NORTHWESTERN
UNIVERSITY

Electrical Engineering and Computer Science Department

Technical Report
NWU-EECS-07-04
July 26, 2007

Human-driven Optimization

Bin Lin

Abstract

The optimization problems associated with adaptive and autonomic computing systems are often difficult to pose well and solve efficiently. A key challenge is that for many applications, particularly interactive applications, the user or developer is unlikely or unable to provide either the objective function *f*, or constraints. It is a key problem encountered broadly in adaptive and autonomic computing.

This dissertation argues for using **human-driven optimization** techniques to solve optimization problems. In particular, it consists of two core ideas. In **human-driven specification**, we use direct human input from users to pose specific optimization problems, namely to determine the objective function *f* and expose hidden constraints. Once we have a well-specified problem, we are left with the need to search for a solution in a very large solution space. In **human-driven search**, we use direct human input to guide the search for a good solution, a valid configuration x that optimizes *f(x)*.

My research happens in three contexts. The main context is the Virtuoso system for utility and grid computing based on virtual machines (VMs) interconnected with overlay networks. Virtuoso provides instances of optimization problems and a framework for evaluating solutions to them. In particular, the virtual execution environment of Virtuoso makes possible low-level, application- and developer-independent adaptation mechanisms such as CPU reservations, VM migration, overlay topology configuration and routing, and network reservations. The high-level optimization problem in Virtuoso is how to dynamically optimize, at run-time, the performance of existing, unmodified distributed applications running on existing, unmodified operating systems. These applications can be batch, batch parallel and interactive applications. The second context of my research is power management for laptops. Existing Dynamic voltage and Frequency Scaling (DVFS) techniques are commonly used to reduce power consumption

but they are conservative and pessimistic about both the user and the processor. How to further prolong battery life and reduce heat dissipation is the problem that is addressed. The last context of my research is IT configuration, a process whereby individual components are assembled and adjusted to construct a working solution. Visible complexity—of setting configuration knobs, installing and updating software, diagnosing and repairing problems, and so on—is a challenge for IT. How to reduce such complexity is a challenging problem. As the first step, I explored the decision complexity presented to the non-expert system administrator, which represents a significant part of the whole IT complexity.

To show the feasibility and effectiveness of my techniques, in this dissertation, I describe how I address increasingly difficult optimization problems in the Virtuoso context using human-driven specification or search. Those problems cover single machine CPU scheduling, multiple machine CPU scheduling, and multiple machine VM mapping for interactive (desktop), batch and batch parallel applications. I also present how I apply human-driven techniques to solving power management problems on laptop computers. In general, solving each of these problems involves the design and development of systems mechanisms, adaptive algorithms and user interfaces. I evaluate each element of my work through a user study. I also discuss my work on modeling user decision complexity in IT configuration systems, as the first step towards applying human-driven techniques in that domain.

This work is in part supported by the NSF (awards ANI-0093221, ANI-0301108, and EIA-0224449), the DOE via ORNL, and by gifts from VMware, Dell, and Symantec.

Keywords: Adaptive systems; Human-computer interaction; Resource virtualization; Power management

NORTHWESTERN UNIVERSITY

Human-driven Optimization

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Bin Lin

EVANSTON, ILLINOIS

December 2007

ABSTRACT

Human-driven Optimization

Bin Lin

The optimization problems associated with adaptive and autonomic computing systems are often difficult to pose well and solve efficiently. A key challenge is that for many applications, particularly interactive applications, the user or developer is unlikely or unable to provide either the objective function f, or constraints. It is a key problem encountered broadly in adaptive and autonomic computing.

This dissertation argues for using **human-driven optimization** techniques to solve optimization problems. In particular, it consists of two core ideas. In **human-driven specification**, we use direct human input from users to pose specific optimization problems, namely to determine the objective function f and expose hidden constraints. Once we have a well-specified problem, we are left with the need to search for a solution in a very large solution space. In **human-driven search**, we use direct human input to guide the search for a good solution, a valid configuration x that optimizes $f(x)$.

My research happens in three contexts. The main context is the Virtuoso system for utility and grid computing based on virtual machines (VMs) interconnected with overlay networks. Virtuoso provides instances of optimization problems and a framework for evaluating solutions to them. In particular, the virtual execution environment of Virtuoso makes possible low-level, application- and developer-independent adaptation mechanisms such as CPU reservations, VM migration, overlay topology configuration and routing, and network

reservations. The high-level optimization problem in Virtuoso is how to dynamically optimize, at run-time, the performance of existing, unmodified distributed applications running on existing, unmodified operating systems. These applications can be batch, batch parallel and interactive applications. The second context of my research is power management for laptops. Existing Dynamic voltage and Frequency Scaling (DVFS) techniques are commonly used to reduce power consumption but they are conservative and pessimistic about both the user and the processor. How to further prolong battery life and reduce heat dissipation is the problem that is addressed. The last context of my research is IT configuration, a process whereby individual components are assembled and adjusted to construct a working solution. Visible complexity—of setting configuration knobs, installing and updating software, diagnosing and repairing problems, and so on—is a challenge for IT. How to reduce such complexity is a challenging problem. As the first step, I explored the decision complexity presented to the non-expert system administrator, which represents a significant part of the whole IT complexity.

To show the feasibility and effectiveness of my techniques, in this dissertation, I describe how I address increasingly difficult optimization problems in the Virtuoso context using human-driven specification or search. Those problems cover single machine CPU scheduling, multiple machine CPU scheduling, and multiple machine VM mapping for interactive (desktop), batch and batch parallel applications. I also present how I apply human-driven techniques to solving power management problems on laptop computers. In general, solving each of these problems involves the design and development of systems mechanisms, adaptive algorithms and user interfaces. I evaluate each element of my work through a user study. I also discuss my work on modeling user decision complexity in IT configuration systems, as the first step towards applying human-driven techniques in that domain.

Thesis Committee

Peter A. Dinda, Northwestern University, Committee Chair

Fabián E. Bustamante, Northwestern University

Bruce Gooch, Northwestern University

Aaron B. Brown, IBM

To my parents, Jiangying Li and Bizhi Lin

Acknowledgments

First of all, I would like to thank my advisor, Professor Peter A. Dinda, for his support throughout my Ph.D. study. His insights, inspiration and guidance have helped me find my own path and overcome many obstacles in this journey. He always suggests that I think broadly and independently, and encourages me to pursue what I am interested in. I have learned a lot about teaching, researching and thinking from him. I aspire to be as good an researcher as he is.

Second, I want to thank the rest of my thesis committee, Fabián E. Bustamante, Aaron B. Brown and Bruce Gooch for their encouragement and support. Fabián's singular ability to ask embarrassing questions and an insistence on clear explanations of complex ideas greatly helped me to refine my work. When I was an intern in the IBM T.J. Watson Research Center, Aaron provided me with wise counsel. I want to thank him for his continuous support beyond my internship. Bruce gave insightful comments since my thesis proposal from a perspective of a graphics and HCI researcher.

I would like to acknowledge that parts of this work are joint with others. The user comfort study was a joint work with Ashish Gupta. Don Norman gave very valuable comments in the beginning of that project. In the button-driven scheduling of VMs, Dong Lu contributed to the model of describing a CPU intensive process's performance as a function of its nice value. The first generation of the application trace-driven simulator was developed by Ananth I. Sundararaj. He also participated in the verification and validation of that version of the simulator. The user- and process-driven power management research is joint work with Arindam Mallik, Gokhan Memik and Robert P. Dick. Arindam focused more on the process-driven aspect in that project. The decision complexity work was done during my internship in IBM T.J. Watson Research Center. I would like to thank Joseph

L. Hellerstein for guidance and advice in valuable weekly meetings when I was in IBM. I want to thank my previous IBM colleague, John C. Thomas, Christopher Ward, Melissa J. Buco and Heiko Ludwig, for very useful suggestions and also conversation during daily lunches. John also gave helpful comments on my work on the direct user control of CPU scheduling.

I am grateful to the participants in all of our studies. In particular, I want to thank Conrad Albrecht-Buehler for interesting and intellectually stimulating discussions about user interfaces.

On a more personal note, I am deeply appreciative for the support and encouragement of all of my friends, especially Ananth I. Sundararaj. Additionally, I want to thank my previous office mate Pinku Surana and current office mate Ashish Gupta for many interesting conversations over the years.

My parents have been an vital support throughout my Ph.D. studies including the completion of this dissertation.

Contents

List of Figures **16**

List of Tables **22**

1 Introduction **23**

1.1 Problem space and assumptions 24

1.2 Contexts 25

1.2.1 Virtual machines and Virtuoso 25

1.2.2 Power management 28

1.2.3 IT configuration complexity 29

1.3 Challenges in user interfaces 30

1.4 Outline of dissertation 31

2 Measuring and Understanding User Comfort With Resource Borrowing **35**

2.1 System design 38

2.1.1 Testcases and exercise functions 39

2.1.2 Resource exercisers 40

2.1.3 Testcase execution and system monitoring 42

2.1.4 Client interface 43

2.2 Controlled study 45

2.2.1 Perspective of the user 45
2.2.2 Testcase details 46
2.2.3 Results 49
2.3 Advice to implementors 57
2.4 Conclusions 57

3 User-driven Scheduling of Interactive Virtual Machines 60
3.1 Mechanism 63
3.1.1 Overall framework 63
3.1.2 Control mechanism 65
3.1.3 Control model 67
3.1.4 Control algorithm 68
3.2 Experiments 74
3.2.1 Experimental setup 74
3.2.2 Metrics 75
3.2.3 Experimental results and analysis 75
3.3 Observations 80
3.4 Conclusions 80

4 Putting the User in Direct Control of Scheduling His Interactive Virtual Machine 82
4.1 VSched 83
4.1.1 Abstraction 83
4.1.2 Type-II versus type-I VMMs 85
4.2 System design 86
4.2.1 Algorithms 86
4.2.2 Mechanisms 87

4.2.3 Structure 89
4.3 Evaluation 92
4.3.1 Methodology 92
4.3.2 Randomized study 93
4.3.3 Deterministic study 93
4.3.4 I/O 98
4.4 Mixing batch and interactive VMs 99
4.5 Summary 101
4.6 User interface 102
4.7 User study 104
4.7.1 Particulars 104
4.7.2 Process 105
4.7.3 Qualitative Results 108
4.7.4 Quantitative Results 115
4.8 Conclusions 118

5 Time-sharing Parallel Applications With Performance Isolation and Control 120
5.1 Motivation 120
5.2 Global controller 123
5.2.1 Inputs 124
5.2.2 Control algorithm 124
5.3 Evaluation 126
5.3.1 Experimental framework 126
5.3.2 Range of control 127
5.3.3 Schedule selection and drift 128
5.3.4 Evaluating the control algorithm 130

5.3.5 Summary of limits of control algorithm 131
5.3.6 Dynamic target execution rates 132
5.3.7 Ignoring external load 133
5.3.8 NAS IS Benchmark 134
5.3.9 Time-sharing multiple applications 135
5.3.10 Effects of local disk I/O 138
5.3.11 Effects of physical memory use 140
5.4 Conclusions 141

6 Application Trace-driven Simulator for Virtuoso 142
6.1 First generation simulator 144
6.1.1 Input 145
6.1.2 Output 148
6.1.3 Data structures 149
6.1.4 Basic operations 149
6.1.5 Verification and validation 151
6.2 Second generation simulator 153
6.2.1 Periodic real-time scheduling model 153
6.2.2 VM migration 156
6.2.3 User input to the simulator 156
6.2.4 Verification and validation 157
6.3 Summary 159

7 Direct User Control of Scheduling and Mapping of Virtual Machines 160
7.1 Specific problem 160
7.2 Interface design 162
7.3 Game design 164

7.4 Configuration 166
7.5 User study 166
7.5.1 Testcases and process 167
7.5.2 Results 169
7.6 Summary of the VM scheduling game 172
7.7 Issues in interface 173
7.8 Interface design revisit 174
7.8.1 Game design revisit 177
7.9 User study 178
7.9.1 Testcases and process 179
7.9.2 Results 182
7.10 Conclusions 186

8 User- and Process-Driven Dynamic Voltage and Frequency Scaling 189
8.1 User-driven frequency scaling 192
8.1.1 Pessimism about the user 193
8.1.2 Technique 195
8.2 Process-driven voltage scaling 200
8.2.1 Pessimism about the CPU 200
8.2.2 Technique 203
8.3 Evaluation 204
8.3.1 UDFS 205
8.3.2 PDVS 206
8.3.3 UDFS+PDVS (CPU dynamic power, trace-driven simulation) . . 208
8.3.4 UDFS+PDVS (System power and temperature measurement) . . 212
8.3.5 Discussion 215

8.4 Conclusions 219

9 Related work **222**

9.1 Users and direct user input 222

9.2 Human decision making 223

9.3 Service-level agreements (SLAs) 224

9.4 Process and VM scheduling 225

9.5 Real-time scheduling 226

9.6 Scheduling parallel applications 227

9.7 Feedback-based control 228

9.8 Dynamic voltage and frequency scaling 228

9.9 Dynamic thermal management 230

9.10 Games with a purpose 230

10 Conclusions **231**

10.1 Summary of contributions 233

10.2 Future work 238

10.2.1 Power-aware multi-core scheduling 238

10.2.2 Optimization problems in wide-area distributed systems 239

10.2.3 Online games for solving optimization problems 240

Bibliography **241**

Appendices 259

A Towards an Understanding of Decision Complexity in IT Configuration **260**

A.1 Introduction 260

A.2 Complexity model for experts 263
A.3 Model and hypothesis 264
A.4 Approach 267
A.5 User study design 268
A.5.1 Experiment and test cases 268
A.5.2 Perspective of the user 271
A.5.3 Implementation 273
A.5.4 Two-stage User Study 274
A.6 Results and Analysis 275
A.6.1 Metrics 275
A.6.2 Qualitative results 275
A.6.3 Quantitative results 277
A.6.4 Summary and advice to designers 281
A.7 Next Steps 282
A.8 Conclusions 285

B User Study Instructions for Chapter 2 **286**

C User Study Instructions for Chapter 4 **290**

D User Study Instructions for Chapter 7 **298**
D.1 Instructions for VM scheduling game 298
D.2 Instructions for VM scheduling & mapping game 303

E User Study Instructions for Chapter 8 **313**

List of Figures

2.1 Structure of UUCS. 38
2.2 The Analysis Phase. 38
2.3 Testcases. 40
2.4 Step and ramp testcases. 41
2.5 Client design. 41
2.6 Client interface. The menu and full application interface can be disabled. 44
2.7 Machine configuration. 45
2.8 Testcase descriptions for the 4 tasks (given in random order). 47
2.9 Breakdown of runs. 48
2.10 CDF of discomfort for CPU. 49
2.11 CDF of discomfort for Memory. 51
2.12 CDF of discomfort for Disk. 51
2.13 User sensitivity by task and resource. 52
2.14 f_d by task and resource. 52
2.15 $c_{0.05}$ by task and resource. (* indicates insufficient information) 53
2.16 c_a by task and resource, including 95% confidence intervals. (* indicates insufficient information) 53
2.17 Significant differences based on user-perceived skill level. 55
2.18 CDFs by resource and task. 59

3.1 Structure of UVMS. 63
3.2 Client interface. 64
3.3 Nice control experiment 1. 66
3.4 Nice control experiment 2. 66
3.5 Model evaluation experiment I. 69
3.6 Model evaluation experiment II. 70
3.7 TCP Reno-like control algorithm for VM priority. 71
3.8 Linearization. 72
3.9 Linearization. 73
3.10 Experiment I: nonlinear control scheme. 76
3.11 Experiment II: linear control scheme. 76
3.12 Experiment III: adaptive control scheme. 76
3.13 Comparison of compute time extracted by a batch VM and button press interarrival times for different algorithms. 79

4.1 Structure of VSched. 89
4.2 A detailed VSched schedule for three VMs. 91
4.3 Testbed Machines 91
4.4 Evaluation scenarios. 92
4.5 Miss rate as a function of utilization, Random study on Machine 1 (2 GHz P4, 2.4 kernel). 94
4.6 Distribution of missed percentage of slice; Random study on Machine 1 (2 GHz P4, 2.4 kernel). 95
4.7 Miss rate as a function of period and slice for Machine 1 (2 GHz P4, 2.4 kernel). 96

4.8 Distribution of miss times when utilization is exceeded for Machine 1 (2 GHz P4, 2.4 kernel). 97
4.9 Summary of performance limits on three platforms. 97
4.10 Performance of time-sensitive I/O (ripping an Audio CD) (2 GHz P4, 512MB RAM, 2.6 kernel, Mandrake Linux). 98
4.11 Summary of qualitative observations from running various interactive applications in an Windows VM with varying period and slice. The machine is also running a batch VM simultaneously with a (10 min, 1 min) constraint. 99
4.12 Control interface. The combination of (a) and (c) is used in our study. . . 102
4.13 User A: Tracks, cost versus time. 109
4.14 User B: Tracks, cost versus time. 111
4.15 User C: Tracks, cost versus time. 112
4.16 User C: Cost, efficiency versus time. 113
4.17 Summary of user responses in study. 114
4.18 Statistics of the lowest costs reported by users in study. 116
4.19 Duration to lowest cost. 117

5.1 Structure of global control. 123
5.2 Compute rate as a function of utilization for different $(period, slice)$ choices. 127
5.3 Elimination of drift using global feedback control; 1:1 comp/comm ratio. 129
5.4 System in stable configuration for varying comp/comm ratio. 129
5.5 System in oscillation when error threshold is made too small; 1:1 comp/comm ratio. 131
5.6 Response time of control algorithm; 1:1 comp/comm ratio. 132
5.7 Response time and threshold limits for the control algorithm. 132

5.8 Dynamically varying execution rates; 1:1comp/comm ratio. 133
5.9 Performance of control system under external load; 3:1 comp/comm ratio; 3% threshold. 134
5.10 Running NAS benchmark under control system; 3% threshold. 135
5.11 Running of two Patterns benchmarks under the control system, 1:1 comp/comm ratio. 136
5.12 Running multiple Patterns benchmarks; 3:1 comp/comm ratio; 3% threshold. 137
5.13 Performance of control system with a high (145:1) comp/comm ratio and varying local disk I/O. 138
5.14 Performance of control system with 10 MB/node/iter of disk I/O and varying comp/comm ratios. 139
5.15 Running two Patterns benchmarks under the control system; high (130:1) comp/comm ratio. The combined working set size is slightly less than the physical memory. 140

6.1 Structure of the simulator 144
6.2 Sample input application trace, 3 nodes, Patterns benchmark, bus topology, 3 iterations. 146
6.3 XPVM output for Patterns benchmark executing on 3 hosts. 147
6.4 The simulator core. 150
6.5 The check_vsched function 154

7.1 Initial interface for VM scheduling game 163
7.2 User background versus average global efficiency 170
7.3 User self-evaluation versus average global efficiency 171

7.4 Interface for CPU scheduling & VM mapping game; the user is trying to migrate the ball in the left-most resource box to the second resource box to the left. 175
7.5 Final screen 178
7.6 User performance 183
7.7 User performance and user background 184
7.8 Percentage of users who find the optimal mapping; 95% confidence interval. 185
7.9 Duration to the optimal mapping. 186

8.1 User pessimism. 194
8.2 The frequency for UDFS schemes during FIFA game for a representative user. 198
8.3 Minimum stable V_{dd} for different operating frequencies and temperatures. 202
8.4 Frequency over time for UDFS1 and UDFS2, aggregated over 20 users. . 207
8.5 Power reduction for Windows DVFS and DVFS+PDVS 208
8.6 Comparison of UDFS algorithms, UDFS+PDVS, and Windows XP DVFS (CPU Dynamic Power). Chebyshev bound-based $(1-p)$ values for difference of means from zero are also shown. 210
8.7 Comparison of UDFS algorithms, UDFS+PDVS, and Windows XP DVFS (measured system power with display off). Chebyshev bound-based $(1-p)$ values for difference of means from zero are also shown. 213
8.8 Mean and peak temperature measurement. 216
8.9 Average number of user events. 216
8.10 Number of voltage transitions 218
8.11 Power improvement in the multitasking environment. Chebyshev bound-based $(1-p)$ values for difference of means from zero are also shown. . 218

A.1 The screen-shot of a running testcase. 269
A.2 User rating and time; Avg Std for time over all testcases: 4368 milliseconds 275
A.3 Error rate and time; Avg Std for time over all testcases: 4368 milliseconds 276
A.4 Mapping 282
A.5 Steps 283

B.1 Tray interface. 286

D.1 Game interface. 299
D.2 Game interface. 304
D.3 Migration. 306
D.4 final screen. 309

E.1 Tray interface. 313

List of Tables

1.1 Problem space 25

7.1 User background 178
7.2 Relationship between user self-rating and his score 186

A.1 High-level model of decision making 264
A.2 Sub-factors within guidance 265
A.3 Route planning domain based on the model 265
A.4 Summary of test cases; a $\times$ means the parameter is not presented while a check means the opposite. 270
A.5 Summary of complexity factors 274
A.6 Baseline analysis of variability for time 277
A.7 Analysis of complexity factors for time 278
A.8 Pair-wise test for time 279
A.9 Baseline analysis of variability for rating 280
A.10 Analysis of complexity factors for rating 280
A.11 Pair-wise test for rating 281

Chapter 1

Introduction

The optimization problems associated with adaptive and autonomic computing systems are often difficult to pose well and solve efficiently. This dissertation explores using direct human input to solve optimization problems. The core of the dissertation describes the design, implementation, and evaluation of system mechanisms, adaptive algorithms and user interfaces to support and use direct human input.

Optimization is an activity that aims at finding, ideally, the best (i.e., optimal) solution to a problem, and at worst a high quality solution. For optimization to be meaningful there must exist an objective function f to be optimized and at least one feasible solution x, i.e., a solution which does not violate the constraints.

Before we can solve the optimization problem, however, we need a clear statement of it. A key challenge is that for many applications, particularly interactive applications, the user or developer is unlikely or unable to provide either the objective function f, or the constraints. This is a fundemental problem encountered broadly in adaptive and autonomic computing.

In my thesis, I explored **human-driven optimization**. It consists of two ideas. In **human-driven specification**, I explored how to *use direct human input from users to pose specific problems, namely to determine the objective function f and the constraints*. Once

we have a well specified problem, we are left with the need to search for a solution in a very large solution space. In **human-driven search**, I explored how to *use direct human input to guide the search for a good solution, a valid configuration x that optimizes $f(x)$*.

My work takes place in three contexts. The main context is Virtuoso, a system for utility and grid computing that is based on virtual machines (VMs) interconnected with overlay networks. I will demonstrate the feasibility of my human-driven techniques in Virtuoso (Chapter 3, 4, 5, 6, 7). In addition, I will show that my approach can be used to solve optimization problems in the second context, power management for laptops, demonstrating CPU power reductions of $>$ 20% (Chapter 8). The last context of my work is IT configuration systems. As the first step towards applying human-driven techniques on IT domain, I will discuss how we model user decision complexity in IT configuration (Appendix A).

1.1 Problem space and assumptions

For the objective functions and constraints of an optimization problem, I assume that the application's user or system administrator knows about them, at least implicitly. I believe this must be true for an interesting range of applications, as does the wider community of researchers in adaptive and autonomic computing.

In terms of the search for an appropriate configuration, there are two possible approaches: automatic search and manual search. I consider both when using a human input method. Table 1.1 highlights the specific part of the problem space I explored.

In the remainder of this chapter, I will describe the context of my work in Virtuoso and adaptation mechanisms. I will also introduce power management problems and the concept of IT configuration complexity. An outline of this dissertation can be found in the end of this chapter.

	Known formal objective function and constraints	Informal objective function and constraints derived from direct human input
Automatic search	My colleague Ananth I. Sundararaj's thesis explored this.	My thesis (human-driven specification) (Chapter 2, 3, 5, 8)
Manual search	My thesis (human-driven search) (Chapter 7)	My thesis (human-driven search) (Chapter 4)

Table 1.1: Problem space

1.2 Contexts

1.2.1 Virtual machines and Virtuoso

In general, systems that simulate multiple copies of a machine on itself and add an indirection layer between software and physical hardware are called virtual machine systems. The simulated machines are called the virtual machines (VMs) [70]. The indirection layer is called the virtual machine monitor (VMM). Virtual machines provide many benefits, such as the capability of multiplexing several VMs on a single physical resource, isolation, and security.

Virtuoso is middleware for virtual machine distributed computing that very closely emulates the process of buying, configuring, and using an Intel-based computer or collection of computers from a web site, a process with which many users and certainly all system administrators are familiar. Instead of a physical computer, 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 in terms of console display, devices, and the network. Virtuoso currently uses VMware GSX Server, a type-II virtual machine [70], as its virtual machine monitor, though other VMMs can in principle be substituted, and our model could easily be employed in a type-I VMM.

Each host computer in Virtuoso can support multiple VMs, each of which can run a

different operating system. VMs communicate using a custom overlay network [180, 182] that infers CPU and network demands and measures the underlying network using the VMs naturally occurring traffic. Virtuoso is intended to support a wide range of uses from interactive desktop users to batch parallel users. It provides many adaptation and resource reservation mechanisms that can be applied to an existing, unmodified application and operating system running in a collection of VMs:

- Mapping of VMs to hosts and migration of VMs [180]
- Overlay topology [180]
- Forwarding on the overlay topology [180]
- CPU reservations [109]
- Network reservations [105]

Details about the Virtuoso implementation [164], its virtual networking system [182], its application topology inference system [74], its integration with the Wren network inference system [77], its dynamic adaptation system [180, 183], and its optical network reservation system [105] can be found in the references, as can a detailed case for grid computing on virtual machines [58], a more recent discussion of the role of VMs in this area [99], and an introduction to the state of the art in virtualization [57].

At a high level, Virtuoso seeks to use inferred information about the application, network, and hosts to engage the adaptation and reservation mechanisms in order to increase the application's performance. Generally, this is a challenging, NP-hard optimization problem [184]. My fellow student Ananth I. Sundararaj has a complete formalization as a part of his thesis [181]. I use the formalization where necessary, although for most problems that I address, formalization is not known.

Here is the Virtuoso optimization problem that I am addressing in this dissertation.

I seek to find a **configuration** x such that an **objective function** $f(x)$ is maximized while x also obeys a set of given **constraints**. I consider a configuration consists of

- a local periodic real-time schedule of VMs on each host,
- a mapping from VMs to hosts,

The constraints determine what valid configurations are and include

- schedulability (e.g., there must exist a feasible schedule on each host),
- 0/1 constraints (e.g., a VM must be mapped onto exactly one host),
- placement (e.g., some VM may have to be placed on a specific host), and
- **hidden constraints** (e.g., user specifies that some VM must be specifically scheduled).

The 0/1 constraints play a key role in making variants of this problem NP-hard.

Virtuoso is designed to support a wide range of workloads that its simple user-level abstraction makes possible. My work will focus on three workload types:

- Interactive workloads which occur when using a remote VM to substitute for a desktop computer. These workloads include desktop applications, web applications and games.[1]
- Batch workloads, such as scientific simulations or analysis codes. These workloads are commonplace in grid computing [60].

[1]It is debatable to what extent a remote VM could replace a desktop and what the permissible limits on the latency from the VM to the client are, but there are certainly a wide range of interactive applications which can be successfully used remotely using modern display techniques. For example, Lai and Neih demonstrated successful use of thin clients for several desktop applications, including video, despite > 1000 miles between client and server [104].

- Batch parallel workloads, such as scientific simulations or analysis codes that can be scaled by adding more VMs. These are also commonplace in grid computing.

Adaptation mechanisms

In an effort to adapt applications running in distributed computing environment to the dynamically changing computational and networking resources to achieve high performance, there have been numerous attempts in different settings such as load balancing in networks of shared processors [38, 79, 201], solutions to workflow problems, component placement problems and support for heavyweight applications in computational grids [12, 100, 122], adaptation, load balancing and fault tolerance in message passing and parallel processing systems spread over heterogeneous resources [4, 73, 126, 165], distributed mobile applications [142], automated runtime tuning systems [23] and extensions to commercial standards such as QuoIN/CORBA [208]. However most of the approaches are very application-specific and require considerable user or developer effort.

1.2.2 Power management

Research on power management for laptops and embedded systems to prolong battery life and reduce heat dissipation has attracted attention from both micro-architecture and systems communities for years. Dynamic Voltage and Frequency Scaling (DVFS) is one of the most commonly used power reduction techniques in high-performance processors and is the most important OS power management tool. DVFS varies the frequency and voltage of a microprocessor in real-time according to processing needs. Although there are different versions of DVFS, at its core DVFS adapts power consumption and performance to the current workload of the CPU. Specifically, existing DVFS techniques in high-performance processors select an operating point (CPU frequency and voltage) based on the utilization of the processor. While this approach can integrate information available to the OS kernel,

such control is conservative and pessimistic about both the user and processor. A core question is how low can CPU frequency be without annoying the user? In Chapter 8, we will present our work on user- and process- driven dynamic voltage and frequency scaling techniques.

1.2.3 IT configuration complexity

A significant fraction of modern systems management revolves around *configuration*, a process whereby individual components are assembled and adjusted to construct a working solution. Human-driven configuration procedures occur at every stage in the lifecycle of hardware and software artifacts, from installation and deployment of new system components, to ongoing maintenance and tuning of existing components, to upgrading, replacing, or decommissioning obsolete components. For many systems, these human-driven configuration procedures represent a significant operational cost, often dominating total cost of ownership. The optimization problem here is to optimize the simplicity of managing the systems, where the objective function is dependent on the system administrator performing management tasks. To address this, new model and innovative approach must be adopted.

A key complexity challenge lies in improving the experience of the non-expert system administrator—the person providing IT support in a small-business environment, who is confronted by decisions during the configuration process. **Decision complexity** is the complexity of figuring out for the first time what steps to follow and what decisions to make while performing a complex configuration process. In Appendix A, I will present my work on understanding decision complexity in IT configuration.

1.3 Challenges in user interfaces

Applying my concept of direct human input to the optimization problems in the above contexts poses the following challenges.

Frequency of input: In each of these contexts, I will show in later chapters 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 my view, sensible low-frequency input will take one of three forms:

1. Evaluations of a current configuration, resulting in user-specific utility functions.
2. Specifications of configurations.
3. Directions for search within a space of configurations.

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.

Mechanism transition: In Virtuoso, 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 [34], 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 can be thought of as a point within a multi-dimensional space. If a dimension is categorical (for example, a VM can be mapped to one of several choices), it is difficult to present it using an easily understood external interface.

Dimensionality: It is relatively easy to map two dimensions of a control directly to two dimensions of an interface, 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 and its neighborhood.

Service-level agreements (SLAs) Interactive applications used by ordinary end-users are migrating to service-oriented systems that span shared server resources. One example is the move to "software as a service" (SAAS), leveraging Web 2.0 technologies like AJAX. Services like docs.google.com and salesforce.com are well known. A second example is virtual machine-based desktop-replacement computing, as in the Collective [26], Internet Suspend/Resume [160], and our Virtuoso system [164].

Service-oriented systems must schedule resources such as the CPU so that the relevant service-level agreements (SLAs) are honored. 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?

1.4 Outline of dissertation

This section describes the overall organization of the dissertation.

Chapter 2 presents a controlled user study of the effects of resource contention on the comfort of the users of typical interactive desktop applications on Microsoft Windows. We discovered considerable diversity in the tolerance that users have for resource contention. This diversity argues for per-user tailoring of utility functions, and motivates later on work on using direct user input. The study further provides evidence for the feasibility of human-driven specification through a button-press feedback mechanism.

Chapter 3 describes my work on scheduling an interactive VM through simple button-press feedback from the user. We show that using direct user feedback, it is possible to balance between providing high average computation rates to the non-interactive VMs while keeping the users of the interactive VMs happy. This work provides evidence for the feasibility of using human-driven specification to solve CPU scheduling problem on a single machine.

Chapter 4 presents the design, implementation and evaluation of VSched - a periodic real-time scheduler, and shows that we can use it to schedule both batch and interactive VMs. The tool further enables the design of a joystick input interface which allows the user to directly control the CPU schedule of his VM. The two dimensions (period, slice) of CPU schedule is mapped directly to the two dimensions of the joystick. Our user study showed that even a naive user can use the interface, combined with an on-screen display of VM cost, to quickly balance between the comfort of the environment and its cost. The results show the feasibility of using human-driven search to solve CPU scheduling problem on a single machine.

In Chapter 5, we apply a new approach to time-sharing parallel applications with performance isolation. based on local VScheds combined with a global feedback controller. It provides a simple way for the user/administrator to control execution rate of applications while maintaining efficiency. The work demonstrates that it is feasible to apply human-driven specification to CPU scheduling problem on multiple machines.

In Chapter 6, we describe the design of an application trace-driven simulator to facilitate the research on human-driven control of a collection of VMs.

In Chapter 7, we discuss the design, implementation and evaluation of a game-like interface for a user to easily control and optimize both the CPU schedules of his VMs and the mapping of VMs to hosts. We evaluated the interface, which is connected with the simulator, through extensive user studies. The results show that it is feasible to put the naive user in direct control of both CPU scheduling and VM mapping of a collection of VMs. It provides evidence for human-driven search. It further explored the idea of solving optimization problems through game-play by common people.

Chapter 8 summarizes two new, independently-applicable power reduction techniques for power management on processors that support dynamic voltage and frequency scaling (DVFS). In PDVS (process-driven voltage scaling), a CPU-customized profile is derived offline that encodes the minimum voltage needed to achieve stability at each combination of CPU frequency and temperature. In UDFS (user-driven frequency scaling), on the other hand, dynamically adapts CPU frequency to the individual user and the workload through direct user feedback. Combining PDVS and the best UDFS scheme reduces measured system power by 49.9% (27.8% PDVS, 22.1% UDFS), averaged across all our users and applications, compared to Windows XP DVFS. UDFS provides evidence for applying human-driven specification to solving power management problems.

Chapter 9 discusses related work to this dissertation.

Finally, in Chapter 10, we summarize the conclusions of this dissertation and highlight opportunities for additional research.

Appendix A presents our work on modeling decision complexity in IT configuration systems. Decision complexity is the complexity faced by a non-expert system administrator—the person providing IT support in a small-business environment, who is confronted by decisions during the configuration process, and is a measure of how easy or hard it is to

identify the appropriate sequence of configuration actions to perform in order to achieve a specified configuration goal. As the first step towards a complete model of decision complexity, we conducted an extensive user study of decision making in a carefully-mapped analogous domain (route planning), and showed how the results of that study suggest an initial model of decision complexity applicable to IT configuration. The model identifies the key factors affecting decision complexity and highlights several interesting results, including the fact that decision complexity has significantly different impacts on user-perceived difficulty than on objective measures like time and error rate.

Appendix B, C, D and E provide written protocols and forms used in our user studies discussed in Chapter 2, 4, 7 and 8, respectively.

Chapter 2

Measuring and Understanding User Comfort With Resource Borrowing

This dissertation argues for using direct human input to solve optimization problems. In this chapter, we describe the design and development of a sophisticated distributed application for directly measuring user comfort with the borrowing of CPU time, memory space, and disk bandwidth. Using this tool, we conducted a controlled user study with qualitative and quantitative results that are of direct interest to the designers of grid and thin-client systems. We found that resource borrowing can be quite aggressive without creating user discomfort, particularly in the case of memory and disk. The results of the study further revealed an important fact that resources needed to keep the user happy are highly dependent on the user as well as the application. In other words, the traditional assumption that systems can optimize for a canonical user is invalid. The successful use of direct user feedback in this work provides evidence for the feasibility of human-driven specification.

Many widely used distributed computing platforms and applications exploit available resources on existing host computers, exploiting the fact that most of these systems are dramatically under-utilized [3, 42, 138], a technique we refer to as *resource borrowing.* Examples in scientific computing include Condor [61, 117], Sprite [44], Entropia [30], SETI@Home [179], Protein Folding at Home [106], DESChall [37], and the Google

Toolbar [72]. The majority of such systems and applications run on Microsoft Windows platforms, and they are deployed on hundreds of thousands (SETI@Home) to millions (Google) of hosts. Examples in peer-to-peer content distribution systems include commercial tools such as Kazaa [163] and Gnutella [153], as well as academic projects [84, 93, 155, 174]. Some of these systems are deployed on millions of Windows hosts.

There have also been numerous proposals to consolidate desktop computers and servers onto clusters [58, 158] to simplify their administration and exploiting their dramatic underutilization to make computing more economical. Interactive users would then use thin clients [161] to access their processes. In effect, each user will see a slowed machine due to resources borrowed for other users.

In both cases, several fundamental questions arise about user's interaction with resource borrowing:

1. What level of resource borrowing leads to user discomfort for a significant fraction of users?
2. How does the level depend on which resource or combination of resources is borrowed?
3. How does the level depend on the user's context (the foreground task)?
4. How does the level depend on the user, factoring out context?
5. How does the level depend on the time dynamics of resource borrowing?
6. How does the level depend on the raw power of the host?

Current systems assume very conservative answers to these questions because if they cause the user to feel the machine is slower than is desirable, the user is likely to disable them. For example, the default behavior in Condor, Sprite(Process Migration) and SETI@Home is to execute only when they are quite sure the user is away, when the screen saver has been activated. Other systems run at a very low priority, or they simply ask the user to specify constraints on resource borrowing, something that few ordinary users understand. If less conservative resource borrowing does not lead to significantly increased user discomfort,

the performance of systems like Condor and SETI@Home could be increased through techniques such as linger-longer scheduling [156].

There is indirect evidence that resource borrowing need not be especially conservative in that many users are willing to install peer-to-peer tools, run services such as Microsoft's IIS and Kazaa, Gnutella, etc, on their desktop computers. Certainly, the extremely low utilization of CPU cycles, which has held true for over 10 years, suggests that if the "right" cycles were used, those "in between" the cycles the user is using, there would be little for the user to perceive. The priority-based schedulers of modern operating systems approximate this.

Despite indirect evidence, there exist no quantitative, empirical measurements that could be used to answer the above questions. This may seem surprising to some readers, as this would appear to be an excellent problem in human-computer interaction or psychology. However, the work in those areas has concentrated on the impact of latency on user-perceived utility of the system [49, 103], and on user frustration to different user interfaces [101, 151]. Within the systems community, related work has examined the performance of end-user operating systems using latency as opposed to throughput [51], and suggested models for interactive user workload [11].

In response to this lack of information, we have developed a system, the Understanding User Comfort System (UUCS). UUCS is a distributed Windows application similar to SETI@Home in design. A UUCS client emulates resource borrowing of CPU, memory and disk on the user's machine in a throttled manner [157] encoded in a *testcase* provided by a server. The user operates the machine as normal, but may express discomfort by clicking on an icon or pressing a hot-key. Resource borrowing stops immediately if discomfort is expressed or when the testcase is finished. The point of discomfort, if any, is returned to the server along with contextual information. By analyzing the results of applying a particular set of testcases to a particular set of users, we can qualitatively and quantitatively

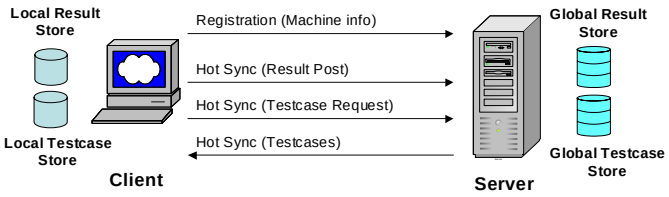


Figure 2.1: Structure of UUCS.

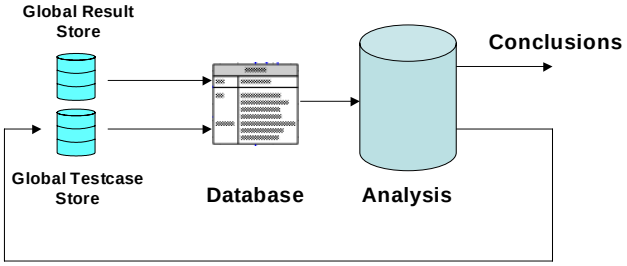


Figure 2.2: The Analysis Phase.

characterize user comfort.

Using UUCS, which we describe in detail in Section 2.1, we have conducted a carefully controlled user study done at Northwestern University to address the questions raised earlier. We describe this study and its results in detail in Section 2.2. In Section 2.3, we provides advice to implementors based on our study results.

More information about the UUCS system can be found in a separate technical report [75]. The written protocol and the form the user filled out can be found in Appendix B.

2.1 System design

UUCS consists of a server and a client, as shown in Figure 2.1. Both are Windows applications that store testcases and results on permanent storage in text files. A testcase contains functions that describe how to “exercise” a collection of resources.

Using its local testcase and result stores, the client can operate disconnected from the

server. There are two interactions between the two, both initiated by the client. When the client is initially run, it registers with the server, providing it with a detailed snapshot of the hardware and software of the client machine, and allowing the server to associate a globally unique identifier with the client. Subsequently, at particularly selected times when the client is connected to the network, the client initiates a "hot sync" with the server. New testcases, which can be added to the server at any time, are downloaded by the clients, while new results are uploaded back to the server.

Hot syncing operates at user-defined intervals and acquires a growing random sample of testcases from the server. This, combined with local random choice of testcases and Poisson arrivals of testcase execution, is designed to make a collection of clients execute a random sample with respect to testcases, users, and times. This is the mode of operation that we plan to use in our future Internet-wide study. A UUCS client can also be configured to behave deterministically, executing a predefined set of commands from a local file. We use this feature in our controlled study.

In addition to the client and server, we have developed a set of tools for creating, viewing, and manipulating testcases, and for importing testcase results into a database. An additional set of tools is then used to analyze the results and guide us to other interesting testcases (Figure 2.2).

2.1.1 Testcases and exercise functions

Testcases encode the details of resource borrowing for various resources. A testcase consists of a unique identifier, a sample rate, and a collection of exercise functions, one for each resource that will be used during the execution of the testcase (the *run*). An exercise function is a vector of values representing a time series sampled at the specified rate. Each value indicates the level of contention (the extent of resource borrowing, described in Section 2.1.2) for a resource at the corresponding time into the testcase. For example,

Name	Description
$step(x,t,b)$	contention of zero to time b, then x to time t
$ramp(x,t)$	ramp from zero to x over times 0 to t
sin	sine wave
saw	sawtooth wave
expexp	Poisson arrivals of exponential-sized jobs (M/M/1)
exppar	Poisson arrivals of Pareto-sized jobs (M/G/1)

Figure 2.3: Testcases.

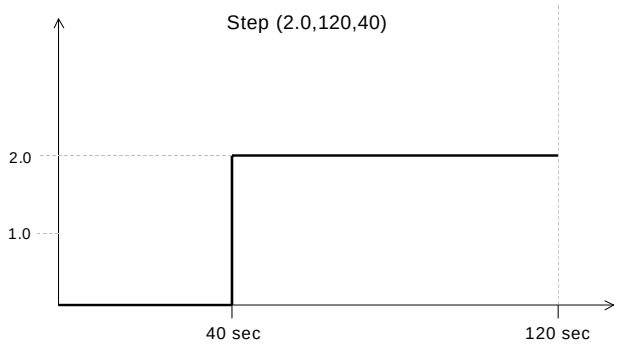
consider a sample rate of 1 Hz, and the vector $[0, 0.5, 1.0, 1.5, 2.0]$ for the CPU resource. This exercise function persists from 0 to 5 seconds from the start of the testcase. From 3 to 4 seconds into the testcase, it indicates that contention of 1.5 should be created and subsequently 2.0 in the next second.

Our testcase tools let us generate testcases of many different kinds, as summarized in Figure 2.3. In our controlled study, we use a small set of *step* and *ramp* testcases with different parameters. Figure 2.4 shows examples for $step(2.0, 120, 40)$ and $ramp(2.0, 120)$ respectively.

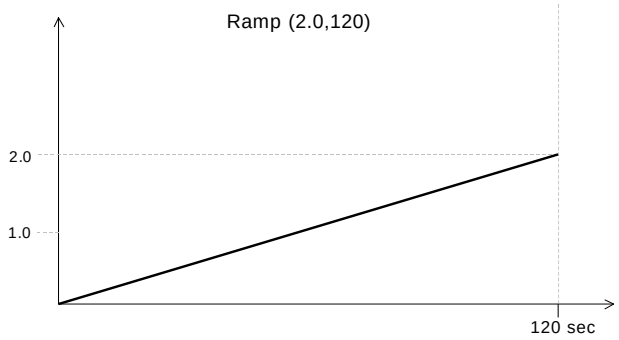
2.1.2 Resource exercisers

Resource exercisers are important components of the client that apply the contention described by an exercise function. There are three exercisers: CPU, memory, and disk.They run on the same priority as other applications.

The CPU exerciser implements time-based playback of the exercise function, as we describe and evaluate in detail in earlier work [43]. Consider the previous example, where we are asked to create a contention of 1.5 from 3 to 4 seconds. Two threads with carefully calibrated busy-wait loops will execute for one second. These loops split the one second interval into a number of subintervals, whose duration is computed by calibration, each larger than the scheduling resolution of the machine. The first loop will only execute busy



(a) Step testcase



(b) Ramp testcase

Figure 2.4: Step and ramp testcases.

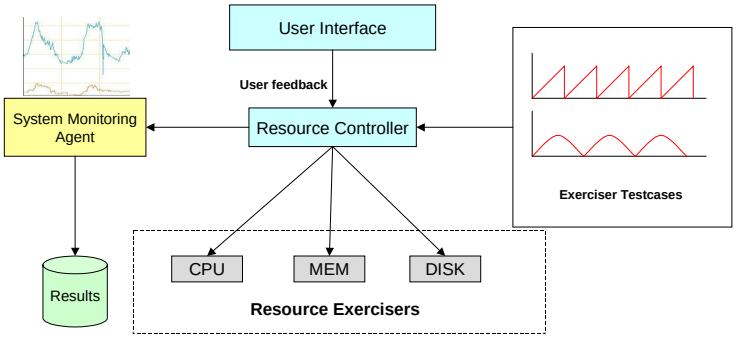


Figure 2.5: Client design.

subintervals,with no sleeps. The second executes busy subintervals with probability 0.5, calling ::Sleep in other subintervals. The effect is that if there is another busy thread in the system (the display loop of a first person shooter game, for example), that thread will execute at a rate $1/(1.5+1) = 40$ % that of the maximum possible rate on the system, the CPU exerciser having borrowed 60% of the CPU. This is experimentally verified to a contention level of 10 for equal priority threads.

The disk exerciser operates nearly identically to the CPU exerciser, except its goal is to create contention for disk bandwidth. The busy operation here is a random seek in a large file (2x the memory of the machine) followed by a write of a random amount of data. The write is forced to be write-through with respect to the windows buffer cache and synced with respect to the disk controller. Contention here has the effect of slowing down the I/O of another I/O-busy thread similarly to the CPU exerciser. This is experimentally verified to a contention level of 7 for equal priority threads.

The memory exerciser is considerably different. It interprets contention as the fraction of physical memory it should attempt to allocate. It keeps a pool of allocated pages equal to the size of physical memory in the machine and then touches the fraction corresponding to the contention level with a high frequency, making its working set size inflate to that fraction of the physical memory. This ensures borrowing of physical memory by the desired amount. We avoid contention levels greater than one because this immediately results in thrashing which is not only very irritating to all users (as it affects interactivity drastically), but also very difficult to stop punctually.

2.1.3 Testcase execution and system monitoring

When a testcase is executed, the appropriate exercisers are started up, passed their exercise functions, synchronized, and then let run. A high priority GUI thread watches for a click or hot-key stroke. If this occurs, the exercisers are immediately stopped and their resources

released. The testcase run is over when user expresses discomfort feedback or the exercise functions are exhausted with any feedback.

A considerable amount of information is stored as the result of the testcase run, including the CPU, memory and disk resource measurements for entire duration of the testcase, system processes information, foreground process information, feedback time etc. A number of technical articles were very useful in the development of the resource exercisers and monitors [13, 16, 32, 40, 63, 65, 97, 140, 145]. Figure 2.5 illustrates the structure of the UUCS client.

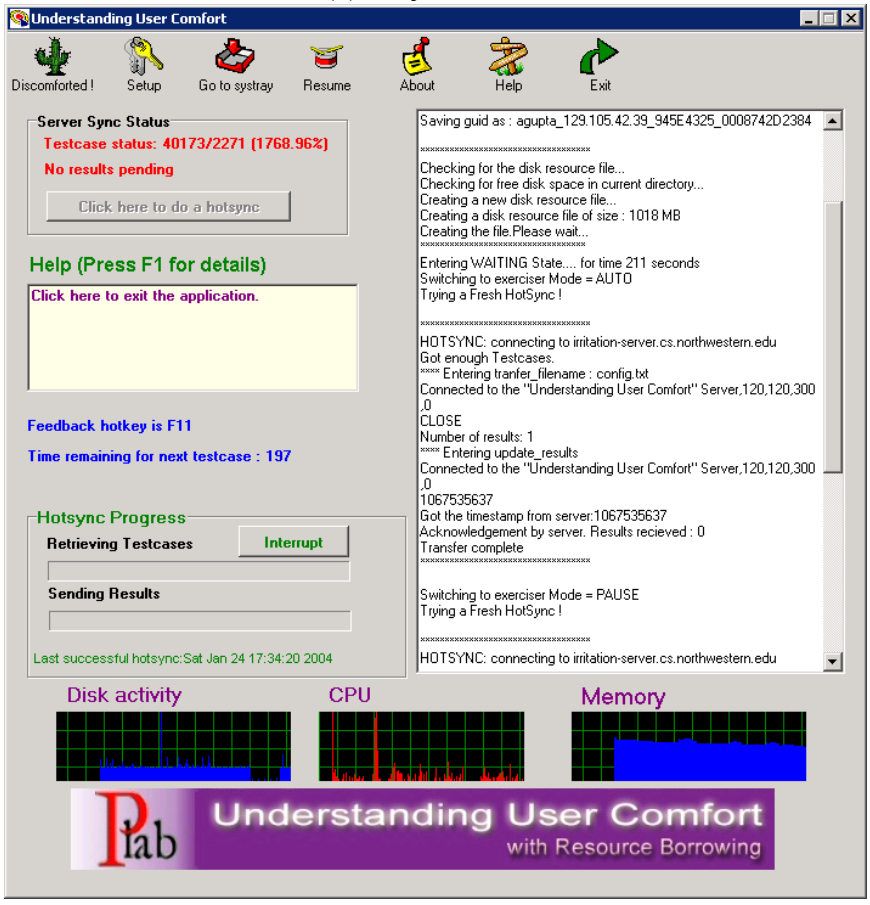
Combined with the detailed registration information for the client, the data we collect is stored in text-based form for later communication back to the server. For the remainder of this paper, we use the following data:

1. The client guid,
2. The testcase guid,
3. Whether the testcase run was terminated due to user feedback or testcase exhaustion,
4. The time offset into the testcase at which irritation or exhaustion was reported, and
5. The last five values used in each exercise function at and immediately before the point of user feedback.

2.1.4 Client interface

Figure 2.6 shows the graphical interface of the UUCS client. The most basic interface is the tray interface (Figure 2.6(a)), in which a user can only express discomfort, either by clicking on the tray icon or by pressing a hot-key (F11 here). The remainder of the interface can be disabled, and is disabled in our controlled study. If it is enabled, the user

(a) Tray interface



(b) Application

Figure 2.6: Client interface. The menu and full application interface can be disabled.

Machine Configuration	
Hardware Configuration	2.4 GHz P4, 512 MB 80 GB , Dell Optiplex GX270 17 in monitor
Operating System	Windows XP
Applications Installed	Word 2002, Powerpoint 2002, IE 6, Quake 2
Network Configuration	100 Mbps Ethernet

Figure 2.7: Machine configuration.

can pop up a detailed view (Figure 2.6(b)) of the operation of the application which permits various other controls like pausing.

2.2 Controlled study

Using the UUCS, we ran a controlled study at Northwestern to help us answer the questions posed in the introduction. This study had a limited number of participants, but because of the careful control of factors, we can directly address many of the questions.

2.2.1 Perspective of the user

The 35 users in our study consisted primarily of graduate and undergraduate students from the Northwestern engineering departments. Anecdotal evidence suggests that this group is more sensitive to resource borrowing than others. We advertised for participants via flyers and email. Each user was given remuneration for participating. The machine configuration used in the control study is shown in Figure 2.7. Two such machines were set up in separate, private environments.

The duration of the study for each user was 84 minutes. The user:

1. Filled out a questionnaire. The key questions were user self-evaluations as “Power User”, “Typical User”, or “Beginner” for use of PCs, Windows, Word, Powerpoint,

Internet Explorer, and Quake. (5 minutes).

2. Read a one page handout. (5 minutes).

3. Acclimatized themselves to the performance of our machine by using the above applications. (10 minutes).

4. Performed the following tasks:

   (a) Word processing using Microsoft Word (16 minutes): Each user had to type in a non-technical document with limited formatting.

   (b) Presentation making using Microsoft Powerpoint (16 minutes): Each user was required to duplicate a presentation consisting of complex diagrams involving drawing and labeling, from a hard copy of a sample presentation.

   (c) Browsing and research with Internet Explorer (16 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 conducted searches for related material on the web and saved it. This task involved multiple application windows.

   (d) Playing Quake III (16 minutes): Quake III is a well known first person shooter game. There were no constraints on user's gameplay.

As the user performed the tasks, the UUCS client executed in the background and ran specific testcases. It recorded all the system and contextual information as well as the user feedbacks, which were later used to generate the results.

