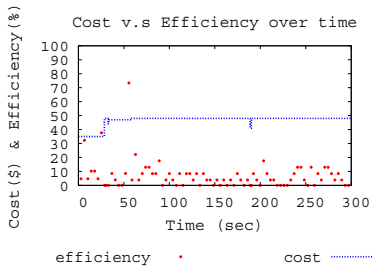
8 Conclusions and future work

We have described and evaluated a technique for putting even naive users in direct, explicit control of the scheduling of their interactive computing environments through the combination of a joystick and an on-screen display of cost. In so doing, we have demonstrated that with such input it is *possible* and *practical* to adapt the schedule dynamically to the user, letting him trade off between the comfort of the environment and its cost. Because the tolerance for cost and the comfort with a given schedule is highly dependent on both the applications being used and on the user himself, this technique seems very fruitful both for tailoring computing environments to users and making them cheaper for everyone.

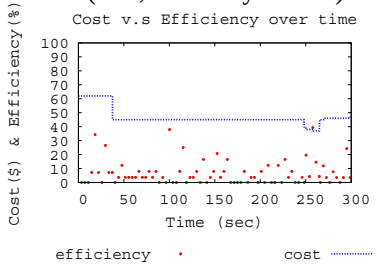
We are currently exploring how to extend our results to scheduling other resources, combinations of resources, and in power management, with a particular focus on the distributed adaptation problem described in the previous section.

References

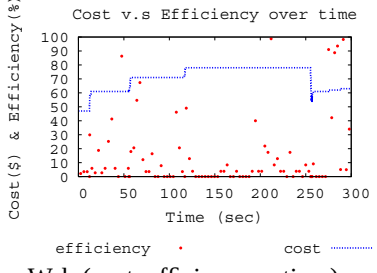
[1] BALAN, R. K., GERGLE, D., SATYANARAYANAN, M., AND HERBSLEB, J. Simplifying cyber foraging for mobile devices. Tech. Rep. CMU-CS-05-157, Computer



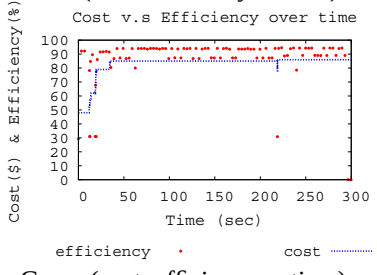
Word (cost, efficiency v. time)



Powerpoint (cost, efficiency v. time)

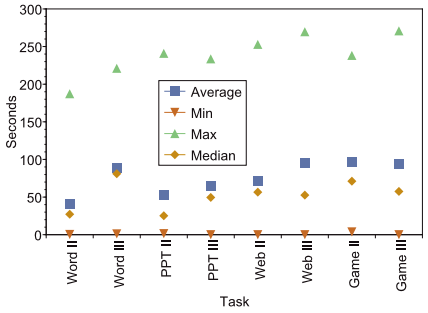


Web (cost, efficiency v. time)

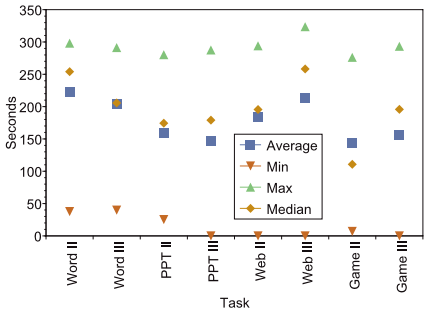


Game (cost, efficiency v. time)

Figure 6: User C: Cost, efficiency versus time.



(a) Duration to first encounter of lowest cost



(b) Duration to last encounter of lowest cost

Figure 9: Duration to lowest cost.

Science Department, Carnegie Mellon University, August 2005.

[2] BANSAL, N., AND HARCHOL-BALTER, M. Analysis of srpt scheduling: Investigating unfairness. In *Proceeds of the 2001 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (2001), pp. 279–290.

[3] BENNETT, J., AND ZHANG, H. Worst-case fair weighted fair queueing. In *Proceedings of IEEE INFOCOM 1996* (March 1996), pp. 120–127.