2.2.2 Testcase details

Our testcases were designed to help us address the questions in the introduction. They were either of the type ramp or step (Section 2.1.1), or blank. Blank testcases allow us to

No.	Resource	Type	Word Parameters	Powerpoint	Internet Explorer	Quake
1	CPU	Ramp	7.0,0	2.0,0	2.0,0	1.3,0
2	Blank					
3	Disk	Ramp	7.0,0	8.0,0	5.0,0	5.0,0
4	Memory	Ramp	1.0,0	1.0,0	1.0,0	1.0,0
5	CPU	Step	40,5.5	40,0.98	40,1	40,0.5
6	Disk	Step	40,5	40,6	40,4	40,5.0
7	Blank					
8	Memory	Step	40,1	40,1.0	40,1	40,1

Figure 2.8: Testcase descriptions for the 4 tasks (given in random order).

test for the background level of discomfort, while ramps allow us to test user tolerance to borrowing. Steps combined with ramps are used to test for sensitivity to one element of time dynamics. Each task had 8 associated testcases, each 2 minutes long. They are run in a random order for each 16-minute task. We call the execution of a testcase during a specific task by a specific user a *run*.

The regions of resource usage where interactivity is affected are different for each task. For example, in Word very high values of CPU contention (around 3) are needed to affect interactivity at all, while in Quake, CPU contention values in the region of 0.2 to 1.2 cause drastic effects. To observe the onset of discomfort, the parameters for the testcases for each task had to be chosen carefully. This calibration was done by having one of the authors use the applications while running a large number of testcases with different parameters, selecting those testcases which affected interactivity. Figure 2.8 shows the specific testcases used for each task.

It is important to point out that our calibration procedure was subjective. It could have been the case that our testcases were too aggressive or too lax, discomforting too many or too few users. However, our results suggest that we have captured a wide range of behavior. Figure 2.9 counts the runs, grouped by the task, whether the testcase was blank or not, and whether the user expressed discomfort or did not react (testcase exhausted). As

Total		
	Non-Blank testcases	Blank
Discomforted	295	33
Exhausted	47	212
MS Word		
Discomforted	48	0
Exhausted	20	59
Prob of discomfort from blank testcase		0.00
MS Powerpoint		
Discomforted	71	0
Exhausted	4	60
Prob of discomfort from blank testcase		0.00
Internet Explorer		
Discomforted	50	14
Exhausted	17	50
Prob of discomfort from blank testcase		0.22
Quake		
Discomforted	126	19
Exhausted	6	43
Prob of discomfort from blank testcase		0.30

Figure 2.9: Breakdown of runs.

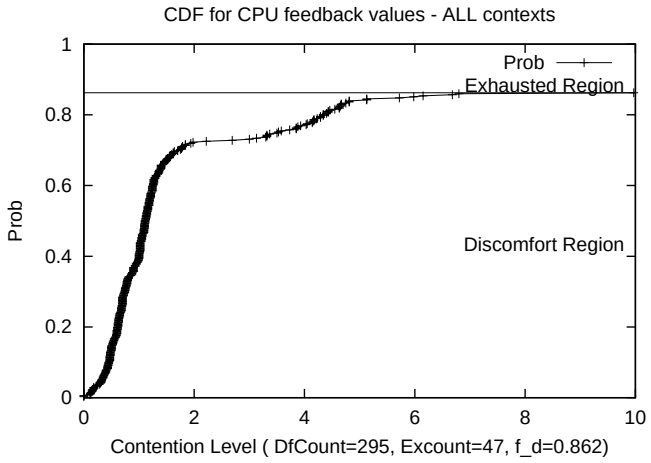


Figure 2.10: CDF of discomfort for CPU.

we can see, there were few reactions to blank testcases, and most, but not all, non-blank testcases caused discomfort at some level.

2.2.3 Results

We now address the questions posed in the introduction using empirical cumulative distribution functions and informal factor analysis. We describe the results below, along with additional observations.

What level of resource borrowing leads to user discomfort for a significant fraction of users?

From the perspective of an implementor this is a key question. We can answer this question using cumulative probability distributions (CDF) derived from running our ramp testcases, aggregated across contexts to convey a general view of each resource.

Figures 2.10-2.12 show CDFs for CPU, Memory and Disk aggregated over all the tasks. The horizontal axis is the level of contention for each resource. The vertical axis is the cumulative fraction of users discomforted. As the level of borrowing increases, users

are increasingly likely to be irritated. This is the *discomfort region*. Finally, some users do not become discomforted in the range of levels explored. We refer to this as the *exhausted region.* The graph labeled with the number of runs that ended in discomfort ($DfCount$) and exhaustion ($ExCount$). There is some probability that a user will feel discomforted even when no resource borrowing (blank testcase) is occurring. We refer to this as the *noise floor* and it is reflected in Figure 2.9.

To make our discussion easier, we derive three metrics from the CDFs. The first is f_d, the fraction of testcases which provoke discomfort,

$$f_d = \frac{DfCount}{DfCount + ExCount}$$

A low value of f_d indicates that the range of contention applied in that context for resource borrowing doesn't affect interactivity significantly.

The second metric is $c_{0.05}$, the contention level that discomforts 5% of the users. This is the 5th-percentile of the CDFs. This value is of particular interest to implementors as it provides them with a level that discomforts only a tiny fraction of users.

The third metric is c_a, the average contention level at which discomfort occurs. This is useful in comparing classes of users. Figures 2.14, 2.15, and 2.16 show the three metrics.

Figure 2.10 shows the CDF for CPU borrowing. Notice that even at CPU contention levels of 10, more than 10% of users do not become irritated. More importantly, we can reach contention levels of 0.4 while irritating fewer than 5% of the users ($c_{0.05,cpu} \simeq 0.4$). This corresponds to consuming 40% of the processor when there are no competing threads.

Figure 2.11 shows the CDF for memory. Notice that almost 80% of users are unfazed even when nearly all their memory is consumed ($f_d = 0.21$). Furthermore, aggregating over the four contexts, it appears we can easily borrow 33% of memory while irritating fewer than 5% of users ($c_{0.05,memory} \simeq 0.33$) in general.

Figure 2.12 shows the CDF for disk bandwidth. Almost 70% of users are comfortable

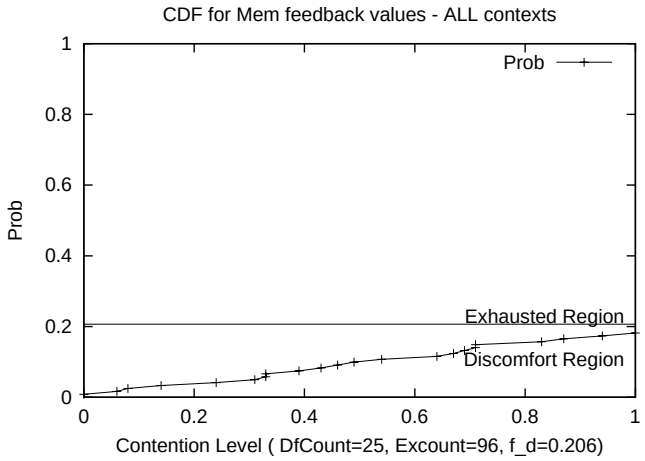


Figure 2.11: CDF of discomfort for Memory.

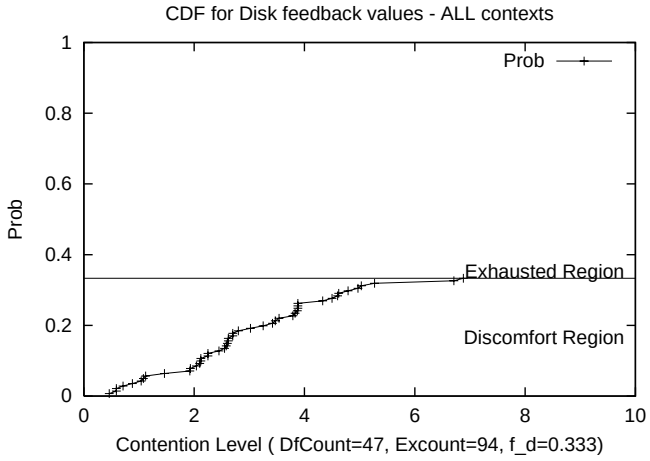


Figure 2.12: CDF of discomfort for Disk.

	CPU	Memory	Disk	Total
Word	L	L	L	L
Powerpoint	M	L	L	M
IE	M	M	H	M
Quake	H	M	M	H
Total	M	L	L	

Figure 2.13: User sensitivity by task and resource.

	CPU	Memory	Disk
Word	0.71	0.00	0.10
Powerpoint	0.95	0.07	0.17
IE	0.75	0.30	0.61
Quake	0.95	0.45	0.29
Total	0.86	0.21	0.33

Figure 2.14: f_d by task and resource.

even with seven competing tasks ($f_d = 0.33$). Furthermore, we can easily execute a single disk writing task, capable of consuming the whole disk bandwidth if run alone, while irritating fewer than 5% of the users ($c_{0.05,disk} \simeq 1.11$). We found this result remarkably counterintuitive as we ourselves tend to become uncomfortable when large amounts of unexplained disk I/O occurs on our desktops. The Dell machines we used for the study are remarkably quiet and have very dim disk lights. We suspect that it is the limited feedback about disk activity that leads to users accepting far higher amounts of disk contention than they otherwise might.

How does the level depend on which resource or combination of resources is borrowed?

Consulting the columns of Figure 2.18 (located at the end of paper) as well as the aggregated CDFs shown earlier clearly shows the strong dependence on the type of resource. Within the contention levels explored by the ramp testcases for each resource, users are much more tolerant with borrowing of memory and disk. This observation is qualitative as

	CPU	Memory	Disk
Word	3.06	*	3.28
Powerpoint	1.00	0.64	3.84
IE	0.61	0.31	2.02
Quake	0.18	0.08	0.69
Total	0.35	0.33	1.11

Figure 2.15: $c_{0.05}$ by task and resource. (* indicates insufficient information)

	CPU	Memory	Disk
Word	4.35 (3.97,4.72)	*	4.20 (1.89,6.51)
Powerpoint	1.17 (1.11,1.24)	0.64 (0.21,1.06)	4.65 (3.67,5.63)
IE	1.20 (1.07,1.33)	0.55 (0.39,0.71)	3.11 (2.69,3.52)
Quake	0.64 (0.58,0.69)	0.55 (0.37,0.74)	1.19 (0.86,1.52)
Total	1.47 (1.31,1.64)	0.58 (0.46,0.71)	2.97 (2.54,3.41)

Figure 2.16: c_a by task and resource, including 95% confidence intervals. (* indicates insufficient information)

the testcases for each resource are different, but within the levels explored this holds true.

The varying tolerance by resource also shows up in our aggregated f_d, c_5 and c_a metrics, the totals rows of Figures 2.14, 2.15, and 2.16. An important point to note is that the high f_d value of CPU (0.87), does not mean that the probability of discomforting users by borrowing CPU is 0.87. This probability depends on the contention. To determine this probability, a level must be chosen and the CDFs consulted as described in the previous section.

How does the level depend on the user's context?

Figure 2.18 shows the CDF for each context and resource pair. We see dramatic differences in the reactions to resource borrowing between different contexts. Consulting the rows illustrates this. It is clearly the case that the user's tolerance for resource borrowing depends not only on the resource, but also on what the user is doing.

The totals row of Figure 2.16 shows the average level at which discomfort occurs for the CPU contention for the four tasks. For an undemanding application like Word, the CPU contention can be very high (> 4) without significant affecting interactivity. However, with finer-grain interactivity, as in Powerpoint and Quake, the average level is much lower. This is likely due to the more aggressive CPU demands of these applications. Still, for the most aggressive application, Quake, results show that a thread with fractional contention of 0.3 can still be run with a low probability of discomfort.

We used the same testcase for memory in all four tasks, growing the working set from zero to nearly the full memory size. The effect of memory borrowing is minimal in the case of Word (no discomfort recorded) and Powerpoint. IE and Quake are much more sensitive to memory borrowing, with more instances of discomfort ($f_d = 0.3$ and $f_d = 0.45$, respectively). For IE and Quake, value of $c_{0.05,mem}$ is 0.31 and 0.08 respectively, meaning that Quake users become irritated at much lower levels. It appears that once

Application	Resource	Background	Rating Pair	P-value	Avg. Difference
Quake	CPU	PC	Power, Typical	0.006	0.176
Quake	CPU	Windows	Power, Typical	0.031	0.137
Quake	CPU	Quake	Power, Typical	0.001	0.224
Quake	CPU	Quake	Typical, Beginner	0.031	0.139
IE	Disk	Windows	Power, Typical	0.004	1.114
IE	Mem	Windows	Power, Typical	0.011	0.354

Figure 2.17: Significant differences based on user-perceived skill level.

office applications like Word and Powerpoint form their working set, significant portions of the remaining physical memory can be borrowed with marginal impact. This seems to be less true for IE and Quake, where there memory demands are more dynamic.

Disk bandwidth can be borrowed with little discomfort in typical office applications. In Word and Powerpoint, the fraction of testcases ending in discomfort was small ($f_d = 0.096$ and $f_d = 0.166$ respectively), in wide range covered by the testcases. IE and Quake are more sensitive. Here we run the identical testcase and find that IE is more sensitive ($f_d = 0.61$). This may be expected as IE caches files and users were asked to save all the pages, resulting in more disk activity.

Figure 2.9 shows that users express feedback even when there is no testcase running. We note that users exhibit this behavior only in IE and Quake. Quake is a very demanding application in which jitter quickly discomforts users. There are sources of jitter on even an otherwise quiescent machine. Discomfort in IE depends to some extent on network behavior.

Figure 2.13 summarizes our judgement of user sensitivity to resource borrowing by resource and task. Note that the totals are not derived from the columns but represent overall judgements from the study of the CDFs (Figure 2.18).

How does the level depend on the user, factoring out context?

Users' comfort with resource borrowing depends to a small extent on their perceived skill level. We asked our users to rate themselves as {Power User, Typical User, or Beginner} in each of {PC Usage, Windows, Word, Powerpoint, IE, and Quake}.

We compared the average discomfort contention levels for the different groups of users defined by their self-ratings for each context/resource combination using unpaired t-tests. In some cases, we found significant differences, summarized in Figure 2.17. The largest differences were for the combination Quake/CPU. For example, a Quake Power User will tolerate 0.224 less CPU contention than a Quake Typical User at a significance level (p-value) of 0.001. Even the users' self-rating for general Windows and PC use can lead to interesting differences in their tolerance. For example, for CPU, the differences between discomfort levels for Power and Typical users are quite drastic with $p = 0.002$ (PC Background) and $p = 0.010$ (Windows Background). Applications which have higher resource requirements show greater differences between user classes.

These results expose the psychological component to comfort with resource borrowing. Experienced or power users have higher expectations from the interactive application than beginners. When we borrow resources it may be helpful to ask the user to rate himself.

How does the level depend on the time dynamics of resource borrowing?

For this question, we have only preliminary results. We tested the "frog in the pot" hypothesis. A perhaps apocryphal rule in French cooking is that when boiling a frog, it is best to place the frog in the water before starting to heat it. The frog will not react to the slowly rising temperature, while a frog dumped unceremoniously into boiling water will immediately jump out. We paired ramp and step testcases in our study to explore if a similar phenomenon might be true of user comfort with resource borrowing—that a user

would be more tolerant of a slow ramp that a quick step to the same level. We did observe the phenomenon in Powerpoint/CPU—the majority of users tolerated higher levels in the ramp testcase. However, we were surprised at how few frog in the pot instances we could find.

2.3 Advice to implementors

Based on our study, we can offer the following guidance to implementors of distributed computing and thin-client frameworks.

- Borrow disk and memory aggressively, CPU less so.
- Build a throttle. Your system can benefit from being able to control its borrowing at a fine granularity similar to the UUCS client.
- Exploit our CDFs (Figures 2.10-2.12) to set the throttle according to the percentage of users you are willing to irritate. As we collect more data, the CDF estimates will improve.
- Know what the user is doing. Their context greatly affects the right throttle setting.
- Consider using irritation feedback directly in your application.

2.4 Conclusions

We have described the design and implementation of a system for measuring user comfort with resource borrowing, as well as a carefully controlled study undertaken with the system.

The end result has three components. First, we provided a set of empirical cumulative distribution functions that show how to trade off between the level of borrowing of CPU,

memory, and disk resources and the probability of discomforting an end-user. Second, we describe how resource type, user context (task), and user-perceived expertise affect these CDFs. Finally, we have made initial observations on how the time dynamics of resource borrowing, and the overall environment affect the level.

Surprisingly, disk and memory can be borrowed quite aggressively with little user reaction, while CPU can also be borrowed liberally. Our observations formed the basis of advice for the implementors of distributed computing and thin-client frameworks.

This work provides evidence for the feasibility of human-driven specification part of my thesis. That is, it is possible to use direct human input from users to determine at least some objective function and constraints, how resource borrowing and user comfort are related in this case. Our study shows that user comfort with resource borrowing is highly dependent on the applications being used and on the user. There is considerable diversity in the tolerance that users have for resource contention. This diversity argues for per-user tailoring of utility functions. It reveals opportunities for new approaches to systems problems that draw on feedback or other input from the end-user.

In the next chapter, we will discuss how we explore using user feedback directly in the scheduling of interactive VMs.

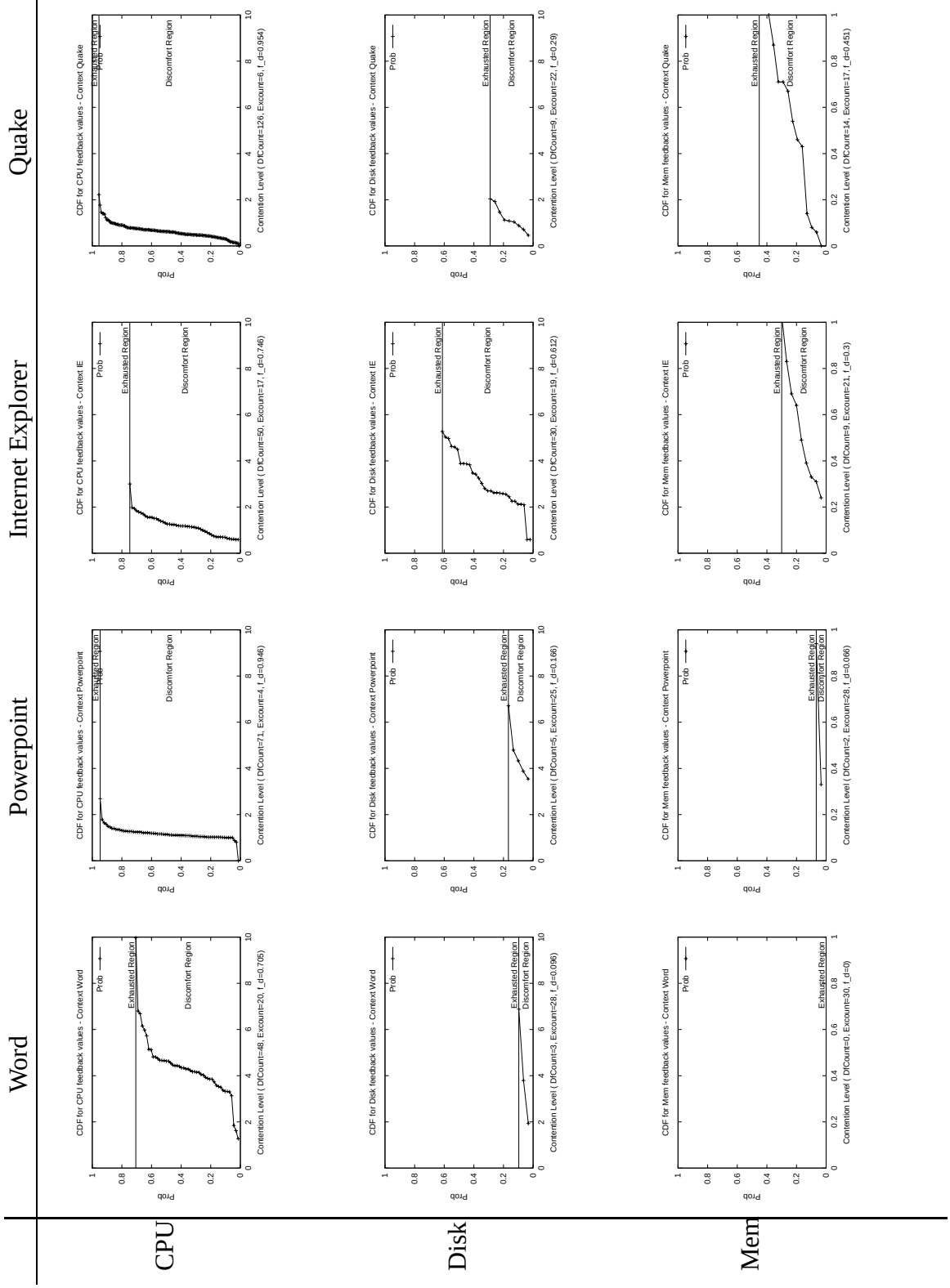


Figure 2.18: CDFs by resource and task.

Chapter 3

User-driven Scheduling of Interactive Virtual Machines

This dissertation argues for using direct human input to solve optimization problems. In the last chapter, we described using a button feedback mechanism to study user comfort with resource borrowing systems. In this chapter, we are going to look at a CPU scheduling problem in Virtuoso systems, which we want to solve using human-driven specification.

In Virtuoso, while a VM can support a very wide range of applications, we particularly want to be able to gracefully handle long-running non-interactive applications, such as scientific simulations, parallel programs, and grid computing applications, as well as interactive applications, such as desktop productivity applications and games. VMs running noninteractive applications and VMs running interactive applications may have to coexist on the same host machine. *How can we schedule or control the interactive VMs so that their users remain happy while not over-penalizing the noninteractive VMs?*

In the previous chapter, we studied the the tolerance of the interactive user for contention for CPU, memory, and disk resources. One of the purposes for characterizing user tolerance to resource borrowing is to inform the scheduling of interactive and non-interactive VMs.

We have shown that user comfort with resource borrowing is highly dependent on

the applications being used. Because of this complexity, determining the exact single scheduling goal for an interactive VM in a particular context is challenging. In response, we explored using direct user feedback in the scheduling process. This chapter presents our initial results in applying direct user feedback, specifically the use of a "discomfort button" similar to that used in our user comfort work, to control the CPU use of interactive VMs.

Virtuoso uses VMware GSX Server as its virtual machine monitor (VMM). GSX is a "type II" VMM [71], meaning that it executes as a process on an underlying operating system, Linux in our case. As such, we can control, to some extent, the priority of all the processes and the OS (Windows XP here) running in the VM by manipulating the nice level of the VMM process. The basic idea here is to modulate the Linux "nice" level of an interactive VM's VMM in response to the passing of time and the user pressing the "discomfort button."

There is a tension between user participation and the average compute rate of non-interactive VMs; the more frequently we expect the user to press the button, the faster the non-interactive VMs can operate. We propose to let the administrator resolve this tension by setting a target mean interarrival time between button presses. One example of how an administrator might set the target interarrival time is in response to the cost of running the VM with more costly VMs having a longer interarrival time target, and the interactive VM user paying according to the interarrival time he is assigned. The more the user pays, the better interactivity he can get. The control system's goal is to manipulate the interactive VM's nice level to maintain this set interarrival time with as little variance as possible.

We explore three control algorithms, all of which are in their infancy. The first two are straightforward translations of the TCP Reno congestion control algorithm. For us a congestion event is the user button press and the equivalent to the congestion window is the linearized nice level of the interactive VM. Acknowledgments are replaced with the simple

passage of time. After a button press, the priority of the interactive VM is maximized. It then declines exponentially in a slow start-like phase and eventually linearly increases at a rate r. Other VMs have the potential to run increasingly faster during this process. At some point, through the combination of the low priority of the interactive VM, the workload inside it, and the workload on the other VMs, the user becomes discomforted and the process repeats.

The third algorithm is also similar to TCP Reno, but here we also control the rate of linear increase from button press to button press, making r a function of time, with the goal of achieving the target interarrival time between button presses.

There has been some related work on controlling CPU allocation by adjusting priorities and scheduler parameters [53, 80] or by the nice mechanism [137], but none has used direct user feedback to facilitate the scheduling of both interactive and non-interactive programs.

In the following, we begin by explaining the details of our framework, which we refer to as the User-driven Virtual Machine Scheduling or UVMS. We describe the extent of control we have using the nice mechanism, how we linearize this mechanism to simplify the remainder of the system, and how our three control algorithms work. Next, we describe very early performance results using the system. Although these results are largely anecdotal at this point, they are quite interesting. Using user feedback to schedule the interactive VMs, the batch machines can make much faster progress compared with simply running the batch machines at a low priority. However, it is unclear whether the user feedback rate is sufficient to control the interarrival time. In the absence of such control, the system may be too frustrating for the end user.

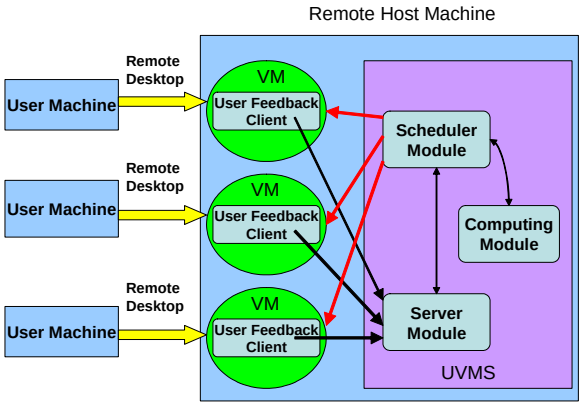


Figure 3.1: Structure of UVMS.

3.1 Mechanism

We have prototyped a mechanism to explore the user-driven scheduling of interactive VMs. This section describes our experimental framework, the control mechanism we use, and three algorithms that react to user feedback using the control mechanism.

3.1.1 Overall framework

User-driven Virtual Machine Scheduling, as shown in Figure 3.1, involves a remote host machine and local client machines. The remote host machine, one of the machines in our cluster, runs VMs. We install Windows XP Professional in a VM created using VMWare GSX Server 2.5. VMWare is configured to use bridged networking, which makes the Windows VM appear as a new and independent machine in the cluster. To access the VM from local machines, we enable the Remote Desktop service of Windows XP professional (also known as Terminal Services.)

The local client machine can be any machine away from the cluster. We use a remote desktop client to connect to the remote Windows VM. Note that there are various methods to connect to a remote VM's display, such as X11 and VNC. The reason why we use

Figure 3.2: Client interface.

remote desktop is that it can achieve better interactivity compared with other methods. Note that multiple VMs can be hosted simultaneously in the same remote host machine. In the current state of this research, we only study the case of a single VM. Underneath VMWare, the host machine runs Red Hat Linux 9.0 with a uniprocessor kernel to simplify the analysis. The host machine is an IBM x335 (2 GHz Xeon, 1.5 GB RAM).

Our UVMS control system consists of a client that runs in the VM and a scheduler that runs on top of Linux in the host machine. We modified the UUCS client, developed in the Understanding User Comfort Project [75, 76], to be our UVMS client. The UVMS client can monitor user's activities and capture user discomfort feedback. Figure 3.2 shows the most basic graphical interface of the UVMS client as it appears in the toolbar of the Windows VM. A user can express discomfort, either by clicking on the tray icon or by simply pressing a hot-key (e.g. F11).

UVMS runs under Linux on the host machine, side by side with the VMs. It consists of three modules:

- Server Module
- Priority Scheduler Module
- Computing Module

The client synchronizes with the server module whenever it starts running, ends, or captures user discomfort feedback.

The priority scheduler module is responsible for applying control algorithms to set the priority of the VMs. It also records realtime information about the VM process and

the scheduler itself, including process id, priority, running time, and so on. The data we collect is stored in text-based form.

To simulate the non-interactive VMs competing for CPU in the system, the computing module launches and monitors a computing process which keeps running a unit busy loop (a loop which finishes computing in a certain unit of time, e.g. 0.01 seconds). We measure the amount of computation as the number of unit busy loops finished.

3.1.2 Control mechanism

By default, processes in Linux are scheduled using the SCHED_OTHER policy. In this policy, each process has a dynamic priority level. Linux ranks the processes based on their priorities and executes the highest priority process. Processes have an initial static priority that is called the nice value. This value ranges from -20 to 19 with a default of zero. The smaller the nice value, the higher the priority. The dynamic priority is calculated as a function of the nice value and the task's interactivity (e.g. sleeping time of a process versus time it spends in runnable state) [125]. The user influences the system's scheduling decisions by changing the process's nice value [115].

We control the process of the VM running in the remote host machine (the core process used by the VMware virtual machine monitor (VMM) for the execution path within the VM). The UVMS scheduler monitors the execution of the VM process and adjusts its priority at run time through the nice mechanism, based on user feedback. As a first step, we did two experiments to validate the control capability of our scheduler.

Experiment 1: In this experiment, we measure the running time of a busy loop program in the VM that is similar to the computing module, with different nice values of the VM process. Note that we run the competing computing module process with a nice value of 0 to compete with the VM process for CPU. We sweep the nice value of the VM process from -20 to 19. For each value, we run the program 15 times and calculate the average. As

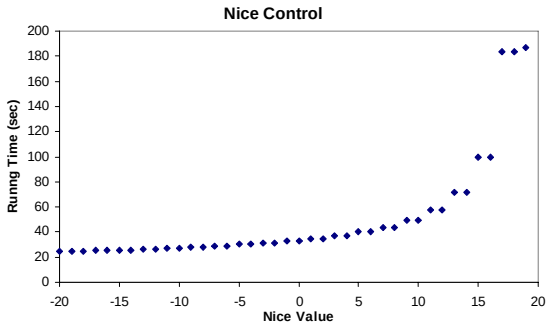


Figure 3.3: Nice control experiment 1.

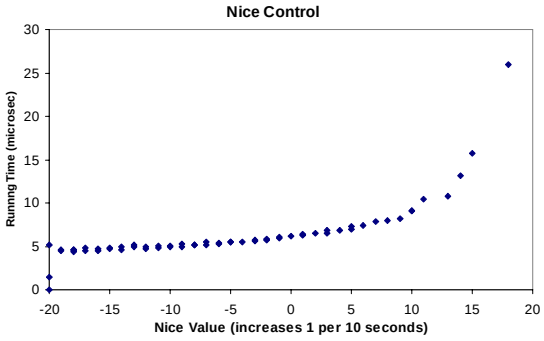


Figure 3.4: Nice control experiment 2.

shown in Figure 3.3, with nice value -20, the average running time is 24.32 seconds. As we increase the nice value from -20 to 19, the running time gradually increases to about 50 seconds and then abruptly steps to 185.99 seconds.

We repeat the experiment three times with different unit running time of the computing module. We find the same trend in all graphs. This experiment shows that by controlling the nice value of the VM process, we can influence the performance of the programs inside the VM by a factor of around 8.

Note that, surprisingly, the nice control mechanism does not behave linearly.

Experiment 2: In this experiment, we study the sensitivity of programs inside the VM to the nice value of VM process. We increase the nice value of the VM process from -

20 to 19 at set intervals. We increase the nice value by 1 every 5 seconds. At the same time, we record, at a fine granularity, the running time of a unit computation inside the VM. From Figure 3.4, we can see that with nice value -20, the running time of the unit computation is 5 microseconds. The running time increases smoothly as the nice value of the VM increases. As before, we are running a competing compute module process with nice value equal to 0 in the host machine to compete with the VM process for CPU.

We repeat the experiment three times, with different unit computations in the computing program and with different intervals of increasing priority. We find the same trend in all graphs.

3.1.3 Control model

How does a CPU intensive process's performance change with its nice value and nice values of other competing processes? In this section, we set up a simple yet effective model to describe a CPU intensive process's performance as a function of its nice value and other competing processes' nice values.

We assume that the process's nice value ranges between -20 and 19, as is reported in the man page that comes with current Linux kernel 2.4.20-8. Note that the possible difference of the nice value range between platforms won't effect our model as long as we know the range on a specific platform. For modeling convenience, we map the nice value from [-20, 19] to [1, 40]. This mapping can be converted back easily. In the paper, we call the mapped nice value *normalized nice value*.

Let P_x be the percentage of CPU dedicated to process x, and T_{fx} be the execution time of process x given $P_x = 1$. Then we have

$$T_x = \frac{T_{fx}}{P_x} \tag{3.1}$$

where T_x is the execution time of process x. Equation 3.1 holds for any CPU intensive

applications with deterministic compute cycle requirements.

Assume there are m CPU intensive processes running in the system, each with nice value n_1, n_2, n_3 ... n_m. Then P_x can be modeled using

$$P_x = \frac{40 - n_x}{\sum_{i=1}^{m}(40 - n_i)} \tag{3.2}$$

where x is between 1 and m. We call $40 - n_x$ the *complementary nice value* of process x. Equation 3.2 means that the percentage of CPU cycles that process x can achieve equals the ratio of its complementary nice value over the sum of all CPU intensive process's complementary nice values.

Combining Equation 3.1 and Equation 3.2, we derive

$$T_x = \frac{T_{fx} \times \sum_{i=1}^{m}(40 - n_i)}{40 - n_x} \tag{3.3}$$

We did experiments to validate our model. Figures 3.5 and 3.6 show two examples of the experimental data versus predicted data given by the model. The experiment data is from experiment 1 as we discussed in Section 3.1.2. Clearly, we can see the model produces satisfactory prediction results until the normalized nice value exceeds 38, where the experimental data flattens out while our model shoots much higher.

3.1.4 Control algorithm

We seek a scheduling algorithm that balances the comfort of interactive VM users and the progress of non-interactive VMs. In other words, we want to maximize the CPU usage of the non-interactive VMs, modeled in our framework with the compute module, subject to a constraint on the discomfort of the the interactive VM users. Our innovation is to have an interactive VM user directly report his discomfort.

We borrow a simple but well-known algorithm as our starting point. The TCP congestion control algorithm [17, 55, 173, 195] is designed to adapt the size of the congestion

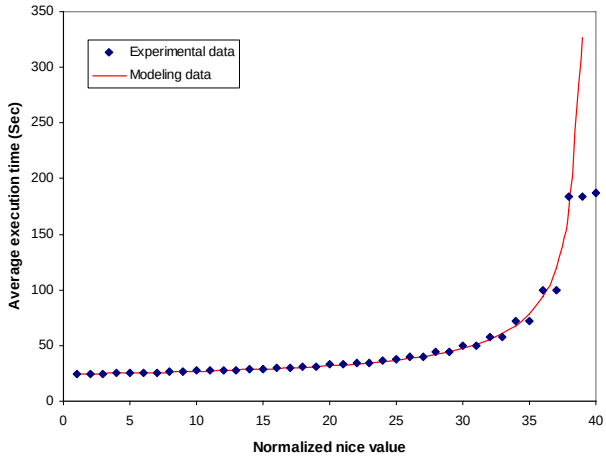


Figure 3.5: Model evaluation experiment I.

window (and hence the send rate) dynamically to the available bandwidth in the path. For us a congestion event is the user button press and the equivalent to the congestion window is the nice value of the VM. Acknowledgments are replaced with the simple passage of time. In effect, the priority of the interactive VM starts out at maximum, and then declines with the passage of time until the user presses the button, at which point the priority is restored to the maximum and the cycle repeats.

Each of our algorithms has two state variables

- Current control value, r (this is the nice level of the VM)
- Threshold, r_t

and three major components:

- Slow Start
- Additive-Increase, Multiplicative-Decrease
- Reaction to the user feedback

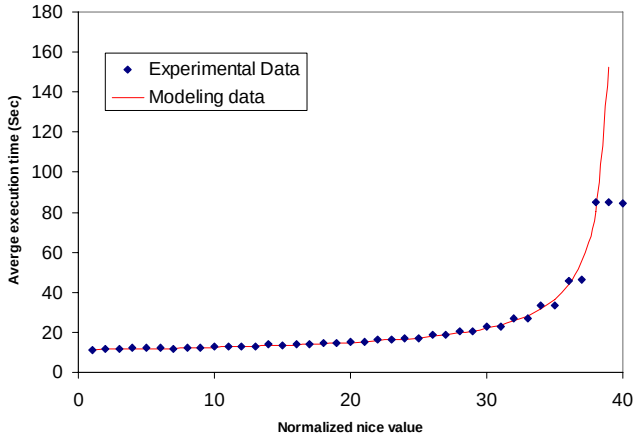


Figure 3.6: Model evaluation experiment II.

We begin with an algorithm with two control parameters:

- Slow Start speed, α
- Additive-Increase speed, β.

Slow Start: If $r < r_t$, we increase r exponentially fast with time (e.g. 2^{α}), assuming that the performance of the VM is less likely to be affected under low nice values (i.e. high priorities).

Additive-Increase, Multiplicative-Decrease: If no user feedback is received and $r \geq r_t$, we increase r linearly with time, $r \propto \beta t$.

Reaction to the user feedback: When the user expresses his discomfort at level r we immediately set $r_t = r/2$, and set r to the initial (lowest) priority.

Figure 3.7 illustrates the execution of the algorithm. On top of this general TCP Reno lookalike, we implemented three extended control algorithms based on nonlinear and linear control schemes. Experimental results of our algorithms will be discussed in Section 3.2.

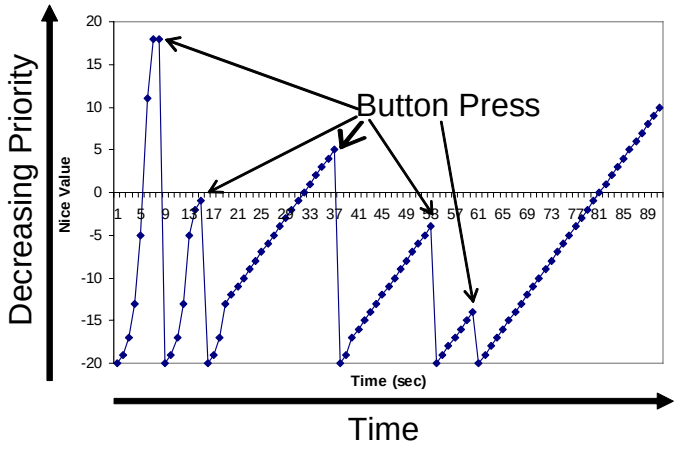


Figure 3.7: TCP Reno-like control algorithm for VM priority.

Nonlinear control scheme

By nonlinear, we mean that changing a function input does not proportionally change the output. As discussed in Section 3.1.2, by directly manipulating the nice value (40 levels from -20 to 19) of the VM process, we can nonlinearly influence the performance of the programs inside the VM by factor of around 8. Based on this scheme, we apply the general TCP control algorithm directly, using r as the nice level, and as a result, we get our nonlinear control algorithm.

Linear control scheme

As discussed in Section 3.1.3, Equation 3.3 models the impact of the nice value on the compute rate of a process. Using this model, we can linearize our control mechanism, as shown in Figure 3.8, by these means:

- From Figure 3.3, we can see that the initial 2/3 of the curve is almost linear. We divide this part as evenly as possible into 7, assigning control values 1 to 7. These

Control value	Nice value	Experimental data	Normalized model value
1	-20	24.36	1.00
2	-7	29.14	1.25
3	-1	32.98	1.48
4	3	37.08	1.72
5	5	40.00	1.89
6	7	43.94	2.13
7	9	49.51	2.45
9	11	57.71	2.94
11	13	71.75	3.75
16	15	99.56	5.38
28	17	183.61	10.25

Figure 3.8: Linearization.

values ensure distinguishable differences in the running time.

- For the tail of the curve, we can see that the running time increases very quickly with even small changes in the nice values. To preserve the smallest granularity of the nice value changes, we divide this part into 4 and assign to it discontinuous control values 9, 11, 16 and 28.

- The time intervals between control values being changed reflect the differences between the corresponding control values.

We use discontinuous control values here to show that we adapt the time intervals of applying control values to the growing Y axis difference between two consecutive control values.

Figure 3.9 shows that with the above linearization to 11 control values and the adaptation of intervals, we can achieve very good linear control. And we did experiments to evaluate the TCP control algorithm based on this linear control scheme.

The linear control scheme has the same control parameters as the nonlinear scheme. The *r* value of the nonlinear scheme is transformed via the mapping derived above before

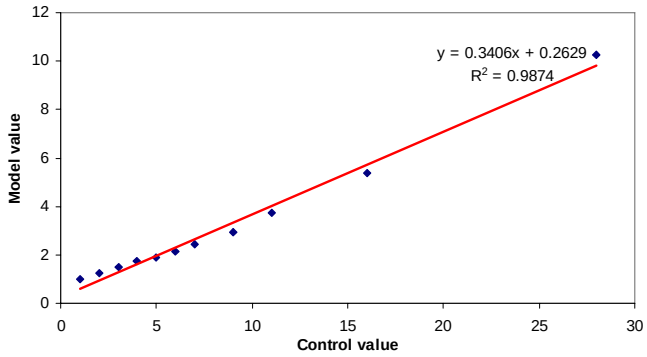


Figure 3.9: Linearization.

it is applied.

Adaptive control scheme

Through experiments, we observe that the Additive-Increase and Multiplicative-Decrease phase in our TCP control algorithm is most often dominated by the linear increase, while the interarrival time between button presses is quite varied. Recall that we would ideally want the user to only have to press the discomfort button at relatively periodic and deterministic points. We extended our TCP control algorithm to better adapt to user feedback, to control not only the impact of background processes, but also the degree of user attention necessary.

In the adaptive algorithm, certain control parameters become state variables:

- Rate of increase, ρ
- Slow Start speed, $\alpha = f(\rho)$
- Additive-Increase speed, $\beta = g(\rho)$

Adaptive reaction to the user feedback: The rate of increase ρ controls the rate of exponential and linear increase from user feedback to user feedback. In addition to our

original TCP control algorithm, we introduce an adaptive function to control the rate of increase:

$$\rho_{i+1} = \rho_i \left(1 + \gamma \times \frac{T_i - T_{AVI}}{T_{AVI}}\right) \tag{3.4}$$

Here ρ_i is the rate of increase. T_i is the latest interarrival time between user feedbacks. T_{AVI} is the target mean interarrival time between user feedbacks, expected by the user or set by the administrator. γ controls the sensitivity to the feedback. We applied this adaptive algorithm to the linearized control scheme.

3.2 Experiments

Using the UVMS, we addressed the questions posed in the introduction, and we compared the various control algorithms described in the previous section. At present, we have studied only a single user (Lin), so our results are preliminary, but interesting and promising.

3.2.1 Experimental setup

The user used his own desktop in his office to connect to the remote Windows VM using the Windows Remote Desktop client in full screen mode. The user used this VM as his desktop during the day. The only difference from his physical desktop was the existence of the user feedback button. The UVMS client spoke to the UVMS scheduler as described earlier. UVMS recorded the system information as well as the user feedback, which was later used to generate the results.

We did three experiments corresponding to the three control algorithms we discussed in Section 3.1.4. The duration of each experiment was approximately 1 hour.

The user's activities included typical tasks he performs daily, for example:

- Browsing with Internet Explorer

- Word processing using Microsoft Word
- Presentation preparation using Microsoft Powerpoint
- Listening to MP3 music using Microsoft Media Player
- Playing games like DemonStar [39] (a 2D shooting game)

3.2.2 Metrics

Recall that our goal is to simultaneously provide high average computation rates to the non-interactive VMs while keeping the users of the interactive VMs happy. And both workloads are run on tightly-coupled computing resources such as clusters. We use two metrics to evaluate the algorithms.

The *interarrival time between user feedbacks* is the interval between two consecutive user feedbacks. This measurement helps us understand how the user feels (e.g. comfort, happiness) when interacting with the VM. Ideally, the user would prefer that such feedbacks are far between on average with very low jitter. We will consider both the average interarrival time and its standard deviation.

The *compute rate* is the rate of computation of the competing process launched by the computing module of UVMS. This metric represents the computation done by other VMs and non-VM processes running in the same host machine. As we mentioned before, we measure the amount of computation as the number of unit busy loops finished. We would like this rate to be as high as possible.

3.2.3 Experimental results and analysis

We show results for our three control algorithms.

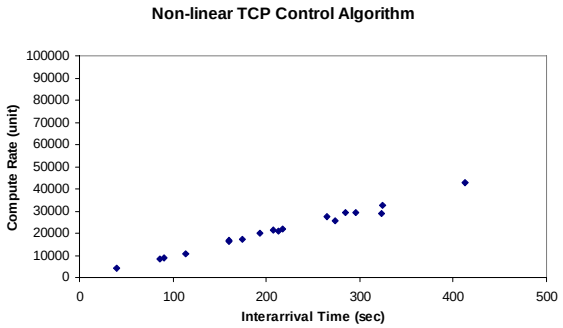


Figure 3.10: Experiment I: nonlinear control scheme.

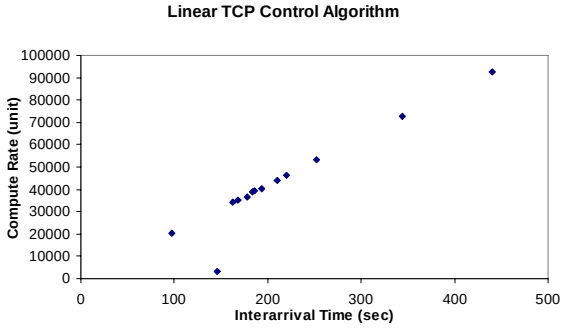


Figure 3.11: Experiment II: linear control scheme.

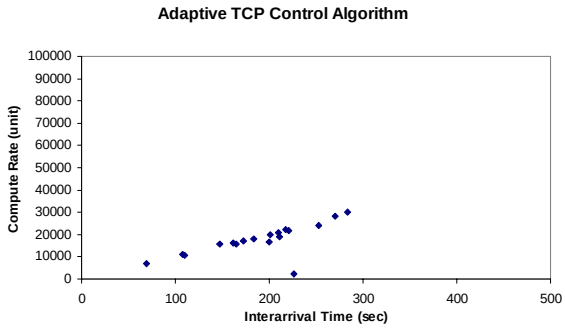


Figure 3.12: Experiment III: adaptive control scheme.

Experiment I: nonlinear control scheme.

Figure 3.10 shows the relationship between the compute rate of the noninteractive process and the interarrival times of the user feedbacks. Each data point in the figure represents the interarrival time between two consecutive user feedbacks and the amount of computation accomplished in the interval.

The average interarrival time between user feedbacks is 213.17 seconds, with a standard deviation of 97.37 seconds. The average nice value at which user feels discomfort is 12.12, with a standard deviation of 6.89. The average compute rate is 21223.61 units, with a standard deviation of 9804.84 units.

Experiment II: linear control scheme.

Figure 3.11 shows the relationship between compute rate and interarrival time. The average interarrival time between user feedbacks is 213.89 seconds, with a standard deviation of 89.28 seconds. The average nice value at which user feels discomfort is 14.67, with a standard deviation of 2.06. The average compute rate is 42824.62 units per interarrival, with a standard deviation of 21981.35 units.

Does the linear control scheme *really* work better than the nonlinear control scheme? Although both experiments have almost the same average interarrival time between user feedbacks with similar standard deviations, the user felt more comfortable in Experiment I. More specifically, in Experiment I, the user felt that the machine always slowed down gradually, while in Experiment II, he often felt discomforted by the abrupt slowdown of the machine. One possible reason for this may be that in the additive-increase phase of Experiment I, we increase the nice value by 1 in certain interval (e.g. 10 seconds) using nonlinear control of 40 levels of nice values, while we use discontinuous control values with limited levels in the other two experiments.

Observe that in both Experiment I and II, the standard deviation of the interarrival time is very large. While it is clear that the system was able to react quickly to provide better performance to the user, the points in time at which the user had to express discomfort showed a lot of jitter, depending mostly on what applications he was using. In other words, it was difficult for the button pressing to become a habit.

Based on this observation, we did experiment III, testing the adaptive TCP control algorithm, which explicitly tries to reduce the variance of the discomfort interarrival error (i.e., the selected interarrival time minus the actual discomfort interarrival time of the user).

Experiment III: adaptive control scheme

Figure 3.12 shows the relationship with compute rate and time interval. The target interarrival time (T_{AVI}) is set to 240 seconds (based on the user experience in Experiment I and II). The control parameter γ is set to 0.5, which makes the algorithm very sensitive to user feedback.

The average interarrival time between user feedbacks is now 189.44 seconds, with a standard deviation of 56.60 seconds. The average nice value at which user feels discomfort is 10.28, with a standard deviation of 2.78. The average compute rate is 17610.56, with a standard deviation of 6980.43.

As we can see, Experiment III achieved a much lower deviation of interarrival time, compared with Experiment I and II. The user felt discomforted at more predictable points in time, as we hoped for.

What compute rate we can deliver to other VMs and non-VM processes? In all three experiments, we can see that the larger the interarrival time is, the higher the compute rate is. The reason is the larger the interarrival time of user feedbacks is, the higher the average nice value (lower priority) the interactive VM process has, and the more CPU time other processes will get. However, a large interarrival time also means that user can

Control algorithm	Accumulated compute time (loop iters)	Average compute time (loops/button press)	Std. Dev. compute time (loops/button press)	Average interarrival (seconds)	Std. Dev. interarrival (seconds)
Exp I nonlinear	382025	21223.61	9804.84	213.17	97.37
Exp II linear	556720	42824.62	21981.35	213.89	89.28
Exp III adaptive	316990	17610.56	6980.43	189.44	56.60
nice -20	116987	N/A	N/A	N/A	N/A
nice -1	133650	N/A	N/A	N/A	N/A

Figure 3.13: Comparison of compute time extracted by a batch VM and button press interarrival times for different algorithms.

tolerate the slowdown of the machine for a long time without being discomforted.

To further study how high a compute rates we can achieve by using UVMS, we calculated the accumulated compute cycles in Experiment I through III. We also ran two more experiments to calculate the accumulated compute rate of the computing process when competing with VM process without UVMS scheduler running. The nice value of the VM process was -20 in one experiment and -1 in another one. Note that in all the experiments we did, the nice value of the computing process was always -20 (highest priority).

Figure 3.13 summarizes our experimental results. For each of the control algorithms, we show the total amount of compute time that was accumulated during the fixed experimental period by the batch VM, the average compute time per button press and its standard deviation, and the average interarrival time between button presses and its standard deviation. For comparison, we also show the accumulated compute time for another possible configuration, leaving the batch VM at the default priority level and increasing the priority level (decreasing the nice level) of the interactive VM.

With the linear and nonlinear control approaches, the user has to tolerate being discomforted at any time—the variation in interarrival time for button presses is high. However, the batch VM can extract the most time. For the linear scheme, the batch VM can extract almost five times as much time as when no user-driven priority manipulation is done. With

the adaptive control approach, the user is more comfortable as the variation in interarrival time is considerably lower. Here the batch VM still extracts three times as much time as compared to doing without user-driven priority manipulation.

3.3 Observations

Figure 3.13 clearly illustrates that it is possible for batch VMs running under our system, competing with interactive VMs that are scheduled using direct user input, to get much more computation time than the Linux dynamic scheduler, with or without static priority manipulations (nice levels), permits. That is, we have demonstrated the utility of direct user input into the scheduling process. However, there is a fundamental dissonance in the scheme that forced us to discard it.

In the system, the interactive user indicates he is willing to press the button periodically. The more he pays, the longer the expected interval between button presses. Unfortunately, with a long expected interval, the adaptive algorithm gets very little measurement input, making it harder for *any* control algorithm to do well. Conversely, the algorithm is likely to do better with short intervals, but these are lower paying customers.

Our conclusion is that while it is critical for a user to be able to asynchronously indicate to the system that he is displeased, he needs to be able to convey more than a single bit of information. This led us to a far superior and practical approach, which we describe in the next chapter.

3.4 Conclusions

We have described the initial design of a scheduling system UVMS, that uses direct user feedback to balance between providing high average computation rates to the non-interactive VMs while keeping the users of the interactive VMs happy. We showed the extent of con-

trol we have using the nice mechanism, how we linearize this mechanism to simplify the remainder of the system, the design of control algorithms and how our three control algorithms work. We also described very early experimental results using the system.

Although our results are largely anecdotal at this point, they are promising. Using feedback it is possible to provide interactive performance while noninteractive VMs progress much faster than would otherwise be possible. However, it appears that more information is needed from the user, perhaps in the form of a real-time schedule.

This work provides evidence for the feasibility of human-driven specification part of my thesis. That is, it is possible to use direct human input from users to determine at least some objective function and constraints, what is the correct priority of user's interactive VM relative to the other VMs in the system in this case.

Chapter 4

Putting the User in Direct Control of Scheduling His Interactive Virtual Machine

This dissertation argues for using direct human input to solve optimization problems. In the last chapter, we showed an initial work on incorporating direct button feedback into the CPU scheduling. The conclusion is that while it is critical for a user to be able to asynchronously indicate to the system that he is displeased, he needs to be able to convey more than a single bit of information. In this chapter, we discuss the design and development of a periodic real-time scheduler. 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 and the necessity of a novel user interface.

4.1 VSched

As discussed in 1, Virtuoso is designed to support a wide range of workloads including interactive workloads, batch workloads and batch parallel workloads.

Today, both sequential and parallel batch jobs are often scheduled using advance reservations [91, 168] such that they will finish by some deadline. Resource providers in Virtuoso price VM execution according to interactivity and compute rate constraints; thus, its scheduling model must be able to validate and enforce these constraints.

An important challenge in Virtuoso is how to schedule a workload-diverse set of VMs on a single physical machine so that interactivity does not suffer and batch machines meet both their advance reservation deadlines and gang scheduling constraints. It is that challenge that VSched addresses.

VSched schedules a collection of VMs on a host according to the model of independent periodic real-time tasks. Tasks can be introduced or removed from control at any point in time through a client/server interface. Virtuoso uses this interface to enforce compute rate and interactivity commitments a provider has made to a VM.

4.1.1 Abstraction

The periodic real-time model is a unifying abstraction that can provide for the needs of the various classes of applications described above. In the periodic real-time model, a task is run for *slice* seconds every *period* seconds. Typically, the periods start at time zero. Using earliest deadline first (EDF) schedulability analysis [118], 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 offers soft real-time guarantees. Because the Linux kernel does not have pri-

ority inheritance mechanisms, nor known bounded interrupt service times, it is impossible for a tool like VSched to provide hard real-time guarantees to ordinary processes. Nonetheless, as we show in our evaluation, for a wide range of periods and slices, and under even fairly high utilization, VSched almost always meets the deadlines of its tasks.

In typical soft and hard embedded real-time systems, the $(period, slice)$ constraint of a task is usually measured in microseconds to low milliseconds. VSched is unusual in that it supports periods and slices ranging into days. While fine, millisecond and sub-millisecond ranges are needed for highly interactive VMs, much coarser resolutions are appropriate for batch VMs.

It is important to realize that the ratio $slice/period$ defines the *compute rate* of the task.

Batch VMs Executing a VM under the constraint $(period, slice)$ for T seconds gives us at least $slice \times \lfloor T/period \rfloor$ seconds of CPU time within T seconds. In this way, the periodic real-time model can be used to express a deadline for the entire execution of the batch VM.

Batch parallel VMs A parallel application may be run in a collection of VMs, each of which is scheduled with the same $(period, slice)$ constraint. If each VM is given the same schedule and starting point, then they can run in lock step, avoiding the synchronization costs of typical gang scheduling.[1] If the constraint accurately reflects the application's compute/communicate balance, then there should be minimal undesired performance impact as we control the execution rate. As the schedule is a reservation, the application is impervious to external load.

Interactive VMs Based on an in-depth study of users operating interactive applications such as word processors, presentation graphics, web browsers, and first-person shooter

[1] Note, however, that this does introduce the need for synchronized clocks, with the bounds on synchronization decreasing with the granularity of the application.

games (Chapter 2), we have reached a number of conclusions about how to keep users of such applications happy [76]. The points salient to this chapter are that the CPU rates and jitter needed to keep the user happy is highly dependent on the application and on the user. We believe we need to incorporate direct user feedback in scheduling interactive applications running in VMs.

In Chapter 3, we explored using a single "irritation button" feedback mechanism to control VM priority. This approach proved to be too coarse-grained. The two-dimensional control possible with the $(period, slice)$ mechanism is much finer-grained. An important design criterium for VSched is that a VM's constraints can be changed very quickly (in milliseconds) so that an interactive user can improve his VM's performance immediately or have the system migrate it to another physical machine if his desired $(period, slice)$ is impossible on the original machine. We discuss this further in Section 4.6.

4.1.2 Type-II versus type-I VMMs

VSched is a user-level program that runs on Linux and schedules other Linux processes. We use it here to schedule the VMs created by VMware GSX Server. GSX is a type-II virtual machine monitor, meaning that it does not run directly on the hardware, but rather on top of a host operating system, in this case Linux. A GSX VM, including all of the processes of the guest operating system running inside, appears as a process in Linux, which is then scheduled by VSched.

While type-II VMMs are by far the most common on today's hardware and VSched's design lets it work with processes that are not VMs, it is important to point out that periodic real-time scheduling of VMs could also be straightforwardly applied in type-I VMMs. A type-I VMM runs directly on the underlying hardware with no intervening host OS. In this case, the VMM schedules the VMs it has created just as an OS would schedule processes. Just as many OSes support the periodic real-time model, so could type-I VMMs. Our

argument for scheduling VMs using the periodic real-time model still applies.

4.2 System design

VSched uses the schedulability test of the earliest-deadline-first (EDF) algorithm [118, 120] to do admission control and EDF scheduling to meet deadlines. It is a user-level program that uses fixed priorities within Linux's SCHED_FIFO scheduling class and SIGSTOP / SIGCONT to control other processes, leaving aside some percentage of CPU time for processes that it does not control. By default, VSched is configured to be work-conserving for the real-time processes it manages, allowing them to also share these resources and allowing non-real-time processes to consume time when the real-time processes are blocked. The resolution at which it can schedule depends on timer resolution in the system, and thus its resolution depends on the Linux kernel version and the existence of add-on high-resolution timers. VSched consists of a parent and a child process that communicate via a shared memory segment and a pipe. The following describes the design of VSched in detail.

4.2.1 Algorithms

A well-known dynamic-priority algorithm is EDF (Earliest Deadline First). It is a preemptive policy in which tasks are prioritized in reverse order of the impending deadlines. The task with the highest priority is the one that is run. We assume that the deadlines of our tasks occur at the ends of their periods, although this is not required by EDF.

Given a system of n independent periodic tasks, there is a fast algorithm to determine if the tasks, if scheduled using EDF, will all meet their deadlines:

$$U(n) = \sum_{k=1}^{n} \frac{slice_k}{period_k} \leq 1 \tag{4.1}$$

Here, $U(n)$ is the total utilization of the task set being tested. Equation 4.1 is both a necessary and sufficient condition for any system of *n* independent, preemptable tasks that have relative deadlines equal to their respective periods to be schedulable by EDF [120].

4.2.2 Mechanisms

SCHED_FIFO Three scheduling policies are supported in the current Linux kernel: SCHED_FIFO, SCHED_RR and SCHED_OTHER. SCHED_OTHER is the default universal time-sharing scheduler policy used by most processes. It is a preemptive, dynamic-priority policy. SCHED_FIFO and SCHED_RR are intended for special time-critical applications that need more precise control over the way in which runnable processes are selected for execution. Within each policy, different priorities can be assigned, with SCHED_FIFO priorities being strictly higher than SCHED_RR priorities which are in turn strictly higher than SCHED_OTHER priorities. SCHED_FIFO priority 99 is the highest priority in the system and it is the priority at which the scheduling core of VSched runs. The server front-end of VSched runs at priority 98. No other processes at these priority levels are allowed.

SCHED_FIFO is a simple preemptive scheduling policy without time slicing. For each priority level in SCHED_FIFO, the kernel maintains a FIFO queue of processes. The first runnable process in the highest priority queue with any runnable processes runs until it blocks, at which point it is placed at the back of its queue. When VSched schedules a VM to run, it sets it to SCHED_FIFO and assigns it a priority of 97, just below that the VSched server front-end. No other processes at this priority level are allowed.

The following rules are applied by the kernel: A SCHED_FIFO process that has been preempted by another process of higher priority will stay at the head of the list for its priority and will resume execution as soon as all processes of higher priority are blocked again. When a SCHED_FIFO process becomes runnable, it will be inserted at the end

of the list for its priority. A system call to `sched_setscheduler` or `sched_setparam` will put the SCHED_FIFO process at the end of the list if it is runnable. No other events will move a process scheduled under the SCHED_FIFO policy in the queue of runnable processes with equal static priority. A SCHED_FIFO process runs until either it is blocked by an I/O request, it is preempted by a higher priority process, or it calls `sched_yield`. The upshot is that the process that VSched has selected to run is the one with the earliest deadline. It will run whenever it is ready until VSched becomes runnable.

Timers After configuring a process to run at SCHED_FIFO priority 97, the VSched core waits (blocked) for one of two events using the `select` system call. It continues when it is time to change the currently running process (or to run no process) or when the set of tasks has been changed via the front-end.

The resolution that VSched can achieve is critically dependent on the available timer. Under the standard 2.4.x Linux kernel, the timer offers 10 ms resolution. For many applications this is sufficient. However, especially interactive applications, such as games or low-latency audio playback require finer resolution. When running on a 2.6.x Linux kernel, VSched achieves 1 ms resolution because the timer interrupt rate has been raised to 1000 Hz. The UTIME component of KURT-Linux [83] uses the motherboard timers to deliver asynchronous timer interrupts with resolution in the tens of μs. In VSched, we call `select` with a non-null timeout as a portable way to sleep with whatever precision is offered in the underlying kernel. Since UTIME extends `select`'s precision when it's installed, VSched can offer sub-millisecond resolution in these environments. Note, however, that the overhead of VSched is considerably higher than that of UTIME, so the resolution is in the 100s of μs.

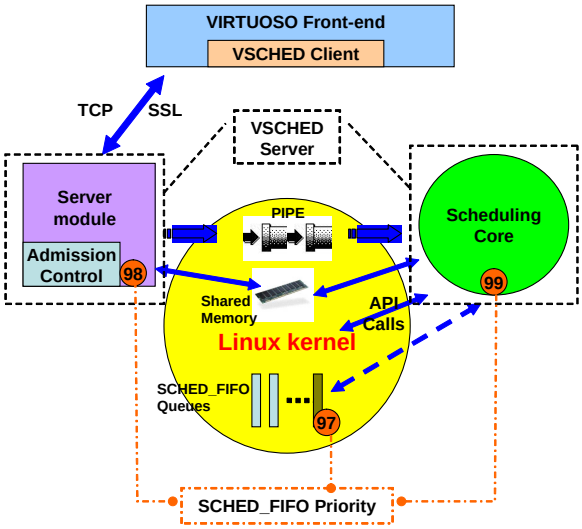


Figure 4.1: Structure of VSched.

SIGSTOP/SIGCONT By using EDF scheduling to determine which process to raise to highest priority, we can assure that all admitted processes meet their deadlines. However, it is possible for a process to consume more than its slice of CPU time. By default, when a process's slice is over, it is demoted to SCHED_OTHER. VSched can optionally limit a VM to exactly the slice that it requested by using the SIGSTOP and SIGCONT signals to suspend and resume the VM, similar to how control was asserted in GLUnix [68]. Although this adds overhead, we envision this as critical in a commercial environment.

4.2.3 Structure

VSched consists of a server and a client, as shown in Figure 4.1. The VSched server is a daemon running on Linux that spawns the scheduling core, which executes 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 two mechanisms. First, they share a memory segment which contains an array that describes the current

tasks to be scheduled as well as their constraints. Access to the array is guarded via a semaphore. The second mechanism is a pipe from server to core. The server writes on the pipe to notify the core that the schedule has been changed.

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. For example, the specification (3333, 1000 ms, 200 ms) would mean that process 3333 should be run for 200 ms every 1000 ms. 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. In either case, it replies to the client.

VSched allows a remote client to find processes, pause or resume them, specify or modify their real-time schedules, and return them to ordinary scheduling. Any process, not just VMs, can be controlled in this way.

Admission control VSched's admission control algorithm is based on Equation 4.1, the admissibility test of the EDF algorithm. As we mentioned above, it is both a necessary and sufficient condition. Instead of trying to maximize the total utilization, we allow the system administrator to reserve a certain percentage of CPU time for SCHED_OTHER processes. The percentage can be set by the system administrator when starting VSched.

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.

Since a task can miss its deadline only at a period boundary, the scheduling core makes

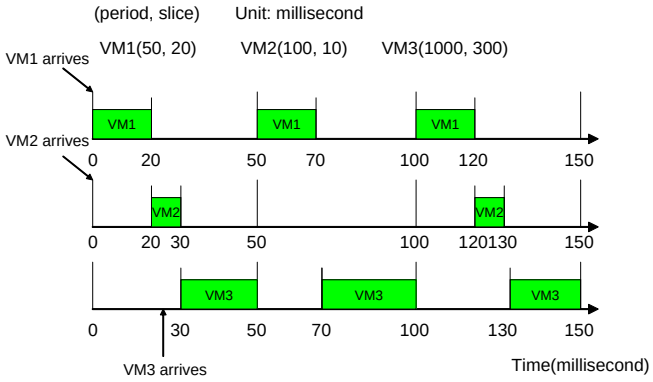


Figure 4.2: A detailed VSched schedule for three VMs.

Machine 1: Pentium 4, 2.00GHz, 512MB Mem, Linux version 2.4.20-31.9 (Red Hat Linux 9.0)
Machine 2: Dual CPUs (Pentium III Coppermine, 1.0 GHZ), 1G Mem, non-SMP Linux kernel 2.4.18 patched with KURT 2.4.18-2
Machine 3: Pentium 4, 2.00GHz, 512MB Mem, Linux version 2.6.8.1 (Red Hat Linux 9.0)

Figure 4.3: Testbed Machines

scheduling decisions only at period boundaries, i.e., at the points when a task exhausts its slice for the current period, or when the server indicates that the task set and its constraints have changed. In this way, unlike a kernel-level scheduler [2, 7, 8, 25, 81, 136], VSched is typically invoked only at the rate of the task with the smallest period.

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.

Figure 4.2 illustrates the scheduling of three virtual machines with different arrival times.

Kernel version	Machine (from Fig. 4.3)	Utilization Range	Period Range	Slice Range	Deadlines per combination
Linux kernel 2.4.20-31.9	1	10% - 99% (increasing by 10%)	1016 ms - 16 ms (decreasing by 40 ms)	105.8 ms - 1.6 ms	1000
KURT 2.4.18-2	2	1% - 99% (increasing by 1%)	10.1 ms - 1.1 ms (decreasing by 1 ms)	9.999 ms - 0.011 ms	2000
Linux kernel 2.6.8.1	3	1% - 99% (increasing by 1%)	101 ms - 1 ms (decreasing by 10 ms)	99.99 ms - 0.01 ms	2000

Figure 4.4: Evaluation scenarios.

4.3 Evaluation

Our evaluation focuses on the resolution and utilization limits of VSched running on several different platforms. We answer the following questions: what combinations of period and slice lead to low deadline miss rates and what happens when the limits are exceeded?

We ran our evaluation in three different environments, as shown in Figure 4.3. The key differences between these environments are the processor speed (1 GHz P3 versus 2 GHz P4) and the available timers (2.4 kernel, 2.4 with KURT, and 2.6 kernel). For space reasons, we present results for machine 1 only, a stock Red Hat installation that is the most conservative of the three. Additional results are available at virtuoso.cs.northwestern.edu.

We also consider the effects of VSched on time-sensitive local I/O devices in this section. The next section looks at user-perceived quality of audio and video I/O. In all cases except for local I/O, we are running the application in the VM and scheduling the VM.

4.3.1 Methodology

Our primary metric is the *miss rate*, the number of times we miss the deadlines of a task divided by the total number of deadlines. For tasks that miss their deadlines, we also

collect the *miss time*, the time by which the deadline was overrun. We want to understand how the miss rate varies with period and slice (or, equivalently, period and utilization), the number of VMs, and by how much we typically miss a deadline when this happens.

We evaluate first using randomly generated testcases, a testcase being a random number of VMs, each with a different $(period, slice)$ constraint. Next, we do a careful deterministic sweep over period and slice for a single VM. Figure 4.4 shows the range of parameters used.

4.3.2 Randomized study

Figure 4.5 shows the miss rates as a function of the total utilization of the VMs for one through four VMs. Each point corresponds to a single randomly generated testcase, while the line represents the average miss rate over all the testcases. The miss rates are low, independent of total utilization, and largely independent of the number of VMs after two VMs. Going from one to two VMs introduces the need for more frequent context switches.

Figure 4.6 shows the distribution of the ratio of miss time to slice size, with the line showing the maximum. All misses that do occur miss by less than 9%.

4.3.3 Deterministic study

In this study, we scheduled a single VM, sweeping its period and slice over the values described in Figure 4.4. Our goal was to determine the maximum possible utilization and resolution, and thus the safe region of operation for VSched on the different platforms.

Figure 4.7 shows the miss rate as a function of the period and slice for Machine 1. The top graph is a 3D representation of this function, while the bottom graph is a contour map of the function. This is evidence that utilizations to within a few percent of 100% are possible with nearly 0% miss rate.

Deadline misses tend to occur in one of two situations:

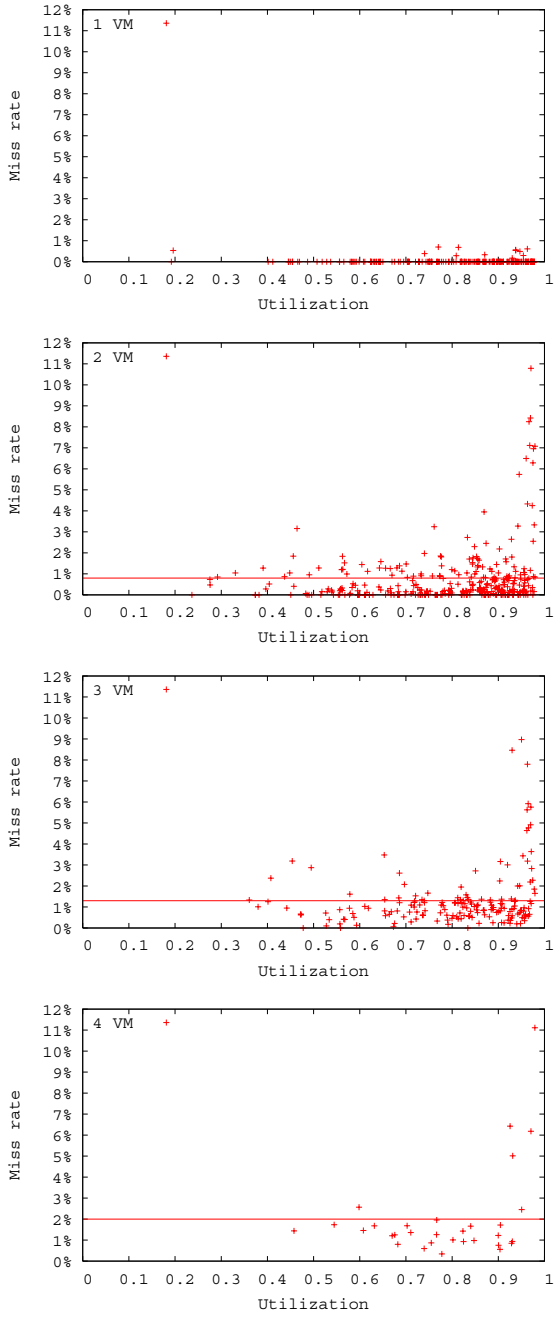


Figure 4.5: Miss rate as a function of utilization, Random study on Machine 1 (2 GHz P4, 2.4 kernel).

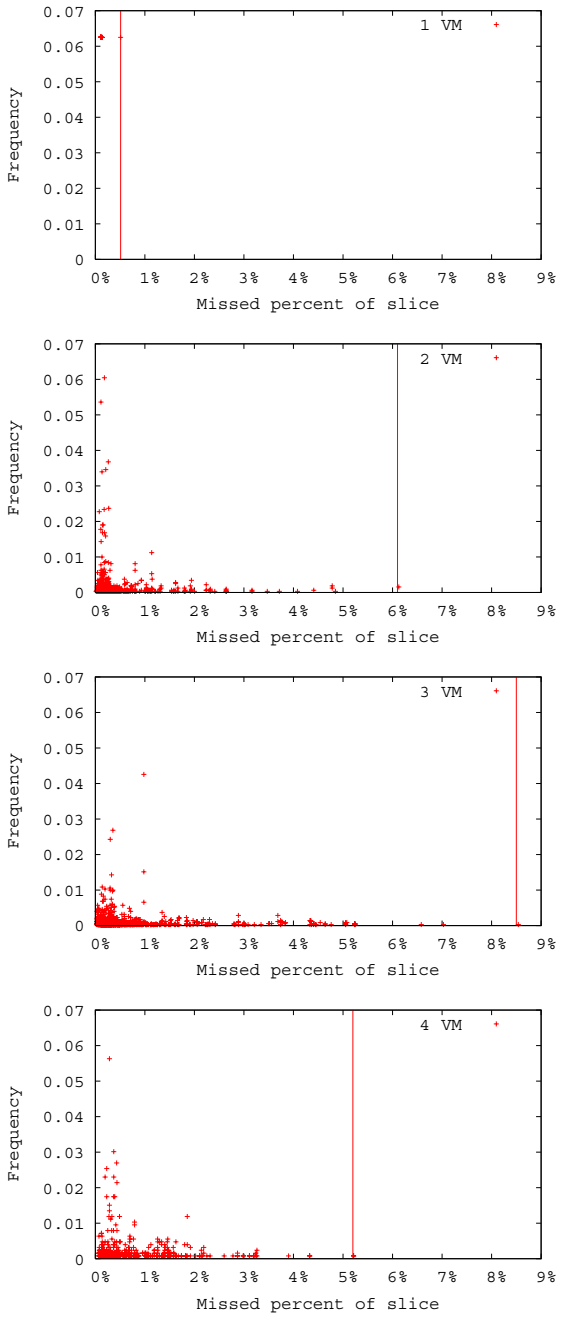
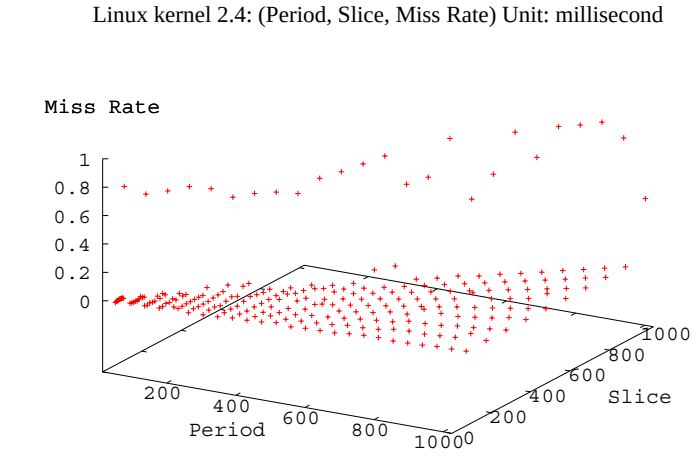
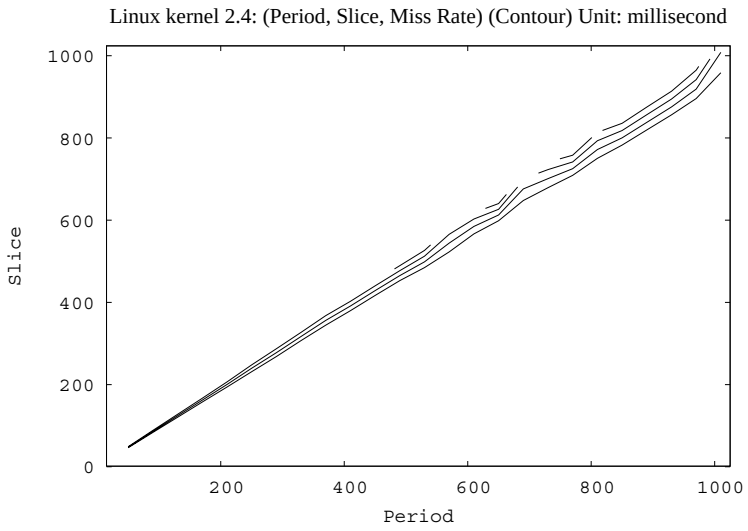


Figure 4.6: Distribution of missed percentage of slice; Random study on Machine 1 (2 GHz P4, 2.4 kernel).



(a) 3D



(b) Contour

Figure 4.7: Miss rate as a function of period and slice for Machine 1 (2 GHz P4, 2.4 kernel).

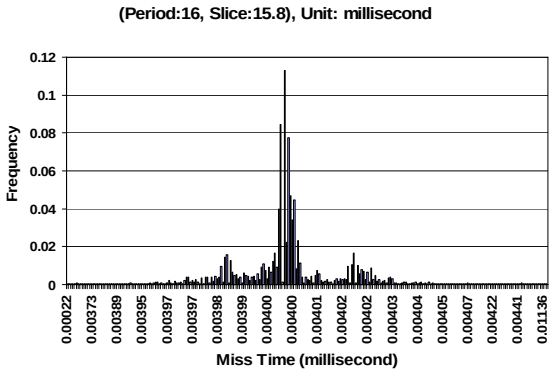


Figure 4.8: Distribution of miss times when utilization is exceeded for Machine 1 (2 GHz P4, 2.4 kernel).

Configuration	Maximum Utilization	Minimum Resolution
Machine 1	0.90	10 ms
Machine 2	0.75	0.2 ms
Machine 3	0.98	1 ms

Figure 4.9: Summary of performance limits on three platforms.

- Utilization misses: The utilization needed is too high (but less than 1).
- Resolution misses: The period or slice is too small for the available timer and VSched overhead to support.

Figure 4.8 illustrates utilization misses on Machine 1. Here, we are requesting a period of 16 ms (feasible) and a slice of 15.8 ms (feasible). However, this utilization of 98.75% is too high for to be able to schedule it. VSched would require slightly more than 1.25% of the CPU. The figure shows a histogram of the miss times. Notice that the vast majority of misses miss by less than 405 μs, less than 3% of the period.

Figure 4.9 summarizes the utilization and resolution limits of VSched running on our different configurations. Beyond these limits, miss rates are close to 100%, while within these limits, miss rates are close to 0%.

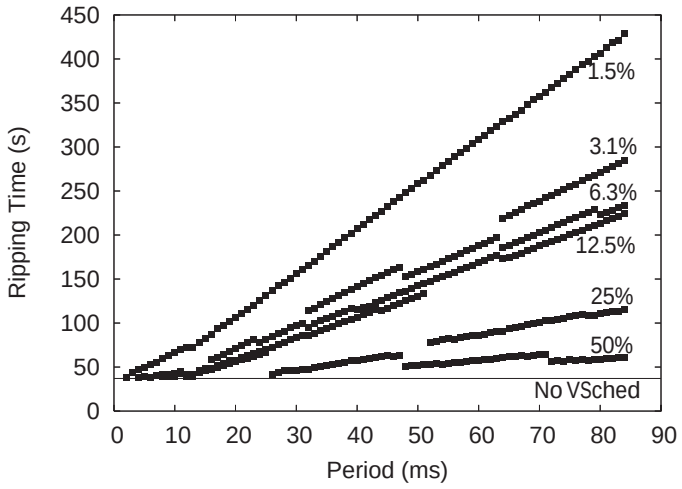


Figure 4.10: Performance of time-sensitive I/O (ripping an Audio CD) (2 GHz P4, 512MB RAM, 2.6 kernel, Mandrake Linux).

4.3.4 I/O

As VSched schedules only the CPU and, unlike SCHED_OTHER, provides no priority boost for a process that has just completed I/O, a natural question is how much I/O, particularly time-sensitive I/O, suffers. Figure 4.10 illustrates the performance of ripping a track from an audio CD using cdparanoia, where cdparanoia is scheduled according to different periods and utilizations. Note that here we are scheduling the cdparanoia application directly (no VM is involved). Reading from CD is extremely time sensitive as a buffer overrun results in a very expensive seek. The time to rip the track without any VSched scheduling is 37 seconds with 5% CPU utilization, which is nearly identical to not using a VM at all. It is clearly possible for VSched to schedule cdparanoia so that it achieves similar performance to SCHED_OTHER at a similar utilization.

(period, slice)(ms) (ms)	Quake(with sound)	MP3 playback	MPEG(with sound) playback	Web Browsing
5, 1	good	good	tiny audio jitter	good
6, 1	good	good	tiny audio jitter	good
7, 1	tiny jitter	good	tiny audio jitter	good
8, 1	small jitter	tiny jitter	tiny jitter	jitter
9, 1	jitter	noisy	tiny jitter	jitter
10, 1	jitter	noisy	jitter	jitter
15, 1	jitter	noisy	jitter	jitter
20, 1	jitter	noisy	jitter	jitter
30, 1	jitter	noisy	jitter	jitter
20, 10	small jitter	small jitter	jitter	small jitter
30, 10	jitter	noisy	jitter	jitter
50, 10	jitter	noisy	jitter	jitter
100, 80	jitter	noisy	jitter	good
200, 100	jitter	noisy	jitter	jitter
300, 100	jitter	noisy	jitter	jitter

Figure 4.11: Summary of qualitative observations from running various interactive applications in an Windows VM with varying period and slice. The machine is also running a batch VM simultaneously with a (10 min, 1 min) constraint.

4.4 Mixing batch and interactive VMs

To see the effect of VSched on an interactive VM used by real users, we ran a small study. The users in our study consisted of four graduate students from the Northwestern Computer Science Department. Each user ran an interactive VM with fine-grained interactive programs together with a batch VM and reported his observations. The test machine had the following configuration:

- Pentium 4, 2.20GHz, 512MB Mem, Linux version 2.6.8.1-12mdk (Mandrake Linux 10.1)
- VMware GSX Server 3.1
- VSched server running as a daemon

- Interactive VM running Windows XP
- Batch VM running Red Hat Linux 7.3. A process was started in the batch VM that consumed CPU cycles as fast as possible and periodically sent a UDP packet to an external machine to report on progress.

Each user tried the following activities in his VM:

- Listening to MP3 music using MS Media Player
- Watching an MPEG video clip using MS Media Player
- Playing QUAKE II [87] (3D first person shooter)
- Browsing the web using Internet Explorer, including using multiple windows, Flash Player content, saving pages, and fine-grained view scrolling.

We set the batch VM to run 1 minute every 10 minutes (10% utilization). The user was given control of the period and slice of his interactive VM. For each activity, the user tried different combinations of period and slice to determine qualitatively which were the minimum acceptable combinations. Figure 4.11 summarizes our observations. For each activity, we present the worst case, i.e., the observations of the most sensitive user.

These qualitative results are very promising. They suggest that by using VSched we can run a mix of interactive and batch VMs together on the same machine without having them interfere. The results also indicate that there is considerable headroom for the interactive VMs. For example, we could multiplex nearly 8 Windows VMs with users comfortably playing QUAKE II in each of them on one low-end P4 computer. Given the fast reaction time of VSched to a schedule change (typically within a few milliseconds), we have high hopes that the end-users of interactive machines will be able to dynamically adjust their VM's constraints for changing needs. The same holds true for the users of batch VMs.

Indeed, the VSched abstraction provides for a continuum from fine-grained interactivity to very coarse-grained batch operation, all on the same hardware.

4.5 Summary

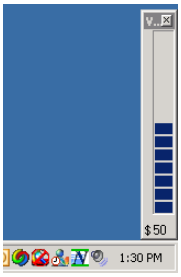
We have motivated the use of the periodic real-time model for virtual-machine-based distributed computing; the model allows us to straightforwardly mix batch and interactive VMs and allows users to succinctly describe their performance demands. We have designed and implemented a user-level scheduler for Linux that provides this model. We evaluated its performance on several different platforms and found that we can achieve very low deadline miss rates up to quite high utilizations and quite fine resolutions. Our scheduler has allowed us to mix long-running batch computations with fine-grained interactive applications such as first-person-shooter games with no reduction in usability of the interactive applications. It also lets us schedule parallel applications, effectively controlling their utilization without adverse performance effects, and allowing us to shield them from external load.

VSched is publicly released and can be downloaded from http://virtuoso.cs.northwestern.edu.

A natural next step after we developed VSched is to decide how to choose schedules straightforwardly for all kinds of VMs, and how to incorporate direct human input into the scheduling process.

For batch VMs with I/O and batch parallel VMs we envision the user manipulating the schedule to achieve a needed application-specific execution rate or efficiency. Alternatively, for an application that is run often, a user can readily map out the relationship between $(period, slice)$ and execution rate, which we will discuss in the next chapter, and then make that relationship available to others.

It is challenging to decide $(period, slice)$ constraints for interactive VMs in which users



(a) GUI

(b) Force Feedback Joystick

(c) Non-centering Joystick

(d) Precision Non-centering Joystick

Figure 4.12: Control interface. The combination of (a) and (c) is used in our study.

may have varying demands.

4.6 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 feedback[2], a non-centering joystick, and a precision non-centering joystick. These interfaces are illustrated in Figure 4.12. 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 as a starting point. In such a joystick, the control stalk maintains its current deflection even after the user removes his hand. The horizontal and vertical deflection are naturally 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 4.12(b)) to produce a non-centering joystick (Figure 4.12(c)) by removing a spring. This joystick is not as precise as a precision non-centering joystick (Figure 4.12(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,

[2]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.

the influence of the overhead is minimal, and the cost is effectively the utilization of the user's VM.

4.7 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.

4.7.1 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 (2GHz P4, 512MB mem, 17" monitor, 100mbit 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.

4.7.2 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 [176], 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 4.12(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 4.12(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 can be found in Appendix C.

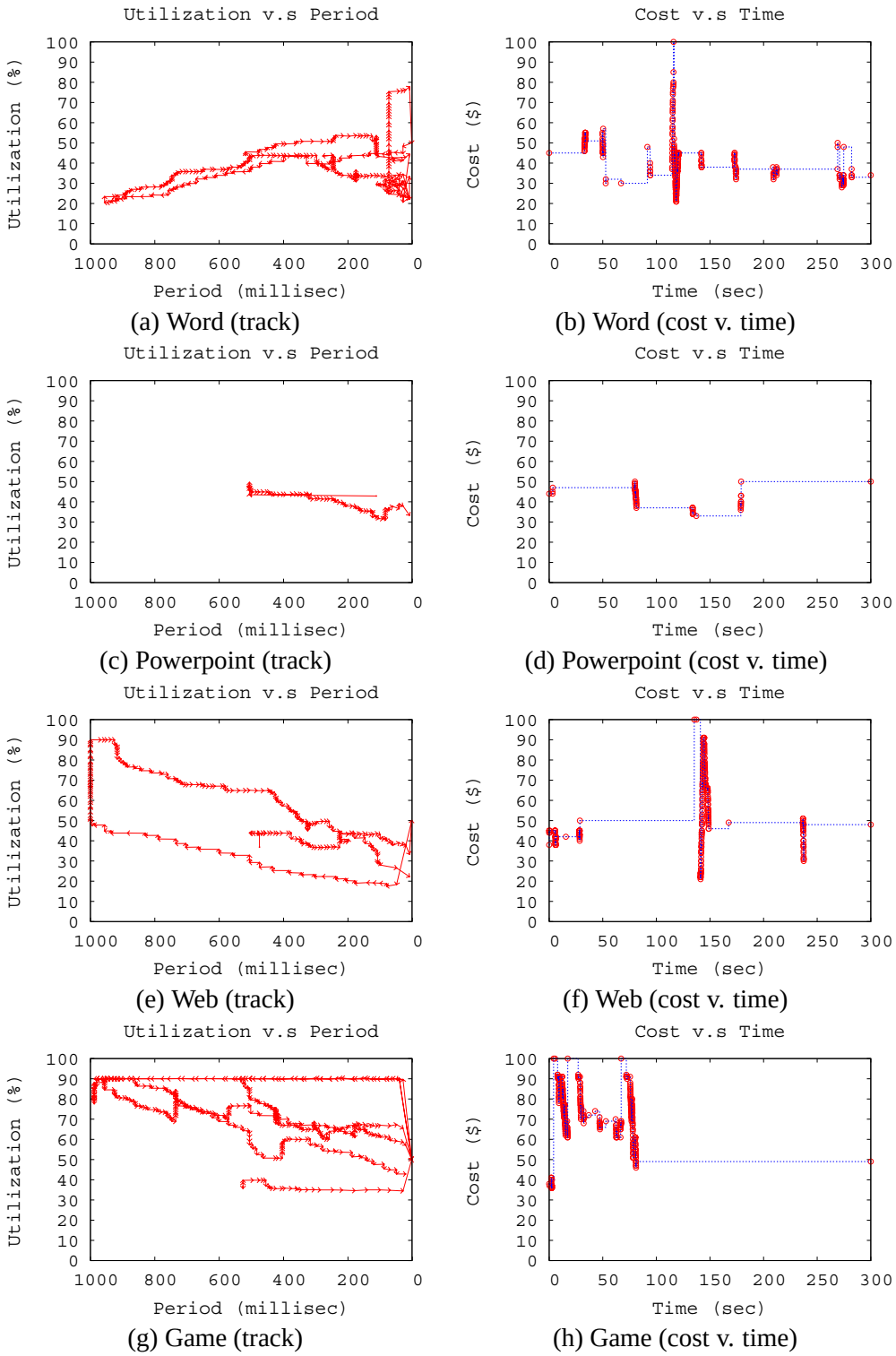
As the user performed the tasks, we recorded the following information:

- 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.

4.7.3 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 4.12(a)). We present the traces of three users as examples of different ways



(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 4.13: User A: Tracks, cost versus time.

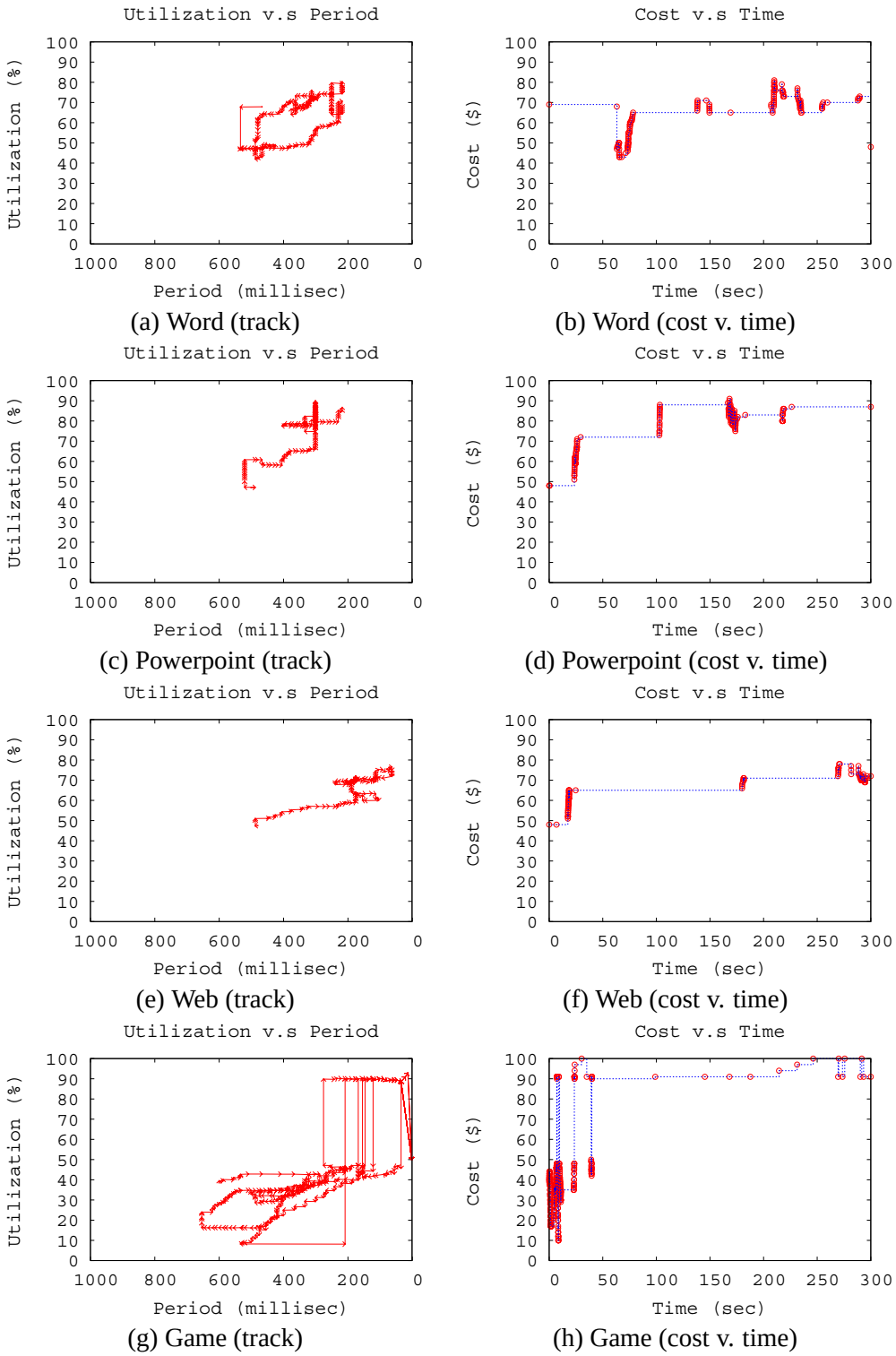
of interacting with the system, ranging from focusing first on the cost to focusing primarily on the comfort of the application.

Figure 4.13 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 4.13 is hunting through the space, smoothly changing the joystick to look for better prices. This is particularly evident in Figure 4.13(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 4.14 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 4.15 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



(a) Word (track)

(b) Word (cost v. time)

(c) Powerpoint (track)

(d) Powerpoint (cost v. time)

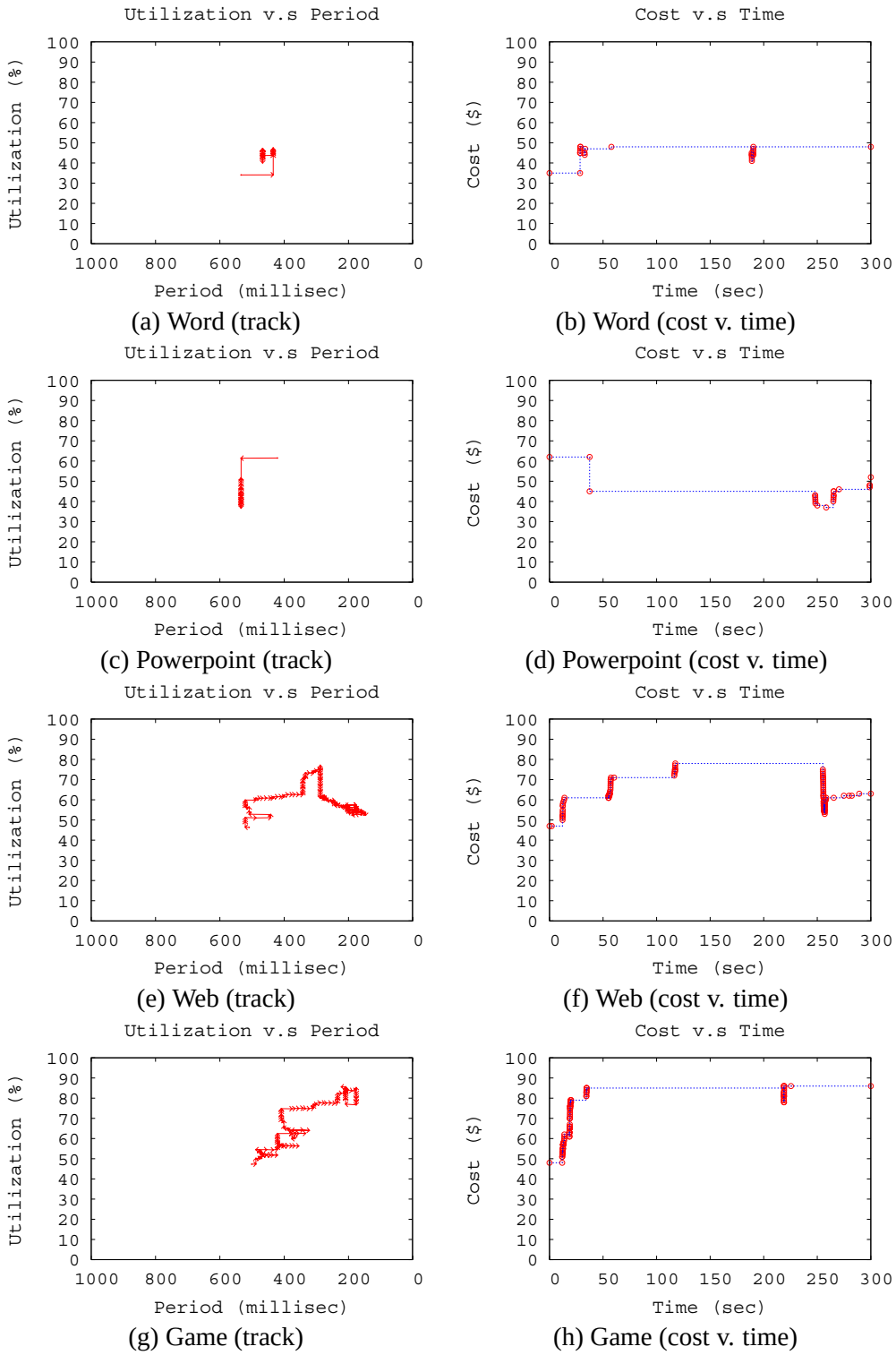
(e) Web (track)

(f) Web (cost v. time)

(g) Game (track)

(h) Game (cost v. time)

Figure 4.14: User B: Tracks, cost versus time.



(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 4.15: User C: Tracks, cost versus time.

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.

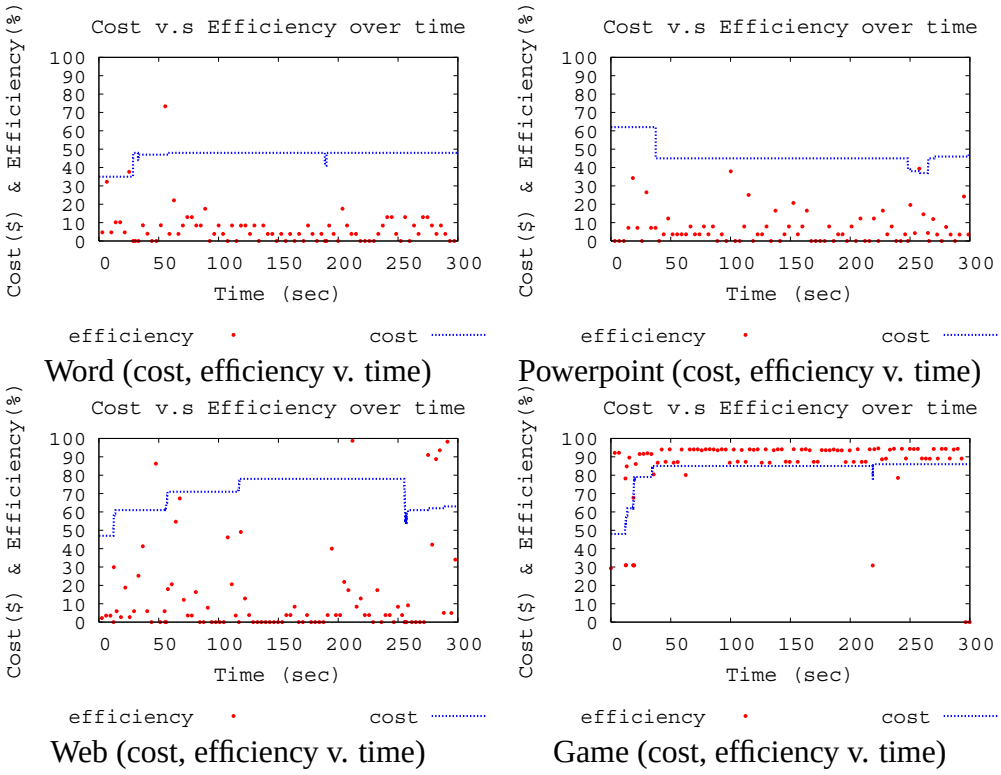


Figure 4.16: User C: Cost, efficiency versus time.

For this last user, Figure 4.16 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

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 4.17: Summary of user responses in study.

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 jitter of the application.

4.7.4 Quantitative Results

Figure 4.17 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.

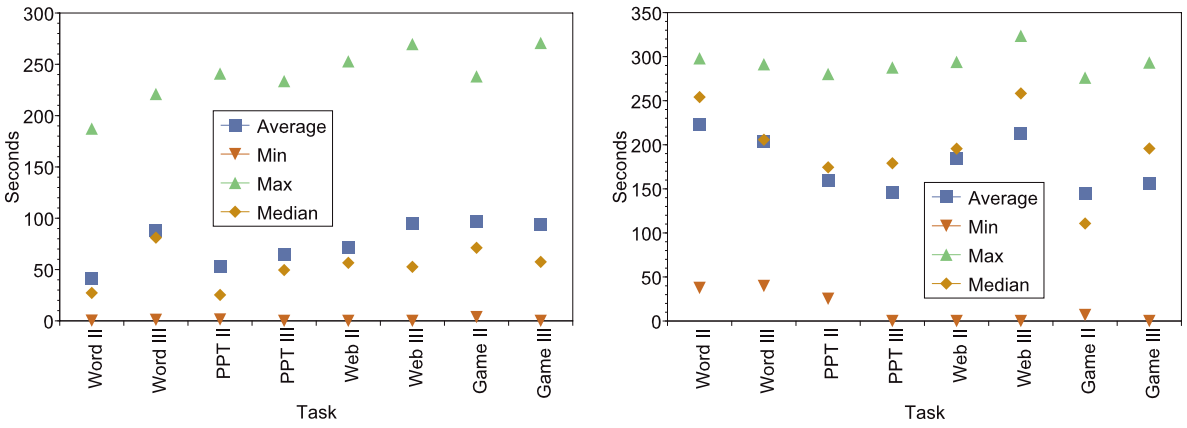
Despite the disparity among the applications and the users, there was little disparity in the users' reactions. There were only two situations where a significant fraction of the users had difficulty finding a comfortable or comfortable and low-cost setting. 28% of users had difficulty finding a comfortable 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 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 4.18 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.

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



(a) Duration to first encounter of lowest cost (b) Duration to last encounter of lowest cost

Figure 4.19: Duration to lowest cost.

- 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 (Chapter 2).

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

Figure 4.19(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 4.19(b) shows identical statistics for the duration to the last time the lowest cost occurred. Note that one of 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.

4.8 Conclusions

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.

This work provides evidence for the feasibility and effectiveness of human-driven

search part of my thesis. That is, it shows that it is possible to use direct human input to guide the search for a good configuration. Using VSched's joystick control, 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.

Chapter 5

Time-sharing Parallel Applications With Performance Isolation and Control

This dissertation argues for using direct human input to solve optimization problems. In previous chapters, we showed how we used human-driven specification and search to solve the CPU scheduling problem on a single machine. In this chapter, we apply human-driven specification to an optimization problem on multiple machines.

5.1 Motivation

Tightly-coupled computing resources such as clusters are typically used to run batch parallel workloads. An application in such a workload is typically communication intensive, executing synchronizing collective communication. The Bulk Synchronous Parallel (BSP) model [186] is commonly used to understand many of these applications. In the BSP model, application execution alternates between phases of local computation and phases of global collective communication. Because the communication is global, the threads of execution on different nodes must be carefully scheduled if the machine is time-shared. If a thread on one node is slow or blocked due to some other thread unrelated to the application, all of the application's threads stall.

To avoid stalls and provide predictable performance for users, almost all tightly-coupled computing resources today are space-shared. In space-sharing [178], each application is given a partition of the available nodes, and on its partition, it is the *only* application running, thus avoiding the problem altogether by providing complete performance isolation between running applications. Space-sharing introduces several problems, however. Most obviously, it limits the utilization of the machine because the CPUs of the nodes are idle when communication or I/O is occurring. Space-sharing also makes it likely that applications that require many nodes will be stuck in the queue for a long time and, when running, block many applications that require small numbers of nodes. Finally, space-sharing permits a provider to control the response time or execution rate of a parallel job at only a very coarse granularity. Though it can be argued theoretically that applications can be always built such that computation and I/O overlap all the time, thus preventing stalls, practically speaking, this is rarely the case. We propose a new self-adaptive approach to time-sharing parallel applications on tightly-coupled computing resources like clusters, *performance-targetted feedback-controlled real-time scheduling*. The goals of our technique are to provide

- performance isolation within a time-sharing framework that permits multiple applications to share a node, and
- performance control that allows the administrator to finely control the execution rate of each application while keeping its resource utilization automatically proportional to execution rate.

Conversely, the administrator can set a target resource utilization for each application and have commensurate application execution rates follow.

In performance-targetted feedback-controlled real-time scheduling, each node has a periodic real-time scheduler, VSched (Chapter 4).

Once an administrator has set a target execution rate for an application, a global controller determines the appropriate constraint for each of the application's threads of execution and then contacts each corresponding local scheduler to set it. The controller's input is the desired application execution rate, given as a percentage of its maximum rate on the system (i.e., as if it were on a space-shared system). The application or its agent periodically feeds back to the controller its current execution rate. The controller automatically adjusts the local schedulers' constraints based on the error between the desired and actual execution rate, with the added constraint that utilization must be proportional to the target execution rate. In the common case, the only communication in the system is the feedback of the current execution rate of the application to the global controller, and synchronization of the local schedulers through the controller is very infrequent. Section 5.2 describes the global controller in detail.

It is important to point out that our system schedules the CPU of a node, not its physical memory, communication hardware, or local disk I/O. Nonetheless, in practice, we can achieve quite good performance isolation and control even for applications making significant use of these other resources, as we show in our detailed evaluation (Section 5.3). Mechanisms for physical memory isolation in current OSes and VMMs are well understood and can be applied in concert with our techniques. As long as the combined working set size of the applications executing on the node does not exceed the physical memory of the machine, the existing mechanisms suffice. Communication has significant computational costs, thus, by throttling the CPU, we also throttle it. The interaction of our system and local disk I/O is more complex. Even so, we can control applications with considerable disk I/O.

The primary contributions of this work to the state of the art are the following:

- We have described, implemented, and evaluated a new approach to time-sharing

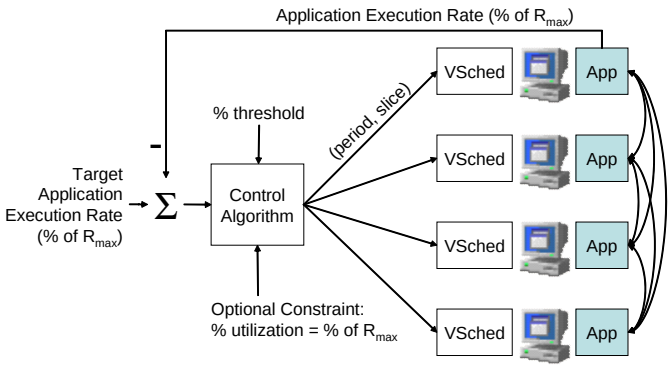


Figure 5.1: Structure of global control.

parallel applications with performance isolation. The approach is based on periodic real-time scheduling of the nodes combined with global control of the real-time constraints.

- We have demonstrated that this approach also provides a simple way to control the execution rate of applications while maintaining efficiency.

5.2 Global controller

The control system consists of a centralized feedback controller and multiple host nodes, each running a local copy of VSched, as shown in Figure 5.1. A VSched daemon is responsible for scheduling the local thread(s) of the application(s) under the yoke of the controller. The controller sets $(period, slice)$ constraints using the mechanisms described in Chapter 4. Currently, the same constraint is used for each VSched. One thread of the application, or some other agent, periodically communicates with the controller using non-blocking communication.