[4] BHOLA, S., AND AHAMAD, M. Workload modeling for highly interactive applications. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (1999), pp. 210–211. Extended version as Technical Report GIT-CC-99-2, College of Computing, Georgia Tech.

[5] BOWMAN, H., BLAIR, L., BLAIR, G. S., AND CHETWYND, A. G. A formal description technique supporting expression of quality of service and media synchronization. In *Proceedings of the International COST Workshop* (November 1994), no. 882 in Lecture Notes in Computer Science, Springer, pp. 145–167.

[6] CHU, H.-H., AND NARHSTEDT, K. Cpu service classes for multimedia applications. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems* (June 1999).

[7] CLARK, C., FRASER, K., HAND, S., HANSEN, J. G., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. Live migration of virtual machines. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)* (2005).

[8] DOURISH, P. Evolution in the adoption and use of collaborative technologies. In *Proceedings of the ECSCW Workshop on the Evolving Use of Groupware* (September 1999).

[9] DUDA, K. J., AND CHERITON, D. R. Borrowed-virtual-time (bvt) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles* (1999), ACM Press, pp. 261–276.

[10] EMBLEY, D. W., AND NAGY, G. Behavioral aspects of text editors. *ACM Computing Surveys 13*, 1 (January 1981), 33–70.

[11] ENDO, Y., WANG, Z., CHEN, J. B., AND SELTZER, M. Using latency to evaluate interactive system performance. In *Proceedings of the 1996 Symposium on Operating Systems Design and Implementation* (1996).

[12] GOLDBERG, R. Survey of virtual machine research. *IEEE Computer* (June 1974), 34–45.

[13] GUPTA, A., AND DINDA, P. A. Inferring the topology and traffic load of parallel programs running in a virtual machine environment. In *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing (JSPPS 2004* (June 2004).

[14] GUPTA, A., LIN, B., AND DINDA, P. A. Measuring and understanding user comfort with resource borrowing. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC 2004)* (June 2004).

[15] GUPTA, A., ZANGRILLI, M., SUNDARARAJ, A., HUANG, A., DINDA, P., AND LOWEKAMP, B. Free network measurement for virtual machine distributed computing. In *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2006).

[16] HOOVER, C., HANSEN, J., KOOPMAN, P., AND TAMBOLI, S. The amaranth framework: Policy-based quality of service management for high-assurance computing. *International Journal of Reliability, Quality, and Safety Engineering* (December 2001).

[17] JONES, M., ROSU, D., AND ROSU, M.-C. Cpu reservations and time constraints: Efficient, predictable scheduling of independent activities. In *Proceedings of the 16th ACM Symposium on Operating System Principles (SOSP)* (1997).

[18] KLEIN, J. T. Computer response to user frustration. Master's thesis, Massachusetts Institute of Technology, 1999.

[19] KOMATSUBARA, A. Psychological upper and lower limits of system response time and user's preferance on skill level. In *Proceedings of the 7th International Conference on Human Computer Interaction (HCI International 97)* (August 1997), G. Salvendy, M. J. Smith, and R. J. Koubek, Eds., vol. 1, IEE, pp. 829–832.

[20] LAI, A., AND NIEH, J. Limits of wide-area thin-client computing. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (2002).

[21] LANGE, J., SUNDARARAJ, A., AND DINDA, P. Automatic dynamic run-time optical network reservations. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC)* (July 2005), pp. 255–264.

[22] LIN, B., AND DINDA, P. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In *Proceedings of ACM/IEEE SC 2005 (Supercomputing)* (November 2005).

[23] LIU, C. L., AND LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM 20*, 1 (January 1973), 46–61.

[24] LIU, J. *Real-time Systems*. Prentice Hall, 2000.

[25] LOTLIKA, R., VATSAVAI, R., MOHANIA, M., AND CHAKRAVARTHY, S. Policy scheduler advisor for performance management. In *Proceedings of the 2nd IEEE International Conference on Autonomic Computing (ICAC)* (June 2005).