5.2.1 Inputs

The maximum application execution rate on the system in application-defined units is R_{max}. The set point of the controller is supplied by the user or the system administrator through a command-line interface that sends a message to the controller. The set point is r_{target} and is a percentage of R_{max}. The system is also defined by its threshold for error, ε, which is given as percentage points. The inputs Δ_{slice} and Δ_{period} specify the smallest amounts by which the slice and period can be changed. The inputs min_{slice} and min_{period} define the smallest slice and period that VSched can achieve on the hardware.

The current utilization of the application is defined in terms of its scheduled period and slice, $U = slice/period$. The user requires that the utilization be proportional to the target rate, that is, that $r_{target} - \varepsilon \leq U \leq r_{target} + \varepsilon$.

The feedback input $r_{current}$ comes from the parallel application we are scheduling and represents its current execution rate as a percentage of R_{max}. To minimize the modification of the application and the communication overhead, our approach only requires high-level knowledge about the application's control flow and only a few extra lines of code.

5.2.2 Control algorithm

The control algorithm (or simply the algorithm) is responsible for choosing a $(period, slice)$ constraint to achieve the following goals

1. The error is within threshold: $r_{current} = r_{target} \pm \varepsilon$, and
2. That the schedule is efficient: $U = r_{target} \pm \varepsilon$.

The algorithm is based on the intuition and observation that application performance will vary depending on which of the many possible $(period, slice)$ schedules corresponding to a given utilization U we choose, and the best choice will be application dependent and vary

with time. For example, a finer grain schedule (e.g. (20ms, 10ms)) may result in better application performance than coarser grain schedules (e.g. (200ms, 100ms)). At any point in time, there may be multiple "best" schedules.

The control algorithm attempts to automatically and dynamically achieve goals 1 and 2 in the above, maintaining a particular execution rate r_{target} specified by the user while keeping utilization proportional to the target rate.

We define the error as

$$e = r_{current} - r_{target}.$$

At startup, the algorithm is given an initial rate r_{target}. It chooses a $(period, slice)$ constraint such that $U = r_{target}$ and *period* is set to a relatively large value such as 200 ms. The algorithm is a simple linear search for the largest *period* that satisfies our requirements.

When the application reports a new current rate measurement $r_{current}$ and/or the user specifies a change in the target rate r_{target}, e is recomputed, followed by:

- If $|e| > \varepsilon$ decrease *period* by Δ_{period} and decrease *slice* by Δ_{slice} such that $slice/period = U = r_{target}$. If $period \leq min_{period}$ then we reset *period* to the same value as used at the beginning and again set *slice* such that $U = r_{target}$.

- If $|e| \leq \varepsilon$ do nothing.

It should be noticed that the algorithm always maintains the target utilization and searches the $(period, slice)$ space from larger to smaller granularity, subject to the utilization constraint. The linear search is, in part, done because multiple appropriate schedules may exist. We do not preclude the use of algorithms that walk the space faster, but we have found our current algorithm to be effective.

5.3 Evaluation

In presenting our evaluation, we begin by explaining the experimental framework. Then we show the range of control that the scheduling system has made available. This is followed by an examination of using the algorithm described above to prevent the inevitable drift associated with simply using a local real-time scheduler. Next, we examine the performance of the algorithm in a dynamic environment, showing their reaction to changing requirements. We then illustrate how the system remains impervious to external load despite the feedback. Next, we show how the system scales as it controls increasing numbers of parallel applications. Finally, we examine the effects of local disk I/O and memory contention.

5.3.1 Experimental framework

As mentioned previously, Bulk Synchronous Parallel (BSP [67]) model is used to characterize many of the batch parallel workloads that run in tightly coupled computing resources such as clusters. In most of our evaluations we used a synthetic BSP benchmark, called Patterns, written for PVM [66]. Patterns is described in more detail in a previous paper [74], but the salient points are that it can execute any BSP communication pattern and run with different compute/communicate (comp/comm) ratios and granularities. In general, we configure Patterns to run with an all-to-all communication pattern on four nodes of our IBM e1350 cluster (Intel® Xeon® 2.0 GHz, 1.5 GB RAM, Gigabit Ethernet interconnect, Linux 2.4.20). Each node runs VSched, and a separate node is used to run the controller. Note that all of our results involve CPU and network I/O.

We also evaluated the system using an NAS (NASA Advanced Supercomputing) benchmark. In particular, we use the PVM implementation of the IS (Integer Sort) benchmark developed by White et al. [198]. It performs a large integer sort, sorting keys in parallel

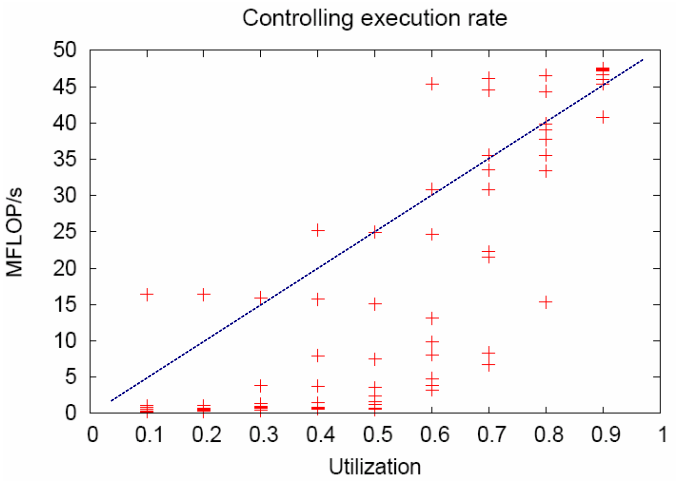


Figure 5.2: Compute rate as a function of utilization for different $(period, slice)$ choices.

as seen in large scale computational fluid dynamic (CFD) applications. IS combines integer computation speed and communication with, unlike Patterns, different nodes doing different amounts of computation and communication.

5.3.2 Range of control

To illustrate the range of control possible using periodic real-time scheduling on the individual nodes, we ran Patterns with a compute/communicate ratio of 1:2, making it quite communication intensive. Note that this configuration is conservative: it is far easier to control a more loosely coupled parallel application with VSched. We ran Patterns repeatedly, with different $(period, slice)$ combinations. Figure 5.2 shows these test cases. Each point is an execution of Patterns with a different $(period, slice)$, plotting the execution rate of Patterns as a function of Patterns utilization on the individual nodes. Notice the line on the graph, which is the ideal control curve that the control algorithm is attempting to achieve, control over the execution rate of the application with proportional utilization ($r_{current} = r_{target} = U$). Clearly, there *are* choices of $(period, slice)$ that allow us to meet

all of the requirements.

5.3.3 Schedule selection and drift

Although there clearly exist $(period, slice)$ schedules that can achieve an execution rate with (or without) proportional utilization, we cannot simply use only the local schedulers for several reasons:

- The appropriate $(period, slice)$ is application dependent because of differing compute/communicate ratios, granularities, and communication patterns. Making the right choice should be automatic.
- The user or system administrator may want to dynamically change the application execution rate r_{target}. The system should react automatically.
- Our implementation is based on a *soft* local real-time scheduler. This means that deadline misses will inevitably occur and this can cause timing offsets between different application threads to accumulate. We must monitor and correct for these slow errors. Notice that this is likely to be the case for a hard local real-time scheduler as well if the admitted tasks vary across the nodes.

Figure 5.3 illustrates what we desire to occur. The target application execution rate is given in iterations per second, here being 0.006 iterations/second. The current execution rate $r_{current}$ is calculated after each iteration and reported to the controller. This is Patterns running with a 1:1 compute/communicate ratio on two nodes. The lower curve is that of simply using VSched locally to schedule the application. Although we can see that the rate is correct for the first few iterations, it then drifts downward, upward, and once again downward over the course of the experiment. The roughly straight curve is using VSched, the global controller, and the control algorithm. We can see that the tendency to drift has been eliminated using global feedback control.

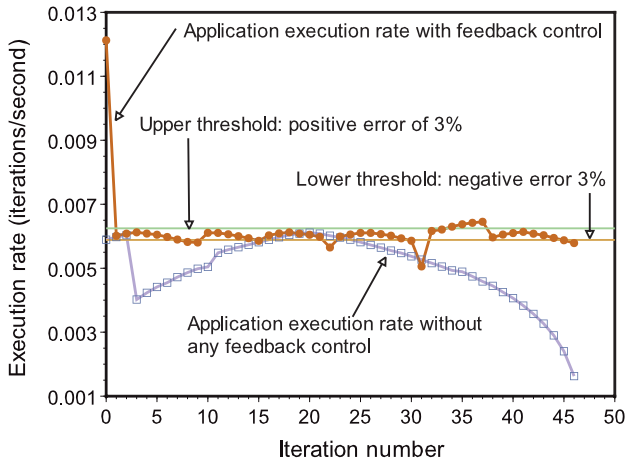
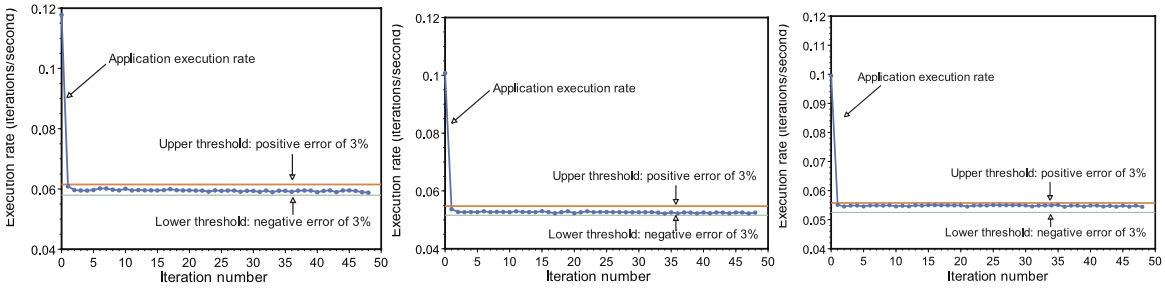


Figure 5.3: Elimination of drift using global feedback control; 1:1 comp/comm ratio.



(a) high (5:1) comp/comm ratio

(b) medium (1:1) comp/comm ratio

(c) low (1:5) comp/comm ratio

Figure 5.4: System in stable configuration for varying comp/comm ratio.

5.3.4 Evaluating the control algorithm

We studied the performance of the control algorithm using three different compute / communicate ratios (high (5:1) ratio, medium (1:1) ratio, and low (1:5) ratio), different target execution rates r_{target}, and different thresholds ε. In all cases $\Delta_{period} = 2$ ms, where Δ_{period} is the change in period effected by VSched when the application execution rate goes outside of the threshold range, the *slice* is then adjusted such that $U = r_{target}$.

Figure 5.4 shows the results for high, medium, and low test cases with a 3% threshold. We can see that the target rate is easily and quickly achieved, and remains stable for all three test cases. Note that the execution rate of these test cases running at full speed without any scheduling are slightly different. $r_{current}$ is calculated in the end of every iteration.

Next, we focus on two performance metrics:

- Minimum threshold: What is the smallest ε below which control becomes unstable?
- Response time: for stable configurations, what is the typical time between when the target execution rate r_{target} changes and when the $r_{current} = r_{target} \pm \varepsilon$?

Being true for all feedback control systems, the error threshold will affect the performance of the system. When the threshold ε is too small, the controller becomes unstable and fails because the change applied by the control system to correct the error is even greater than the error. For our control algorithm, when the error threshold is $< 1\%$, the controller will become unstable. Figure 5.5 illustrates this behavior. Note that while the system is now oscillating, it appears to degrade gracefully.

Figure 5.6 illustrates our experiment for measuring the response time. The target rate is changed by the user in the middle of the experiment. Our control system quickly adjusts the execution rate and stabilizes it. It shows that the response time is about 32 seconds, or two iterations, for the case of 1:1 compute/communicate ratio. The average response

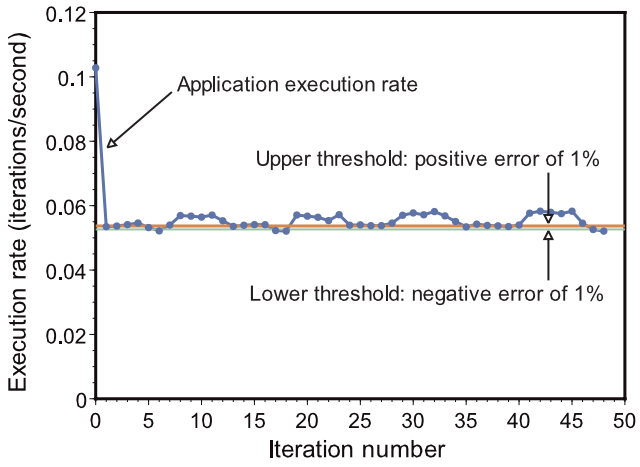


Figure 5.5: System in oscillation when error threshold is made too small; 1:1 comp/comm ratio.

time over four test cases (1 high, 2 medium, and 1 low compute/communicate ratios) is 30.68 seconds. In all cases, the control algorithm maintains $U = r_{target}$ as an invariant by construction.

5.3.5 Summary of limits of control algorithm

Figure 5.7 summarizes the response time, communication cost to support the feedback control, and threshold limits of our control system. Overall we can control with a quite small threshold ε. The system responds quickly, on the order of a couple of iterations of our benchmark. The communication cost is minuscule, on the order of just a few bytes per iteration. Finally, these results are largely independent of the compute/communicate ratio.

The exceptionally low communication involved in performance-targetted feedback-controlled real-time scheduling is a natural consequence of the deterministic and predictable periodic real-time scheduler being used on each node.

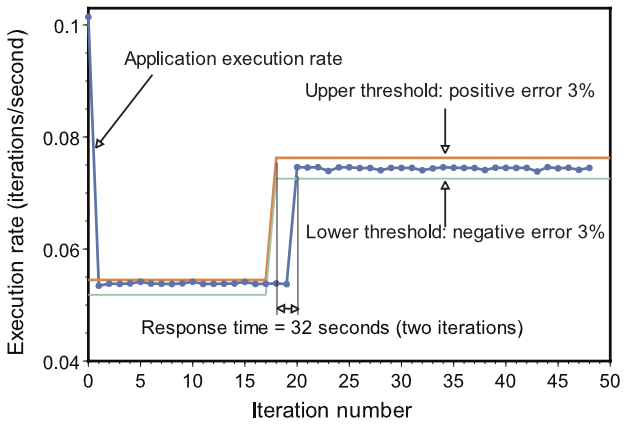


Figure 5.6: Response time of control algorithm; 1:1 comp/comm ratio.

High (5:1) compute /communicate ratio			Medium (1:1) compute /communicate ratio			Low (1:5) compute /communicate ratio		
Response time	Threshold limit	Feedback comm. cost	Response time	Threshold limit	Feedback comm. cost	Response time	Threshold limit	Feedback comm. cost
29.16 s	2 %	32 bytes /iter	31.33 s	2 %	32 bytes /iter	32.01 s	2 %	32 bytes /iter

Figure 5.7: Response time and threshold limits for the control algorithm.

5.3.6 Dynamic target execution rates

As we mentioned earlier, using the feedback control mechanism, we can dynamically change the target execution rates and our control system will continuously adjust the real-time schedule to adapt to the changes. To see how our system reacts to user inputs over time, we conducted an experiment in which the user adjusted his desired target rate four times during the execution of the Patterns application. As shown in Figure 5.8, the control algorithm works well. After the user changes the target rate, the algorithm quickly adjusts the schedule to reach the target.

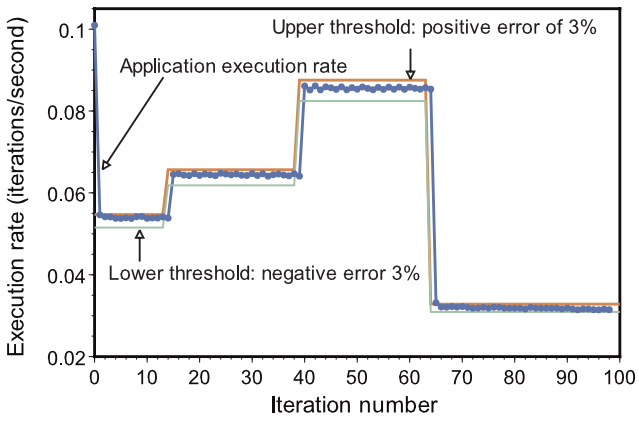


Figure 5.8: Dynamically varying execution rates; 1:1comp/comm ratio.

5.3.7 Ignoring external load

Any coupled parallel program can suffer drastically from external load on any node; the program runs at the speed of the slowest node. We have previously shown that the periodic real-time model of VSched can shield the program from such external load, preventing the slowdown [109]. Here we want to see whether our control system as a whole can still protect a BSP application from external load.

We executed Patterns on four nodes with the target execution rate set to half of its maximum rate. On one of the nodes, we applied external load, a program that contends for the CPU using load trace playback techniques [43]. Contention is defined as the average number of contention processes that are runnable. Figure 5.9 illustrates the results. At roughly the 15th iteration, an external load is placed on one of the nodes in which Patterns is running, producing a contention of 1.0. We note that the combination of VSched and the feedback controller are able to keep the performance of Patterns independent of this load. We conclude that our control system can help a BSP application maintain a fixed stable performance under a specified execution rate constraint despite external load.

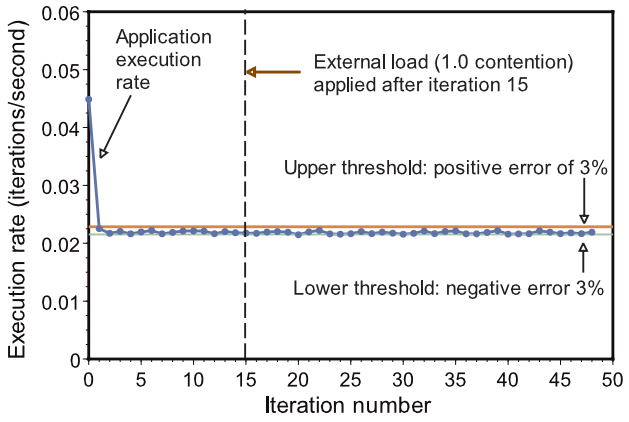


Figure 5.9: Performance of control system under external load; 3:1 comp/comm ratio; 3% threshold.

5.3.8 NAS IS Benchmark

When we ran the NAS IS (Integer Sort) benchmark without leveraging our control system, we observed that different nodes have different CPU utilizations. This is very different from the Patterns benchmark, which does roughly the same amount of computation and communication on each node. In our experiment, for a specific configuration of NAS IS executing on four nodes, we observed an average utilization of $\sim$28% for two nodes and $\sim$14% average utilization for the other two nodes.

This variation has the potential to challenge our control system, since in our model we assume the same target utilization U on each node, and we apply the same schedule on each node. We ran an experiment where we set the target utilization to be half of the maximum utilization among all nodes, i.e. 14%. Figure 5.10 illustrates the performance in this case. We can see that the actual execution rate is successfully brought to within ε of the target rate.

We are currently designing a system in which the global controller is given the freedom

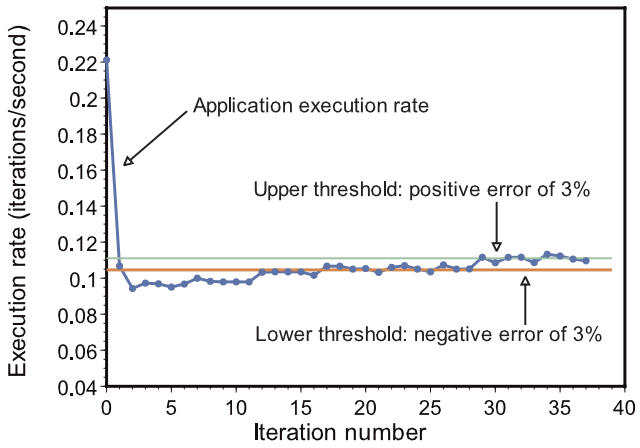


Figure 5.10: Running NAS benchmark under control system; 3% threshold.

to set a different schedule on each node thus making our control system more flexible.

5.3.9 Time-sharing multiple applications

To see how well we can provide time-sharing for multiple parallel applications, we simultaneously executed multiple Patterns benchmarks on the same four nodes of our cluster.

Figure 5.11 shows the results of running two Patterns applications, each configured with a 1:1 compute/communicate ratio. One was configured with a target rate of 30%, with the other set to 40%. We can clearly see that the actual execution rates are quickly brought to within ε of the target rates and remain there for the duration of the experiment.

Next, we consider what happens as we increase the number of Patterns benchmarks running simultaneously. In the following, each Patterns benchmark is set to execute with identical 10% utilization. We ran Patterns with a 3:1 compute/communicate ratio. Figure 5.12 shows our results. Each graph shows the execution rate (iterations/second) as a function of the iteration, as well as the two 3% threshold lines. Figure 5.12(a) contains two such graphs, corresponding to two simultaneously executing Patterns benchmarks, (b) has three, and so on.

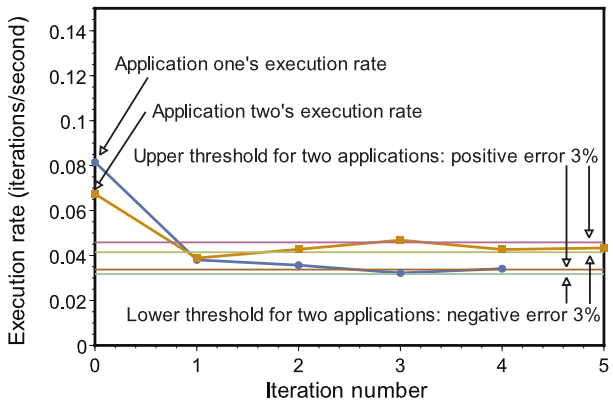
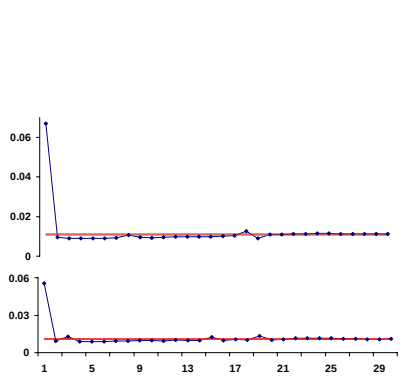


Figure 5.11: Running of two Patterns benchmarks under the control system, 1:1 comp/comm ratio.

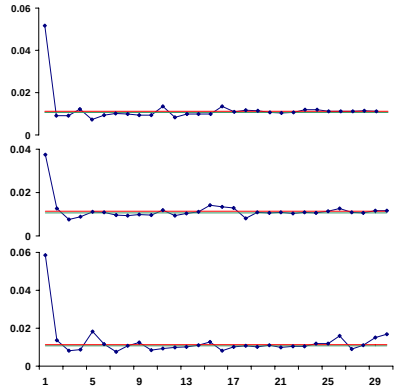
Overall, we maintain reasonable control as we scale the number of simultaneously executing benchmarks. Further, over the thirty iterations shown, in all cases, the average execution rate meets the target, within threshold.

We do notice a certain degree of oscillation when we run many benchmarks simultaneously. Our explanation is as follows. When VSched receives and admits a new schedule sent by the global controller, it will interrupt the current task and re-select a new task (perhaps the previous one) to run based on its deadline queue. As the number of parallel applications increases, each process of an application on an individual node will have a smaller chance of running uninterrupted throughout its slice. In addition, there will be a smaller chance of each process starting its slice at the same time.

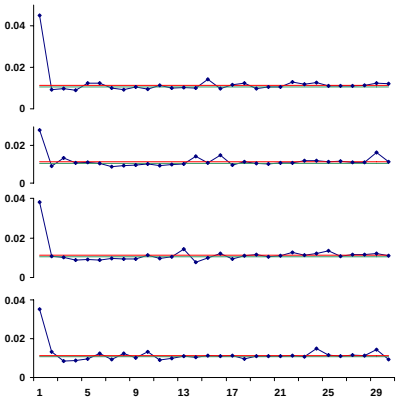
The upshot is that even though the process will continue to meet its deadlines locally, it will be less synchronized with processes running on other nodes. This results in the application's overall performance changing, causing the global controller to be invoked more often. Because the control loop frequency is less than the frequency of these small performance changes, the system begins to oscillate. However, the degradation is graceful,



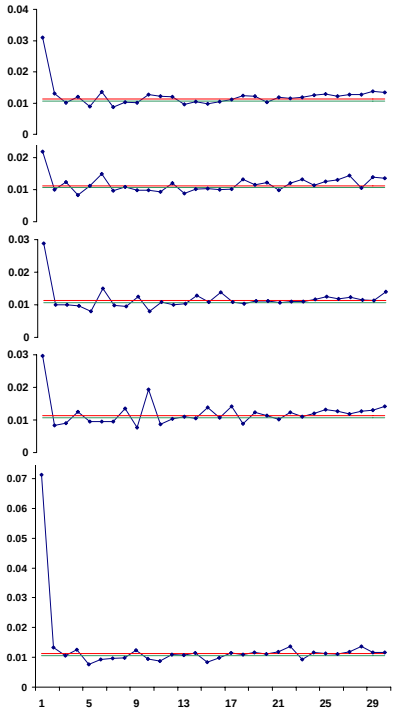
(a) 2 benchmarks



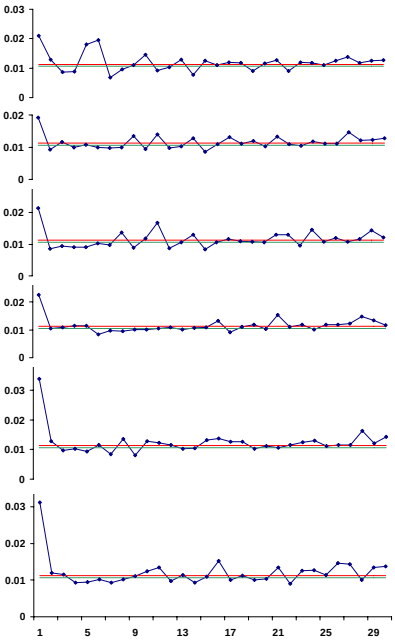
(b) 3 benchmarks



(c) 4 benchmarks

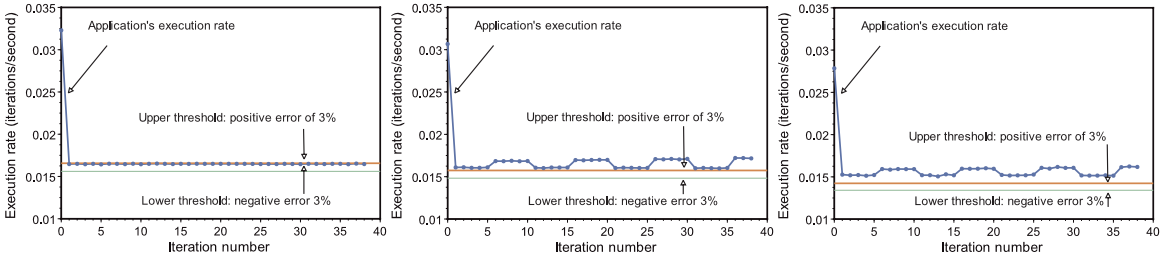


(d) 5 benchmarks



(e) 6 benchmarks

Figure 5.12: Running multiple Patterns benchmarks; 3:1 comp/comm ratio; 3% threshold.



(a) 10 MB/node/iter I/O (b) 20 MB/node/iter I/O (c) 40 MB/node/iter I/O

Figure 5.13: Performance of control system with a high (145:1) comp/comm ratio and varying local disk I/O.

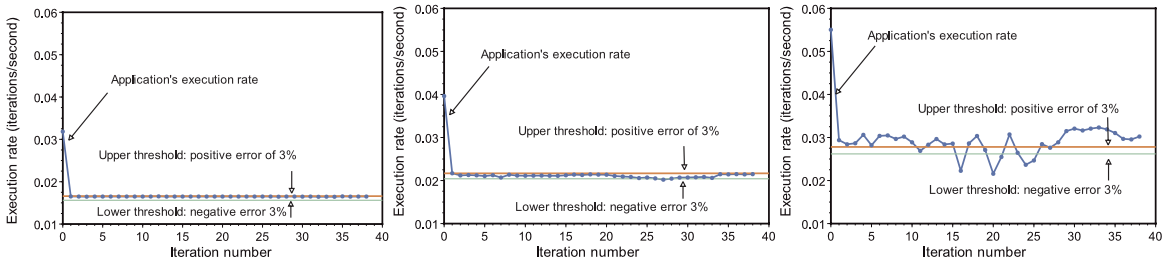
and, again, the long term averages are well behaved.

5.3.10 Effects of local disk I/O

Although we are only scheduling the CPU resource, it is clear from the above that this is sufficient to isolate and control a BSP application with complex collective communications of significant volume. Is it sufficient to control such an application when it also extensively performs local disk I/O?

To study the effects of local disk I/O on our scheduling system, we modified the Patterns benchmark to perform varying amounts of local disk I/O. In the modified Patterns, each node writes some number of bytes sequentially to the local IDE hard disk during each iteration. It is ensured that the data is written to the physical disk by using `fsync()` call.

In our first set of experiments, we configured Patterns with a very high (145:1) compute/communicate ratio, and 0, 1, 5, 10, 20, 40, and 50 MB per node per iteration of local disk I/O. Our target execution rate was 50% with a threshold of 3%. Figure 5.13 shows the results for 10, 20, and 40 MB/node/iter. 0, 1, 5 are similar to 10, while 50 is similar to 40. For up to 10 MB/node/iter, our system effectively maintains control of the application's execution rate. As we exceed this limit, we develop a slight positive bias; the application runs faster than desired despite the restricted CPU utilization. The dominant part of the



(a) high (4900:1) comp/comm ratio

(b) medium (2:1) comp/comm ratio

(c) low (1:3.5) comp/comm ratio

Figure 5.14: Performance of control system with 10 MB/node/iter of disk I/O and varying comp/comm ratios.

time spent on local disk I/O is spent waiting for the disk. As more I/O is done, a larger proportion of application execution time is outside of the control of our system. Since the control algorithm requires that the CPU utilization be equal to the target execution rate, the actual execution rate grows. In the second set of experiments, we fixed the local disk I/O to 10 MB/node/iter (the maximum controllable situation in the previous experiment) and varied the compute/communicate ratio, introducing different amounts of network I/O. We used a target rate of 50%. We used seven compute/communicate ratios ranging from 4900:1 to 1:3.5. Figure 5.14 shows the results for 4900:1, 2:1, and 1:3.5. For high to near 1:1 compute/communicate ratios, our system can effectively control the application's execution rate even with up to 10 MB/node/iteration of local I/O, and degrades gracefully after that.

Our system can effectively control the execution rates of applications performing significant amounts of network and local disk I/O. The points at which control effectiveness begins to decline depends on the compute/communicate ratio and the amount of local disk I/O. With higher ratios, more local disk I/O is acceptable. We have demonstrated control of an application with a 1:1 ratio and 10 MB/node/iter of local disk I/O.

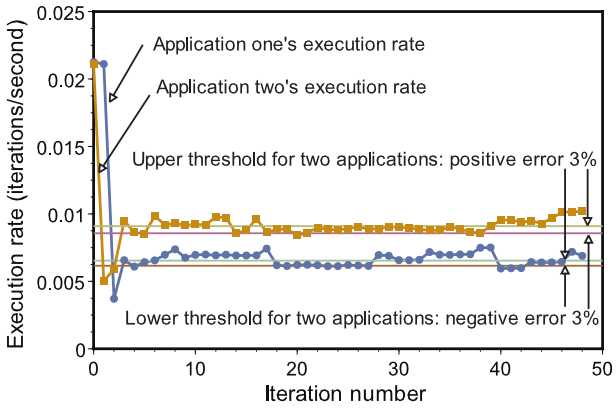


Figure 5.15: Running two Patterns benchmarks under the control system; high (130:1) comp/comm ratio. The combined working set size is slightly less than the physical memory.

5.3.11 Effects of physical memory use

Our technique makes no attempt to isolate memory, but the underlying node OS certainly does so. Is it sufficient?

To evaluate the effects of physical memory contention on our scheduling system, we modified the Patterns benchmark so that we could control its working set size. We then ran two instances of the modified benchmark simultaneously on the four nodes of our cluster. We configured the first instance with a working set of 600 MB and a target execution rate of 30%, while the second was configured with a working set size of 700 MB and a target rate of 40%. Both instances had a compute/communicate ratio of around 130:1. The combined working set of 1.3 GB is slightly less than the 1.5 GB of memory of our cluster nodes.

We used the control algorithm to schedule the two instances, and Figure 5.15 shows the results of this experiment. We see that despite the significant use of memory by both instances, our system maintains control of both applications' execution rates.

Our results suggest that unless the total working set on the machine is exceeded, phys-

ical memory use has little effect on the performance of our scheduling system. It is important to point out that most OS kernels, including Linux, have mechanisms to restrict the physical memory use of a process. These mechanisms can be used to guarantee that the physical memory pressure on the machine does not exceed the supply. A virtual machine monitor such as Xen or VMware provides additional control, enforcing a physical memory limit on a guest OS kernel and all of its processes.

5.4 Conclusions

We have proposed, implemented, and evaluated a new self-adaptive approach to time-sharing parallel applications on tightly coupled compute resources such as clusters. Our technique, performance-targetted feedback-controlled real-time scheduling, is based on the combination of local scheduling using the periodic real-time model and a global feedback control system that sets the local schedules. The approach performance-isolates parallel applications and allows administrators to dynamically change the desired application execution rate while keeping actual CPU utilization automatically proportional to the application execution rate. Our implementation takes the form of a user-level scheduler for Linux and a centralized controller. Our evaluation shows the system to be stable with low response times. The thresholds needed to prevent control instability are quite reasonable. Despite only isolating and controlling the CPU, we find that memory, communication I/O, and local disk I/O follow.

This work shows the effectiveness of human-driven specification on an otherwise difficult multi-machine optimization problem. In this case, the administrator directly specifies the objective function — desired target execution rate for each of his parallel applications, our system then automatically adjusts the applications' real-time schedules to achieve those rates with proportional CPU utilization.

Chapter 6

Application Trace-driven Simulator for Virtuoso

With the increasing complexity of the optimization problems that I address in the next chapter, the design and evaluation of my human-driven techniques would become more difficult and more time-consuming. In this chapter, I discuss the design and development of an application-trace-driven simulator. I decided to develop and use the simulator for the following reasons.

- My proposed human-driven approach focuses on how to allow direct user input in the specification of a adaptation problem and in the search for an optimal or good solution to the problem. In previous chapters, I solved various problems with increasing difficulty involving different real applications. For those interactive applications, it was essential for the user to be able to directly interact with the application. However, for non-interactive applications, such as batch applications, the interactivity between the user and the application itself becomes much less direct and frequent, compared with the interactivity between the user and our human-driven interface. Using an application simulator will let us concentrate on the latter interaction and the design of the interface.

- A non-interactive application typically runs for a long time. Given any user interface, it is unlikely that a system administrator or application user wants to keep monitoring the interface all the time. In reality, he may change some parameters using the interface, leave for a while, come back and adjust the application again based on the current progress of the application. During the design and development of my human-driven techniques, I want to test my interface with real users, for example, in a user study. It would be very time consuming if we use a real non-interactive application which may run for days and receive only sporadic user inputs. With a simulator, we can fast forward the application to those interesting points so that status of the application is updated more frequently to the user, thus triggering more frequent user input.

- The setup and teardown of a batch application running on multiple machines can be tedious and non-trivial. In human-driven search, we want different users to try different configurations and search for the optimal one. This requires that we can quickly re-run the application many times with constant initial state. A simulator makes this straightforwarded.

- Migrating a VM is not instant in a real system. If we allow a user to change the mapping from VMs to hosts through the interface, during the design and test stage, we must allow him to see the effect of migration quickly enough so that he can try a new mapping, considering the huge space of all possible mappings. A simulator will help. We will discuss the VM migration issues in more details later.

After we find a good interface for human-driven search using the combination of the simulator and the interface, we can modify the interface and connect it to the real system and application.

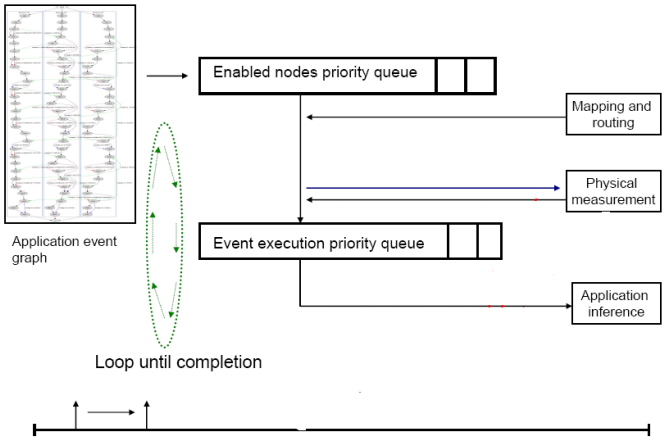


Figure 6.1: Structure of the simulator

In this chapter, I present the design, implementation, verification and validation of our trace-driven simulator.

The first generation of the simulator was developed by my colleague Ananth I. Sundararaj [181]. I implemented an event graph generator to generate traces for the simulator. I also participated in the validation and verification of the original simulator. As part of my dissertation, I designed and implemented the second generation of the simulator. In the remainder of this chapter, for completeness, I will first describe the first generation simulator. More details can be found in [181]. Then I will discuss the design and development of the second generation simulator and its validation.

6.1 First generation simulator

The purpose of the trace-driven simulator is to reasonably simulate the running of applications involving both computation and communication within the Virtuoso environment. Figure 6.1 shows the structure and working of the simulator. The high-level idea is to collect traces of real applications under a tightly controlled setup (our IBM e1350 cluster) and then to replay the traces under real world scenarios to study effects of adaptation.

The simulator described here is modeled on the Virtuoso system. The high-level design is generic and we hope that such a design with minor modifications would also be applicable to other environments, e.g. [15].

Being true for any simulator, our simulator has limitations. It does not model computer system memory or disks. It models computation, communication and the computation costs of communication. The simulator uses real world network measurements as input data, however, it does not uses a real model for time of day effects. Although the simulator takes into account cross traffic, it does not account for sudden instantaneous spikes in CPU or network traffic (unless fed in at the begining of the simulation).

6.1.1 Input

Application trace An application trace is an event graph represented as a directed acyclic graph (DAG). Figure 6.2 shows a sample plot of a small trace generated for a short run of the Patterns benchmark executing on three hosts. An event graph consists of nodes and edges. It starts at a node called start and ends at a node called end. The progression of each application component executing inside a VM is represented by a series of nodes connected by plain or annotated edges. In Figure 6.2, each such application component is enclosed in a rectangular box.

There are four possible types of edges in the event graph:

- Compute edge: Such an edge is labeled with the amount of compute operations that are to be performed on that node starting at that point in time. The origination node of the edge represents the VM state before the start of the computation and the destination node of the edge represents the VM state at the end of the computation.

- Communicate edge: This edge is labeled with the amount of bytes that are to be sent from one VM to another. Again, the origination node represents the state of the

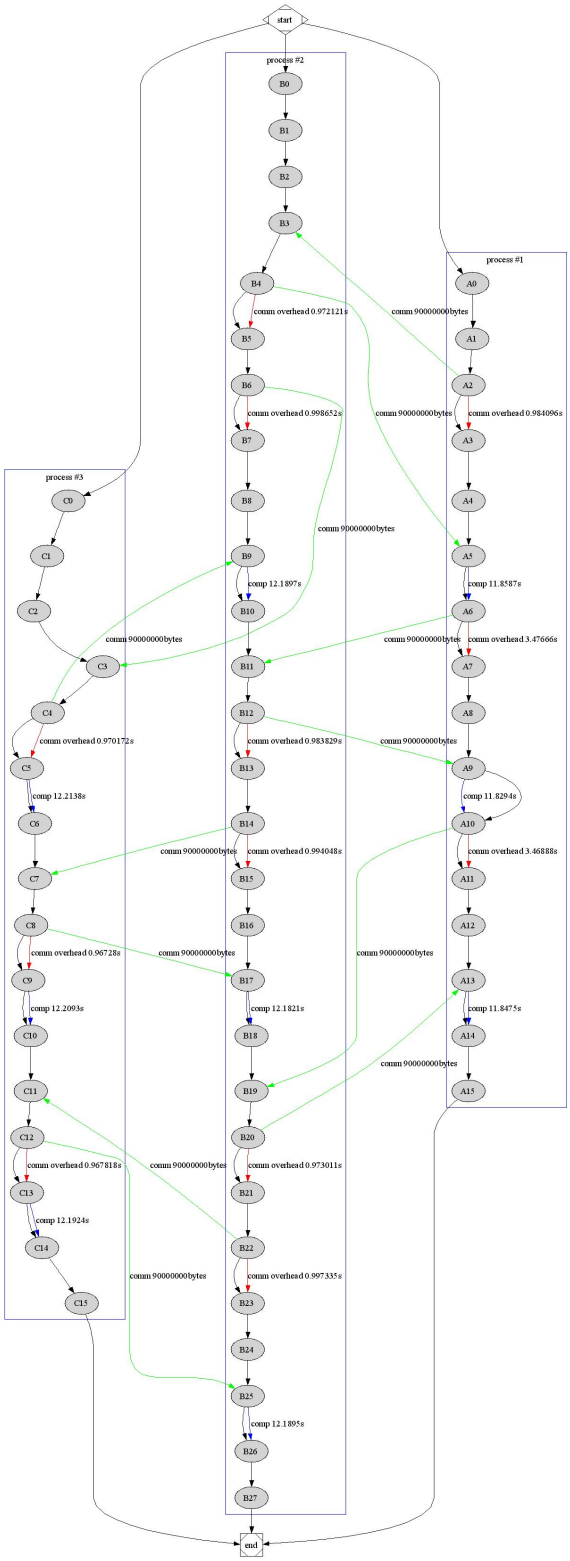


Figure 6.2: Sample input application trace, 3 nodes, Patterns benchmark, bus topology, 3 iterations.

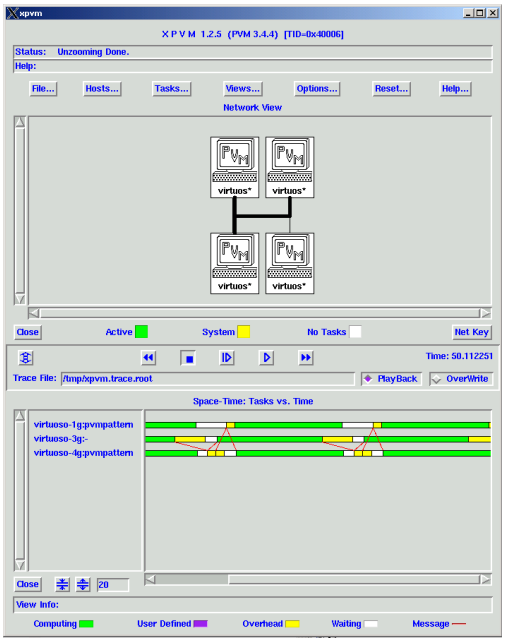


Figure 6.3: XPVM output for Patterns benchmark executing on 3 hosts.

sending VM before it starts sending the data and the destination node represents the state of the receiving VM after it has received all the data.

- Communication overhead edge: This edge captures the computation overhead of communication. The amount of time between starting of "send" operation on a VM until its completion.

- Precedence edge: Some of the edges are not annotated (are instead plain). These edges simply make the graph visually understandable. They are ignored by the simulator.

Trace generation Since we use PVM versions of high performance computing applications, we were able to obtain application trace data using the XPVM [102] tracing facility. XPVM supports a buffering mechanism to reduce the perturbation of user applications

caused by tracing, and a more flexible trace event definition scheme which is based on a self-defining data format. The tracing instrumentation is built into the PVM library. Figure 6.3 shows a snapshot of the XPVM tracing interface for the Patterns benchmark executing on three nodes of our cluster. XPVM will output formatted trace data. I developed an event graph generator which can convert the raw XPVM traces into the DAG input format accepted by the simulator.

Note that for all traces, we gather them by running Patterns on isolated and unloaded nodes interconnected via a Gigabit switch.

Physical system measurement data In our physical system, tools such as Wren [77] and Ganglia [133] can provide system resource measurements. One of the inputs to the simulator is a file containing such data. The file is populated with previously measured data. On the network end, it contains bandwidth and latency measurements. It also contains the CPU frequency of hosts.

Initial VM to host mapping and routing information Since the simulator models the Virtuoso system, it is fed with an initial VM to host mapping and a start topology routing configuration.

6.1.2 Output

At the end of a simulator run, we print out the user time, system time, and idle time for each application component executing in a VM. In addition, we also print out application specific metrics, such as iterations per second (throughput) for the Patterns benchmark, messages per second for a mail application, etc. Additionally, we also maintain a detailed log that records everything that happens in the lifetime of the simulator.

6.1.3 Data structures

During the course of its operation, the simulator maintains two main data structures.

Incoming edges priority queue At startup the simulator reads in the input event graph. It creates a priority queue containing all the nodes in the graph. The priorities are the number of incoming edges in the input application event graph. The lower the number of incoming edges, the higher the priority of that node. Nodes with the highest priority are serviced first.

Event priority queue The simulator also maintains a priority queue containing events to be executed. These events correspond to one of the three annotated edges in the event graph. An event can either be a computation event, a communication event or a computation overhead of communication event. The events in this priority queue are indexed by their estimated completion times. The earlier the completion time of an event, the higher its priority. Events with the highest priority are serviced first.

6.1.4 Basic operations

At startup the simulator loads in the application event graph. It performs a topological sort on the nodes and creates the incoming edges priority queue. The pseudocode shown in Figure 6.4 is at the core of the simulator.

The simulator walks the event graph looking for nodes that have zero incoming edges. This means that the node has no existing dependencies and that we can process all its outgoing edges. For each out going edge, based on whether it is compute, communicate or compute overhead of communicate, we calculate its estimated completion time. This calculation is based on the current VM to host mapping and overlay routing, and the physical system measurement data. We add an event to the event priority queue indexed by this estimated completion time. If the outgoing edge is a plain edge, we then reduce that

```
// 1. Initially all nodes are marked ‘‘unserviced’’
// 2. A Node becomes ‘‘enabled’’ when number of incoming edges
//    to it is zero
// 3. A Node is ‘‘serviced’’ after we have finished
//    processing all its going edges

while(there exists an unserviced node){
    remove the enabled nodes from incoming edges queue;
    foreach (enabled node) {
        foreach (outgoing edge) {
            process the event associated with it;
            add the appropriate event to event priority
            queue to finish at appropriate time;
            if (outgoing edge is plain) {
                reduce the number of incoming edges to the node by 1;
            }
        }
        mark the node as serviced;
    }

    if (there exists an enabled node) {
        next;
        // takes us to begining of top-level while loop
    }
    remove the next event from the event priority queue;
    execute the event;
    modify wallclock and application data structures;
    reduce the number of incoming edges to the node by 1;
}

while (there exists an event in event priority queue) {
    remove the next event from the event priority queue;
    execute the event;
    modify wallclock and application data structures;
}
```

Figure 6.4: The simulator core.

particular nodes incoming edge count by one. Adding an event to the event queue implies that it has started execution (or communication), but has not yet completed. If the event is a communication event, then we modify the physical system measured data to account for the applications own communication.

When we have no nodes with zero incoming edges to process, we execute the next event in the event priority queue. We increment the wallclock to represent the progression in time and we also modify the application data structures (user time, system time, etc.) based on the specifics of the event executed. The incoming edge count of the concerned node is decreased by one. Further, for a communication event we modify the physical system measured data to reflect the completion of this communication event. We then go back to see if we have "enabled" any nodes, i.e. if any nodes have zero incoming edges and then repeat the above mentioned steps.

Once we finished walking the entire graph, we execute the remaining events in the event priority queue and perform the associated actions as above.

6.1.5 Verification and validation

Verification refers to the process of ensuring that the simulator is implemented correctly and validation refers to the process of ensuring that it is representative of the real system.

Verification The simulator was verified using the following techniques [200]:

- Structured walk-through: by explaining the code of the simulator in detail to multiple people, several bugs were identified and addressed through this process.
- Running simplified cases: A synthetic small trace constructed by hand was used as input. we compared the output of the simulator with the manually calculated output. They were very similar in terms of the wall-clock time, total computation time and

communication time. In addition, the log file of the simulator was carefully checked line by line to catch and fix errors.

- Antibugging: Additional checks and outputs were included in the simulator and the log file respectively to point out bugs.
- Degeneracy tests: We checked the response of the simulator for extreme values (multiple VMs mapped on same hosts, applications with only computation or only communication, etc.).

Validation The simulator was validated using the following techniques [200]:

- Expert intuition: This is the most practical and common way to validate a model [200]. Multiple brainstorming meetings of people knowledgeable about virtualized execution systems were called. We validated the assumptions, inputs and outputs.
- Real system measurements: This is the most reliable and preferred way to validate a simulation model [200]. Using XPVM and our task-graph generator (to convert XPVM trace to a form compatible with our simulator), we gathered a Patterns trace on 4 nodes of our cluster. According to patterns output, the ratio of computation over communication was high. The trace contained 5 iterations and the communication pattern was set to be all-to-all. The data in the measurement file was collected via physical measurement tools such as ttcp, ping and top to reflect the current state of the cluster nodes and the isolated network between them. The comparison between patterns output and the simulator output showed that the simulator correctly and closely simulates the running of the trace. We then modified the measurement file by reducing the utilization limits on 2 of those 4 nodes from 100% to 50%. Then, as expected, the total computation time in the output of the simulator nearly doubles,

which is what we expect since the entire Patterns benchmark is slowed down to the speed of the slowest node. This result also agreed with the output of Patterns, with the utilization being throttled using VSched. For example, the difference between the simulated user time on each of simulated hosts and the actual user time on each of actual hosts is less than 5%. A validation was also carried out for a slightly larger application executing among 6 VMs with a low computation to communication ratio.

6.2 Second generation simulator

The first generation of the simulator consists of approximately 5000 lines of Perl code. In the second generation of the simulator, I added the following features

- Simulation of periodic real-time scheduling of VMs running on multiple machines
- Simulation of fine-grained migration cost, and
- Connection of the simulator with a user interface to allow user input during simulation

The latest simulator consists of around 9600 lines of code.

6.2.1 Periodic real-time scheduling model

We want to simulate VSched real-time scheduling (Chapter 4) of VMs on multiple machines. In such a model, a task will run for *slice* seconds every *period* seconds. Because the first generation of the simulator was designed without considering real-time model, extending it becomes difficult.

A new data structure called VSched queue is introduced. It is a priority queue indexed by the deadlines of VSched tasks. A VSched task consists of a VM name, period, slice, deadline and remaining slice in current period. In general, the deadline of a task is the

```
//1. Each host has an associated VSched queue.
//2. Each task in the Vsched queue contains a VM name, period, slice,
//   remaining slice in current period.
//3. A global runnable queue is a priority queue, indexed by task
//   completion time, containing runnable VSched tasks on all hosts.

sub check_vsched {
  check all VSched queues for earliest deadline;
  check event priority queue for earliest communication completion time;
  if(comm_deadline < vsched_deadline){
    enable earliest communicate event in event priority queue & return;
  }

  foreach (VSched queue){
    find runnable VSched task with non-zero remaining slice;
    put VSched task into global runnable queue;
    calculate completion time based on remaining slice & corresponding
    event's compute amount;
    adjust task's remaining slice;
  }

  while (global runnable queue not empty){
    find task with earliest completion time & check event priority queue;
    if(a corresponding compute event for this VM){
      if(compute amount > total computation achievable in remaining slice){
         split event into two, one enabled with compute amount
         proportional to remaining slice, another disabled with
         remaining compute amount;
      } else{
         enable event in event priority queue;
      }
    } else{
      //although there's no corresponding event, the slice is still
      //resevered for its VM
      create a place-holder event, enabled and inserted into event
      priority queue;
    }
  }
}
```

Figure 6.5: The check_vsched function

end of its period. Each simulated host has an associated VSched queue. An enable flag is added to the event structure. All events put into the event priority queue from the event graph are disabled by default. Instead of executing events according to their completion time in the event priority queue, we now select VSched tasks according to their deadlines across all VSched queues. Those selected VSched tasks will then determine which events to enable and run in the event priority queue. Figure 6.5 is the core of a function called check_vsched(). The function is called before we transfer the control to the event priority queue for executing an event in the simulation loop (Figure 6.1)

In this function, we do the following:

- We check the event priority queue for all communication events with completion time equal to the current wallclock. We enable those events.
- We check each VSched queue looking for a VM with the earliest deadline across all hosts. We then look into the event priority queue to find the corresponding event for that VM. If the compute amount of the event is bigger than the total computation achievable within the remaining slice, we split the event into two new events. The first event is enabled and will be executed for *slice* virtual seconds according to the VM's $(period, slice)$ schedule. The second event will contain the remaining compute/communicate overhead amount and is re-inserted into the event priority queue without being enabled.
- A global runnable queue is used to buffer all runnable VSched tasks. We use this queue to sort runnable VSched tasks according to their completion time, because we try to simulate multiple VScheds and we need to synchronize the global wallclock.

Since we "time-slice" events, in the simulator after the event priority queue executes a compute/communicate overhead event, we cannot decrease the incoming edge count of the concerned node unless all sliced events are executed.

Since the application traces are generated by running the application without VSched. The simulator is now fed with a separate file containing initial (period, slice) schedule for each VM.

Note that the current simulator does not simulate the overhead of VSched and deadline misses of soft real-time scheduling.

6.2.2 VM migration

Migrating a VM across distinct physical hosts will incur various cost. The *downtime* refers to the time during which services or applications are entirely unavailable. Our simulator simply simulates a fixed downtime. During the downtime, the VSched schedule of the VM on the source host remains unchanged. On the destination host, the same schedule for this VM is admitted by VSched. However until the migration finishes, the VM does not compute or communicate on either host.

6.2.3 User input to the simulator

An important function of the extended simulator is to allow interrupts from the interface, i.e. the user. The current design is to have the simulator and interface communicate through share files protected by read and write locks. There are two shared files: a shared schedule file for changing VSched schedules and a shared mapping file for changing the VM mappings. For example, whenever the user changes the schedule of a VM, the interface will record the change into the shared schedule file. The simulator, which checks the file periodically, will detect the change and adjust the simulation accordingly. Note that the admission control component of VSched is implemented in the interface instead of the simulator in order to reduce the number of interrupts.

Through the user interface the user can change the mapping from VMs to hosts, for example by migrating a VM from one host to another. Details about the user interface and

human-driven VM migration will be discussed in the next chapter.

Depending on the input trace, in general, the simulation executes faster than that of the real application. However if the simulation is too fast, we will not be able to get enough user feedback during execution. To address this, an adjustable parameter is used to slow down the simulation so that for each compute event, the simulator will sleep for a certain amount of time depending on the difference between virtual time and real elapsed time.

6.2.4 Verification and validation

The verification and validation of the second generation simulator is similar to that of the first generation simulator in term of techniques and steps, except that we now focus on those newly added simulation features. Thus in the following, I will omit those similar steps in verification and validation.

Verification

- Running simplified cases: We took a synthetic small trace and fed the simulator with different (period, slice) schedules. The simulator was instrumented to print out information about current real-time task, VSched queues in each host and the event queue. We compared the output of the simulator with the manually plotted real-time scheduling task graphs (similar to Figure 4.2). They were the same in terms of the wall-clock time, order of tasks, running time of each VSched task and etc. In addition, the log file of the simulator was carefully checked line by line to catch and fix errors.

- Degeneracy tests: We checked the working of the simulator for extreme $(period, slice)$ values (large period with tiny slice, same period and slice for all VMs, etc.).

- Consistency tests: To verify the user input to the simulator, we ran the simulator in step-by-step mode, in which the simulator pauses after executing an event. In different stages of the simulation, we manually change the VSched schedules of some VMs by modifying the shared schedule file between the interface and the simulator. We then verified the status of simulation with manually plotted real-time scheduling task graphs. Similarly for migration, we manually re-map a VM from one host to another by changing shared mapping file and then compared the simulation with manual results.

Validation

- Real system measurements: We gathered a patterns trace on 4 nodes of our cluster. Patterns is configured to do all-to-all communication with high compute/communicate ratio. The trace contained 10 iterations. The data in the measurement file was collected via physical measurement tools such as ttcp, ping and top to reflect the current state of the cluster nodes and the isolated network between them. We fed the simulator with initial VSched schedules for the 4 VMs. We compared the simulator output with output from Patterns run on actual hosts, scheduled by actual VSched on each host with the same initial schedules. The comparision showed that the simulator closely simulates the running of the trace combined with VSched scheduling on multiple machines, for example the difference between the simulated user time on each of simulated hosts and the actual user time on each of actual hosts is less than 5%.

6.3 Summary

In this chapter, we discussed the design and development of an application trace-driven simulator. In the next chapter, we will show how we use this simulator to help the design and development of human-driven technique.

Chapter 7

Direct User Control of Scheduling and Mapping of Virtual Machines

This dissertation argues for using direct human input to solve optimization problems. In Chapter 5, we showed that we can use human-driven specification to set and control the performance of BSP applications. In that work, a global controller applies the same (period, slice) schedule to each application thread running on a collection of hosts.

In VM-based computing systems, such as Virtuoso, distributed applications including BSP applications can be run inside a collection of VMs. Those VMs are mapped to different hosts. The mapping can be dynamically changed through VM migrations. The following is the specific optimization problem that I am addressing in this chapter.

7.1 Specific problem

I consider a configuration x that consists of

- a local periodic real-time schedule of VMs on each host, and
- a mapping from VMs to hosts,

The constraints determine what configurations are valid and include

- schedulability (there must exist a feasible schedule on each host),
- 0/1 constraints (a VM must be mapped onto exactly one host)

For a VM k, I define

$$slice_i = slice_{(compute,i)} + slice_{(idle,i)} \tag{7.1}$$

$slice_{(compute,i)}$ is the portion of the ith real-time *slice* that is used for computing while $slice_{(idle,i)}$ is the portion that is wasted for waiting.

$$efficiency_{(x,i)} = \frac{slice_{(compute,i)}}{slice_i} \tag{7.2}$$

$i = 1,\ldots,n$, n is the total number of CPU slices reserved for VM k during its execution.

Average efficiency for VM k is

$$avg\_efficiency_k = \frac{\sum_{i=1}^{n} efficiency_{(k,i)}}{n} \tag{7.3}$$

Then the objective function is

$$f(x) = \sum_{k=1}^{m} avg\_efficiency_k \tag{7.4}$$

where m is the total number of VMs.

Thus the optimization problem here is how to find a configuration x such that the $f(x)$ is maximized while x also obeys the given constraints.

In this chapter, we present how we use human-driven search to solve this problem. We use the Patterns benchmark as our test application.

In the following, I first consider the optimization problem with a simplified configuration which includes only a local periodic real-time schedule of VMs on each host (the mapping from VMs to hosts is fixed). I describe my first-round interface design, a game-

style problem presentation, and a user study. Then I extend the configuration to include mappings from VMs to hosts. I discuss how I improved the interface and the game, followed by another user study. I then summarize.

7.2 Interface design

Figure 7.1 illustrates the first control interface.

VM list The names of all VMs are listed here.

Control area Each VM is represented as a ball. For a selected ball, the X axis of the Control Area is mapped to the CPU utilization (increasing from left to right) while the Y axis is the period (increasing from bottom to top) in terms of the real-time schedule. By moving a ball, the user can change the CPU schedule of the corresponding VM. In this design, we do not allow the user to control the VM-to-host mapping.

History area The dynamic efficiency history of a selected VM is plotted here. The user can further select how many past history points up to the current time that he wants to watch at one time. The selection will apply to all VMs.

Target areas In Target Area 1, the average efficiency (0 - 1) of each VM is dynamically displayed. The height of each smiley face is proportional to the average efficiency value of its corresponding VM. The value is averaged over the selected number of past history points specified in the History Area. For example, if the user selects “100” in the History Area, the height of a ball in Target Area 1 is then the average efficiency over last 100 efficiency values of that VM. The mouth of each smiley face will change according to its efficiency level.

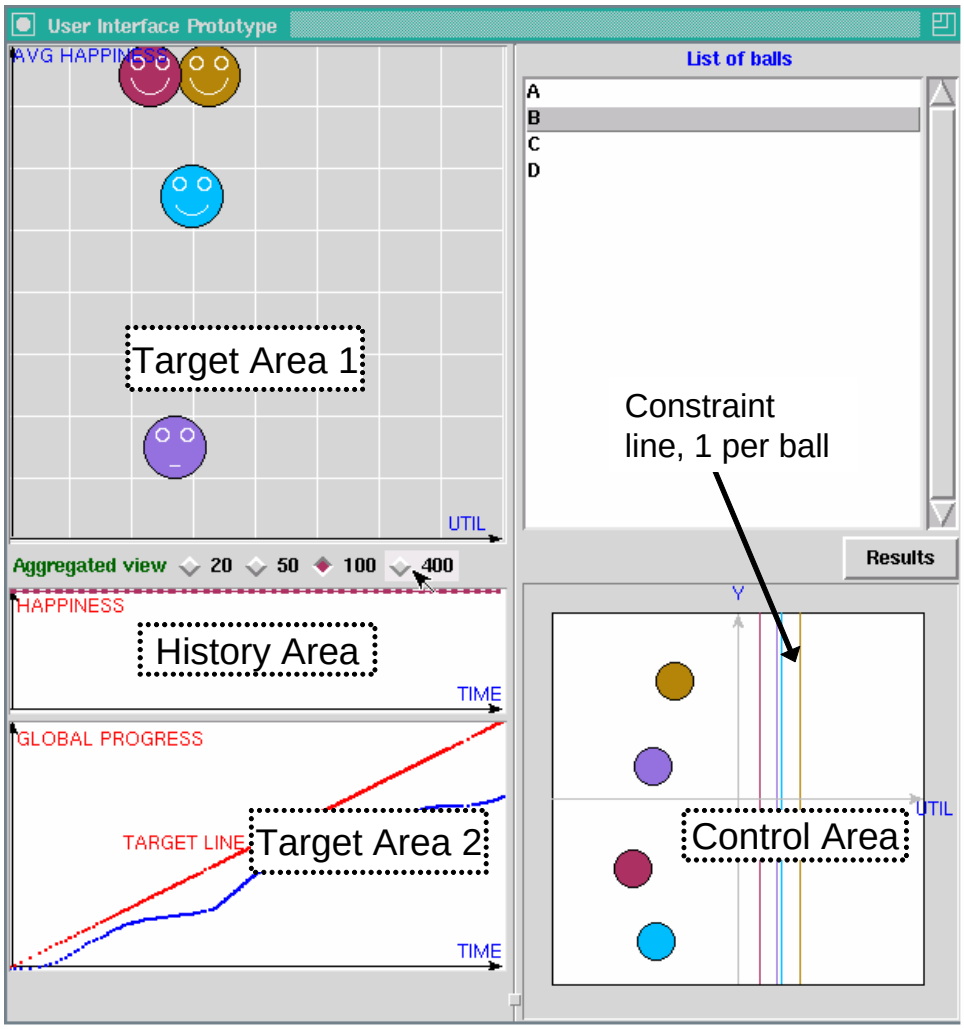


Figure 7.1: Initial interface for VM scheduling game

In Target Area 2, the cumulative amount of computation of all VMs is plotted as a line (in blue) for comparison with a target line (in red). We chose the target line to be the amount of computation in the best case when each VM is mapped to a dedicated node and has 100% utilization. The higher the efficiency of a VM, the more computation that the VM achieves in a period. Thus the efficiency of all VMs will determine the total computational amount. We show the computation line to the user as extra global information, together with the individual information about each VM in the History Area and the Target Area 1.

Constraints We focus on two types of constraints in our problem: the maximum capacity of a host, and the maximum total resource utilization on all hosts for a distributed application. The former is a local constraint while the latter is a global constraint. For example, we can define 100% utilization as the capacity for each host but only allow a application to use up to $90\% \times N$ total utilization, where N is the number of hosts.

In the interface, every ball has a associated constraint line in the Control Area. A ball and its constraint line will be displayed in the same color. The constraint line represents the maximum possible utilization for the VM considering both local and global constraints. The user will not be able to move a ball rightward across its constraint line. Changing a VM's schedule will affect the constraint lines for all other VMs.

7.3 Game design

Based on the problem we described in Section 7.1, I consider two possible presentations of the problem to the user:

- Present a specific optimization problem according to the target application and underlying system. This requires user's knowledge about the application and some understanding of the problem. Thus, the ideal audience will be system administra-

tors and expert application users.

- Present an analogous problem which is understandable to common users without requiring specific background. This approach will allow us to talk to almost unlimited users. Essentially, this will be a game style presentation. However, this approach poses a new challenge to the design in that the analogy must be as simple and as direct as possible. Otherwise, the results of the user playing the game cannot be used as solutions to the original optimization problem.

I started with the first type of presentation. However, I soon realized that it will be very expensive to verify the interface and the presentation through a controlled user study, because it requires a certain population of practicing system administrators and expert application users. I then switched to the second type of design by translating the optimization problem into a generic game. We present the concept of VM efficiency as the happiness of a ball. The cumulative computation is presented as cumulative global progress of all balls, assuming all balls are working together to make progress towards a general goal. The "period" label for the Y axis of the Control Area is replaced with just "Y". And we leave the utilization label there and present it as the utilization of general resources used by balls. The final interface is shown in 7.1. The interface is developed in Perl/TK with approximately 2600 lines of code.

The goals of the game are as follows. By moving balls in the Control Area, the user should

- maximize the happiness levels for all balls, and
- minimize the gap between his global progress line and the target progress line.

In the end of each game, the user will be shown the average happiness level of each ball.

7.4 Configuration

I connected our interface to the simulator, as described in Chapter 6. The simulator is driven by traces from Patterns. We use traces from Patterns configured to do all-to-all communication with roughly 1:1 compute/communicate ratio.

We need to configure a migration downtime for the simulator. Earlier work [34] demonstrates the migration of entire OS instances on a commodity cluster, recording service downtimes as low as 60ms, which is the fastest migration time that I am aware of. Other work [159] shows that a VM can be migrated in 20 minutes or less across 384 kbps DSL. I treat 20 minutes as the upper bound of the downtime. In the simulation setup, we simulate a local cluster with 800 Mbps interconnect and no latency. So a reasonable migration downtime is a time between 60ms and 20 minutes. Also considering the limited time of each of our later user studies, I chose a migration downtime of 20 seconds and enforce that in the simulator.

In this simplified optimization game, we know the solutions. Recall that our current simulator does not simulate the overhead of VSched scheduling and tasks always meet their deadlines. Based on our previous experience with scheduling BSP applications (Chapter 5), we know that the optimal configuration is to apply the same smallest (i.e. the finest) period and maximum utilization to all VMs. For example, if there are only two VMs and both are mapped to the same host, both need to be scheduled to run 50 ms every 100 ms in order to maximize the performance, assuming 100% CPU utilization on the host is allowed.

7.5 User study

We conducted a controlled user study to determine whether end-users could play the game to find optimal configuration for their VMs, or at least a good one.

The 12 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. We advertised for participants via email. Each user was given $10 for participating.

7.5.1 Testcases and process

For all tasks, we set the global utilization constraint to be $90\% \times N$, where N is the number of hosts. For individual hosts, the capacity is 100% while the minimum is 10%. In the background, the maximum period of a VM is 1000 ms and the minimum is 100 ms.

We have 2 types of tasks.

Type I - 1 VM per host We first want to test the simplest case where there is little contention between VMs in terms of resources. We statically map each VM to a dedicated host. The user needs to find out the optimal CPU schedules for all VMs. Considering the global constraint, an optimal configuration is to apply the same 90% utilization and 100ms period to each VM.

Type II - 2 VMs per host We increase the complexity by mapping two VMs to the same host so if there are N hosts, we will have 2N VMs. An optimal configuration is to apply the same 50% utilization and 100ms period to each VM.

Note that the user is not aware of the mapping of VMs to hosts. The difference between two types of tasks is manifested through constraint lines and the behaviors of the balls.

The study consists of 2 warm-up tasks and 6 formal tasks. The 2 warm-up tasks are 5 minutes each and include a 2 ball version of the Type II task and a 3 ball version of the Type I task. The user can acclimatize himself to the interface through the warm-up period. The 6 formal tasks are 8 minutes each and cover 4, 8, and 16 balls with both types of tasks.

We increase the number of balls in order to test the interface as the optimization problem scales.

We had the user fill out a questionnaire in the beginning of the study which asks about his background. The user was then asked to read instructions explaining the game interface, controls as well as the following information:

- The goal of the game is to maximize the happiness levels for all balls (Target Area 1) and minimize the gap between your progress line and the target progress line (Target Area 2).
- If there is only one ball in the game, moving it to the lower-right corner will optimize it and achieve the goal of the game.
- Moving a ball will not only change its own happiness and progress but also influence other balls.
- The global progress of all balls depends on their happiness levels.

The user was asked the following questions after each task:

- Were you able to use the interface to achieve the goal of the game? (Y / N)
- If no, why?

The written protocol and the form the user filled out can be found in the Appendix D.

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

- Efficiency and computation history of every ball (VM), and
- For each move of a ball, the time stamp and the new $(period, utilization)$ for that VM.

The user was unaware of the recording process.

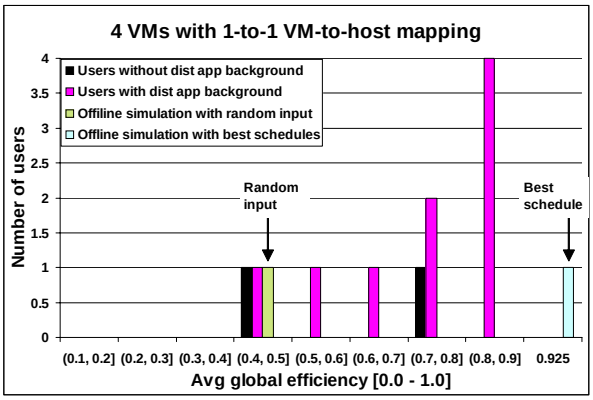
7.5.2 Results

Our game is designed for a general audience. We want to verify if it works for them. We are also interested in finding out whether a user who has a systems background can play our optimization game better than other users.

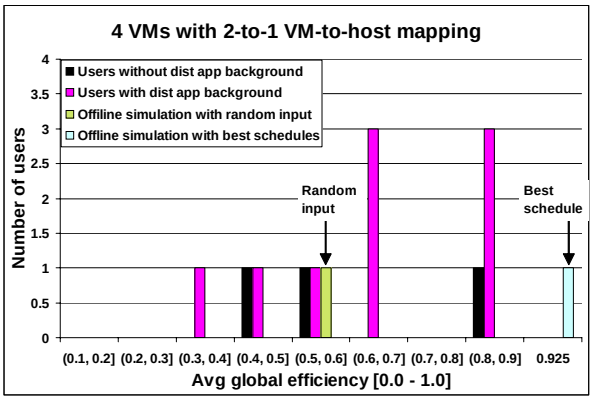
Figure 7.2 is a histogram where the horizontal axis is the average efficiency and the vertical axis is the number of users. Users are differentiated by whether they claimed familiarity with distributed and/or parallel systems. The maximum possible efficiency is also shown. It is not 100% due to communication. The effects of random scheduling decisions are also shown. Note that the resolution of the histogram is 10% and ordering within a bin is not significant.

Because of the limited number of users in this study, it is difficult to draw conclusions statistically from this figure. However we observe the following trends:

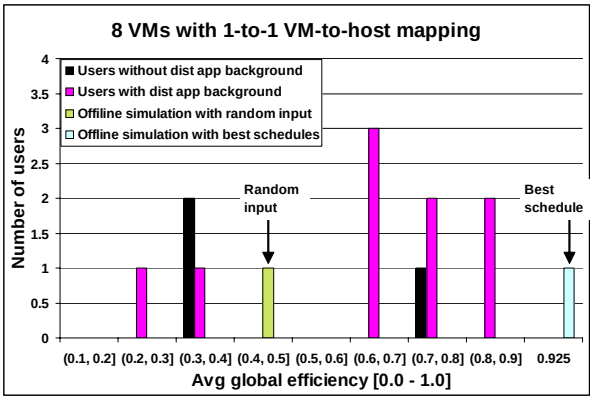
- The upshot is that by playing the game, most users can schedule this collection of VMs with at least some efficiency beyond random, for a small enough problem.
- As we expect, the difficulty of the game increases as the number of VMs increases.
- Although we introduce a hidden constraint in Type II tasks (2 VMs per host), the performance of the users is only very slightly worse than that in Type I tasks (1 VM per host).
- In 4 VM tasks, more than 75% of users do better than random moves. In 8 VM tasks, more than 67% of users perform better than random. Even in the difficult 16 VMs Type I task, around 42% of users achieve average global efficiency higher than that from random decisions.
- Although the figure may suggest that those users who have distributed and/or parallel



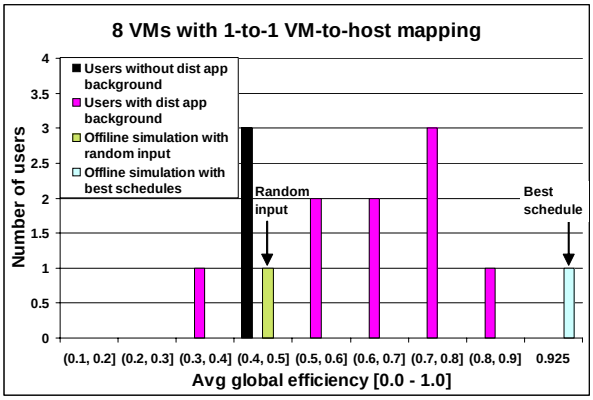
(a) 4 VMs; 1-to-1 mapping



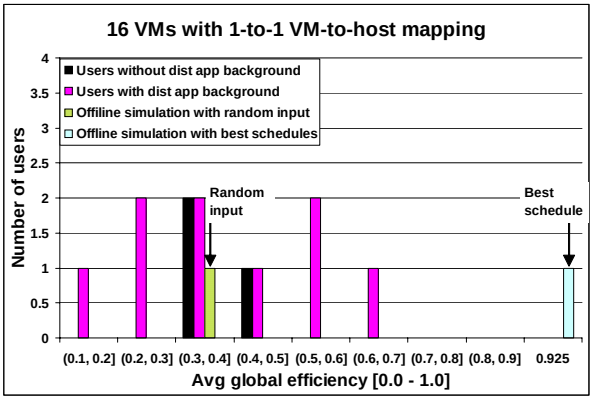
(b) 4 VMs; 2-to-1 mapping



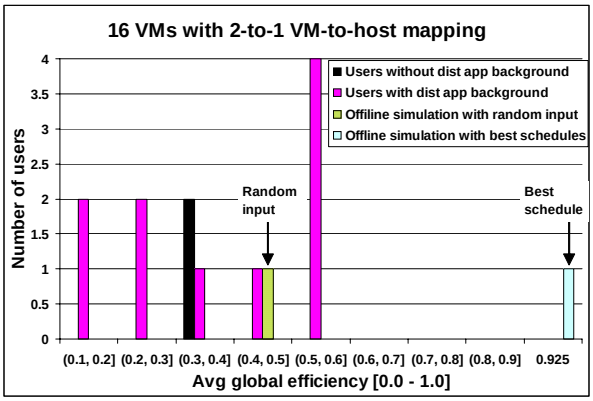
(c) 8 VMs; 1-to-1 mapping



(d) 8 VMs; 2-to-1 mapping

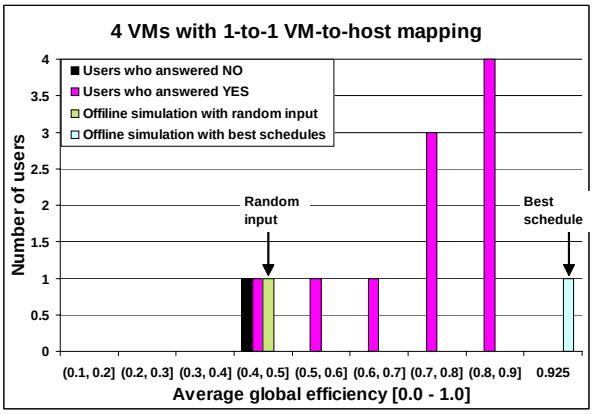


(e) 16 VMs; 1-to-1 mapping

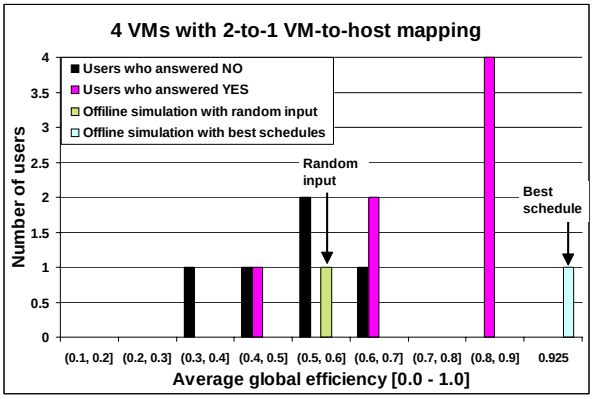


(f) 16 VMs; 2-to-1 mapping

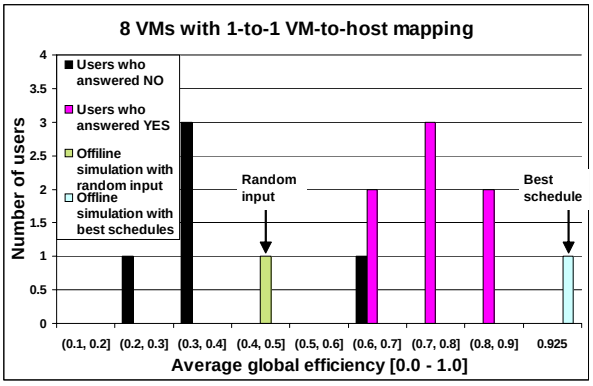
Figure 7.2: User background versus average global efficiency



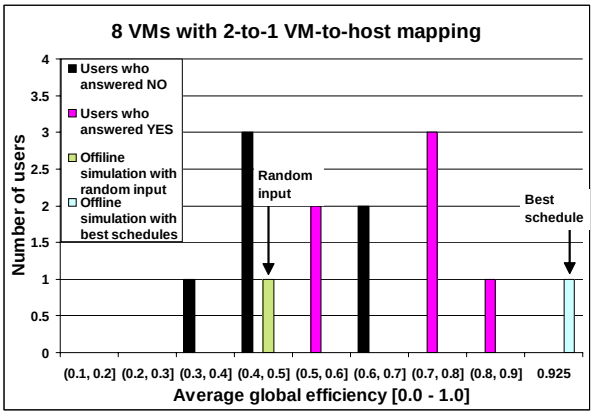
(a) 4 VMs; 1-to-1 mapping



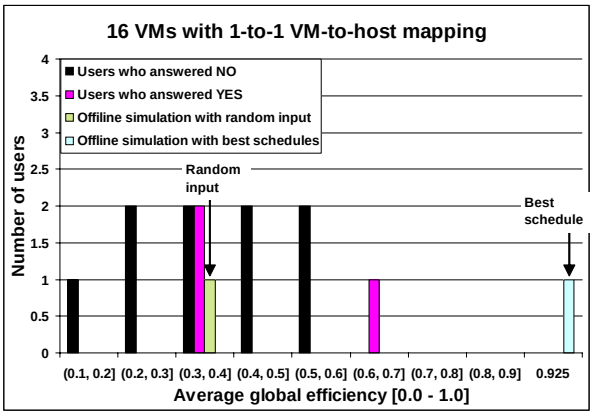
(b) 4 VMs; 2-to-1 mapping



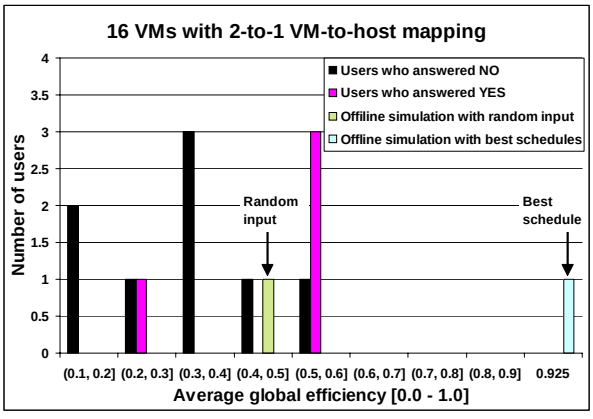
(c) 8 VMs; 1-to-1 mapping



(d) 8 VMs; 2-to-1 mapping



(e) 16 VMs; 1-to-1 mapping



(f) 16 VMs; 2-to-1 mapping

Figure 7.3: User self-evaluation versus average global efficiency

systems background tend to play better than other users, 9 out of 12 users in this study have such background.

- For up to 8 VMs, the best average efficiency achieved by users is very close to the maximum possible efficiency.

Note that in Figure 7.2(a), one of the user's data is missing due to a system bug. That is why we only show two users with background in that figure.

In the end of each task, we asked the user whether he was able to use the interface to achieve the goal of the game ("Yes" or "No"). Figure 7.3 is a histogram that shows how the user's self-evaluation correlates to his performance. We can see that in general, those users who believed that they performed well in the game were in fact doing well if we compare their results with both random decisions and maximum possible efficiency. In addition, they generally performed better than those users who believed that they did not perform well.

7.6 Summary of the VM scheduling game

Overall, the results show that a common user can understand our control interface and can further play the game to optimize the global efficiency of a collection of VMs to a certain extent. The game also gives appropriate feedback to the user on his performance.

To my knowledge, this is the first work that uses direct human control of distributed resource scheduling with game style presentation. The results of the study shows that it is promising to apply human-driven search to solving the optimal problem we discuss here.

7.7 Issues in interface

We notice that users did not perform well in 16 VM tasks compared with smaller tasks. Our explanation to this is follows:

- Our simulator runs slower as the number of VMs increases. As a result, for a fixed period of time, there is less information about efficiency and computation. The user will feel that the display and feedback is slower after each move.

- When there are many balls in both the Control Area and Target Area 1, many of them are easily overlapped which makes it difficult for the user to distinguish balls and further control them, especially when balls have similar schedules.

- Our target compute line is plotted assuming each ball is using 100% of the CPU and is mapped to a dedicated host. As the number of VMs increases, the gap between the target line and the user line becomes bigger even if the user set the optimal schedules for all balls at the very beginning of the game. In other words, the target line is unreachable by the user. The situation is aggravated in those tasks where two VMs are mapped to the same host. As a result, the user tends to move balls around to try new positions even though he has already found the best position. Thus some users reported that they did not achieve the goal of the game even though they did a good job in keeping all balls happy most of the time.

We also obtain useful feedback about the interface and game from users through both our observations and conversations with the user after each study. This led to the following conclusions:

- Our intention to have two goals in the game is forcing the users to consider both short-term efficiency of VMs and long-term trend of total computation (also deter-

mined by efficiency). However, users have difficulty in handling two goals simultaneously. It is also difficult for the user to understand the relationship between the happiness of a ball and the global progress line without knowing the background information about the underlying problem.

- Some users believe that if they were given more configuration-related information, they might be able to perform better.

- Due to communication, the efficiency of a VM is not 100% all the time. Thus, even if the user gives balls the best schedules, he will still notice some fluctuation of the happiness of balls. Many users thus chose to keep moving the balls looking for "better" schedules. Some other users chose to only focus on his global progress line (i.e., Target Area II).

Based on this feedback and the user study results, we decide to improve the interface and game. We also want to solve a more complex problem using the same human-driven approach.

7.8 Interface design revisit

I designed a new control interface based on the observations of the previous study and the need for letting the user directly control the mapping from VMs to hosts. Figure 7.4 shows the new control interface. It consists of the following components.

Short-term Happiness Monitor This is similar to the Target Area 1 in the previous interface except for its name.

History Area This is the same as the previous interface.

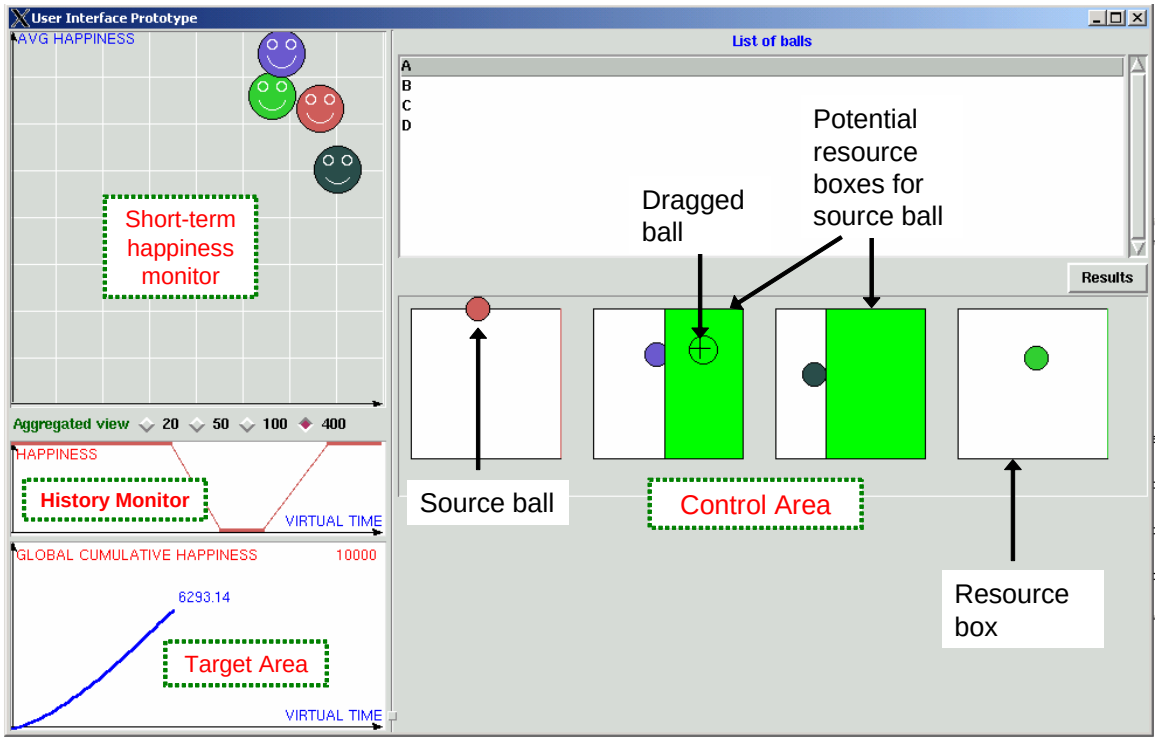


Figure 7.4: Interface for CPU scheduling & VM mapping game; the user is trying to migrate the ball in the left-most resource box to the second resource box to the left.

Target Area In this area, we plot the global cumulative happiness (i.e. efficiency) line (blue) over time. The line shows the sum of the happiness levels across all balls. We also show the current cumulated happiness number. We further show a target number (in red) on the top of the Target Area. The number is selected in a way such that it is a little bit larger than the maximum happiness number achieved by the optimal configuration.

Control Area In this area, each physical host is represented as a white rectangle, called a resource box. Similar to the control in previous game, moving a ball within its resource box will change its CPU schedule. Now whenever the user selects a ball, a dragged ball will show up indicating the current position of the ball. The dragged ball will disappear after the user moves the ball to a new position. The main function of the dragged ball is to facilitate migration, which I will explain next.

Migration The user can now migrate a ball to a different resource box. We provide two ways for the user to do that.

- **Default drag & drop mode** When the user moves a ball close to the borders of its resource box or its constraint line, some highlighted areas (Figure 7.4) may appear to indicate potential new resource boxes for this ball. The width of each highlighted area is proportional to the available utilization left on that resource box. The user can simply drag and drop the ball to any of the highlighted areas to migrate the ball. During the drag & drop, the source ball will not move. After the user drops the dragged ball, the source ball will show up in the new resource box. For example, in Figure 7.4, the user is trying to migrate the ball in the left most resource box to the second resource box to the left.

- **Advanced mode** In the default migration mode, when the user moves a ball close to the borders of its constraint line, the CPU schedule of the corresponding VM

will also change accordingly. The advanced mode allows the user to migrate a ball without changing its schedule. The user can do that by pressing and holding the SHIFT key, and then doing the drag & drop. After the user drags the ball to the destination box and releases the SHIFT key, the ball will be automatically placed in the same relative position in the new resource box. Note that if the user holds the SHIFT key but does not migrate the ball, the schedule will not be changed for that ball. The ball will automatically move back to the starting position before the move, after the user releases the SHIFT key.

As we described in Chapter 6, we simulate a fixed migration cost. We want to manifest that in the interface. In the interface, during the migration, the ball will be frozen for a certain amount of time until the simulator notifies the interface that the migration is complete. During the frozen period, a red circle will appear surrounding the ball indicating that the ball is migrating. The user will not be able to move that ball until the red circle disappears. In addition, we tell the user that during the migration, the ball being migrated will not contribute to the global happiness line, and its own happiness level will not be updated. The happiness level of other balls may be affected due to the temporary absence of the frozen ball.

7.8.1 Game design revisit

Goal In this game, we set only one goal for the user. By controlling his balls, the user wants to push the global cumulative happiness line to as high as possible, towards the target number. The user also wants the slope of the line to be as steep as possible. The slope determines the growth speed of the line. We tell the user that the ceiling of the Target Area may or may not be reachable.

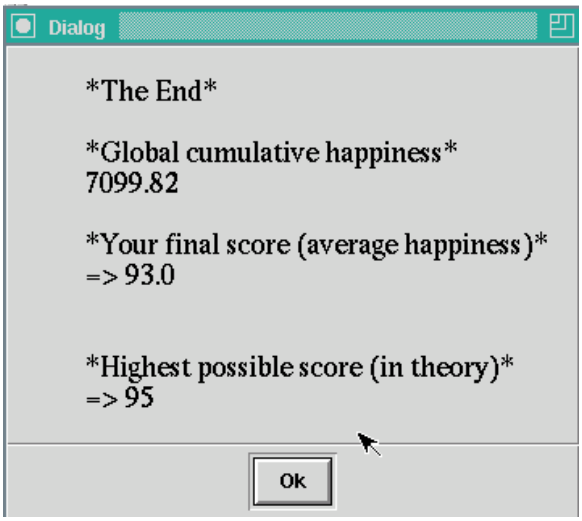


Figure 7.5: Final screen

Number of users	Occupation	Major
17	Student	Biology, Chemistry, Material Science Computer Science, Economics ...
2	Housewife	
1	Professor	
1	Waiter	

Table 7.1: User background

Final screen In the end of each game, the user will see a final screen as shown in Figure 7.5. The user's final score is calculated based on the global cumulative happiness value averaged across all balls and all history points. The highest possible score is the score achieved by applying the optimal strategy. The user was told that the closer his score to the highest possible score, the better he performed.

7.9 User study

We conducted a user study to determine whether end-users could use our new interface and play our new game to find both optimal CPU schedules and mapping for their VMs.

The 21 users in our study consisted of people with various backgrounds, which is different from last study. Table 7.1 shows their background. We advertised for participants via email. Each user was given $12 for participating. Note that, among those users, only two participated in the previous study.

7.9.1 Testcases and process

For all tasks, we set the global total utilization constraint to be $90\% \times N$, where N is the number of hosts. For individual hosts, the maximum local utilization is 100% and the minimum is 10%. The maximum period is 1000 ms and the minimum is 100 ms.

We have 3 types of tasks.

Type I - schedule only In this type of task, VMs are initially mapped to optimal hosts (one VM per host) so the user only need to find the optimal CPU schedules for them. Of course, the user is allowed to migrate VMs if he wants. The optimal schedules are to apply the same maximum finest schedule (90% utilization, 100ms period) to each VM.

Type II - schedule + mapping I In this type of task, all VMs are initially mapped to the same host. The user not only needs to decide the CPU schedules for his VMs but also need to migrate the VMs in order to achieve the highest score. Since the number of VMs is the same as the number of hosts, the optimal mapping is to map one VM to one dedicated host. The optimal CPU schedule setting is the same as in Type I.

Type III - schedule + mapping II Similar to type II, all VMs are initially mapped to one host. The user still wants to optimize both CPU schedules and mapping. However, there are more VMs than hosts. If there are N VMs, we only provide (3/4)N hosts. An optimal strategy is to have two VMs sharing each of the (1/4)N hosts and have each of the left (2/4)N hosts runs only one VM. Besides the mapping, the user needs

to apply the same schedule (50% utilization, 100ms period) to each VM in order to achieve the highest average efficiency.

The study consists of 2 warm-up tasks and 9 formal tasks. The 2 warm-up tasks are 4 minutes each and include a 2 ball version of the Type I task and a 3 ball version of the Type II task. The purpose is to allow the user to get familiar with the controls. The 9 formal tasks are 7 minutes each and cover 4, 8, and 10 balls with all three types of tasks.

Similar to what we did in previous study, we present the optimization problem as a game to the user by hiding all background information. Each VM is represented as a ball. The efficiency of the VM is described as happiness. Each host is now described as a individual resource box. Different from the previous interface, we now hide the utilization label in both the Control Area and the Happiness Monitor. And we simply tell the user that the X axes of these two areas are the same.

We had the user fill out a questionnaire in the beginning of the study which asks about his background. The user was then asked to read instructions that explain the game interface and controls, and state the following:

- In general, moving a ball downward and/or rightward within its resource box will consume more resources. However all positions in the Control Area are valid.

- The more resources you assign to a ball, the more likely that your ball will become happier, but if a ball consumes more resources than it can use, it will become unhappy. In addition, all balls share a global resource pool, so if one ball gets more resources, the other balls will get less. Try to discover the relationship among balls will help you select the correct resource usage level.

- If all balls are happy most of the time in the Short-term Happiness Monitor, your cumulative global line will certainly grow fast and high, but keep in mind that your

only goal is to get the global happiness line as high as possible and its slope as steep as possible. You may observe a certain degree of fluctuation of the happiness of balls. If the fluctuation is temporary and short-term, it does not necessarily mean that you need to adjust the balls. However, if the fluctuation is persistent and/or displays a certain pattern, it is then necessary for you to make some adjustment.

- Your final score is calculated based on the global cumulative happiness averaged across all balls and all history points. The closer your score to the highest possible score, the better you perform. Although we show the highest score in theory in the end of each task, we do not know the highest achievable score in practice.

The user was asked the following questions after each task:

- What is your final score?
- How do you evaluate your performance in this game session? (1 (bad) to 10 (outstanding))
- If bad, why?

The written protocol and the form the user filled out can be found in Appendix D.
As the user performed the tasks, we recorded the following information:

- Efficiency and computation history of every ball
- For each move of a ball, the time stamp and the new $(period, utilization)$ for the corresponding VM.
- For each migration of a ball, the time stamp, the source host, the destination host and the current schedule fo the corresponding VM.

The user was again unaware of the recording process.

7.9.2 Results

Figure 7.6 shows the histogram of users' efficiency score. The horizontal axis is the average efficiency and the vertical axis is the number of users. It also shows the maximum possible efficiency and effects of random mapping and random scheduling decisions. Note that the resolution of the histogram is 5% and ordering within a bin is not significant.

For the same number of VMs, as we expect, the performance of the users decreases as the difficulty of the task increases. The intuition is that the difficulty of the game is related to the dimensions of controls and also the number of constraints.

If we look into each type of task, in Type I tasks, more than 90% of users do better than random moves. In Type II tasks, where we introduce the mapping, 86% of users perform better than random except in the 8 VM task (57%). Type III tasks appear to be difficult for the users. However, in the Type III 10 VM task, more than 76% of users are better than a user who did pure random moves. An interesting observation is that, for the same type of tasks, as we increase the number of VMs from 4 to 8, users' scores decreases a little bit. However as we further increase the number of VMs to 10, users do better than 8 VM tasks. This is showing to some extent that, up to 10 VMs, the effectiveness of our interface and game is largely independent of the number of VMs, although we can not exclude the possible bias caused by some learning effect since we did not randomize the order of tasks.

If we look at the best scores achieve by users, we find that except in Type III 4 and 8 VM tasks, their best scores, in all other tasks, are as good as those achieved by an oracle who knows everything about the background and the underlying optimizaion problem.

In Figure 7.7, users are differentiated by whether they claimed familiarity with distributed and/or parallel systems or not. Overall, users with related background do not noticeably perform better than other users. This supports our claim that the game style presentation is targeted at common users.

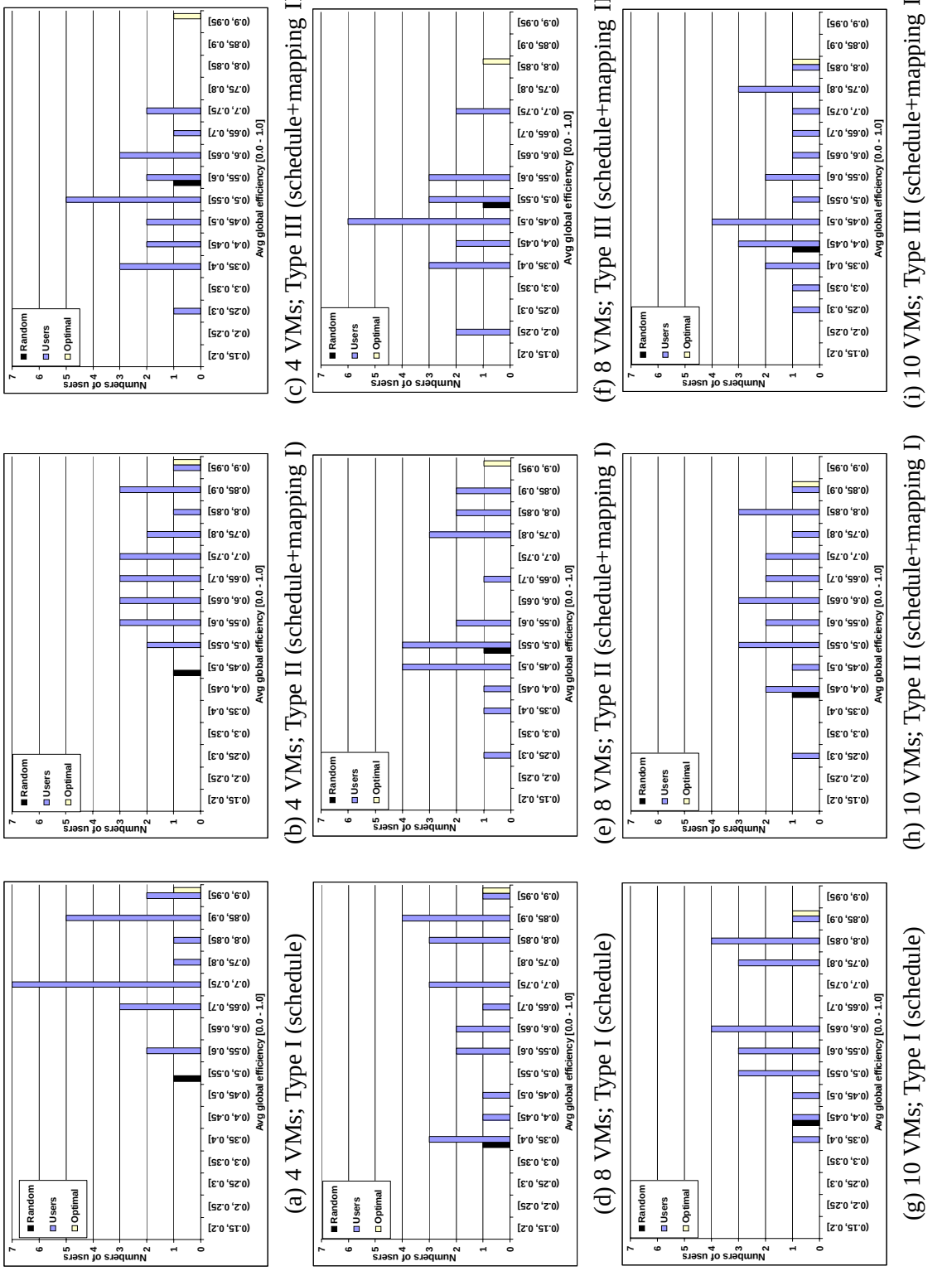


Figure 7.6: User performance

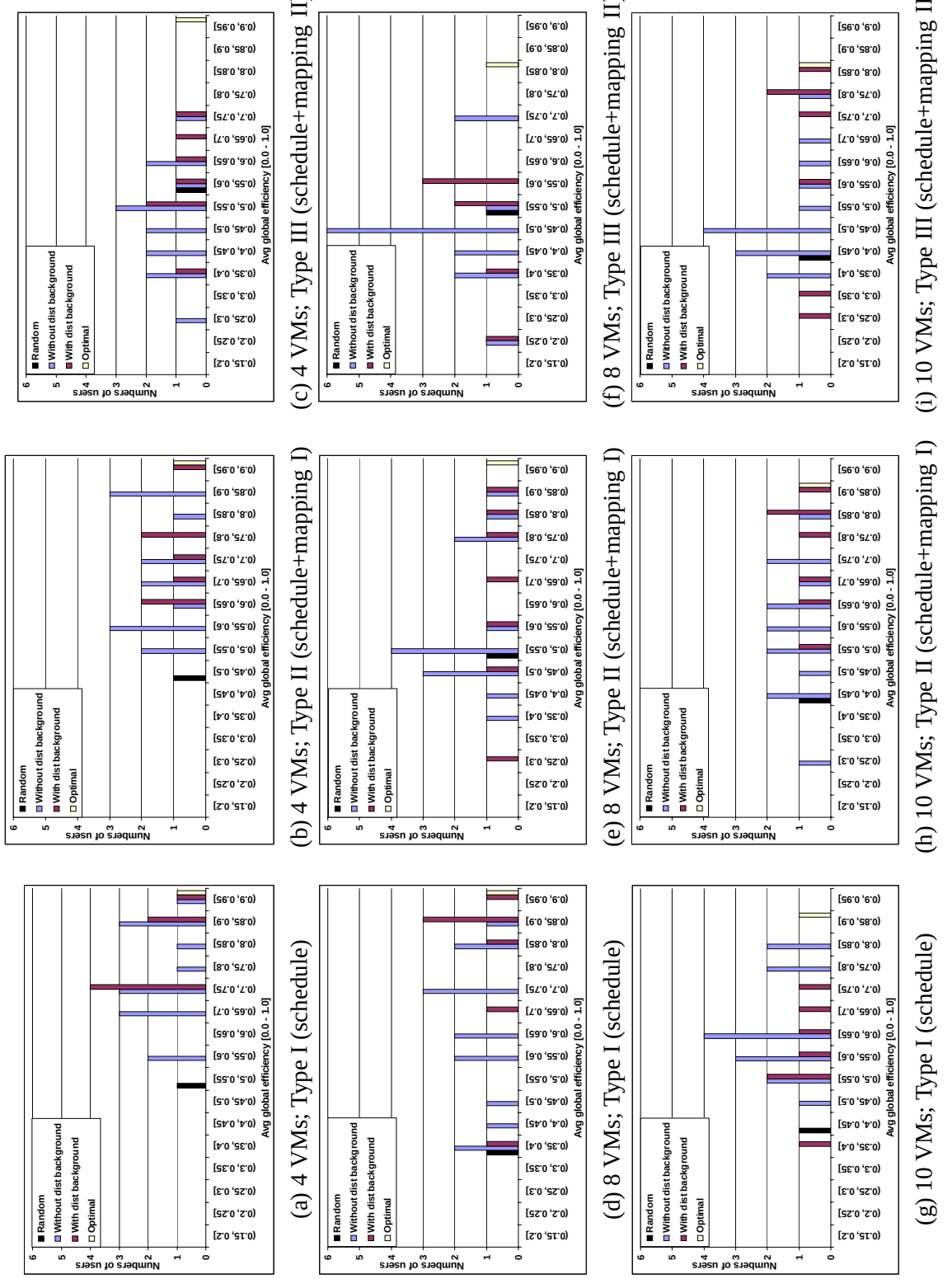


Figure 7.7: User performance and user background

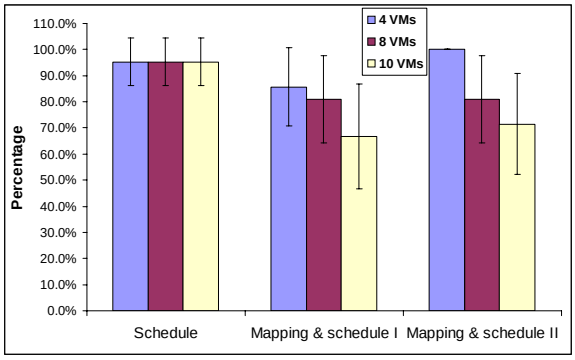


Figure 7.8: Percentage of users who find the optimal mapping; 95% confidence interval.

Mapping In Figure 7.8, we show the percentage of users who successfully found the optimal VM mapping and kept the mapping unchanged until the end of a task. We only count those users whose final mapping of VMs to hosts is optimal. We can see that more than 65% of users were able to find the optimal VM mapping for all tasks. From 4 VMs to 8 VMs, fewer users were able to find the optimal mapping. Interestingly, more users had found the optimal mapping for 10 VMs than for 8 VMs. We also notice that in Type I tasks, the percentage in 10 VM tasks is even higher than that in 4 VM tasks, which we believe is because of the users' unfamiliarity with the game in the beginning of the study.

We further show the statistics for those users who found the optimal mapping in Figure 7.9. The Y axis is the time for the user to find the optimal mapping. Clearly, as the difficulty (both task type and number of VMs) of tasks increases, on average users spent more time in finding the optimal mapping. In general, users were able to find it in around 2-3 minutes in a 7 minute task.

User's self-rating as mentioned earlier, we had the user evaluate his performance in the end of each task after he was shown his score and maximum score in theory. Table 7.2 shows that there is certainly correlation between the score and user's self evaluation. For Type III 8 VM task, the coefficient is low. The number results from two users evaluating

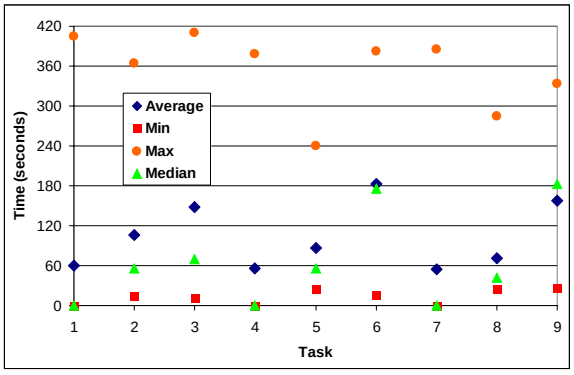


Figure 7.9: Duration to the optimal mapping.

Task	Correlation coefficient
4 VMs, Type I (schedule)	0.75
4 VMs, Type II (schedule+mapping I)	0.61
4 VMs, Type III (schedule+mapping II)	0.69
8 VMs, Type I (schedule)	0.77
8 VMs, Type II (schedule+mapping I)	0.78
8 VMs, Type III (schedule+mapping II)	0.46
10 VMs, Type I (schedule)	0.77
10 VMs, Type II (schedule+mapping I)	0.75
10 VMs, Type III (schedule+mapping II)	0.73

Table 7.2: Relationship between user self-rating and his score

themselves much better than what their score showed. They probably believed that they had done their best.