[26] LOYALL, J. P., BAKKEN, D. D., SCHANTZ, R. E., ZINKY, J. A., KARR, D. A., VANEGAS, R., AND ANDERSON, K. R. QoS aspect languages and their runtime integration. In *Proceedings of the 4th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR)* (1998), Springer-Verlag.

[27] MACLEAN, A., CARTER, K., LOVSTRAND, L., AND MORAN, T. User-tailorable systems: pressing the issues with buttons. In *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 1990), ACM Press, pp. 175–182.

[28] MCKUSICK, M., BOSTIC, K., KARELS, M., AND QUARTERMAN, J. *The Design and Implementation of the 4.4BSD Operating System*. Addison-Wesley Longman, 1996.

[29] NAGARAJA, K., OLIVEIRA, F., BIANCHINI, R., MARTIN, R., AND NGUYEN, T. Understanding and dealing with operator mistakes in internet services. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)* (December 2004).

[30] NELSON, M., LIM, B.-H., AND HUTCHINS, G. Fast transparent migration for virtual machines. In *Proceedings of the USENIX Annual Technical Conference* (2005).

[31] NIEH, J., AND LAM, M. The design, implementation, and evaluation of SMART: A scheduler for multimedia applications. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles* (October 1997).

[32] NOBLE, B. D., SATYANARAYANAN, M., NARAYANAN, D., TILTON, J. E., FLINN, J., AND WALKER, K. R. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles* (1997).

[33] RAJKUMAR, R., LEE, C., LEHOCZKY, J., AND SIEWIOREK, D. A resource allocation model for QoS management. In *Proceedings of the IEEE Real-Time Systems Symposium* (December 1997).

[34] REYNOLDS, C. J. The sensing and measurement of frustration with computers. Master's thesis, Massachusetts Institute of Technology Media Laboratory, 2001.

[35] RICHARDSON, T., STAFFORD-FRASER, Q., WOOD, K., AND HOPPER, A. Virtual network computing. *IEEE Internet Computing 2*, 1 (January/February 1998).

[36] ROMANO, P. Itu-t recommendation t.128 (application sharing). Tech. rep., ITU, March 1997.

[37] SCHMIDT, B., LAM, M., AND NORTHCUTT, J. The interactive performance of slim: A stateless thin client architecture. In *Proceedigns of the 17th ACM Symposium on Operating Systems Principles (SOSP 1999)* (December 1999), pp. 32–47.

[38] SHOYKHET, A., LANGE, J., AND DINDA, P. Virtuoso: A system for virtual machine marketplaces. Tech. Rep. NWU-CS-04-39, Department of Computer Science, Northwestern University, July 2004.

[39] STRICKER, L. J. The true deceiver. *Psychological Bulletin*, 68 (1967), 13–20.

[40] STRICKLAND, J., FREEH, V., MA, X., AND VAZHKUDAI, S. Governor: Autonomic throttling for aggressive idle resource scavenging. In *Proceedings of the 2nd IEEE International Conference on Autonomic Computing (ICAC)* (June 2005).

[41] SUNDARARAJ, A., AND DINDA, P. Towards virtual networks for virtual machine grid computing. In *Proceedings of the 3rd USENIX Virtual Machine Research And Technology Symposium (VM 2004)* (May 2004).

[42] SUNDARARAJ, A., GUPTA, A., , AND DINDA, P. Increasing application performance in virtual environments through run-time inference and adaptation. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC)* (July 2005), pp. 47–58.

[43] SUNDARARAJ, A., SANGHI, M., LANGE, J., AND DINDA, P. An optimization problem in adaptive virtual environments. In *Proceedings of the Seventh Workshop on Mathematical Performance Modeling and Analysis (MAMA)* (June 2005).

[44] WALDSPURGER, C. A., AND WEIHL, W. E. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the First Symposium on Operating Systems Design and Implementation* (1994).

[45] WHITAKER, A., COX, R., AND GRIBBLE, S. Configuration debugging as search: Finding the needle in the haystack. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)* (December 2004).

[46] ZINKY, J. A., BAKKEN, D. E., AND SCHANTZ, R. E. Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems 3*, 1 (April 1997), 55–73.