7.10 Conclusions

The contributions of this work include the following:

- We show that it is possible for a common user to solve the discussed optimization problem by giving him direct control of the system.
- Our interface enables two dimensional controls for the user for his VMs: CPU sched-

ule and VM mapping. Our game design transforms the complex multiple machine optimization problem into a generic game playable by a common user.

- Given the limited scale of the problem studied here, our results show that most users are able to optimize a collection of VMs to different extents without knowing related background.
- For those users who were successful, they achieved the goal quickly even though they were faced with a large solution space (CPU schedule, mapping of VMs to hosts, hidden constraints).
- We improved our interface and game based on results from previous study. We show that the scalability is effective.
- We provide the following advice for designing such a interface and game in the context that we are discussing.
  - The simpler the control is, the more easily the user will understand. For example, we use drag & drop operation in our control which can be easily understood by anyone who has experience in Windows OS.
  - The analogy between the game and the underlying problem should be as direct as possible and as simple as possible.
  - Give the user only one goal to pursue.
  - Give the user constant feedback on his performance.

Overall, we introduced a new approach to solving a complex optimization problem by human beings. We demonstrated the feasibility of our approach through extensive user studies. Although the optimization problems discussed in this chapter have known objective functions and can be arguably solved approximately by heuristic or greedy algorithms,

we are the first to apply human-driven search approach to solving these problems. We envision that our approach can be extended and applied to solving more complex optimization problems including the complete NP-complete problem in Virtuoso.

In the future, we consider comparing human-driven search solutions with heuristic or greedy algorithms for the same optimization problem(s). It will also be interesting to see whether we can re-design the interface for expert users who understand the systems and problem, and whether they can use the interface to find better solutions compared with naive users.

Chapter 8

User- and Process-Driven Dynamic Voltage and Frequency Scaling

This dissertation argues for using direct human input to solve optimization problems in systems. In previous chapters, we showed that we can use human-driven techniques to solve increasing difficult optimization problems in a VM-based computing environment. To further show the feasibility and effectiveness of human-driven optimization, in this chapter we present an innovative solution to the power management problem. For completeness, a process-driven technique will also be discussed. This is joint work with Arindam Mallik, Gokhan Memik and Robert P. Dick.

Dynamic Voltage and Frequency Scaling (DVFS) is one of the most commonly used power reduction techniques in high-performance processors and is the most important OS power management tool. DVFS is generally implemented in the kernel and it varies the frequency and voltage of a microprocessor in real-time according to processing needs. Although there are different versions of DVFS, at its core DVFS adapts power consumption and performance to the current workload of the CPU. Specifically, existing DVFS techniques in high-performance processors select an operating point (CPU frequency and voltage) based on the utilization of the processor. While this approach can integrate information available to the OS kernel, such control is pessimistic.

Existing DVFS techniques are pessimistic about the user. Indeed, they ignore the user, assuming that CPU utilization or the OS events prompting it are sufficient proxies. A high CPU utilization simply leads to a high frequency and high voltage, regardless of the user's satisfaction or expectation of performance.

Existing DVFS techniques are pessimistic about the CPU. They assume worst-case manufacturing process variation and operating temperature by basing their policies on loose worst-case bounds given by the processor manufacturer. A voltage level for each frequency is set such that even the slowest shipped processor of a given generation will be stable at the highest specified temperature.

In response to these observations, on which we elaborate in Sections 8.1.1 and 8.2.1, we have developed two new power management techniques that can be readily employed independently or together. In particular, we introduce the following techniques.

User-Driven Frequency Scaling (UDFS) uses direct user feedback to drive an online control algorithm that determines the processor frequency (Section 8.1.2). Processor frequency has strong effects on power consumption and temperature, both directly and also indirectly through the need for higher voltages at higher frequencies. The choice of frequency is directly visible to the end-user as it determines the performance he sees. There is considerable variation among users with respect to the satisfactory performance level for a given workload mix. UDFS exploits this variation to customize frequency control policies dynamically to the *individual* user. Unlike previous work (Chapter 9), our approach employs direct feedback from the user during ordinary use of the machine.

Process-Driven Voltage Scaling (PDVS) creates a custom mapping from frequency and temperature to the minimum voltage needed for CPU stability (Section 8.2.2), taking advantage of process variation. This mapping is then used online to choose the operating voltage by taking into account the current operating temperature and frequency. Researchers have shown that process variation causes IC speed to vary up to 30% [14]. Hence, using

a single supply voltage setting does not exploit the variation in timing present among processors. We take advantage of this variation via a customization process that determines the slack of the *individual* processor, as well as its dependence on operating temperature. This offline measurement is used online to dynamically set voltage based on frequency and temperature.

We evaluate our techniques independently and together through user studies conducted on a Pentium M laptop running Windows applications. Our studies, described in detail in Section 8.3, include both single task and multitasking scenarios. We measure the overall system power and temperature reduction achieved by our methods. Combining PDVS and the best UDFS scheme reduces measured system power by 49.9% (27.8% PDVS, 22.1% UDFS), averaged across all our users and applications, compared to the Windows XP DVFS scheme. The average temperature of the CPU is decreased by 13.2°C on average. Using user trace-driven simulation to evaluate the CPU in isolation, we find average CPU dynamic power savings of 57.3% (32.4% PDVS, 24.9% UDFS), with a maximum reduction of 83.4%. In a multitasking environment, the same UDFS+PDVS technique reduces the CPU dynamic power by 75.7% on average.

Experimental setup

Our experiments were done using an IBM Thinkpad T43p with a 2.13 GHz Pentium M-770 CPU and 1 GB memory running Microsoft Windows XP Professional SP2. Although eight different frequency levels can be set on the Pentium M-770 processor, only six can be used due to limitations in the SpeedStep technology.

In all of our studies, we make use of three application tasks, some of which are CPU intensive and some of which frequently block while waiting for user input:

- Creating a presentation using Microsoft PowerPoint 2003 while listening to background music using Windows Media Player 10. The user duplicates a presentation

consisting of complex diagrams involving drawing and labeling, starting from a hard copy of a sample presentation.

- Watching a 3D Shockwave animation using the Microsoft Internet Explorer web browser. The user watches the animation and is encouraged to press the number keys to change the camera's viewpoint. The animation was stored locally. Shockwave options were configured so that rendering was done entirely in software on the CPU.

- Playing the FIFA 2005 Soccer game. FIFA 2005 is a popular and widely-used First Person Shooter game. There were no constraints on user gameplay.

In the following sections, we describe the exact durations of these tasks for each user study and additional tasks the user was asked to undertake. In general, our user studies are double-blind, randomized, and intervention-based. The default Windows DVFS scheme is used as the control. We developed a user pool by advertising our studies within a private university that has many non-engineering departments. We selected a random group of users from among those who responded to our advertisement. While many of the selected users were CS, CE, or EE graduate students, our users included staff members and undergraduates from the humanities. Each user was paid $15 for participating. Our studies ranged from number of users $n = 8$ to $n = 20$, as described in the material below.

8.1 User-driven frequency scaling

Current DVFS techniques are pessimistic about the user, which leads them to often use higher frequencies than necessary for satisfactory performance. In this section, we elaborate on this pessimism and then explain our response to it: user-driven frequency scaling (UDFS). Evaluations of UDFS algorithms are given in Section 8.3.

8.1.1 Pessimism about the user

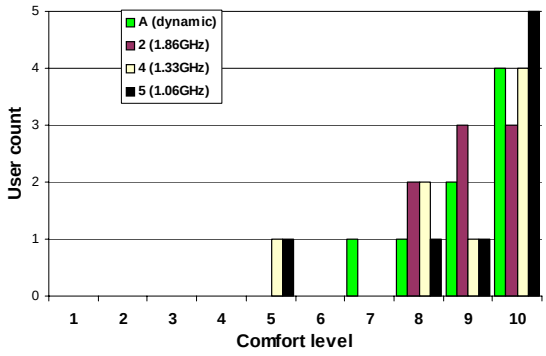
Current software that drives DVFS is pessimistic about the individual user's reaction to the slowdown that may occur when CPU frequency is reduced. Typically, the frequency is tightly tied to CPU usage. A burst of computation due to, for example, a mouse or keyboard event brings utilization quickly up to 100% and drives frequency, voltage, temperature, and power consumption up along with it. CPU-intensive applications also cause an almost instant increase in operating frequency and voltage.

In both cases, the CPU utilization (or OS events that drive it) is functioning as a proxy for user comfort. Is it a good proxy? To find out, we conducted a small ($n = 8$) randomized user study, comparing four processor frequency strategies including dynamic, static low frequency (1.06 GHz), static medium frequency (1.33 GHz), as well as static high frequency (1.86 GHz). The dynamic strategy is the default DVFS policy used in Windows XP Professional. Note that the processor maximum frequency is 2.13 GHz. We allowed the users to acclimate to the full speed performance of the machine and its applications for 4 minutes and then carry out the tasks described in Section 8, with the following durations:

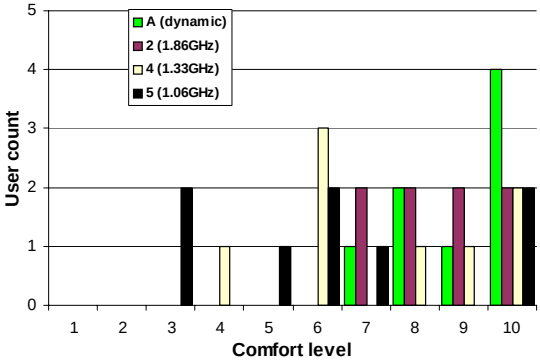
- PowerPoint (4 minutes in total, 1 minute per strategy)
- Shockwave (80 seconds in total, 20 seconds per strategy)
- FIFA (4 minutes in total, 1 minute per strategy)

Users verbally ranked their experiences after each task / strategy pair on a scale of 1 (discomfort) to 10 (very comfortable). Note that for each application and user, strategies were tested in random order.

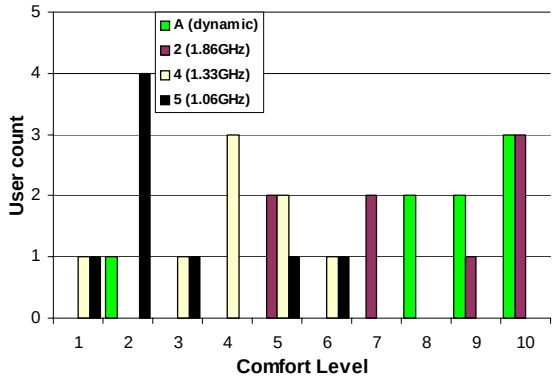
Figure 8.1 illustrates the results of the study in the form of overlapped histograms of the participants' reported comfort level for each of four strategies. Consider Figure 8.1(a), which shows results for the PowerPoint task. The horizontal axis displays the range of



(a) Microsoft PowerPoint.



(b) 3D Shockwave animation.



(c) FIFA Game.

Figure 8.1: User pessimism.

comfort levels allowed in the study and the vertical axis displays the count of the number of times that level was reported. The other graphs are similar.

User comfort with any given strategy is highly dependent on the application. For PowerPoint, the strategies are indistinguishable in their effectiveness. For this task, we could simply set the frequency statically to a very low value and never change it, presumably saving power. For animation, a higher static level is preferred but the medium and high frequencies are statistically indistinguishable from the dynamic strategy despite not using as high a frequency. For the game, the high static setting is needed to match the satisfaction level of the dynamic strategy. However, that setting does not use the highest possible frequency, which was used by the dynamic strategy throughout the experiment.

Comfort with a given strategy is strongly user-dependent, i.e., it is important to note that for any particular strategy, there is considerable spread in the reported comfort levels. In addition to the power-specific results just described, in Chapter 2 and 4 we have also demonstrated a high variation in user tolerance for performance in other contexts. Our dynamic policy automatically adapts to different users and applications. Hence, it can reduce power consumption while still achieving high user satisfaction.

8.1.2 Technique

To implement user-driven frequency scaling, we have built a system that consists of client software that runs as a Windows toolbar task as well as software that implements CPU frequency and temperature monitoring. In the client, the user can express discomfort at any time by pressing the F11 key (the use of other keys or controls can be configured). These events drive the UDFS algorithm. The algorithm in turn uses the Windows API to control CPU frequency. We monitor the CPU frequency using Windows Performance Counter and Log [135] and temperature using CPUCool [202].

It is important to note that a simple strategy that selects a static frequency for an appli-

cation (and/or for a user) is inadequate for three reasons. First, each user will be satisfied with a different level of performance for each application. Finding these levels statically would be extremely time consuming. Second, typical users multitask. Capturing the effects of multiple applications would necessitate examining the power set of the application set for each individual user, resulting in a combinatoric explosion in the offline work to be done. Finally, even when a user is working with a single application, the behavior of the application and the expected performance varies over time. Applications go through phases, each with potentially different computational requirements. In addition, the user's expected performance is also likely to change over time as the user's priorities shift. For these reasons, a frequency scaling algorithm should dynamically adjust to the individual user's needs.

Responding to these observations, we designed algorithms that employ user experience feedback indicated via button presses.

UDFS1 algorithm

UDFS1 is an adaptive algorithm that can be viewed as an extension/variant of the TCP congestion control algorithm. The TCP congestion control algorithm [17, 55, 173, 195] is designed to adapt the send rate dynamically to the available bandwidth in the path. A congestion event corresponds to a user button press, send rate corresponds (inversely) to CPU frequency, and TCP acknowledgments correspond to the passage of time.

UDFS1 has two state variables: r, the current control value (CPU frequency, the smaller the value, the higher the frequency.) and r_t (the current threshold, integervalue). Adaptation is controlled by three constant parameters: ρ, the rate of increase, $\alpha = f(\rho)$, the slow start speed, and $\beta = g(\rho)$, the additive increase speed. Like TCP, UDFS1 operates in three modes, as described below.

- Slow Start (Exponential Increase): If $r < r_t$, we increase r exponentially fast with

time (e.g., $r \propto 2^{\alpha t}$). Note that frequency settings for most processors are quantized and thus the actual frequency changes abruptly upon crossing quantization levels.

- User event avoidance (Additive Increase): If no user feedback is received and $r \geq r_t$, r increases linearly with time, $r \propto \beta t$.
- User event (Multiplicative Decrease): When the user expresses discomfort at level r we immediately set $r_t = r - 1$ and set r to the initial (highest) frequency.

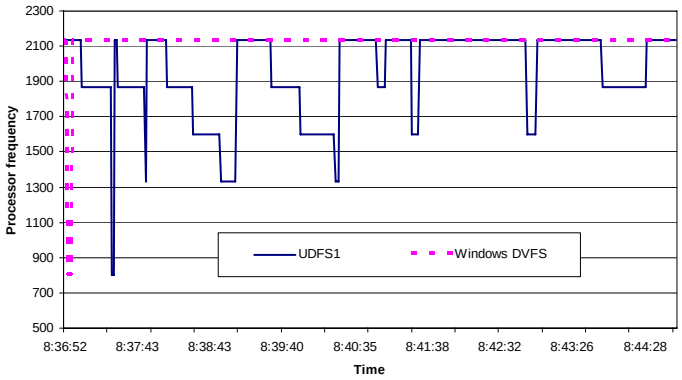
This behavior is virtually identical to that of TCP Reno, except for the more aggressive setting of the threshold.

Unlike TCP Reno, we also control ρ, the key parameter that controls the rate of exponential and linear increase from button press to button press. In particular, for every user event, we update ρ as follows:

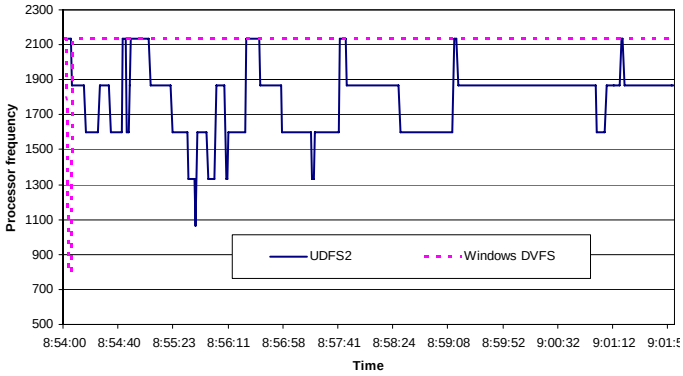
$$\rho_{i+1} = \rho_i \left(1 - \gamma \times \frac{T_i - T_{AVI}}{T_{AVI}}\right)$$

where T_i is the latest inter-arrival time between user events. T_{AVI} is the target mean inter-arrival time between user events, as currently preset by us. γ controls the sensitivity to the feedback.

We set our constant parameters ($T_{AVI} = 120, \alpha = 1, \beta = 1, \gamma = 0.8$) based on the experience of two of the authors using the system. These parameters were subsequently used when conducting a user study to evaluate the system (Section 8.3). Ideally, we would empirically evaluate the sensitivity of UDFS1 performance to these parameters. However, it is important to note that any such study would require having real users in the loop, and thus would be quite slow. Testing five values of each parameter on 20 users would require 312 days (based on 8 users/ day and 45 minutes/user). For this reason, we decided to choose the parameters based on qualitative evaluation by the authors and then "close the loop" by evaluating the whole system with the choices.



(a) UDFS1 scheme



(b) UDFS2 scheme

Figure 8.2: The frequency for UDFS schemes during FIFA game for a representative user.

Figure 8.2(a) illustrates the execution of the UDFS1 and Windows DVFS algorithms for a typical user during the FIFA game task. Note that Windows DVFS causes the system to run at the highest frequency during the whole execution period except the first few seconds. On the other hand, the UDFS1 scheme causes the processor frequency to increase only when the user expresses discomfort (by pressing F11). Otherwise, it slowly decreases.

UDFS2 algorithm

UDFS2 tries to find the lowest frequency at which the user feels comfortable and then stabilize there. For each frequency level possible in the processor, we assign an interval t_i, the time for the algorithm to stay at that level. If no user feedback is received during the interval, the algorithm reduces the frequency from r_i to r_{i+1}, i.e., it reduces the frequency by one level. The default interval is 10 seconds for all levels. If the user is irritated at control level r_i, we update all of our intervals and the current frequency level as follows:

$$\begin{aligned} t_{i-1} &= \alpha t_{i-1} \\ t_k &= \beta t_k, \forall k : k \neq i-1 \\ i &= \min(i-1, 0) \end{aligned}$$

Here $\alpha > 1$ is the rate of interval increase and $\beta < 1$ is rate of interval decrease. In our study, $\alpha = 2.5$ and $\beta = 0.8$. This strategy is motivated by the conjecture that the user was comfortable with the previous level and the algorithm should spend more time at that level. Again, because users would have to be in the inner loop of any sensitivity study, we have chosen the parameters qualitatively and evaluated the whole system using that choice, as described in Section 8.3. Figure 8.2(b) illustrates the execution of the algorithm for a representative user in the FIFA game task. Note that UDFS2 settles to a frequency of

approximately 1.86 GHz, after which little interaction is needed.

8.2 Process-driven voltage scaling

Current DVFS techniques are pessimistic about the processor, which leads them to often use higher voltages than necessary for stable operation, especially when they have low temperatures. We elaborate on this pessimism and then explain our response to it, process-driven voltage scaling (PDVS). PDVS is evaluated in Section 8.3.

8.2.1 Pessimism about the CPU

The *minimum stable voltage* of a CPU is the supply voltage that guarantees correct execution for given process variation and environmental conditions. It is mainly determined by the critical path delay of a circuit. This delay consists of two components: transistor gate delay and wire delay. Gate delay is inversely related to the operating voltages used in the critical paths of the circuit. Furthermore, temperature affects the delay. In current technologies, carrier mobility in MOS transistors decreases with increasing temperature. This causes the circuits to slow down with increasing temperature. Wire delay is also temperature-dependent and increases under higher current/temperature conditions. The maximum operating frequency (F_{max}) varies in direct proportion to the sustained voltage level in the critical timing paths, and inversely with temperature-dependent RC delay [193].

In addition to the operating conditions, which dynamically change, process variation has an important impact on the minimum voltage sufficient for stable operation. Even in identical environments, a variation in timing slack is observed among the manufactured processors of the same family. As a result, each processor reacts differently to changes. For example, although two processors can run safely at 2.8 GHz at the default supply voltage, it is conceivable that these minimum supply voltages will differ. Customizing voltage

choices for individual processors adapts to, and exploits, these variations. Despite these known effects of process variation and temperature on minimum stable voltage, DVFS ignores them: for a given frequency, traditional DVFS schemes use a single voltage level for all the processors within a family at all times.

The dynamic power consumption of a processor is directly related to frequency and supply voltage and can be expressed using the formula $P = V^2CF$, which states that power is equal to the product of voltage squared, capacitance, and frequency. In addition to its direct impact on the power consumption, reliable operation at increased frequency demands increased supply voltage, thereby having an indirect impact on power consumption. Generally, if the frequency is reduced, a lower voltage is safe.

As processors, memories, and application-specific integrated circuits (ASICs) are pushed to higher performance levels and higher transistor densities, processor thermal management is quickly becoming a first-order design concern. The maximum operating temperature of an Intel Pentium Mobile processor has been specified as 100°C [89, 90]. As a general rule of thumb, the operating temperature of a processor can vary from 50°C to 90°C during normal operation. Thus, there is a large difference between normal and worst-case temperatures.

We performed an experiment that reveals the relationship between operating frequency and minimum stable voltage of the processor at different temperature ranges. We used Notebook Hardware Control (NHC) [92] to set a particular V_{dd} value for each operating frequency supported by the processor. When a new voltage value is set, NHC runs an extensive CPU stability check. Upon failure, the system stops responding and computer needs to be rebooted. We execute a program that causes high CPU utilization and raises the temperature of the processor. When the temperature reaches a desired range, we perform the CPU stability check for a particular frequency at a user-defined voltage value.

Figure 8.3 shows the results of this study for the machine described in Section 8. For

Operating Freq. (MHz)	Nominal Voltage (v)	Stable V_{dd} (V) at temp ranges (°C)			
		52–57	62–67	72–77	82–87
800	0.988	0.736	0.736	0.736	0.736
1,060	1.068	0.780	0.780	0.780	0.780
1,200	1.100	0.796	0.796	0.796	0.796
1,330	1.132	0.844	0.844	0.860	0.876
1,460	1.180	0.876	0.892	0.908	0.924
1,600	1.260	0.908	0.924	0.924	0.924
1,860	1.324	1.004	1.004	1.020	1.020
2,130	1.404	1.084	1.100	1.116	1.116

Figure 8.3: Minimum stable V_{dd} for different operating frequencies and temperatures.

reference, we also show the nominal core voltage given in the datasheet [90]. Note that the nominal voltage is the voltage used by all the DVFS schemes by default. The results reveal that, even at the highest operating temperature, the minimum stable voltage is far smaller than the nominal voltage. The results also show that at lower operating frequencies, the effect of temperature on minimum stable voltage is not pronounced. However, temperature change has a significant impact on minimum stable voltage at higher frequencies. In particular, at 1.46 GHz, the core voltage value can vary by 5.6% for a temperature change of 30°C. This would reduce dynamic power consumption by 11.4%.

As the results shown in Figure 8.3 illustrate, there is an opportunity for power reduction if we exploit the relationship between frequency, temperature, and the minimum stable voltage. The nominal supply voltage specified in the processor datasheet has a large safety margin over the minimum stable voltages. This is not surprising: worst-case assumptions were unnecessarily made at a number of design stages, e.g., about temperature. Conventional DVFS schemes are therefore pessimistic about particular *individual* CPUs, often choosing higher voltages than are needed to operate safely. They also neglect the effect of temperature, losing the opportunity to save further power.

8.2.2 Technique

We have developed a methodology for exploiting the process variation described in Section 8.2.1 that can be used to make *any* voltage and frequency scaling algorithm adapt to individual CPUs and their temperature, thereby permitting a reduction in power consumption.

Our technique uses offline profiling of the processor to find the minimum stable voltages for different combinations of temperature and frequency. Online temperature and frequency monitoring is then used to set the voltage according to the profile. The offline profiling is virtually identical to that of Section 8.2.1 and needs to be done only once. Currently, it is implemented as a watchdog timer-driven script on a modified Knoppix Live CD that writes the profile to a USB flash drive. To apply our scheme, the temperature is read from the online sensors that exist in the processor. The frequency, on the other hand, is determined by the dynamic frequency scaling algorithm in use. By setting the voltage based on the processor temperature, frequency, and profile, we adapt to the operating environment. While the frequency can be readily determined (or controlled), temperature changes dynamically. Hence, the algorithm has built-in filtering and headroom to account for this fact. Our algorithm behaves conservatively and sets the voltage such that even if there is a change of 5°C in temperature before the next reading (one Hertz rate), the processor will continue working correctly.

A reader may at this point be concerned that our reduction of the timing safety margin from datasheet norms might increase the frequency of timing errors. However, PDVS carefully determines the voltage required for reliable operation for each processor; that is, it finds the *individual* processor's safety margin. Moreover, it decreases the operating temperature of the processor, which reduces the rates of lifetime failure processes. If characteristics of processors change as a result of wear, PDVS can adapt by infrequently, e.g.,

every six months, repeating the offline characterization process. To determine processor reliability when using reduced operating voltage, we ran demanding programs test the stability of different processor components, e.g., the ALU, at lower voltages. We have set the processor to work at modified supply voltages as indicated in Figure 8.3. The system remained stable for approximately two months, at which point we terminated testing. Although observing the stable operation of one machine does not prove reliability, it is strong evidence.

8.3 Evaluation

We now evaluate UDFS and PDVS in isolation and together. We compare against the native Windows XP DVFS scheme, displaying reductions in power and temperature.

Our evaluations are based on user studies, as described in Section 8 and elaborated upon here. For studies not involving UDFS, we trace the user's activity on the system as he uses the applications and monitor the selections DVFS makes in response. For studies involving UDFS, the UDFS algorithm is used online to control the clock frequency in response to user button presses. We begin by describing a user study of UDFS that provides both independent results and traces for later use. Next, we consider PDVS as applied to the Windows DVFS algorithm. We then consider UDFS with and without PDVS, comparing to Windows DVFS. Here, we examine both dynamic CPU power (using simulation driven from the user traces) and system power measurement (again for a system driven from the user traces). In measurement, we consider not only power consumption, but also CPU temperature. Finally, we discuss a range of other aspects of the evaluation of the system.

The following claims are supported by our results:

- UDFS effectively employs user feedback to customize processor frequency to the individual user. This typically leads to significant power savings compared to exist-

ing dynamic frequency schemes that rely only on CPU utilization as feedback. The amount of feedback from the user is infrequent, and declines quickly over time as an application or set of applications is used.

- PDVS can be easily incorporated into any existing DVFS scheme, such as the default Windows scheme, and leads to dramatic reductions in power use by lowering voltage levels while maintaining processor stability.

- In most of the cases, the effects of PDVS and UDFS are synergistic: the power reduction of UDFS+PDVS is more than the sum of its parts.

- Multitasking increases the effectiveness of UDFS+PDVS.

- Together and separately, PDVS and UDFS typically decrease CPU temperature, often by large amounts, increasing both reliability and longevity. In addition, the effects of PDVS and UDFS on temperature are synergistic.

8.3.1 UDFS

To evaluate the UDFS schemes, we ran a study with 20 users. Experiments were conducted as described in Section 8. Each user spent 45 minutes to

1. Fill out a questionnaire stating level of experience in the use of PCs, Windows, Microsoft PowerPoint, music, 3D animation video, and FIFA 2005 (2 minutes) from among the following set: “Power User”, “Typical User”, or “Beginner”;

2. Read a one page handout (2 minutes);

3. Acclimate to the performance of our machine by using the above applications (5 minutes);

4. Perform the following tasks for UDFS1: Microsoft PowerPoint plus music (4 minutes); 3D Shockwave animation (4 minutes); FIFA game (8 minutes); and

5. Perform the same set of tasks for UDFS2.

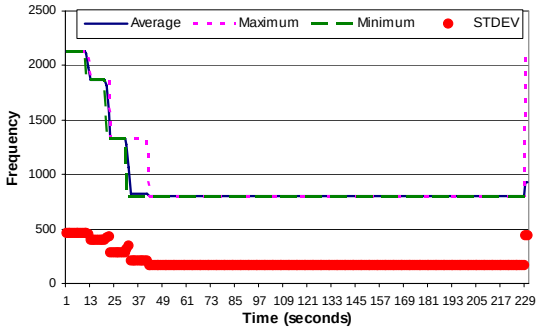
Each user was instructed to press the F11 key upon discomfort with application performance. We recorded each such event as well as the CPU frequency over time.

The written protocol and the form the user filled out can be found in Appendix E.

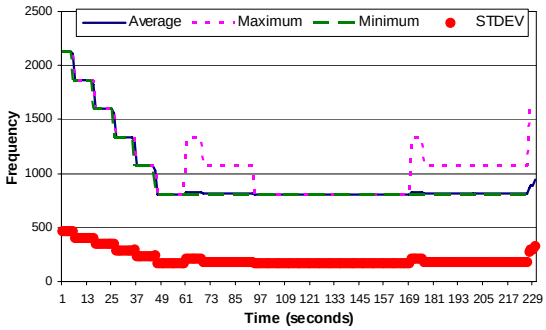
Figure 8.4 illustrates the performance of the two algorithms in our study. The two columns represent UDFS1 and UDFS2 and the three rows represent the three applications. Each graph shows, as a function of time, the minimum, average, maximum, and standard deviation of CPU frequency, aggregated over our 20 users. Notice that almost all users felt comfortable using PowerPoint while the processor was running at the lowest frequency. As one might expect, the average frequency at which users are comfortable is higher for the Shockwave animation and the FIFA game. There is large variation in acceptable frequency among the users for the animation and game. Generally, UDFS2 achieves a lower average frequency than UDFS1. For both algorithms it is very rare to see the processor run at the maximum CPU frequency for these applications. Even the most sophisticated users were comfortable with running the tasks with lower frequencies than those selected by the dynamic Windows DVFS scheme. Sections 8.3.3 and 8.3.4 give detailed, per-user results for UDFS (and UDFS+PDVS).

8.3.2 PDVS

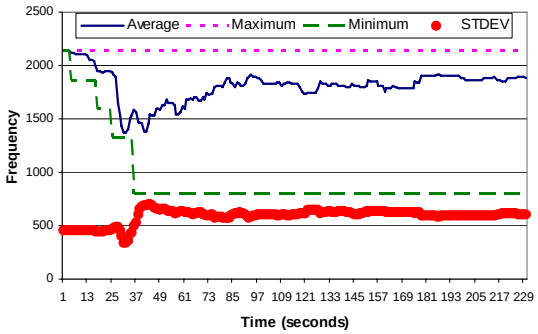
Using the experimental setup described in Section 8, we evaluate the effects of PDVS on the default Windows XP DVFS scheme. In particular, we run the DVFS scheme, recording frequency, then determine the power saving possible by setting voltages according to PDVS instead of using the nominal voltages of DVFS.



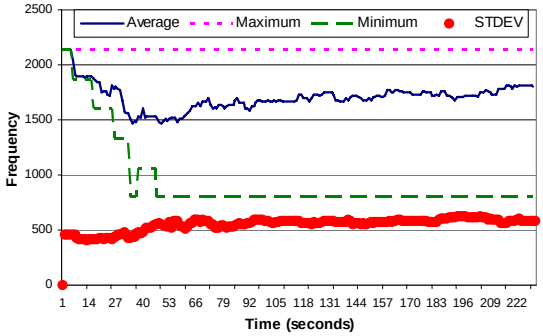
(a) UDFS1 - PowerPoint



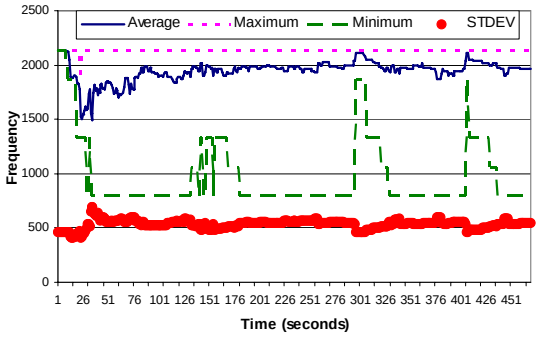
(b) UDFS2 - PowerPoint



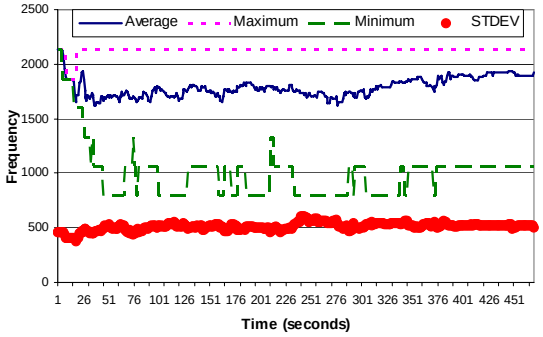
(c) UDFS1 - Shockwave



(d) UDFS2 - Shockwave



(e) UDFS1 - Game



(f) UDFS2 - Game

Figure 8.4: Frequency over time for UDFS1 and UDFS2, aggregated over 20 users.

Application	Power Reduction (%) over Max Frequency	
	DVFS	DVFS+PDVS
PowerPoint + Music	83.08	90.67
3D Shockwave Animation	3.19	40.67
FIFA Game	1.69	39.69

Figure 8.5: Power reduction for Windows DVFS and DVFS+PDVS

Figure 8.5 illustrates the average results, comparing stock Windows DVFS and our DVFS+PDVS scheme. The baseline case in this experiment is running the system with the highest possible CPU frequency and its corresponding nominal voltage. The maximum power savings due to dynamic frequency scaling with nominal voltages are observed for PowerPoint. For this application, the system ran at the lowest clock frequency most of the time, resulting in a reduction of 83.1% for the native DVFS scheme. DVFS+PDVS reduces the power consumption by 90.7%. For PowerPoint, adding PDVS to DVFS only reduces power slightly.

For the Shockwave animation and the FIFA game, the power reductions due to dynamic frequency scaling are negligible because the Windows DVFS scheme runs the processor at the highest frequency most of the time. DVFS+PDVS, however, improves the energy consumption of the system by approximately 40%, compared to the baseline. These results clearly demonstrate the benefits of process-driven voltage scaling.

8.3.3 UDFS+PDVS (CPU dynamic power, trace-driven simulation)

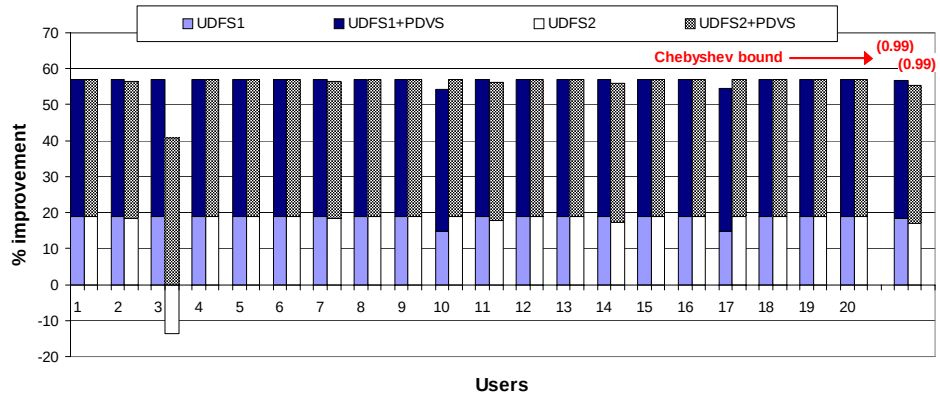
To integrate UDFS and PDVS, we used the system described in Section 8.1.2, recording frequency over time. We then combine this frequency information with the offline profile and techniques described in Sections 8.2.1 and 8.2.2 to derive CPU power savings for UDFS with nominal voltages, UDFS+PDVS, and the default Windows XP DVFS strategy. We calculate the power consumption of the processor. We have also measured online the

power consumption of the overall system, as described in Section 8.3.4.

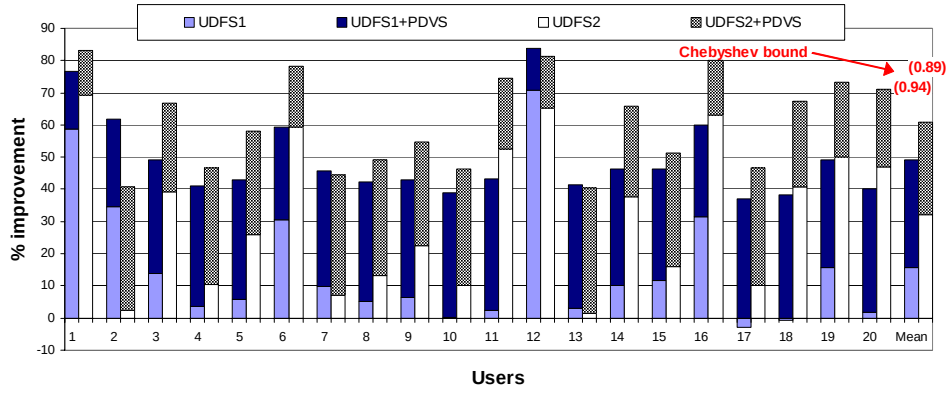
We conducted a user study ($n = 20$) with exactly the same structure presented in Section 8.1.2, except that Windows XP DVFS was also considered. Figure 8.6 presents both individual user results and average results for UDFS1, UDFS1+PDVS, UDFS2, and UDFS2+PDVS. In each case, power savings over the default Windows DVFS approach are reported. To interpret the figure, first choose an application. Next, note the last two bars on the corresponding graph. These indicate the average performance of UDFS1 and UDFS2, meaning the percentage reduction in power use compared to Windows DVFS. Each bar is broken into two components: the performance of the UDFS algorithm without PDVS is the lower component and the improvement in performance of the algorithm combined with PDVS is the upper component. The remaining bars on the graph have identical semantics, but represent user-specific information.

For PowerPoint, UDFS1+PDVS and UDFS2+PDVS reduce power consumption by an average of 56%. The standalone UDFS algorithms reduce it by an average of 17–19%. User 3 with UDFS2 is anomalous. This user pressed the feedback button several times and as a result spent most of the time at high frequencies.

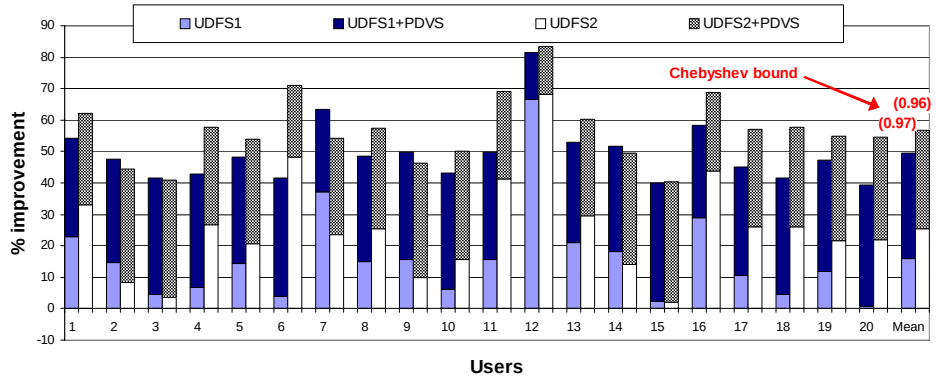
For the Shockwave animation, we see much more mixed responses from the users, although on average we reduce power by 55.1%. On average, UDFS1 and UDFS2 independently reduce the power consumption by 15.6% and 32.2%, respectively. UDFS2 performs better for this application because the users can be satisfied by ramping up to a higher frequency rather than the maximum frequency supported by the processor. Note that UDFS1 immediately moves to the maximum frequency on a button press. User 17 with UDFS1 is anomalous. This user wanted the system to perform better than the hardware permitted and thus pressed the button virtually continuously even when it was running at the highest frequency. Adding PDVS lowers average power consumption even more significantly. On average, the power is reduced by 49.2% (UDFS1+PDVS) and 61.0% (UDFS2+PDVS) in



(a) Microsoft PowerPoint



(b) 3D Shockwave animation



(c) FIFA game

Figure 8.6: Comparison of UDFS algorithms, UDFS+PDVS, and Windows XP DVFS (CPU Dynamic Power). Chebyshev bound-based $(1-p)$ values for difference of means from zero are also shown.

the combined scheme.

There is also considerable variation among users for the FIFA game. Using conventional DVFS, the system always runs at the highest frequency. The UDFS schemes try to throttle down the frequency over the time. They therefore reduce the power consumption even in the worst case (0.9% and 2.1% for UDFS1 and UDFS2, respectively) while achieving better improvement, on average (16.1% and 25.5%, respectively). Adding PDVS improves the average power savings to 49.5% and 56.7% for UDFS1 and UDFS2, respectively.

For the Shockwave animation and the FIFA game, we see a large variation among users, but in all cases the combination of PDVS and UDFS leads to power savings over Windows DVFS. On average, in the best case, the power consumption can be reduced by 57.3% over existing DVFS schemes for all three applications. This improvement is achieved by combining the UDFS2 (24.9%) and PDVS (32.4%) schemes.

UDFS and PDVS are synergistic. The UDFS algorithms let us dramatically decrease the average frequency, and PDVS's benefits increase as the frequency is lowered. At higher frequencies, the relative change from the nominal voltage to the minimum stable voltage is lower than that at lower frequencies. In other words, the power gain from shifting to the minimum stable voltage is higher at the lower frequencies. However, at higher frequencies, PDVS also gains from the variation in minimum stable voltage based on temperature as shown in Figure 8.3. These two different advantages of the PDVS result in power improvements at at a wide range of frequencies.

UDFS+PDVS mean results have statistical significance even with weak bounds. Figure 8.6 shows mean improvements across our 20 users. Normality assumptions hold neither for the distribution of individual user improvements nor for the error distribution of the mean. Instead, to discard the null hypothesis, that our mean improvements for UDFS+PDVS are not different from zero, we have computed the p value for discarding

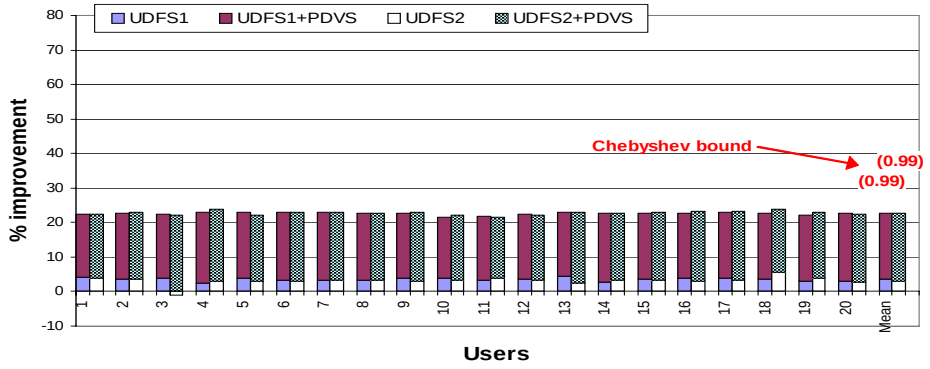
the null hypothesis using Chebyshev bounds, which are looser but rely on no such assumptions. As can be seen from the figure, $1 - p$ is quite high, indicating that it is extremely unlikely that our mean improvements are due to chance. We use Chebyshev bounds similarly for other results.

User self-reported level of experience correlates with power improvement. For example, for FIFA, experienced users expect faster response from the system causing the system to run at higher frequencies, resulting in smaller power improvements. Our interpretation is that familiarity increases both expectations and the rate of user feedback to the control agent, making annoyance with reduced performance more probable and thus leading to higher frequencies when using the UDFS algorithms.

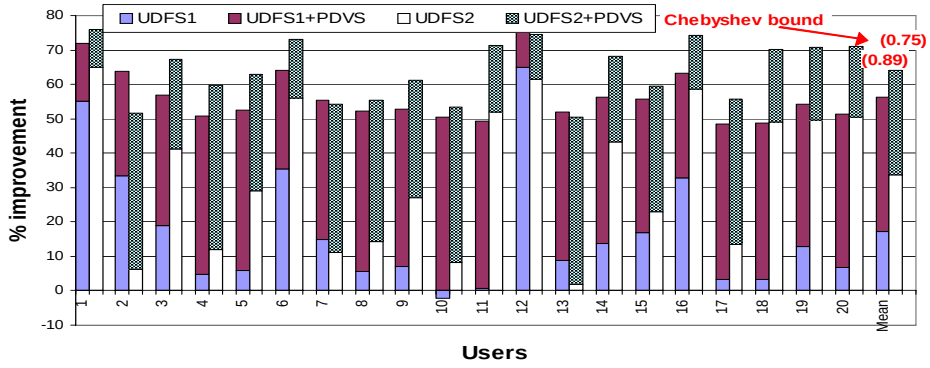
8.3.4 UDFS+PDVS (System power and temperature measurement)

To further measure the impact of our techniques, we replay the traces from the user study of the previous section on our laptop. The laptop is connected to a National Instruments 6034E data acquisition board attached to the PCI bus of a host workstation running Linux, which permits us to measure the power consumption of the entire laptop. The sampling rate is 10 Hz. During the measurements, we have turned off the display of the laptop to make our readings more comparable to the CPU power consumption results of the previous section. Ideally, we would have preferred to measure CPU power directly for one-to-one comparison with results of the previous section, but we do not have the surface mount rework equipment needed to do so.

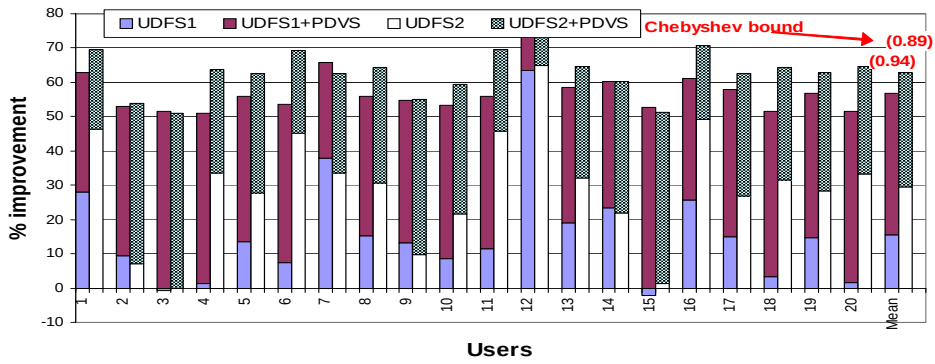
Power: Figure 8.7 presents results for UDFS1, UDFS1+PDVS, UDFS2, and UDFS2+PDVS, showing the power savings over the default Windows DVFS approach. The Chebyshev bounds indicate that the mean improvements are extremely unlikely to have occured by chance.



(a) Microsoft PowerPoint



(b) 3D Shockwave animation



(c) FIFA Game

Figure 8.7: Comparison of UDFS algorithms, UDFS+PDVS, and Windows XP DVFS (measured system power with display off). Chebyshev bound-based $(1-p)$ values for difference of means from zero are also shown.

For PowerPoint, UDFS1+PDVS and UDFS2+PDVS reduce power consumption by averages of 22.6% and 22.7%, respectively. For the Shockwave animation, although we see much more variation, UDFS1 and UDFS2 reduce the power consumption by 17.2% and 33.6%, respectively. Using UDFS together with PDVS lowers average power consumption by 38.8% and 30.4% with UDFS1 and UDFS2, respectively. The FIFA game also shows considerable variation among users. On average, we save 15.5% and 29.5% of the power consumption for UDFS1 and UDFS2, respectively. Adding PDVS improves the average power savings to 56.8% and 62.9% over Windows DVFS with UDFS1 and UDFS2, respectively.

On average, the power consumption of the overall system can be reduced by 49.9% for all three applications. This improvement is achieved by combining the UDFS2 scheme (22.1%) and PDVS scheme (27.8%).

The results presented in the previous section, and in this section, cannot be directly compared because the previous section reports the simulated power consumption of the CPU and this section reports the measured power consumption of the laptop. However, some conclusions can be drawn from the data in both sections. For applications like PowerPoint, where the CPU consumes only a small fraction of the system power, the benefit on system power is low. On the other hand, for the applications that originally result in high CPU power consumption, the system power savings can be substantial due to the reduction in dynamic power as well as the operating temperatures and consequently leakage power.

Temperature: We used CPUCool [202] to measure CPU temperature in the system. Figure 8.8 shows the mean and peak temperatures of the system when using the different combinations of DVFS, PDVS, and UDFS schemes. The values reported for UDFS and UDFS+PDVS are the averages over 20 users.

In all cases, the UDFS1 and UDFS2 schemes lower the temperature compared to the

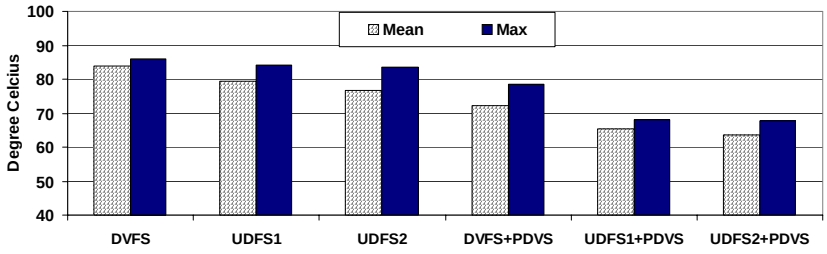
Windows native DVFS scheme due to the power reductions we have reported in the previous sections. The maximum UDFS temperature reduction is seen in the case of the UDFS2 scheme used for the Shockwave application (7.0°C). On average, for all 3 applications, the UDFS1 and UDFS2 schemes reduce the mean temperature of the system by 1.8°C and 3.8°C, respectively. Similarly, PDVS reduces the mean system temperature by 8.7°C on average for the three applications. The best improvement is observed for the FIFA game, where temperature decreases by 12.6°C.

The combination of PDVS and UDFS is again synergistic, leading to even greater temperature reductions than PDVS or UDFS, alone. For the Shockwave application, UDFS2+PDVS reduces the mean temperature by 19.3°C. The average temperature reductions in all three applications by the UDFS1+PDVS and UDFS2+PDVS schemes are 12.7°C and 13.7°C, respectively. Our 13.2°C claim averages these two.

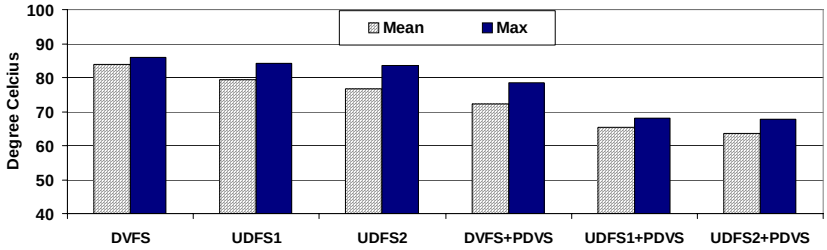
8.3.5 Discussion

We now discuss the degree of user interaction needed to make UDFS work, the CPU reliability and longevity benefits of our techniques, and the effects of multitasking.

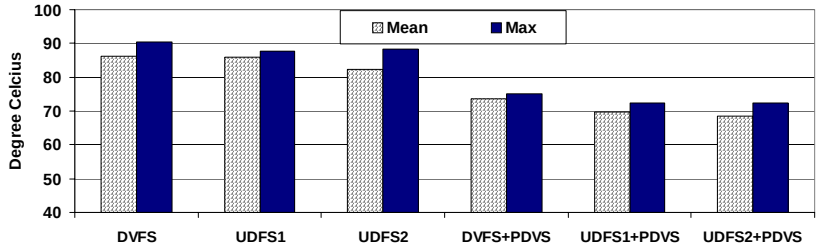
User interaction: While PDVS can be employed without user interaction, UDFS requires occasional feedback from the user. Minimizing the required rate of feedback button presses while maintaining effective control is a central challenge. Our current UDFS algorithms perform reasonably well in this respect, but could be improved. Figure 8.9 presents the average number of annoyance button presses over a 4 minute period for both versions of UDFS algorithms in our 20 user study. Generally, UDFS2 requires more frequent button presses than UDFS1, because a single press only increments the frequency. The trade-off is that UDFS1 generally spends more time at the maximum frequency and thus is more power hungry. On average, a user pressed a button every 8 minutes for PowerPoint, ev-



(a) Microsoft PowerPoint



(b) 3D Shockwave animation



(c) FIFA game

Figure 8.8: Mean and peak temperature measurement.

Algorithms	PowerPoint 4 min	3D animation 4 min	FIFA Game 4 min	FIFA Game 4 min
UDFS1	0.35	11.85	5.10	3.42
UDFS2	0.60	14.25	6.50	3.82

Figure 8.9: Average number of user events.

ery 18 seconds for the Shockwave animation, and every 50 seconds for the FIFA game. During the course of the study, for the 3D animation, there were some extreme cases in which the user kept pressing the button even when the processor was running at the highest frequency. This can be explained by the user's dissatisfaction with the original quality of the video or the maximum performance available from the CPU, over which we had no control. If we omit the three most extreme cases from both maximum and minimum number of annoyances, on average a user presses the annoyance button once every 30 seconds for the Shockwave application.

We also note that the system adapts to users quickly, leading to a reduced rate of button presses. In the Figure 8.9, we show both the first and second 4 minute interval for the FIFA game. The number of presses in the second interval is much smaller than the first. Our interpretation is that once a stable frequency has been determined by the UDFS scheme, it can remain at that frequency for a long time, without requiring further user interaction.

Figure 8.10 records the average number of voltage transitions for the six different schemes used in our study. A voltage transition is caused either due to a button press or a significant change in operating temperature. For the PowerPoint application, we observe a reduction in the number of transitions because the spikes observed for DVFS do not occur for UDFS1 and UDFS2. On the other hand, the 3D animation and FIFA Game applications have more voltage transitions than observed with Windows native DVFS, because they aim to reduce power by adjusting throttle and, in effect, voltage. In contrast, conventional DVFS keeps the system at the highest frequency during the entire interval. The increase in the number of transitions for the PDVS schemes implemented on top of UDFS are caused by the extra voltage transitions due to changing temperature at a given frequency level.

Applications	DVFS	DVFS+PDVS	UDFS1	UDFS1+PDVS	UDFS2	UDFS2+PDVS
PowerPoint+Music	11.00	11.00	4.40	4.65	6.55	6.50
3D Animation	3.00	4.00	10.30	11.50	16.3	17.55
FIFA Game	6.00	6.00	18.06	18.05	28.85	29.30

Figure 8.10: Number of voltage transitions

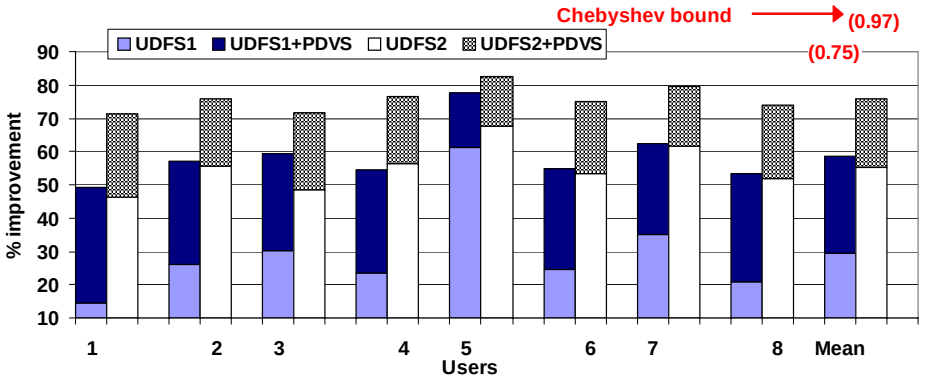


Figure 8.11: Power improvement in the multitasking environment. Chebyshev bound-based $(1-p)$ values for difference of means from zero are also shown.

Reliability and longevity: In addition to its direct impact on power consumption, our techniques may ultimately improve the lifetime reliability of a system. Earlier research [169] showed that the effect of operating temperature on integrated circuit's mean time to failure (MTTF) is exponential. As we show in Section 8.3.4, our schemes can reduce the operating temperature by 13.2°C on average, thereby potentially reducing the rate of failure due to temperature-dependant processes such as electromigration.

Traditionally, the required supply voltage of a processor is reported at the maximum operating temperature of the system. Therefore, at temperatures below the maximum rated temperature, timing slack exists. As long as the current temperature is below the highest rated operating temperature, the operating voltage can be reduced below the rated operating voltage without reducing reliability below that of the same processor operating at the rated voltage and at the maximum temperature.

Multitasking: A natural question to ask is whether the extremely simple "press the button" user feedback mechanism we use in UDFS is sufficient for describing user preferences in a multitasking environment. To see the effect of UDFS in a multitasking environment, we conducted a small study ($n = 8$) similar to that of Section 8.3.1. Instead of several consecutive tasks, the user was asked to watch a 3D animation using Microsoft Internet Explorer while listening to MP3 music using Windows Media Player in compact mode with visualization.

Figure 8.11 shows the measured system power improvements compared to Windows DVFS. On average, the power consumption of the overall system is reduced by 29.5% and 55.1% for UDFS1 and UDFS2, respectively. Adding PDVS improves the average power savings to 58.6% and 75.7% for UDFS1 and UDFS2, respectively. Although these results are preliminary, combined with the results from the combined PowerPoint+MP3 task described in Section 8.3.1, they suggest that the simple feedback mechanism is sufficient in a multitasking environment. It is clearly a better proxy of the user's satisfaction than the CPU utilization of the combined task pool.

8.4 Conclusions

We have identified processor and user pessimism as key factors holding back effective power management for processors with support for DVFS. In response, we have developed and evaluated the following new, process- and user-adaptive DVFS techniques: process-driven voltage scaling (PDVS) and user-driven frequency scaling (UDFS). These techniques dramatically reduce CPU power consumption in comparison with existing DVFS techniques. Extensive user studies show that we can reduce power on average by over 50% for single task and over 75% for multitasking workloads compared to the Microsoft Windows XP DVFS scheme. Furthermore, CPU temperatures can be markedly decreased

through the use of our techniques. PDVS can be readily used along with any existing frequency scaling approach. UDFS requires that user feedback be used to direct processor voltage and frequency control. PDVS and UDFS are synergistic. UDFS leads to lower average frequencies and PDVS allows great decreases in voltage at low frequencies. We are currently exploring machine learning techniques to develop UDFS algorithms that require even less input from the user.

UDFS provides evidence for the feasibility of human-driven specification part of my thesis beyond Virtuoso domain. That is, it is possible to use direct human input from users to determine at least some objective function and constraints, what is the lowest CPU frequency that can satisfy the user in this case.

So far, we have applied the technique of letting the user convey a single bit of information to the systems in three different problems: understanding and measuring user comfort with resource borrowing (Chapter 2), user-driven scheduling of interactive virtual machines (Chapter 3), and power management for laptops. The technique works in the user-driven scheduling context, but the button feedback is less desirable than the two-dimensional joystick control of Chapter 4. The technique works very well in both the user comfort and power management problems. Here I would like to point out some differences between these two works. In the user comfort problem, by using irritation feedback, we only seek the relationship between user comfort and resource borrowing. Our results can then be used online without future feedback by implementors of distributed computing and thin-client frameworks to intelligently borrow resources or decide necessary resource share of desktop replacement VMs without irritating the user. In the power management problem, button-press feedback directly drives adaptive algorithms that drive processor frequency setting. As a reward for being sporadically interrupted, the user prolongs his battery life for his laptop.

Despite these differences, we have showed that given considerable variation in user

satisfaction with any given interactive setting, the user can and should directly inform the systems software of his satisfaction with the current delivered performance, which results from the current interactive setting.

Chapter 9

Related work

The work described in this dissertation is related to work in a number of other areas.

9.1 Users and direct user input

Direct user input is represented in HCI and other fields. For example, early work [45, 131] used buttons as on-screen objects that encapsulated code to enable tailoring of applications. Weighted fair queuing allows users to explicitly weight each of their processes, controlling the CPU share given to each. PSDoom [27] represented processes as monsters in a first person shooter game. "Wounding" a "monster process" decreases its priority. With enough "damage", the process is killed. 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', the user 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. Without such knowledge, the user can easily live-lock or even crash the system. Ours is the first scheduling system to incorporate direct user input from even naive users.

Other related work in HCI research and psychology has concentrated on the impact of latency on user-perceived utility of the system [49, 103], and on user frustration with different user interfaces [101, 151]. Earlier work [24] also studied the effect of computer response time on user response time.

Within the systems community, related work has examined the performance of end-user operating systems using latency as opposed to throughput [51], and suggested models for interactive user workload [11]. The TIPME [50] tool monitors information about high-level GUI events and transitions on operating system state. When the user experiences unacceptable performance, he presses a hot-key sequence and describes the problem. TIPME concentrates on diagnosing the sources of user-perceived delays, but it cannot diagnose resource contention problems.

9.2 Human decision making

Understanding decision complexity would appear to be in the purview of human-computer interaction research and psychology. However, the work in those areas [9, 36, 62, 94, 98, 147, 166, 172, 199] has concentrated on understanding how human beings make decisions in general. And the cognitive or perceptual models in those field are very complex and not practical to be directly borrowed to benchmark complexity. In addition, none of those models were developed under the specific goal of understanding how non-expert system administrators make decisions in performing a complex configuration process.

For example, the traditional normative models of decision making prescribe that people assign either an objective or subjective value to an option and then factor in the opinion's probability [9, 172]. It is almost impossible to measure such perceptual value and probability in the real world including IT configuration, not to mention that research has shown a variety of ways in which people deviate from the normative models. For another example,

the Prospect Theory [98], which provides a general theory of decision making that explains how people's reasoning deviates from normative models, models people's decisions by a descriptive $\pi(p)$ function, which represents the subjective perception of probabilities [172]. Obviously, it is not very practical to calculate such functions in the real world.

9.3 Service-level agreements (SLAs)

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 [143], QuOin [208], and Amaranth [82] assume that SLAs or QoS constraints are supplied to them. Many forms for this information have been proposed, including composable utility functions [149], decision procedures [15], and aspects [127]. Our work (Chapter 4) not only presents a way of discovering appropriate SLAs from end-user input, but also 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 [139], exposing debugging of configurations as a search process [197], adjusting cycle stealing so as to control impact on users [177], and using performance feedback to the administrator to help adjust policy [124]. As far as we are aware, however, no work in autonomic computing directly incorporates the end-user.

9.4 Process and VM scheduling

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 [134], lottery scheduling [194], weighted fair queuing and its derivatives [10], BVT [48], SRPT scheduling [6], and periodic [96, 118] and sporadic [120] hard real-time models. 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.

Existing approaches to scheduling VMs running under a type-II VMM on Linux (and other Unixes) are insufficient to meet the needs of different workloads such as batch VMs, batch parallel and interactive VMs. By default, these VMs are scheduled as ordinary dynamic-priority processes with no timing or compute rate constraints at all. VMware ESX server [187] and virtual server systems such as Ensim [52] improve this situation by providing compute rate constraints using weighted fair queuing [10] and lottery scheduling [194]. However, these are insufficient to schedule a workload-diverse set of VMs on a single physical machine because they either provide no timing constraints or do not allow for the timing constraints to be smoothly varied. Fundamentally, they are rate-based. For example, an interactive VM in which a word processing application is being used may only need 5% of the CPU, but it will need to be run at least every 50 ms or so. Similarly, a VM that is running a parallel application may need 50% of the CPU, and be scheduled together with its companion VMs. The closest VM-specific scheduling approach to ours is the VServer [116] slice scheduling in the PlanetLab testbed [152]. However, these slices are created a priori and fixed. Our VSched tool, discussed in Chapter 4, provides dynamic

scheduling.

9.5 Real-time scheduling

The theory of periodic real-time scheduling dates to the 70s [118].

Periodic real-time scheduling systems for general-purpose operating systems have been developed before. Most relevant to our work is Polze's scheduler [148], which created soft periodic schedules for multimedia applications by manipulating priorities under Windows NT. DSRT [33], SMART [141], and Rialto [96] had similar objectives. In contrast, VSched is a Linux tool, provides remote control for systems like Virtuoso, and focuses on scheduling VMs. Linux SRT, defunct since the 2.2 kernel, was a set of kernel extensions to support soft real-time scheduling for multimedia applications under Linux [88]. The RBED system [162] also provides real-time scheduling for general Linux processes through kernel modifications. The Xen [47] virtual machine monitor uses BVT [48] scheduling with a non-trivial modification of Linux kernel and requires that the hosted operating system be ported to Xen. In contrast to these systems, VSched can operate entirely at user-level.

There have been several hard real-time extensions to Linux. The best known of these are Real-time Linux [207], RTAI [46], and KURT [83]. We examined these tools (and Linux SRT as well) before deciding to develop VSched. For our purposes, the hard real-time extensions are inappropriate because real-time tasks must be written specifically for them. In the case of Real-time Linux, the tasks are even required to be kernel modules. VSched can optionally use KURT's UTIME high resolution timers to achieve very fine-grained scheduling of VMs.

9.6 Scheduling parallel applications

Our work in Chapter 5 ties to gang scheduling, implicit co-scheduling, real-time schedulers, and feedback control real-time scheduling. As far as we aware, we are the first to develop real-time techniques for scheduling parallel applications that provide performance isolation and control. We also differ from these areas in that we show how external control of resource use (by a cluster administrator, for example) can be achieved while maintaining commensurate application execution rates. That is, we can reconcile administrator and user concerns.

The goal of gang scheduling [95, 146] is to "fix" the blocking problems produced by blindly using time-sharing local node schedulers. The core idea is to make fine-grain scheduling decisions collectively over the whole cluster. For example, one might have all of an application's threads be scheduled at identical times on the different nodes, thus giving many of the benefits of space-sharing, while still permitting multiple applications to execute together to drive up utilization, and thus allowing jobs into the system faster. In essence, this provides the performance isolation we seek, while performance control depends on scheduler model. However, gang scheduling has significant costs in terms of the communication necessary to keep the node schedulers synchronized, a problem that is exacerbated by finer grain parallelism and higher latency communication [86]. In addition, the code to simultaneously schedule all tasks of each gang can be quite complex, requiring elaborate bookkeeping and global system knowledge [175].

Implicit co-scheduling [5] attempts to achieve many of the benefits of gang scheduling without scheduler-specific communication. The basic idea is to use communication irregularities, such as blocked sends or receives, to infer the likely state of the remote, uncoupled scheduler, and then adjust the local scheduler's policies to compensate. This is quite a powerful idea, but it does have weaknesses. In addition to the complexity inherent

in inference and adapting the local communication schedule, the approach also doesn't really provide a straightforward way to control effective application execution rate, response time, or resource usage.

9.7 Feedback-based control

The feedback control real-time scheduling project at the University of Virginia [128–130, 170] had a direct influence on our thinking for Chapter 5. In that work, concepts from feedback control theory were used to develop resource scheduling algorithms to give quality of service guarantees in unpredictable environments to applications such as online trading, agile manufacturing, and web servers. In contrast, we are using concepts from feedback control theory to manage a tightly controlled environment, targeting parallel applications with collective communication.

Feedback-based control was also used to provide CPU reservations to application threads running on a single machine based on measurements of their progress [171], for controlling coarse-grained CPU utilization in a simulated virtual server [205], for dynamic database provisioning for web servers [28], and to enforce web server CPU entitlements to control response time [121].

9.8 Dynamic voltage and frequency scaling

Dynamic voltage and frequency scaling (DVFS) is an effective technique for microprocessor energy and power control for most modern processors [18, 69]. Energy efficiency has been a major concern for mobile computers. Fei et al. [56] proposed an energy aware dynamic software management framework that improves battery utilization for mobile computers. However, this technique is only applicable to highly adaptive mobile applications. Researchers have proposed algorithms based on workload decomposition [31], but these

tend to provide power improvements only for memory-bound applications. Wu et al. [203] presented a design framework of a run-time DVFS optimizer in a general dynamic compilation system. The Razor [54] architecture dynamically finds the minimal reliable voltage level. Dhar et al. [41] proposed adaptive voltage scaling that uses a closed-loop controller targeted towards standard-cell ASICs. These schemes are similar to the PDVS scheme. However, our approach is completely operating system controlled and does not require any architectural modifications and therefore incurs no hardware overhead. Intel Foxton technology [196] provides a mechanism for select Intel Itanium 2 processors to adjust core frequency during operation to boost application performance. However, unlike PDVS it does not perform any dynamic voltage setting.

Other DVFS algorithms use task information, such as measuring response times in interactive applications [123, 206] as a proxy for the user. Unlike Vertigo [59], we monitor the *user* instead of the application. Xu et al. proposed novel schemes [204] minimizing energy consumption in real-time embedded systems that execute variable workloads. However, they try to adapt to the variability of the workload rather than to the users. AutoDVS [78] is a dynamic voltage scaling (DVS) system for hand-held devices. They used user activity as an indicator to detect computationally intensive CPU intervals and use that to drive DVS. In contrast, UDFS uses user activity to directly control the frequency of the system. Ranga et al. proposed energy-aware user interfaces [150] based on usage scenarios, but they concentrated on the display rather than the CPU. Gupta et al. [76] and Lin et al. [110] demonstrated a high variation in user tolerance for performance in the scheduling context, variation that we believe holds for power management as well. Anand et al. [1] discussed the concept of a control parameter that could be used by the user. However, they focus on the wireless networking domain, not the CPU. Second, they do not propose or evaluate a user interface or direct user feedback. To the best of our knowledge, the UDFS component of our work is the first to employ direct user feedback instead of a proxy for

the user.

9.9 Dynamic thermal management

Dynamic thermal management is an important issue for modern microprocessors due to the high cost of cooling solutions. Previous work has discussed microarchitectural modeling and optimization based on temperature [35, 154, 167]. Liu and Svensson made a trade-off between speed and supply voltage [119]. Brooks and Martonosi [19] proposed dynamic thermal management for high-performance processors. For portable computers, Transmeta's Crusoe [185] and Intel's Pentium-M [69] are notable commercial products that uses innovative dynamic thermal management. To the best of our knowledge, the PDVS component of our work is the first to consider exploiting process variation via per-CPU customization using profiling. In addition, it is the first scheme to consider temperature in voltage level decisions.

9.10 Games with a purpose

"Games with a purpose" [188] refers to the idea that through online games, people can collectively solve large-scale computational problems. In [189], people play a game to help determine the contents of images by providing meaningful labels for them. [192] tailored a game to collect image metadata. The same approach was adopted in [190, 191].

Our game for a collection of VMs, as presented in Chapter 7, shares a similar idea, although our focus is on using human input approach to solve optimization problems. In general, the human input can be directed to a game, an interface, a simulator and many other things. In term of game design, although we conducted a controlled study on the game and interface, we can imagine extending the framework to become online and access broader audience.

Chapter 10

Conclusions

This dissertation explores human-driven optimization — using direct human input to solve optimization problems in adaptive and autonomic computing systems. Optimization problems associated with these systems are often difficult to pose well and solve efficiently. To address this, we proposed two ideas here. In human-driven specification, we explore how to use direct human input to pose specific problems, namely to determine the objective function and/or the hidden constraints. In human-driven search, we explore how to use direct human input to guide the search for a good solution, a valid configuration that optimizes the objective function.

We motivated the necessity of direct human input by presenting a user study on understanding user comfort with resource borrowing systems. The study revealed an important fact that resources needed to keep the user happy are highly dependent on the user as well as the application. In other words, the traditional assumption that systems can optimize for a canonical user is invalid. The study further provided evidence for the feasibility of human-driven specification through a button-press feedback mechanism. We then applied the same button-press feedback on scheduling interactive VM by manipulating the priority of the VM process under Linux.

We also argued for using periodic real-time scheduling model for VM-based computing

environment. We designed, implemented and evaluated a user-level scheduler (VSched) for Linux that provide this model. The scheduler allows us to mix long-running batch computations with fine-grained interactive applications such as first-person-shooter games with no reduction in usability of the interactive applications. In addition, the scheduler enables the design of a joystick interface for controlling the CPU schedule of a VM. By using a joystick and a on-screen display of real-time cost, a naive user can instantaneously change the performance of his interactive VM and quickly find a trade-off between the cost of his VM and his own comfort. This makes the case for using human-driven search in single machine CPU scheduling problem.

We further combined local VScheds on each node of a cluster with a global feedback control system that can time-share multiple parallel applications with high efficiency while providing performance isolation. The user/administrator can dynamically change the target execution rate of his application. The work shows the effect of using human-driven specification on CPU scheduling across multiple machines.

To show that we can apply human-driven search on optimizing CPU schedules of multiple VMs as well as the VM-to-host mapping, we designed and evaluated a game-like interface, through which the user can control his VMs by simply moving and drag-and-dropping balls. The interface will dynamically display both short-term and long-term efficiency of all VMs. We designed the interface to be generic instead of specific to an application. We used a game style presentation to access broader audience. We conducted two controlled user studies to let naive users play the optimization game using the interface. The results showed the effectiveness of our interface and the feasibility of human-driven search.

We further extended our human-driven approach to address power management problem. We proposed two techniques: user-driven frequency scaling (UDFS) and process-driven voltage scaling (PDVS). UDFS dynamically adapts CPU frequency to the individual

user and the workload through direct user feedback of button-press, unlike currently-used DVFS methods that do not use user feedback. In PDVS, a CPU-customized profile is derived offline that encodes the minimum voltage needed to achieve stability at each combination of CPU frequency and temperature. UDFS makes another case for human-driven specification in a non-Virtuoso domain.

Overall this dissertation advocated human-driven optimization and showed its feasibility by solving increasing difficult optimization problems in Virtuoso and also in power management. This work has the potential to dramatically increase the range of applications for which adaptation is possible and increases the scope of autonomic computing.

In the remainder of this chapter, we summarize the detailed contributions of this dissertation, and show how we intend to extend the framework in the future.

10.1 Summary of contributions

- I co-developed a sophisticated distributed application for directly measuring user comfort with the borrowing of CPU time, memory space, and disk bandwidth. Using this tool, we have conducted a controlled user study with qualitative and quantitative results that are of direct interest to the designers of grid and thin-client systems. We have found that resource borrowing can be quite aggressive without creating user discomfort, particularly in the case of memory and disk. The resources needed to keep a user happy are highly dependent on the application being used. Our observations formed the basis of advice for the implementors of distributed computing and thin-client frameworks. (Chapter 2, [75, 76])

- I proposed human-driven optimization - using direct human input in solving optimization problems in adaptive and autonomic computing. I propose two ideas: human-driven specification and human-driven search. The former uses direct human

input to pose specific problems. The latter uses direct human input to search for a solution to the problem. (Chapter 1)

- I proved the feasibility of my thesis in four different optimization problems: single machine CPU scheduling, multiple machine CPU scheduling, multiple machine CPU scheduling with VM mapping, and power management.

- I designed a scheduling system that uses direct user feedback through button-press to balance between providing high average computation rates to the non-interactive VMs while keeping the users of the interactive VMs happy. I used an adaptive algorithm that exploits the nice mechanism of Linux. Results show that it is possible to provide interactive performance while noninteractive VMs progress much faster than would otherwise be possible. (Chapter 3, [111])

- I proposed real-time scheduling model as the right one for VM-based distributed computing. The model allows us to straightforwardly mix batch and interactive VMs and allows users to succinctly describe their performance demands. (Chapter 4, [109])

- I designed and developed VSched, a user-level scheduler for Linux, and released it online. I evaluated its performance on several different platforms and found that we can achieve very low deadline miss rates up to quite high utilizations and quite fine resolutions. I evaluated the scheduler and showed that we can schedule long-running batch computations with fine-grained interactive applications such as first-person-shooter games with no reduction in usability of the interactive applications. I also applied VSched to scheduling parallel workloads, showing that it can help a BSP application maintain a fixed stable performance despite externally caused load imbalance. (Chapter 4, [109])

- I designed 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 doing so, I 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. (Chapter 4, [110, 112])

- Together with my colleague Ananth I. Sundararaj, we proposed, implemented, and evaluated a new self-adaptive approach to time-sharing parallel applications on tightly coupled compute resources such as clusters. Our technique, performance-targeted feedback-controlled realtime scheduling, is based on the combination of local scheduling using the periodic real-time model and a global feedback control system that sets the local schedules. The approach performance-isolates parallel applications and allows administrators to dynamically change the desired application execution rate while keeping actual CPU utilization automatically proportional to the application execution rate. We evaluated the system and showed it to be stable with low response times. The thresholds needed to prevent control instability are quite reasonable. Despite only isolating and controlling the CPU, we find that memory, communication I/O, and local disk I/O follow. (Chapter 5, [114])

- I developed a event graph generator which can take the raw data from XPVM on monitoring a PVM application, parse it and convert it into a plottable DAG format. The generator further generates a trace file which can be used as the input to a trace-driven simulator. I manually gathered many PVM traces by running Patterns bench-

mark with different configurations in Virtuoso, using XPVM and my event graph generator. Those trace files are not only used by my colleague in his dissertation but also used later on in my user studies. (Chapter 6)

- Virtuoso system simulator: I participated in the verification and validation of the first-generation of an application trace-driven simulator. I designed and developed the second-generation of the simulator, which can simulate the periodic real-time scheduling on multiple hosts, and can handle inputs from the user. The latest simulator allows the user to dynamically change the configuration (CPU schedule, mapping) of a simulated VM in the middle of the simulation. I then combined the simulator with a novel user interface which can dynamically update simulation status to the user while taking his input to change the configuration. (Chapter 6)

- I designed and developed an interface for the user to easily control the CPU scheduling and VM mapping for a collection of VMs. The interface is generic. Using this interface, I designed an optimization game so that even a naive user can play without knowing the background information. I conducted two user studies. The results showed that the interface and game are effective. Most users can quickly optimize the system to a certain extent. The game explores the idea of "games with a purpose" by solving a complex system problem through game-play, although our main focus is to apply human-driven search. The results also shed light into future design of similar interface and game. (Chapter 7)

- Together with my colleagues Arindam Mallik, Gokhan Memik, and Robert P. Dick, we identified processor and user pessimism as key factors holding back effective power management for processors with support for DVFS. In response, we developed and evaluated two new, process- and user-adaptive DVFS techniques: user-driven frequency scaling (UDFS) and process-driven voltage scaling (PDVS). These

techniques dramatically reduce CPU power consumption in comparison with existing DVFS techniques. We conducted extensive user studies showing we can reduce power on average by over 50% for single task- and over 75% for multitasking workloads compared to the Microsoft Windows XP DVFS scheme. Furthermore, CPU temperatures can be markedly decreased through the use of our techniques. UDFS requires that user feedback be used to direct processor voltage and frequency control. PDVS can be readily used along with any existing frequency scaling approach. PDVS and UDFS are synergistic. UDFS leads to lower average frequencies and PDVS allows great decreases in voltage at low frequencies. (Chapter 8, [113, 132])

- I developed a model of decision complexity in the context of understanding the complexity of IT configuration. The model includes three factors: constraints, levels of guidance and consequences. Based on the model, I conduct a carefully controlled online user study in an analogous route-planning domain. The results revealed the important fact that decision complexity has significantly different impacts on user-perceived difficulty than on objective measures like time and error rate. And I identified the key factors affecting decision complexity, which we used to extract some basic guidance for reducing complexity. I also proposed our next step on validating the model in real IT contexts. (Appendix A, [107, 108])

I want to emphasize two fundamental contributions of this dissertation

- I demonstrated that it can be extremely effective to involve human beings in solving optimization problems arising from systems, especially those with unknown objective functions and hidden constraints.

- I developed and evaluated techniques for taking advantage of the considerable variation in user satisfaction in many interactive settings, and using that “harvested variation” to optimize system performance and user comfort.

10.2 Future work

My future research plans is to extend both my scheduling and direct user input work to solve system problems in broader fields, including the following:

10.2.1 Power-aware multi-core scheduling

One natural extension to my current user-driven power management work is to apply the same approach on multi-core platforms. It is an increasing trend for processor manufacturers to combine multi/many CPU cores into single integrated circuit (IC). Such a change in the hardware is not only changing the software programming model but also opening up a new research space for power management. One challenge for current multi-core mobile environment is load-balancing among cores to maximize throughput and resource utilization, while minimizing the power consumption. It will be beneficial for the future OS scheduler to be able to predict and finely control the power consumption of individual core.

One scenario where this approach may be especially successful is runtime monitoring for detecting program bugs. It is increasingly popular to run monitoring tools on different cores than the monitored programs, and hence not to compete for cycles, registers or L1 cache. However typical monitor tools run slower than the monitored program due to extra code that they need to execute for checking. As the result, the program needs to be frequently paused or stopped whenever a certain checking is in progress. Swapping in another program to keep the CPU busy may be a solution but may incur extra OS context switch overhead. An alternative decision is to slow down the program core through frequency and voltage scaling. However the scheduler needs to be intelligent enough in order to save power in the long term. The scheduler need to consider both the variation among applications and its monitoring tools, as well as the history of power saving decisions.

My future plan includes in-depth problem study and literature survey, new scheduling model proposal and development of a real scheduler. I did my initial work [29] on this topic at Intel Research working with Dr. Michael A. Kozuch and I plan to continue/extend it after graduation.

10.2.2 Optimization problems in wide-area distributed systems

I plan to look into adaptation problems in wide-area distributed computing systems and explore human-driven optimization. Problem assumptions and constraints will surely be changed when we are facing even the similar problems while in a wide-area distributed environment. Besides the difference in both the applications and problems, we can expect application users to have different requirement and behaviors than those cluster users. Accordingly, new user input/feedback new techniques need to be proposed, developed and evaluated.

An example of such an adaptation problem is in P2P (peer-to-peer) systems, where people share their own resources (e.g. disk space, network bandwidth and etc) in order to utilize resources from other people. However resource sharing will lead to user irritation if the level of resource sharing is not adapted to individual user, as discovered in our user comfort study. As the result, people tend to be conservative in sharing their own resources. Systems like BitTorrent will punish those users by limiting their access to resources contributed by other people. The whole system will therefore run less efficiently and go into a vicious circle. How to optimize both for the user's comfort and for his desired utilization of the resource, while maximizing the efficiency of the overall P2P system, will be an interesting research problem. Research in the P2P field has been done on general user incentives with the goal to motivate more users to contribute more resources. Recent P2P clients also allow users to specify their acceptable maximum resource share, for example upload rate. However, to my knowledge, none of them can dynamically and simultane-

ously optimize for individual user's comfort, utilization request, and the overall system. In the future I plan to study existing P2P systems, formalize the problem and propose a practical solution to it. Evaluation of the solution will be either through simulation or user study.

10.2.3 Online games for solving optimization problems

A natural extension to the optimization game, presented in Chapter 7, is to target more complex optimization problems, especially those with unknown objective functions and/or constraints. A good example of those problems is the complete NP-complete problem in Virtuoso introduced in the beginning of this dissertation. More specifically, in Virtuoso, how to use inferred information about the application, network, and hosts to engage the adaptation and reservation mechanisms in order to increase the application's performance while satisfying the user. Due to the complexity of the problem, the interface needs to be extended to expose sufficient information to the user. In addition, multi-dimensional ($>$ 3, e.g. CPU schedule, VM mapping and network) control is necessary although it is very challenging. To access more users, we can take the similar approach as in [189] to make a online game. Combining direct human input and a way to exploit mass human intelligence, we see great potential in this future work.

Bibliography

[1] ANAND, M., NIGHTINGALE, E., AND FLINN, J. Self-tuning Wireless Network Power Management. In *The Ninth Annual International Conference on Mobile Computing and Networking (MobiCom)* (2003).

[2] ANDERSON, J., AND SRINIVASAN, A. Pfair scheduling: Beyond periodic task systems. In *Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications* (2000).

[3] ANDERSON, T. E., CULLER, D. E., AND PATTERSON, D. A. A case for now (networks of workstations). *IEEE Micro 15*, 1 (1995), 54–64.

[4] ARABE, J. N. C., BEGUELIN, A., LOWEKAMP, B., SELIGMAN, E., STARKEY, M., AND STEPHAN, P. Dome: Parallel programming in a heterogeneous multi-user environment. Tech. Rep. CMU-CS-95-137, Mellon University, School of Computer Science, April 1995.

[5] ARPACI-DUSSEAU, A. C., CULLER, D. E., AND MAINWARING, A. Scheduling with implicit information in distributed systems. In *ACM Sigmetrics* (1998).

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

[7] BARUAH, S. K., COHEN, N. K., PLAXTON, C. G., AND VARVEL, D. A. Proportionate progress: a notion of fairness in resource allocation. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing* (1993).

[8] BARUAH, S. K., GEHRKE, J., AND PLAXTON, C. G. Fast scheduling of periodic tasks on multiple resources. In *Proceedings of the 9th International Symposium on Parallel Processing* (1995).

[9] BELL, D. E., RAIFFA, H., AND TVERSKY, A., Eds. *Decision Making: Descriptive, Normative and Prescriptive Interactions*. Cambridge University Press, October 19 1988.

[10] BENNETT, J. C., AND ZHANG, H. Worst-case fair weighted fair queueing. In *Proceedings of the IEEE INFOCOM* (1996).

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

[12] BLYTHE, J., DEELMAN, E., GIL, Y., KESSELMAN, C., AGARWAL, A., MEHTA, G., AND VAHI, K. The role of planning in grid computing. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)* (2003).

[13] BOLKER, E. D., DING, Y., FLYNN, W., SHEETZ, D., AND SOMIN, Y. Interpreting Windows NT processor queue length measurements. In *Proceedings of the 31st Computer Measurement Group Conference* (2002).

[14] BORKAR, S., KARNIK, T., NARENDRA, S., TSCHANZ, J., KESHAVARZI, A., AND DE, V. Parameter Variations and Impact on Circuits and Microarchitecture. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)* (2003).

[15] 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 on Multimedia Transport and Teleservices* (1994).

[16] BRACHMAN, B. Ms Windows NT Win32 subsystem kernel thread scheduling simulator. http://www.808multimedia.com/winnt/simulator.htm.

[17] BRAKMO, L. S., O'MALLEY, S. W., AND PETERSON, L. L. Tcp vegas: new techniques for congestion detection and avoidance. In *Proceedings of the conference on Communications architectures, protocols and applications* (1994).

[18] BROCK, B., AND RAJAMANI, K. Dynamic power management for embedded systems. In *Proceedings of the IEEE SOC Conference* (2003).

[19] BROOKS, D., AND MARTONOSI, M. Adaptive Thermal Management for High-Performance Microprocessors. In *Workshop on Complexity Effective Design* (2000).

[20] BROWN, A. B., AND HELLERSTEIN, J. L. An approach to benchmarking configuration complexity. In *Proceedings of the 11th ACM SIGOPS European Workshop* (2004).

[21] BROWN, A. B., AND HELLERSTEIN, J. L. Reducing the cost of it operations–is automation always the answer? In *Proceedings of the 10th Workshop on Hot Topics in Operating Systems (HotOS)* (2005).

[22] BROWN, A. B., KELLER, A., AND HELLERSTEIN, J. L. A model of configuration complexity and its application to a change management system. In *Proceedings of the Ninth IFIP/IEEE International Symposium on Integrated Network Management (IM)* (2005).

[23] BROWN, K. P., MEHTA, M., CAREY, M. J., AND LIVNY, M. Towards Automated Performance Tuning For Complex Workloads. In *Proceedings of the Twentieth International Conference on Very Large Databases* (1994).

[24] BUTLER, T. W. Computer response time and user performance. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (1983).

[25] CHANDRA, A., ADLER, M., AND SHENOY, P. Deadline fair scheduling: Bridging the theory and practice of proportionate-fair scheduling in multiprocessor servers. In *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium (RTSA)* (2001).

[26] CHANDRA, R., ZELDOVICH, N., SAPUNTZAKIS, C., AND LAM, M. The collective: A cache-based system management architecture. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)* (2005).

[27] CHAO, D. Doom as an interface for process management. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (2001).

[28] CHEN, J., SOUNDARARAJAN, G., AND AMZA, C. Autonomic provisioning of databases in dynamic content web servers. In *Proceedings of the 3rd IEEE International Conference on Autonomic Computing* (2006).

[29] CHEN, S., FALSAFI, B., GIBBONS, P. B., KOZUCH, M., MOWRY, T. C., TEODORESCU, R., AILAMAKI, A., FIX, L., GANGER, G. R., LIN, B., AND SCHLOSSER, S. W. Log-Based Architectures for General-Purpose Monitoring of Deployed Code. In *Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability (ASID)* (1996).

[30] CHIEN, A. A., CALDER, B., ELBERT, S., AND BHATIA, K. Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing 63*, 5 (2003), 597–610.

[31] CHOI, K., SOMA, R., AND PEDRAM, M. Dynamic Voltage and Frequency Scaling based on Workload Decomposition. In *Proceedings of The 2004 International Symposium on Low Power Electronics and Design (ISLPED)* (2004).

[32] CHONG, S. How to track a user's idle time. *The Code Project* (Nov 2001). http://www.codetools.com/dll/trackuseridle.asp.

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

[34] 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).

[35] COHEN, A., FINKELSTEIN, F., MENDELSON, A., RONEN, R., AND RUDOY, D. On estimating optimal performance of cpu dynamic thermal management. *IEEE Computer Architecture Letters 2*, 1 (2003), 6.

[36] CROZIER, R. W., AND SVENSON, O. *Decision Making: Cognitive Models and Explanations*, 1st ed. Routledge, Nov 1997.

[37] CURTIN, M., AND DOLSKE, J. A brute force search of des keyspace. *;login:* (May 1998).

[38] CYBENKO, G. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing 7*, 2 (1989), 279–301.

[39] DEMON STAR. Demonstar. http://www.demonstar.co.uk/.

[40] DENVER, A. Using the performance data helper library. *Microsoft Developers Network Library* (March 1997).

[41] DHAR, S., MAKSIMOVIC, D., AND KRANZEN, B. Closedloop adaptive voltage scaling controller for standard cell asics. In *Proceedings of The International Symposium on Low Power Electronics and Design (ISLPED)* (2005).

[42] DINDA, P. A. The statistical properties of host load. *Scientific Programming 7*, 3,4 (1999). A version of this paper is also available as CMU Technical Report CMU-CS-TR-98-175.

[43] DINDA, P. A., AND O'HALLARON, D. R. Realistic CPU workloads through host load trace playback. In *Proceedings of the 5th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR)* (2000).

[44] DOUGLIS, F., AND OUSTERHOUT, J. Transparent process migration: Design alternatives and the Sprite approach. *Software Practice and Experience 21*, 7 (1991), 1–27.

[45] DOURISH, P. Evolution in the adoption and use of collaborative technologies. In *Proceedings of the Sixth European Conference on Computer Supported Cooperative Work (ECSCW)* (1999).

[46] DOZIO, L., AND MANTEGAZZA, P. Real-time distributed control systems using rtai. In *Proceedings of the Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing* (2003).

[47] DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., PRATT, I., WARFIELD, A., BARHAM, P., AND NEUGEBAUER, R. Xen and the art of virtualization, 2003.

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

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

[50] ENDO, Y., AND SELTZER, M. Improving interactive performance using tipme. In *Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (2000).

[51] 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).

[52] ENSIM CORPORATION. http://www.ensim.com.

[53] EPEMA, D. H. J. Decay-usage scheduling in multiprocessors. *ACM Transactions on Computer Systems (TOCS) 16*, 4 (1998), 367–415.

[54] ERNST, D., KIM, N. S., DAS, S., PANT, S., PHAM, T., RAO, R., ZIESLER, C., BLAAUW, D., AUSTIN, T., AND MUDGE, T. Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)* (2003).

[55] FALL, K., AND FLOYD, S. Simulation-based comparisons of tahoe, reno and sack tcp. *SIGCOMM Computer Communications Review 26*, 3 (1996), 5–21.

[56] FEI, Y., ZHONG, L., AND JHA, N. K. An Energy-aware Framework for Coordinated Dynamic Software Management in Mobile Computers. In *IEEE/ACM Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (2004).

[57] FIGUEIREDO, R., DINDA, P., AND FORTES, J., Eds. *Special Issue On Resource Virtualization*. IEEE Computer. IEEE, May 2005.

[58] FIGUEIREDO, R., DINDA, P. A., AND FORTES, J. A case for grid computing on virtual machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003)* (2003).

[59] FLAUTNER, K., AND MUDGE, T. Vertigo: Automatic Performance-Setting for Linux. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)* (2002).

[60] FOSTER, I., AND KESSELMAN, C., Eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[61] FREY, J., TANNENBAUM, T., FOSTER, I., LIVNY, M., AND TUECKE, S. Condor-g: A computation management agent for multi-institutional grids. In *Proceedings of the 10th International Symposium on High Performance Distributed Computing (HPDC)* (2001).

[62] GADOMSKI, A. M. Cognitive decision-making: Toga socio-cognitive approach. White paper, Aug 2006.

[63] GAJIC, Z. Working with keyboard hooks. `http://delphi.about.com/library/weekly/aa101000a.htm`.

[64] GALIC, M., HALLIDAY, A., HATZIKYRIACOS, A., MUNARO, M., PAREPALLI, S., AND YANG, D. *A Secure Portal Using WebSphere Portal V5 and Tivoli Access Manager V4.1*. Redbooks. IBM, Dec 2003.

[65] GEISS, R. Results of some quick research on timing in win32. `http://www.geisswerks.com/ryan/FAQS/timing.html`.

[66] GEIST, A., BEGUELIN, A., DONGARRA, J., JIANG, W., MANCHECK, R., AND SUNDERAM, V. *PVM: Parallel Virtual Machine*. MIT Press, Cambridge, Massachusetts, 1994.

[67] GERBESSIOTIS, A. V., AND VALIANT, L. G. Direct bulk-synchronous parallel algorithms. *Journal of Parallel and Distributed Computing 22*, 2 (1994), 251–267.

[68] GHORMLEY, D. P., PETROU, D., RODRIGUES, S. H., VAHDAT, A. M., AND ANDERSON, T. E. Glunix: A global layer unix for a network of workstations. *Software Practice and Experience 28*, 9 (1998), 929–961.

[69] GOCHMAN, S., AND RONEN, R. The Intel Pentium M Processor: Microarchitecture and Performance. In *Intel Technology Journal* (2003).

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

[71] GOLDBERG, R. Survey of virtual machine research. In *Survey of Virtual Machine Research, IEEE Computer* (June 1974), pp. 34–45.

[72] GOOGLE CORPORATION. Google compute. http://toolbar.google.com/dc/.

[73] GRIMSHAW, A. S., STRAYER, W. T., AND NARAYAN, P. Dynamic, object-oriented parallel processing. *IEEE Parallel Distrib. Technol. 1*, 2 (1993), 33–47.

[74] 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).

[75] GUPTA, A., LIN, B., AND DINDA, P. A system for studying user comfort with resource borrowing. Tech. Rep. NWU-CS-04-28, Department of Computer Science, Northwestern University, 2003.

[76] 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).

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

[78] GURUN, S., AND KRINTZ, C. Autodvs: an automatic, general-purpose, dynamic clock scheduling system for hand-held devices. In *Proceedings of the 5th ACM international conference on Embedded software (EMSOFT)* (2005).

[79] HARCHOL-BALTER, M., AND DOWNEY, A. B. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems 15*, 3 (1997), 253–285.

[80] HELLERSTEIN, J. L. Achieving service rate objectives with decay usage scheduling. *IEEE Transactions on Software Engineering 19*, 8 (1993), 813–825.

[81] HOLMAN, P., AND ANDERSON, J. H. Guaranteeing pfair supertasks by reweighting. In *Proceedings of the 22nd IEEE Real-time Systems Symposium* (2001).

[82] 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* (2001).

[83] HOUSE, S., AND NIEHAUS, D. Kurt-linux support for synchronous fine-grain distributed computations. In *Proceedings of the Sixth IEEE Real Time Technology and Applications Symposium (RTAS)* (2000).

[84] HUA CHU, Y., RAO, S., SHESHAN, S., AND ZHANG, H. Enabling conferencing applications on the internet using an overlay multicast architecture. In *Proceedings of ACM SIGCOMM* (2001).

[85] HUMPHREYS, J., AND TURNER, V. On-demand enterprises and utility computing: A current market assessment and outlook. Tech. Rep. 31513, IDC, July 2004.

[86] HYOUDOU, K., KOZAKAI, Y., AND NAKAYAMA, Y. An implementation of a concurrent gang scheduler for pc cluster systems. In *Proceedings of Parallel and Distributed Computing and Networks* (2004).

[87] ID SOFTWARE. Quakeii. http://www.idsoftware.com/games/quake/quake2/.

[88] INGRAM, D., AND CHILDS, S. The linux-srt integrated multimedia operating system: bringing qos to the desktop. In *Proceedings of the IEEE Real-time Technologies and Applications Symposium (RTAS)* (2001).

[89] INTEL CORPORATION. Intel Pentium M Datasheet. `http://developer.intel.com/design/mobile/pentium-m/documentation.htm`.

[90] INTEL CORPORATION. Intel Pentium M Processor Thermal Management. `http://www.intel.com/support/processors/mobile/pm/sb/CS-007971.htm`.

[91] JACKSON, D. B., SNELL, Q., AND CLEMENT, M. J. Core algorithms of the maui scheduler. In *Proceedings of the 7th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)* (2001).

[92] JAIDER, M. Notebook Hardware Control Personal Edition. `http://www.pbus-167.com/chc.htm/`.

[93] JANNOTTI, J., GIFFORD, D., JOHNSON, K., KAASHOEK, M., AND JR., J. O. Overcast: Reliable multicasting with an overlay network. In *Proceedings of OSDI* (2000).

[94] JEROME, B. R., AND JULIE, S. C. A contribution of cognitive decision models to clinical assessment: Decomposing performance on the bechara gambling task. *Psychological assessment 14* (2002).

[95] JETTE, M. Performance characteristics of gang scheduling in multiprogrammed environments. In *Proceedings of the ACM/IEEE conference on Supercomputing* (1997).

[96] 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).

[97] JOSEPH, H. Keyboard hooks. *The Code Project* (July 2001). `http://www.codeproject.com/dll/keyboardhook.asp`.

[98] KAHNEMAN, D., AND TVERSKY, A. Prospect theory: An analysis of decision under risk. *Econometrica 47*, 2 (1979), 263–91. http://ideas.repec.org/a/ecm/emetrp/v47y1979i2p263-91.html.

[99] KEAHEY, K., DOERING, K., AND FOSTER, I. From sandbox to playground: Dynamic virtual environments in the grid. In *Proceedings of the 5th International Workshop on Grid Computing* (2004).

[100] KICHKAYLO, T., AND KARAMCHETI, V. Optimal resource-aware deployment planning for component-based distributed applications. In *Proceedings of the Thirteenth IEEE International Symposium on High-Performance Distributed Computing (HPDC)* (2004).

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

[102] KOHL, J. A., AND GEIST, G. A. The pvm 3.4 tracing facility and xpvm 1.1. In *Proceedings of the 29th Hawaii International Conference on System Sciences (HICSS)* (1996).

[103] 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)* (1997).

[104] 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).

[105] 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)* (2005).

[106] LARSON, S. M., SNOW, C. D., SHIRTS, M., AND PANDE, V. S. Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. In *Computational Genomics*, R. Grant, Ed. Horizon Press, 2002.

[107] LIN, B., BROWN, A. B., AND HELLERSTEIN, J. L. Towards an understanding of decision complexity in it configuration. In *Proceedings of the 1st ACM Symposium on Computer-Human Interaction for Management of Information Technology (CHIMIT)* (2006).

[108] LIN, B., BROWN, A. B., AND HELLERSTEIN, J. L. Towards an understanding of decision complexity in it configuration. In *Proceedings of the 3rd IEEE International Conference on Autonomic Computing* (2006).

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

[110] LIN, B., AND DINDA, P. Putting the User in Direct Control of CPU Scheduling. Tech. Rep. NWU-EECS-06-07, Department of Electrical Engineering and Computer Science, Northwestern University, August 2006.

[111] LIN, B., DINDA, P., AND LU, D. User-driven scheduling of interactive virtual machines. In *Proceedings of the 5th International Workshop on Grid Computing* (2004).

[112] LIN, B., AND DINDA, P. A. Towards scheduling virtual machines based on direct user input. In *Proceedings of the 1st International Workshop on Virtualization Technology in Distributed Computing (VTDC)* (2006).

[113] LIN, B., MALLIK, A., A.DINDA, P., MEMIK, G., AND DICK, R. Process and user driven dynamic voltage and frequency scaling. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)* (2007).

[114] LIN, B., SUNDARARAJ, A. I., AND DINDA, P. A. Time-sharing parallel applications with performance isolation and control. In *Proceedings of the 4th IEEE International Conference on Autonomic Computing (ICAC)* (2007).

[115] LINUX. Linux man pages : nice.

[116] LINUX VSERVER PROJECT. http://www.linux-vserver.org.

[117] LITZKOW, M. J., LIVNY, M., AND MUTKA, M. W. Condor — a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems (ICDCS)* (1988).

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

[119] LIU, D., AND SVENSSON, C. Trading Speed for Low Power by Choice of Supply and Threshold Voltages. In *IEEE J. Solid-State Circuits* (1993), vol. 28, pp. 10–17.

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

[121] LIU, X., ZHU, X., SINGHAL, S., AND ARLITT, M. Adaptive entitlement control of resource containers on shared servers. In *Proceedings of the IFIP/IEEE International Symposium on integrated Network Management* (2005).

[122] LOPEZ, J., AND O'HALLARON, D. Support for interactive heavyweight services. In *Proceedings of the Thirteenth IEEE International Symposium on High-Performance Distributed Computing (HPDC)* (2001).

[123] LORCH, J., AND SMITH, A. Using User Interface Event Information in Dynamic Voltage Scaling Algorithms. In *Technical Report UCB/CSD-02-1190, Computer Science Division, EECS, University of California at Berkeley* (August 2002).

[124] 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)* (2005).

[125] LOVE, R. *Linux Kernel Development*. SAMS, Developer Library Series, Sept 2003.

[126] LOWEKAMP, B. B., AND BEGUELIN, A. Eco: Efficient collective operations for communication on heterogeneous networks, 1996.

[127] 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).

[128] LU, C., STANKOVIC, J. A., ABDELZAHER, T. F., TAO, G., SON, S. H., AND MARLEY, M. Performance specifications and metrics for adaptive real-time systems. In *Proceedings of 21st IEEE Real-Time Systems Symposium* (2000).

[129] LU, C., STANKOVIC, J. A., TAO, G., AND SON, S. H. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Special issue of Real-Time Systems Journal on Control-Theoretic Approaches to Real-Time Computing 23*, 12 (September 2002), 85–126.

[130] LU, C., WANG, X., AND KOUTSOUKOS, X. Feedback utilization control in distributed real-time systems with end-to-end tasks. *IEEE Transactions on Parallel and Distributed Systems 16*, 6 (2005), 550–561.

[131] MACLEAN, A., CARTER, K., LOVSTRAND, L., AND MORAN, T. User-tailorable systems: pressing the issues with buttons. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (1990).

[132] MALLIK, A., LIN, B., A.DINDA, P., MEMIK, G., AND DICK, R. User-driven frequency scaling. In *IEEE Computer Society Computer Architecture Letters* (2006).

[133] MASSIE, M. L., CHUN, B. N., AND CULLER, D. E. The ganglia distributed monitoring system: Design, implementation and experience. *Parallel Computing 30*, 1 (January 2004).

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

[135] MICROSOFT CORPORATION. Performance Logs and Alerts overview. `http://www.microsoft.com/windows2000/en/advanced/help/`.

[136] MOIR, M., AND RAMAMURTHY, S. Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In *IEEE Real-Time Systems Symposium* (1999).

[137] MORUZZI, C., AND ROSE, G. Watson share scheduler. *Proc. Large Installation Systems Administrator Conf.* (1991), 129–133.

[138] MUTKA, M. W., AND LIVNY, M. The available capacity of a privately owned workstation environment. *Performance Evaluation 12*, 4 (July 1991), 269–284.

[139] 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)* (2004).

[140] NEWCOMER, J. M. Time is the simplest thing. *The Code Project* (June 2000). http://www.codeproject.com/system/simpletime.asp.

[141] 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* (1997).

[142] 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 sixteen ACM Symposium on Operating Systems Principles* (1997).

[143] 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).

[144] OFFICE OF GOVERNMENT COMMERCE, U. *Best Practice for Service Support*. IT Infrastructure Library Series. Stationery Office, June 2000.

[145] O'NEILL, M. Quantifying the accuracy of sleep. *The Code Project* (Mar 2003). http://www.codeproject.com/system/SleepStudy.asp.

[146] OUSTERHOUT, J. K. Scheduling techniques for concurrent systems. In *Proceedings of the Conference on Distributed Computing Systems (ICDCS)* (1982).

[147] PLOUS, S. *The Psychology of Judgment and Decision Making*, 1st ed. McGraw-Hill, January 1 1993.

[148] POLZE, A., FOHLER, G., AND WERNER, M. Predictable network computing. In *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS)* (1997).

[149] 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* (1997).

[150] RANGANATHAN, P., GEELHOED, E., MANAHAN, M., AND NICHOLAS, K. Energy-aware user interfaces and energy-adaptive displays. *Computer 39*, 3 (2006), 31–38.

[151] REYNOLDS, C. J. The sensing and measurement of frustration with computers. Master's thesis, Massachusetts Institute of Technology Media Laboratory, 2001. http://www.media.mit.edu/∼carsonr/pdf/sm_thesis.pdf.

[152] RIPEANU, M., BOWMAN, M., CHASE, J., FOSTER, I., AND MILENKOVIC, M. Globus and planetlab resource management solutions compared. In *Proceedings of the 13th IEEE Symposium on High Performance Distributed Computing (HPDC)* (2004).

[153] RIPEANU, M., FOSTER, I., AND IAMNITCHI, A. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal 6*, 1 (2002).

[154] ROHOU, E., AND SMITH, M. Dynamically Managing Processor Temperature and Power. In *2nd Workshop on Feedback Directed Optimization* (1999).

[155] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)* (2001).

[156] RYU, K. D., AND HOLLINGSWORTH, J. K. Fine-grain cycle stealing for networks of workstations. In *Proceedings of ACM/IEEE Supercomputing (SC)* (1998).

[157] RYU, K. D., HOLLINGSWORTH, J. K., AND KELEHER, P. J. Efficient network and I/O throttling for fine-grain cycle stealing. In *Proceedings of Supercomputing* (2001).

[158] SAPUNTZAKIS, C., BRUMLEY, D., CHANDRA, R., ZELDOVICH, N., CHOW, J., LAM, M. S., AND ROSENBLUM, M. Virtual appliances for deploying and maintaining software. In *Proceedings of the 17th Large Installation Systems Administration Conference (LISA)* (2003).

[159] SAPUNTZAKIS, C. P., CHANDRA, R., PFAFF, B., CHOW, J., LAM, M. S., AND ROSENBLUM, M. Optimizing the migration of virtual computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)* (2002).

[160] SATYANARAYANAN, M., KOZUCH, M., HELFRICH, C., AND O'HALLARON, D. Towards seamless mobility on pervasive hardware. *Pervasive and Mobile Computing 1*, 2 (2005), 157–189.

[161] 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).

[162] SCOTT A. BRANDT, SCOTT BANACHOWSKI, C. L., AND BISSON, T. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proceedings of the 24th IEEE Real-Time Systems Symposium* (2003).

[163] SHARMAN NETWORKS. "the kazaa media desktop". http://www.kazaa.com.

[164] 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.

[165] SIEGELL, B. S., AND STEENKISTE, P. Automatic generation of parallel programs with dynamic load balancing. In *Proceedings of the Thirteenth IEEE International Symposium on High-Performance Distributed Computing (HPDC)* (1994).

[166] SIMON, H. A. *Administrative Behavior*, 4th ed. Free Press, March 1997.

[167] SKADRON, K., STAN, M. R., SANKARANARAYANAN, K., HUANG, W., VELUSAMY, S., AND TARJAN, D. Temperature-aware Microarchitecture: Modeling and Implementation. *ACM Transactions on Architecture and Code Optimization (TACO) 1*, 1 (2004), 94–125.

[168] SNELL, Q., CLEMENT, M., JACKSON, D., AND GREGORY, C. The performance impact of advance reservation meta-scheduling. In *Proceedings of the 6th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)* (2000).

[169] SRINIVASAN, J., ADVE, S. V., BOSE, P., AND RIVERS, J. A. The Case for Lifetime Reliability-Aware Microprocessors. In *The International Symposium on Computer Architecture (ISCA)* (2004).

[170] STANKOVIC, J. A., HE, T., ABDELZAHER, T. F., MARLEY, M., TAO, G., SON, S. H., AND LU, C. Feedback control scheduling in distributed real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium* (2001).

[171] STEERE, D. C., GOEL, A., GRUENBERG, J., MCNAMEE, D., PU, C., AND WALPOLE, J. A feedback-driven proportion allocator for real-rate scheduling. In *Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation* (1999).

[172] STERNBERG, R. J., AND BEN-ZEEV, T. *Complex Cognition: The Psychology of Human Thought*. Oxford University Press, Feb 2001.

[173] STEVENS, R. W. Tcp slow start, congestion avoidance, fast retransmit and fast recovery algorithms. In *Internet RFC 2001* (1997).

[174] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F., AND BALAKRISHNAN, H. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM* (2001).

[175] STRAZDINS, P., AND UHLMANN, J. A comparison of local and gang scheduling on a beowulf cluster. In *Proceedings of the 2004 IEEE International Conference of Cluster Computing* (2004), pp. 55–62.

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

[177] 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)* (2005).

[178] SUBHLOK, J., GROSS, T., AND SUZUOKA, T. Impact of job mix on optimizations for space sharing schedulers. In *Proceedings of Supercomputing* (1996).

[179] SULLIVAN, W. T., WERTHIMER, D., BOWYER, S., COBB, J., GEDYE, D., AND ANDERSON, D. A new major seti project based on project serendip data and 100,000 personal computers. In *Proceedings of the Fifth International Conference on Bioastronomy* (1997), no. 161 in IAU Colloquim.

[180] 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)* (2005).

[181] SUNDARARAJ, A. I. *Automatic, Run-time and Dynamic Adaptation of Distributed Applications Executing in Virtual Environments*. PhD thesis, Northwestern University, Electrical Engineering and Computer Science Department, December 2006. NWU-EECS-06-18.

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

[183] SUNDARARAJ, A. I., GUPTA, A., AND DINDA, P. A. Dynamic topology adaptation of virtual networks of virtual machines. In *Proceedings of the Seventh Workshop on Langauges, Compilers and Run-time Support for Scalable Systems (LCR)* (2004).

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

[185] TRANSMETA CORPORATION. The technology behind the crusoe processor, (2000).

[186] VALIANT, L. G. A bridging model for parallel computation. *Communications of the ACM 33*, 8 (August 1990).

[187] VMWARE. Vmware esx server-cpu resource management. http://www.vmware.com/support/esx/doc/res_cpu_esx.html.

[188] VON AHN, L. Games with a purpose. *IEEE Computer Magazine 39* (June 2006), 92–94.

[189] VON AHN, L., AND DABBISH, L. Labeling images with a computer game. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI)* (2004).

[190] VON AHN, L., GINOSAR, S., KEDIA, M., LIU, R., AND BLUM, M. Improving accessibility of the web with a computer game. In *Proceedings of the SIGCHI conference on Human Factors in computing systems (CHI)* (2006).

[191] VON AHN, L., KEDIA, M., AND BLUM, M. Verbosity: a game for collecting common-sense facts. In *Proceedings of the SIGCHI conference on Human Factors in computing systems (CHI)* (2006).

[192] VON AHN, L., LIU, R., AND BLUM, M. Peekaboom: a game for locating objects in images. In *Proceedings of the SIGCHI conference on Human Factors in computing systems (CHI)* (2006).

[193] WAIZMAN, A., AND CHUNG, C. Resonant free Power Network Design using Extended Adaptive Voltage Positioning (EAVP) Methodology. *IEEE Transactions on Advanced Packaging 24*, 3 (August 2001), 236–244.

[194] 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), Usenix.

[195] WANG, Z., AND CROWCROFT, J. Eliminating periodic packet losses in the 4.3-tahoe bsd tcp congestion control algorithm. *SIGCOMM Computer Communications Review 22*, 2 (1992), 9–16.

[196] WEI, J. Foxton technology pushes processor frequency, application performance. `http://http://www.intel.com/technology/magazine/computing/foxton-technology-0905.htm`.

[197] 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)* (2004).

[198] WHITE, S., ALUND, A., AND SUNDERAM, V. S. Performance of the NAS parallel benchmarks on PVM-Based networks. *Journal of Parallel and Distributed Computing 26*, 1 (1995), 61–71.

[199] WICKENS, C. D., AND HOLLANDS, J. G. *Engineering Psychology and Human Performance*, 3rd ed. Prentice Hall, September 21 1999.

[200] WILEY, R. J. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley- Interscience, February 2001.

[201] WILLEBEEK-LEMAIR, M. H., AND REEVES, A. P. Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Systems 4*, 9 (1993), 979–993.

[202] WOLFRAM PODIEN. CPUCool. `http://www.cpufsb.de/CPUCOOL.HTM`.

[203] WU, Q., REDDI, V., WU, Y., LEE, J., CONNORS, D., BROOKS, D., MARTONOSI, M., AND CLARK, D. W. Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance. In *38th International Symposium on Microarchitecture (MICRO)* (2005).

[204] XU, R., MOSS, D., AND MELHEM, R. Minimizing expected energy in real-time embedded systems. In *Proceedings of the 5th ACM international conference on Embedded software(EMSOFT)* (2005).

[205] XU, W., ZHU, X., SINGHAL, S., AND WANG, Z. Predictive control for dynamic resource allocation in enterprise data centers. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium* (2006).

[206] YAN, L., ZHONG, L., AND JHA, N. K. User-perceived Latency based Dynamic Voltage Scaling for Interactive Applications. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)* (2005).

[207] YODAIKEN, V., AND BARABANOV, M. A real-time linux, 1997.

[208] 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 (1997).

Appendix A

Towards an Understanding of Decision Complexity in IT Configuration

This dissertation argues for using direct human input to solve optimization problems. We believe that human-driven optimization has very broad application. In this appendix, we will describe our work on modeling user decision making in IT configuration in order to reduce configuration complexity. Our ultimate goal is to apply human-driven technique to eventually solving this problem.

A.1 Introduction

As briefly introduced in Chapter 1, complexity is the most significant challenge confronting IT systems today. Complexity hinders penetration of new technology, drastically increases the cost of IT system operation and administration (which today dwarfs the cost of the IT systems themselves [85]), and makes the systems that we build hard to comprehend, diagnose, and repair.

In previous work [20], we argued that complexity can be tackled quantitatively, with a framework that allows system designers to assess the sources of complexity and directly measure the effectiveness of potential complexity improvements. We also introduced an

initial approach to quantifying the complexity of IT configuration and management tasks, based on a model of the sources of configuration complexity and a set of metrics derived from that model [22]. This approach, which we summarize in Section A.2, focuses on complexity as perceived by expert users—for example, experienced system administrators who have long-term experience with the systems they are managing—and is based on a structural analysis of the configuration or administration task itself, assuming all decisions are known and made correctly.

While this expert-focused approach is proving its value in practical application within IBM, the fact remains that its expert-only perspective limits the complexity insights that it can provide. In particular, a key complexity challenge lies in improving the experience of the non-expert system administrator—the person providing IT support in a small-business environment; the administrator who has expertise in one platform but is working for the first time with a new one; the experienced operator trying to deploy a new piece of technology for the first time; the outsourcer trying to apply ITIL best practices [144] but facing decision points within the prescribed processes. In these cases, a different dimension of complexity becomes paramount: the complexity of figuring out for the first time what steps to follow and what decisions to make while performing a complex configuration process. We call this complexity **decision complexity**.

However, quantifying decision complexity is not straightforward. Unlike the expert-only case, we cannot simply analyze a "gold standard" procedure for complexity. Instead, we must understand how configuration decisions are made, what factors influence those decisions, and how those factors contribute to both perceived difficulty as well as objectively-measured quantities like time and error rate. And, since our goal is ultimately to be able to easily quantify points of high complexity, we must build and use this understanding pragmatically, without having to resort to complex cognitive or perceptual modeling.

We quickly realized that the only way to make progress towards these goals was to

formulate an initial model of decision complexity and move rapidly to collect real data to test that model and provide insight into factors that affect decision complexity. We designed and conducted an extensive user study to produce data relating hypothesized decision complexity factors to measured user perception ratings, task time, and error rate. Because of the difficulties of conducting a controlled study on actual IT tasks with a large population of practicing system administrators, we collected data in an alternative, more accessible domain—route planning—with an experiment carefully designed to connect features of decision-making in the route planning domain with analogous features in the IT configuration domain.

Analysis of our study data reveals several interesting results. We found that task time was primarily affected by the number and type of constraints controlling the key decisions, as well as secondarily by the presence of short-term goal-related guidance. User-perceived difficulty was affected primarily by the short-term goal-related guidance factor, with a secondary effect from the presence of status feedback and only minor effects from constraints. Error rate was affected by short-term goal-related guidance and position guidance. The contrasts in these results suggest the hypothesis that decision complexity has multiple influences, and that system designers can optimize differently to minimize time, error rate, and perceived difficulty, respectively.

We have created a model from our study results that relates decision complexity in the route-planning domain to some of the factors discussed above. Because of the construction of our experiment, we believe that this model should apply to decision complexity in the IT configuration complexity domain as well, and that it can be used to extract some initial guidance for system designers seeking to reduce complexity. However, there is still a clear need for further extension and validation of the model in actual IT contexts. We describe some thoughts and future work on how we intend to accomplish that validation. These are the next steps to continue the exploration of this crucial aspect of complexity analysis and

can take us closer to a quantitative framework that can help shape a future with easier-to-manage, less complex IT infrastructures.

The remainder of this appendix is organized as follows. Section A.2 briefly summarizes our previous work in complexity modeling for experts. Chapter 9 discusses the related work. Section A.3 describes our initial hypothesized model for decision complexity that we used to construct the user study, which is in turn described in Sections A.4 and A.5. The results and analysis of our study data are presented in Section A.6. Finally, we describe our next steps in Section A.7, and conclude in Section A.8.

A.2 Complexity model for experts

To provide context for our work on decision complexity, we first summarize our previous work on complexity modeling for experts, as described in [20, 22]. Our previous approach focused on creating a standard framework for modeling and quantifying configuration complexity from the point of view of an expert administrator. The intended use of this model and related metrics was twofold: first, to provide an easy way for system designers to obtain quantitative insight into the sources of complexity in their designs (without the need for costly user studies), and second to serve as the foundation for a competitive complexity benchmark.

The approach we followed is based on process analysis. The input to our expert-level complexity model is a codified record of the actual configuration procedure used to accomplish some administrative task on the IT system under test, captured from actual execution or documentation. This record contains information on the configuration contexts present in the procedure, the detailed sequences of actions performed within those contexts, and the data items and data flow between actions, as managed by the system administrator. The model uses features of that record to extract complexity metrics in three dimensions: (1)

execution complexity, reflecting the complexity of actually performing the needed action sequences; (2) parameter complexity, reflecting the complexity of supplying the correct values of all needed information to the configuration actions; and (3) memory complexity, reflecting the burden of parameter management and data item tracking carried by the system administrator. Metrics are calculated across the entire procedure to allow cross-procedure comparison, and are also computed at a per-action level, allowing identification of complexity "hot spots" and targeting of future development focus.

The metrics computed by our expert-level model are all objective scores, based solely on the structure of the procedure record. Likewise, the procedure record reflects the optimal configuration path as identified by an experienced expert, with no mis-steps or decision branches. Thus the results of the analysis are objective and comparable across systems and environments, and they reflect inherent structural complexities present in the configuration procedures, but they do not capture any of the decision complexity in identifying the right procedure or choosing the correct decision branches within that procedure. Hence the focus of this work is on extending the complexity model to include an initial understanding of the impact of decision complexity.

A.3 Model and hypothesis

Table A.1: High-level model of decision making

Factors	**Definition** **Definition**	**Configuration analogy** **(examples)**
Constraints	Constraining conditions that restrict users to avoid or make certain decisions	compatibility between software products, capabilities of a machine
Guidance	Guiding information on decisions	documentation, previous configuration experience
Consequence	Results from the decision	functionality, performance

To understand decision complexity, we initially approached it with an attempt to build

Table A.2: Sub-factors within guidance

Sub-factors of Guidance	**Definition**	**Configuration analogy (examples)**
Global information	Providing an overview of the situation across a set of short-term goals	A "Redbook" describing the options for combining multiple software products into a solution
Short-term goal-oriented information	Information needed for a particular short-term goal, or goal of current interest is co-located and directly answers the major decision.	A configuration wizard, such as a database tuning wizard
Confounding information	Extraneous or misleading info not related to goals are not presented.	A manual providing application configuration instructions for a different OS platform than the one being used
Position information	Information for identifying relative order of current decision across a set of decisions is provided.	Feedback on results of last configuration action; a task-level progress bar

Table A.3: Route planning domain based on the model

Factors	**Route planning domain**
Constraints	Traffic
Guidance (Global info)	Map, Expert path
Guidance (Goal-oriented info)	GPS
Guidance (Position info)	Current position indicator
Consequence	Reach the destination or not

a low-level model that could capture and compute every aspect of a human-driven configuration procedure. We then realized that such a model requires a detailed understanding of human cognitive processes. This approach is too complex for practical use, so we decided to re-approach the problem from a high level, to understand what factors influence decision making, and how those factors contribute to decision complexity.

To address these questions, we formulated an abstract high-level model. As shown in table A.1, the three major factors we consider in our model are *constraints*, *guidance* and *consequences*. We choose these factors based on results from the HCI literature [199] as well as our own assessment of real IT configuration procedures, where the user is given

various types of guidance and needs to make different decisions while facing various constraints. The decisions made by the user then generate different consequences in term of a specific user goal.

For example, one IT procedure we studied involved the installation of a secured web portal software stack, including a portal server, directory server, application middleware server, and a security infrastructure. The procedure contained several decisions concerning software version selection, feature selection (e.g., should the portal support SSL-based secure access), configuration alternatives (e.g., authentication mechanisms), and sequencing.

In this procedure, guidance was provided in the form of product manuals, a step-by-step "how-to"-style guide [64], and on-screen prompts. The procedure involved several constraints, such as incompatibilities between different versions of the constituent software products, different feature sets across different software versions, and resource consumption requirements. Each of the several decision points in the process (for example, choosing which security protocol to use) resulted in consequences relative to the original goal—either performance or functionality implications in the resulting portal installation, or the ability to achieve the goal state at all. An example of the latter style of consequences is a case where certain product versions could not be co-located on the same machine. If the decision was made to co-locate the incompatible versions, the procedure resulted in a non-working system.

Of the guidance, constraints, and consequences factors, guidance is of particular interest because it is the major source of information that user will consult with in making a decision. Analogous to work in the HCI area [199], we further define the formulation of a guidance system in table A.2. The definition is based on what a good guidance system should provide.

In both tables A.1 and A.2, we give examples in the IT configuration domain to show

the ground on which we build the model. For example, in our portal case study, the "how-to" guide provided global information guidance about the structure of the entire task; specific dialog boxes in the install wizards for the portal's components provided short-term goal-oriented guidance for configuring each separate component. There was little explicit position information except what could be gleaned from matching screenshots in the how-to guide with the on-screen display. Confounding information was present in the standalone documentation for each product component of the overall portal stack.

As stated above, our goal in constructing the 3-facet model of guidance, constraints, and consequences is to obtain a high-level understanding of the forces involved in creating decision complexity for IT operational procedures. Thus with the key factors identified, the next step is to validate their impact on decision complexity, and to begin to quantify their relative effects. If we can do this, we can provide a high-level framework for assessing decisions in IT processes and for providing guidance to system designers seeking to reduce decision complexity.

A.4 Approach

To validate our model, ideally we should conduct a user study where users perform a real IT configuration procedure. However we face some obvious difficulties here. First it is challenging to obtain a large set of users with a consistent level of IT experience, especially those with system administration training. Second, it is difficult to finely tune a real IT configuration procedure to validate each component of our model in a controlled, reproducible environment that allows data collection from large numbers of users.

Facing these challenges, we searched for an alternative domain that would allow us to carefully control its elements, and that offered similar characteristics to the IT configuration domain, so that a model built on it could be mapped back to IT configuration domain.

We ended up settling on the domain of *route planning*.

In route planning, users navigate a set of interconnected paths to arrive at a prespecified destination within certain limits of time and distance traveled. As they navigate, they make multiple decisions based on information available to them at the time. If they are unfamiliar with the map, the users are effectively non-experts, and thus face decision complexity at each branch point. As shown in table A.3, the route planning domain contains examples for all factors that we define in our model. In addition, it is familiar to ordinary users with or without an IT administration background, so user training is unnecessary. Using this domain, we can conduct a user study to learn how people make decisions in the context of performing a prescribed procedure, which in our case is navigating a car from one point to another, and extrapolate the results back to the IT configuration domain. While the mapping is clearly not perfect, we believe that it is sufficient to provide a high-level understanding of how our model factors affect decision complexity, giving us an initial step towards the goal.

A.5 User study design

We designed an on-line user study that could be taken by participants over the web. The study included multiple experiments with different test cases. Each test case varied the levels of our key factors (guidance, constraints, consequences) and measured the user's time, correctness, and reported difficulty ranking.

A.5.1 Experiment and test cases

We designed 3 experiments for our user study. Each user was assigned an experiment randomly after he logged in. Each experiment consists of 6 sequential test cases and 1 warm-up test case in the beginning. We have 10 possible test cases (not including the warm-up)

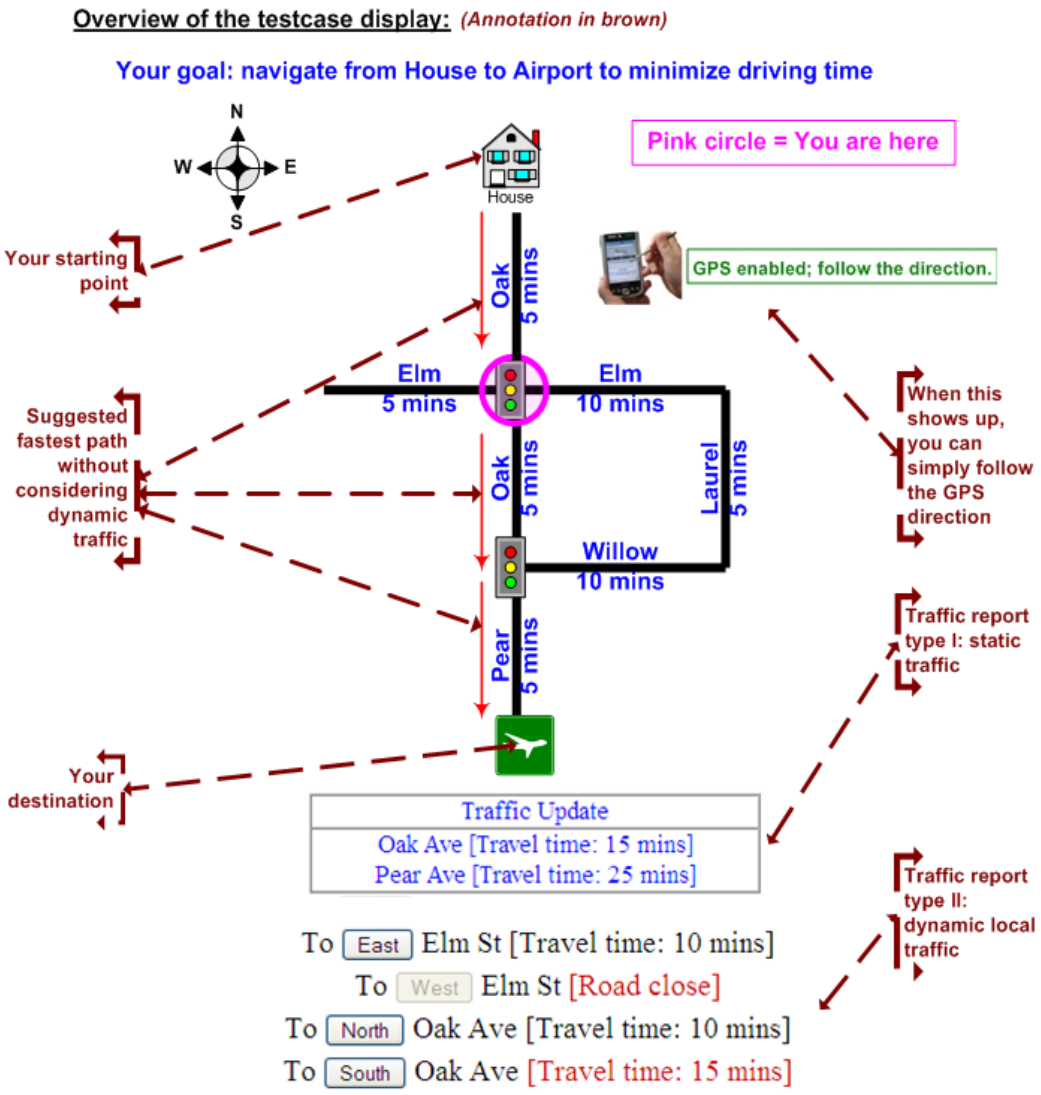


Figure A.1: The screen-shot of a running testcase.

Table A.4: Summary of test cases; a × means the parameter is not presented while a check means the opposite.

No	Pos indicator	Traffic type	Update type	Path diff	Expert path	GPS
1	√	×	×		×	×
2	√	static	travel time		×	×
3	√	dynamic	road close		×	×
4	√	dynamic	travel time		×	×
5	√	×	×		√	×
6	√	dynamic	travel time		√	×
7	√	dynamic	travel time		×	√
8	√	dynamic	road close	bigger	×	×
9	√	dynamic	travel time	bigger	×	×
10	×	×	×		×	×

in total, which we carefully designed and believe will help us find out the answers to the questions that we discussed in previous section A.3.

Table A.4 summarizes the test cases we used in the study. We also carefully selected the set of test cases to be included in each experiment so that we can maximize our data set. The major parameters we built into our test cases are:

- **Traffic:** we have two types of traffic update, representing constraints in our complexity model. **Static update** presents the global traffic updates to the user in the beginning of the test case, while the **dynamic update** only discloses the local traffic to the user when he arrives at the traffic-related intersection or road. This is the equivalent of listening to a traffic report versus running into a traffic jam, and in the IT domain is analogous to prespecified versus unexpected constraints (such as version compatibility). For dynamic update, we further design two types of update: **road close** and **travel time update**. The former is analogous to the constraints in the IT configuration domain that eliminate the viability of one installation path, and cause user to undo and look for a new path, while the latter is an analogy to those

constraints that only change the resulting performance of an otherwise-viable configuration.

- **Expert path:** an expert path is the suggested route for user without considering the traffic. It is analogous to the previous experience a user or expert brings to configure the same system, or the information presented in a "how-to" or step-by-step walkthrough guide.

- **GPS:** similar to the advanced Global Position System people use when driving in the real world, it is analogous to an omniscient expert that directs people during a configuration procedure, which we believe requires the least mental effort from the user in making decisions.

- **Position indicator:** a pink circle on the map indicates current location of the user. It is analogous to the position information defined in Table A.2, i.e. the feedback information in IT context, which provides feedback on the current state of the system and the effect of the previous action.

- **Path differences:** different length of routes from the starting point to the destination reflects different consequences resulted from user's decisions. To study the impact of consequences on the decision complexity, we vary the path difference for different maps so that some maps have small path differences among all possible routes, while some maps have big path differences.

A.5.2 Perspective of the user

In each test case, the user is presented with a map consisting of a series of road segments and intersections. Each road segment is marked with a travel time. The pink circle indicates current position of the user in the map. The goal is to navigate a path from the stating

point (home) to the airport in the minimum amount of driving time, using the navigation buttons at the bottom of the interface. Each test case uses a slightly different map to avoid learning effects; however, all maps are topographically equivalent with the same set of decision points. The optimal path differs across test cases, but note that only one path is optimal in each map. This scenario is roughly equivalent to the IT configuration problem of being given a new system to install/configure and a set of documentation laying out possible system- and resource-dependent sequences of configuration actions. Just as the user has to work out the optimal path through the map, the IT administrator has to make the configuration decisions at each branch point in the IT setup process, based on the state of the system and the visible paths ahead.

To maximize the quality of our data, we requested users not to multi-task or walk away from the system while a test case was in progress. In some test cases, users may have encountered traffic or other obstructions that changed the travel time for certain road segments or rendered them impassable. Users may also have received different levels of guidance that may have helped them to identify the right path. Figure A.1 shows an introductory page, with all possible components annotated. This is what the user saw after logging in and before starting the experiment. Note that not all components showed up in each test case.

In the beginning of the experiment, we ask the user about his or her background.

- What is your gender? (Male / Female)
- Do you have formal or informal training in mathematics, computer science and/or engineering? (Yes / No)
- How long have you been driving? (specify years)
- How often do you drive a car? (Every day / A few times a week / A few times a

month / Rarely / Never–do not drive)

- Do you use online map services like Mapquest, Yahoo Maps, Google Maps, etc when you need to drive to an unfamiliar destination? (Always / Frequently / Occasionally / Never)

- How would you rate your proficiency with map-reading and navigating based on maps? (Excellent / Very good / Good / Mediocre / Poor)

At the end of the set of test cases, we ask the user to rank the test cases according to difficulty on a scale of 1 (easiest) to 6 (most difficult). Note that as the user proceeds through the experiment, he has the opportunity to input a reminder at the end of each test case to help him remember which one is which when he gets to the end of the experiment.

A.5.3 Implementation

We implemented our on-line user study using a JAVA Servlet-based architecture with server-side collection of data, including timings. The web pages are dynamically generated based on the data submitted by the user. The experiment server records user navigation sections (i.e. decision points) as well as the real time he takes to complete each test case. The server also compares the user's path with the optimal path for each map.

We used XML-based experiment configuration files so that we can not only design various test cases and experiments using a standard data format, but also finely control each parameter of the study by simply modifying the corresponding XML file.

We used JPEG images to represent the steps in the experiment. In the beginning of the experiment and after each navigation action, a JPEG image was presented to the user. In our experiments, these were images of the route map with the appropriate information presented to the user (such as their current position, or the suggested expert-supplied path). The implementation consists of approximately 3100 lines of JAVA and 211 JPEG files.

One of our goals in implementing the user study is to design a general framework so that it can be easily exploited for similar experiments. The core of our JAVA Servlet is a general user-driven decision engine which can present information, react and record all according to external XML-based configurations. By supplying different sets of JPEG images, along with corresponding XML files, our experiment framework should be adaptable to explore many other aspects of IT administration and complexity. For example, the map images could be replaced by screen-shots of actual configuration dialogs (with corresponding XML files). We discuss this possibility later in Section A.7 as a possible next step in validating our results in a more directly-IT-relevant context.

A.5.4 Two-stage User Study

Our user study consisted of two stages. In the first stage, 37 users from IBM T.J. Watson Lab participated. In the second stage, we revised the order of test cases in each experiment based on the analysis of the user data from the first stage. Note that we did not change the content of the test cases. 23 users from IBM Almaden Lab, University of California, Berkeley, and Harvard University participated.

In both stages, we advertised for participants via emails. The duration of the study for each user was around 30 minutes. The 10 participants who did the best at the experiments were automatically entered into a random drawing; two won a $50 gift certificate each.

Table A.5: Summary of complexity factors

Model factors	Constraints Constraints	Guidance (global)	Guidance (goal)	Guidance (position)	Consequences
Test case factors	Test case number	Test case order	Found optimal or not		
Background factors	Gender	CS & math background	Driving frequency	Online-map usage	Proficiency with map & reading routing

A.6 Results and Analysis

A.6.1 Metrics

We use three metrics to evaluate the study.

The *AvgTimePerStep* is the average time that users spent in one step in one test case. Note that we have different number of steps in different test cases. The *UserRating* is the average rank specified by users. Recall that users were asked to rank test cases from 1 to 6 in term of difficulty, where 6 indicates the most complex/difficult test case and 1 indicates the easiest one. If they felt that two (or more) test cases have approximately the same level of difficulty, they may give them the same rank. The *ErrorRate* is the percent of users who failed to find the optimal path for a test case. Note that for each test case, we have only one optimal path.

A.6.2 Qualitative results

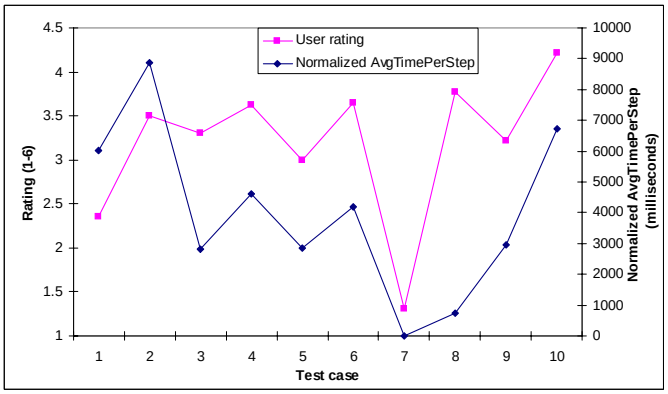


Figure A.2: User rating and time; Avg Std for time over all testcases: 4368 milliseconds

To reduce the variation across users, for each user we normalized his AvgTimePerStep based on test case 7 (see Table A.4), where we provided GPS turn-by-turn guidance. This test case involves no decision making at all on the user's part, and thus reflects each user's

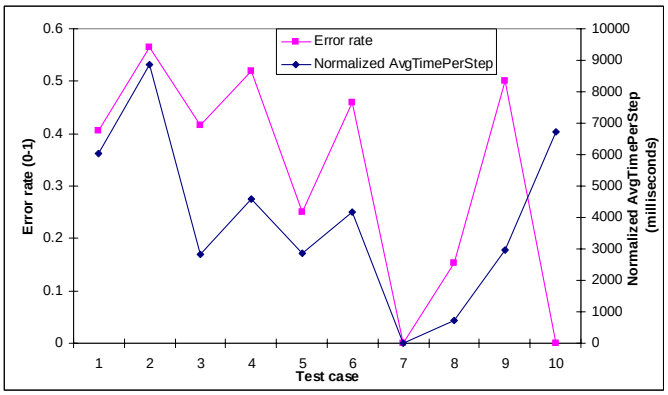


Figure A.3: Error rate and time; Avg Std for time over all testcases: 4368 milliseconds

baseline speed of navigating through the user interface of the study; in all cases each user spent the least amount of time in testcase 7,

Figure A.2 shows that most parts of the trends for UserRating and normalized Avg-TimePerStep are tracked, except for test case 8, which users felt was difficult but in which they only spent a small amount of time. In figure A.3, we see similar tracking between ErrorRate and normalized AvgTimePerStep, except that in test case 10, where all users who did that test case spent more time due to the lack of the position indicator. Interestingly all users were able to find the optimal path in this test case. One possible reason for this is that when there was no position indicator, users had to become more careful in each step and spent more time in tracking their movement and planning their routes. As a result, the ErrorRate was greatly reduced.

Overall, this result confirms that decision complexity has different impacts on:

- User-perceived difficulty
- Objective measures (time and error rate)

Figure A.2 and A.3 bring out some interesting discussion. However we can not draw quantitative conclusions from them because the variation of AvgTimePerStep for each test

case is very large across all users. The average standard deviation of AvgTimePerStep over all test cases is 4368 milliseconds, almost half of the maximum AvgTimePerStep.

In an attempt to gain more insight into the data, we further analyzed the data in two steps, with results discussed in the next section:

- Step I: general statistical analysis; treat each test case measured as an independent data point, with the goal to identify factors that explain the most variance.
- Step II: pair-wise per-user test case comparisons; get more insight into specific effects of factor values, with the goal to remove inter-user variance.

A.6.3 Quantitative results

Table A.5 lists all factors that we identified within the study. The first row lists all factors that we propose in our initial model. We call them *complexity model factors*. The second row includes those test case related factors. The third row shows all background related factors.

Time

Table A.6: Baseline analysis of variability for time

Factor	Sum Squares
Test case #	32.778
Driving years	17.637
Online-map usage	7.260
Residual	45.192

As step I, we conduct an ANOVA (Analysis of Variance) test on AvgTimePerStep using a linear-space regression model. To see how much variance that we can explain, we first include test case number, test case order, and all background related factors (e.g. gender, driving years) in the ANOVA. Since test case number subsumes all of the factors we

Table A.7: Analysis of complexity factors for time

Factor	Sum Squares
Constraints	16.764
Guidance (goal)	11.397
Consequence	1.939

explicitly altered during the experiment, we believe that the variance that can be explained by test case number should be a superset of what can be explained by our model factors. Table A.6 is the summary of the ANOVA. We only list those factors which have significant impact on *Sum of Squares*. As we can see, the maximum variability that can be explained by model factors (those we explicitly varied in the experiment) is 32.778. Interestingly, the length of driving years contributes 17.637 to the Sum of Squares, indicating experience is a significant factor. Other factors are not listed due to their tiny impact. The residual, we believe, comes from random per-user effects that can't be explained by either model factors or user background.

Based on this baseline analysis, we then do an ANOVA test on our model factors (i.e. constraints, levels of guidance, consequence) to identify those factors that explain the most variance. We know from our earlier analysis that at most 32.778 of the sum of squares variance can be explained by these factors. Table A.7 indicates that constraints and short-term goal related guidance have the most impact on time, followed by a small amount of affect from consequences. Other factors have very little impact and are not listed. Note that constraints and guidance together explain 96% of the total variance explainable by model factors.

From this Step I data, we can conclude that the user's decision time is primarily influenced by the presence of constraints, along with goal-directed guidance such as step-by-step instructions. The impact of visible consequences is also present, though at a lower level. The regression fit data confirms this analysis, showing increased predicted step time

when constraints are present, and decreased time when goal-directed guidance is provided or consequences are more visible.

Table A.8: Pair-wise test for time

	1st Study	**95% CT**	**2nd Study**	**95% CT**
Constraints	static traffic > dynamic (road close)	(0.78, 1.07)	static traffic > dynamic (road close)	(1, 1)
	static traffic > without traffic	(0.73, 1.01)	static > dynamic (travel time update)	(0.54, 1.13)
Guidance (goal)	without expert path > with expert path	(0.53, 0.89)		

Next, in step II, we aim to remove inter-user variance and get more insight into specific effects of factor values. Table A.8 summarizes our pair-test analysis, providing 95% confidence intervals. In these tests, we compared the results of a pair of test cases from a single user, to determine a per-user effect of factor differences between the test cases. We then averaged across users to test for a significant cross-population effect. Note that we only list those results which allow us to discount the null hypothesis, that two test cases have no difference, with $> 95\%$ confidence. This result confirms what we found in step I, i.e. constraints and guidance (goal) are two major factors influencing task time. We further discover that statically-presented constraints (like our static traffic) actually increase time compared to dynamic constraints, likely due to the user's need to assess the relevance of the global information at each step of the procedure.

Rating

Similar to our analysis for time, we first do an ANOVA test on UserRating using test case number, test case order, and all background related factors. From table A.9, we can see that the maximum variability that can be explained by the model factors is 51.671. The length of driving years again has some impact although the impact is small compared to that in the time case.

We then feed the model factors into the ANOVA test. Different from what we found in the time case, here short-term goal related guidance is now the top 1 influential factor, followed by position guidance. Constraints however only have small impact on the user's rating.

The results also show that a third factor, Log(order), impacts UserRating, although at a much lower level than Guidance. The Order factor refers to the sequence in which the user was shown the various test cases; the presence of the Log(order) term in the ANOVA implies that there is a bias to users' rating, with higher ratings given later in the sequence.

Table A.11 is the summary for step II - pair-wise test. Although it does not statistically show the impact of guidance (goal), it confirms the impact of position guidance and constraints providing 95% confidence intervals.

Table A.9: Baseline analysis of variability for rating

Factor	Sum Squares
Test case #	51.671
Driving years	7.125
Residual	67.087

Table A.10: Analysis of complexity factors for rating

Factor	Sum Squares
Guidance (goal)	42.272
Guidance (position)	6.278
Log(order)	2.071
Constraints	1.683

Error rate

The analysis of ErrorRate is different from time and rating because we only have one data point per test case, i.e. error rate averaged across all users who finished that test case. So it is hard do any further statistical analysis. However from figure A.3, we can still draw

Table A.11: Pair-wise test for rating

	1st Study	**95% CT**	**2nd Study**	**95% CT**
Guidance (pos)	without pos indicator > with pos indicator	(0.51, 1.05)	without pos indicator > with pos indicator	(0.51, 1.05)
Constraints	static traffic > dynamic traffic (road close)	(0.54, 1.13)		

two conclusions. First, all users were able to find the optimal path in test case 7, where we provided GPS turn-by-turn guidance. So we can conclude that providing short-term goal related guidance will reduce error rate. Second, the error rate in test case 10 is also zero, where the position guidance was not provided. So the conclusion is that error rate will be reduced when guidance (position) is not present, although perceived difficulty and time both increase, illustrating the tradeoffs between different forms of decision complexity.

A.6.4 Summary and advice to designers

The contrasts and complexity in the above results suggest the hypothesis that decision complexity has multiple influences on time, error rate, and user-perceived difficulty, and suggests some rough approaches for reducing complexity along these dimensions.

Depending on its goal, optimization for lower complexity will have a different focus. The examples below illustrate possible design approaches for reducing complexity.

- In the IT configuration domain, an installation procedure with easily-located clear info (e.g. wizard-based prompts) for the next step will reduce both task time and user-perceived complexity, though it is unclear how much it will affect error rate.

- A procedure with feedback on the current state of the system and the effect of the previous action (e.g. message windows following a button press) will reduce user-perceived complexity, but is unlikely to improve task time or error rate.

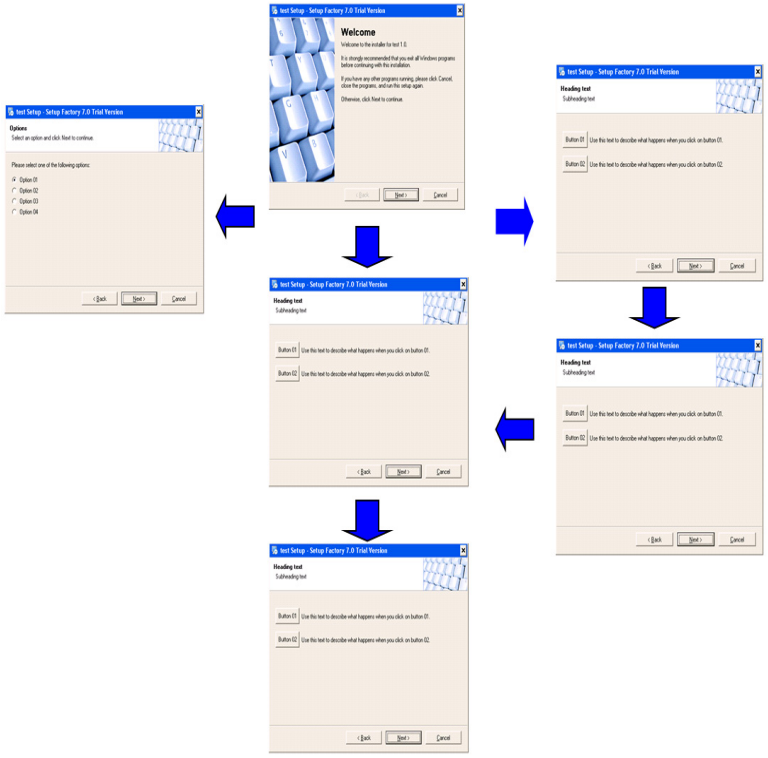


Figure A.4: Mapping

- A procedure that automatically adapts to different software and hardware versions to reduce compatibility constraints will reduce task time, and may also cause a small reduction in perceived complexity.

- Omitting positional feedback (i.e., by not showing users the effects of their actions) may, counterintuitively, increase user accuracy, but at the cost of significantly higher perceived complexity and task time.

A.7 Next Steps

A natural next step following this study will be to extend and validate the model in the IT configuration domain through a controlled user study. Again we are facing the challenge

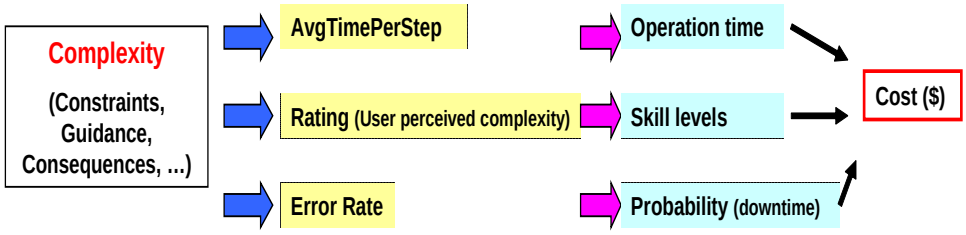


Figure A.5: Steps

of choosing a real scenario, which we can tailor to test various factors of our model. We propose to use a simulated installation process (Figure A.4), where the user has a specific installation goal to achieve and has to go through various decision steps based on provided information (wizard, message windows, buttons...) and choose the right path. For example, the installation process might be to install the web portal software stack mentioned earlier, with the requisite decisions concerning product versions and deployment topology. This approach has the following advantages:

- it is close to a real IT installation process and thus will be familiar to most IT-trained people
- we will have full control over the process
- we can borrow the framework from our route-planning study (on-line experiment engine, test case design etc)

In fact, as described earlier, there exists a mapping between the route-planning domain and the installation domain. For example, the traffic in driving can be seen as analogous to compatibility between software or to machine capacity limits. The global map is analogous to an installation/configuration manual or to a flowchart of the overall process. Likewise, the driving time per road segment can be mapped to the number of features achieved per installation step.

Extrapolating from our earlier results, we can hypothesize that the quality of guidance provided—in terms of overall global configuration flow as well as step-by-step goal-directed guidance—will dominate an IT administrator's perception of decision complexity, whereas the degree of compatibility and software configuration sequencing constraints will dominate the decision time in the installation/configuration process. However, as next steps we need to validate this hypothesis with concrete data from follow-on user studies in the IT domain.

Making use of our current general framework as discussed in section A.5.3, we can expect that conducting these next user studies would be straight-forwarded in terms of implementation.

After validating and refining the model in the actual IT context, the next step to take it further is to start producing mappings from the model-based measures to higher-level measures that speak directly to aspects of IT administration cost. As figure A.5 shows, the idea is to calibrate or map the model measures to higher-level measures such as the time it takes to perform a configuration procedure, the skill level required, and the probability of success at various skill levels. This calibration will almost certainly require the integration of decision complexity with the base complexity measures we developed in previous work [22]. It will additionally require either an extensive user study with trained IT administrators of different skill levels performing real (but controlled) IT tasks, or the collection of a corpus of field data from practicing system administrators performing configuration tasks on production IT environments.

Once we have completed the above calibration to metrics such as time, skill, and error rate for specific configuration procedures, we will then be able to recursively apply our complexity analysis to the collections of IT configuration and administration tasks performed in large IT shops. Here, we will use documented IT management processes to guide the analysis; these may be the aforementioned ITIL best practices [144] or other

multi-role IT processes formally-documented in swimlane format, as described in [21]. Ultimately, our hope is to be able to use such processes to guide an evaluation framework, or benchmark, that can analyze each key process activity for complexity and produce a prediction of the cost incurred by the process (in terms of labor cost and downtime cost). While this is a lofty goal that will not be reached overnight, its realization would provide a tremendous asset in helping to simplify current IT infrastructures and ensure that the new ones we build have the least complexity possible.

A.8 Conclusions

We developed a model of decision complexity in the context of understanding the complexity of configuring computing systems. The model includes three factors: constraints, levels of guidance and consequences. Based on the model, we conducted a carefully controlled user study in an analogous route-planning domain. We discussed both qualitative and quantitative results. We revealed the important fact that decision complexity has significantly different impacts on user-perceived difficulty than on objective measures like time and error rate. And we identified the key factors affecting decision complexity, which we use to extract some basic guidance for reducing complexity. We also proposed our next step on validating the model in real IT contexts. And we described our future work on mapping measures through the model to higher-level measures, which we believe will ultimately bring us to quantitative benchmarks towards less complex, more easily managed IT infrastructures.

Appendix B

User Study Instructions for Chapter 2

We are trying to understand how different users get irritated due to the slow performance of their computers. Thank you for participating in our study. At the end of the study, you will receive a receipt which you can redeem for $15.

During this study, which will take approximately 1.5 hours, you will simply use a Windows computer for four tasks while we make it slower and faster in different ways. **The purpose is to get your feedback when the computer gets unexpectedly slow for your operation.** You can start each task by clicking on appropriately labeled shortcuts on the Windows Desktop. Each task is 16 minutes long and you will receive an alert when it is time to close the task and proceed to the next one. If an error occurs, please ask the proctor for help.

Whenever you feel discomforted by the unexpected slowness of the computer, you should express your irritation using the "I am Irritated" button located on top of the keyboard or by right clicking the irritation icon (Figure B.1) in the system tray:

Figure B.1: Tray interface.

Each time you press the button, the speed of the computer will return to normal and you can resume your task. You may need to express irritation several times during your tasks. Please press the button whenever you feel discomforted by the slowness of the computer. The proctor will give you a short demo before you start.

The Tasks

You will perform the following tasks one after the other.

1. **Adaptation Phase**

   Before beginning the actual tasks, we will simply let you use the computer for 10 minutes. You are recommended to have a feel of these tasks by using the applications (Word, PowerPoint, Internet Explorer and Quake) for couple of minutes. **The purpose is to get a feel for the normal speed of the PC.**

2. **Word Processing (16 minutes)**

   You will use Microsoft Word to type a document, which will be provided to you in hardcopy. You should type at a comfortable pace for you. Please maintain the original formatting of the document as closely as you can.

3. **Presentation Software (16 minutes)**

   You will use Microsoft PowerPoint to draw some diagrams using the drawing features of PowerPoint. The document will be given to you in hard copy. You should work at a comfortable pace. Please try to duplicate the diagram as closely as possible including fonts and positions etc.

4. **Web browsing and searching (16 minutes)**

   The purpose of this task to do normal web browsing which involves opening and closing several IE windows, and also saving several web pages onto the disk. The assigned task is:

You will search and save information about phrases from news stories at `http://dailynews.yahoo.com/`.

(a) You should go to the Yahoo news website and read the top news story briefly (the first paragraph or so). You should then save the page a folder by your name in C:.

(b) Then you should select several keywords or phrases from the story (at least three) and search for these using Google. You can also search for images about these topics using Googles Image search. You should save the results of your searches to the folder by your name in C:.

(c) If you are done with the above, you can go the next news story and repeat.

5. **Playing an Action/Arcade Game (16 minutes)**

   You will play a Windows first-person shooter game. You can play as you choose for the duration. You need to play the game in the windows mode, not in the full screen mode. If you havent played this game before, we can give a short demo.

Notes:

- You can save all the files in a directory by your name in C:.
- Four icons are placed on the desktop named “task1”, “task2” and so on, which start each of the four tasks for you. It also starts a timer which automatically tells you after 16 minutes that time is over.
- At the start and end of each task, please call the proctor for assistance and for setting up the tasks for you.

All the best!

The User Irritation Team

Date/Time:

Name:

Email:

ID:

Proctor Proctor Signature:

Appendix C

User Study Instructions for Chapter 4

Thank you for participating in our study. At the end of the study, you will receive a receipt which you can redeem for $15.

During this study, which will take approximately 1.5 hours, you will simply use a Windows computer for four tasks (Word Processing, Presentation Creation, Web Browsing and Game Playing). Each task has 3 sub-tasks. The proctor will start each task for you. Each task is 15 minutes long with 5 minutes for each sub-task. You will receive an alert when each subtask is done. When you finish a subtask, you will answer several simple questions associated with that subtask, and then ask the proctor to start next subtask for you. If an error occurs, please ask the proctor for help.

Video Taping

We will video tape you during the study. The purpose of the video tape is to help us determine your degree of comfort during the study independently of the questions we will ask you. After we have done so, we will destroy your video tape.

The Tasks

You will perform the following tasks one after the other. Please finish all tasks shown on each page (5 pages in total) before you proceed to next page.

Adaptation Phase I (8 minutes)

Before beginning the actual tasks, we will simply let you use the Windows-based computer for 8 minutes. You should get a feel for the performance of the computer in our four applications (Word, Powerpoint, Internet Explorer, and Quake). During this stage, our control mechanism will be inactive.

When you are done with Adaptation Phase I, please answer the following questions:

- **Do you feel you are familiar with the performance of this computer? (Y / N)**
- **Are you comfortable with these applications? (Y / N)**

Adaptation Phase II (5 minutes)

This phase will familiarize you with our control mechanism, which is a non-centering joystick. Moving the joystick will change the responsiveness of the computer. In general, moving the joystick to the upper-right will make the machine faster, while moving the joystick to bottom-left will slow the machine down. However, the control is not a simple linear control—all joystick positions are valid. You will listen to mp3 music using Windows Media Player and notice how the playback changes when you move the joystick.

When you are done with Adaptation Phase II, please answer the following questions:

- **Do you feel that you understand the control mechanism? (Y / N)**
- **Do you feel that you can use the control mechanism? (Y / N)**

Thanks. Then ask the proctor to start the first task for you.

Word Processing (15 minutes)

You will use Microsoft Word to type a document assigned to you. You should type at a comfortable pace for you. Please maintain the original formatting of the document as closely as you can.

Sub-task I (5 minutes) Youre trying to find a comfortable joystick setting. When the timeout alert shows, answer these 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)**

Thanks. Then ask the proctor to start the next sub-task for you.

Sub-task II (5 minutes)

The proctor will bring up a cost bar for you, which will show the current cost of using the Windows computer. When you move the joystick, both the responsiveness of the computer and current cost will change. Do your best to find a comfortable joystick setting that is of the lowest cost. When the timeout alert shows, answer these 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, whats the cost?**

Thanks. Then ask the proctor to start the next sub-task for you.

Sub-task III (5 minutes)

This sub-task is similar to previous one, except that now we will check for lowest cost through a background computer efficiency measurement, and check for your comfort through analysis of your input through mouse, keyboard and joystick and the video tape. Do your best to find a comfortable joystick setting that is of the lowest cost. When the timeout alert shows, answer these 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, whats the cost?**

Thanks. Then ask the proctor to start the next task for you.

Presentation Software (15 minutes)

You will draw some diagrams using the drawing features of PowerPoint. You should work at a comfortable pace for you. Please try to duplicate the diagram we give you as closely as possible including fonts and positions etc.

Sub-task I (5 minutes) Youre trying to find a comfortable joystick setting. When the timeout alert shows, answer these questions:

- **Did you find that the joystick control was understandable in this application? (Y / N)**
- **Can you find a setting that is comfortable for your PowerPoint preparation? (Y / N)**

Thanks. Then ask the proctor to start the next sub-task for you.

Sub-task II (5 minutes)

The proctor will bring up a cost bar for you, which will show the current cost of using the Windows computer. When you move the joystick, both the responsiveness of the computer and current cost will change. Do your best to find a comfortable joystick setting that is of the lowest cost. When the timeout alert shows, answer these 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, whats the cost?**

Thanks. Then ask the proctor to start the next sub-task for you.

Sub-task III (5 minutes)

This sub-task is similar to previous sub-task, except that now we will check for lowest cost through a background computer efficiency measurement, and check for your comfort through analysis of your input through mouse, keyboard and joystick and the video tape. Do your best to find a comfortable joystick setting that is of the lowest cost. When the timeout alert shows, answer these 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, whats the cost?**

Thanks. Then ask the proctor to start the next task for you.

Web browsing and searching (15 minutes)

You will search and save information about phrases from news stories at `www.cnn.com`.

1. You should go to the CNN website and read the top news story briefly (the first paragraph or so). You should then save the page to the "Top Stories Folder" on the desktop.
2. After saving the page, you should select several keywords or phrases from the story (at least three) and search for these using Google. You can also search for images about these topics using Googles Image search. You should save the results of your searches to the "Top Stories Folder".
3. If you are done with the above, you can go the next news story and repeat.

Sub-task I (5 minutes) Youre trying to find a comfortable joystick setting. When the timeout alert shows, answer these 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)**

Thanks. Then ask the proctor to start the next sub-task for you.

Sub-task II (5 minutes)

The proctor will bring up a cost bar for you, which will show the current cost of using the Windows computer. When you move the joystick, both the responsiveness of the computer and current cost will change. Do your best to find a comfortable joystick setting that is of the lowest cost. When the timeout alert shows, answer these 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, whats the cost?**

Thanks. Then ask the proctor to start the next sub-task for you.

Sub-task III (5 minutes)

This sub-task is similar to previous sub-task, except that now we will check for lowest cost through a background computer efficiency measurement, and check for your comfort through analysis of your input through mouse, keyboard and joystick and the video tape. Do your best to find a comfortable joystick setting that is of the lowest cost. When the timeout alert shows, answer these 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, whats the cost?**

Thanks. Then ask the proctor to start the next sub-task for you.

Playing a First Person Shooter Game (15 minutes)

You will play a Windows first-person shooter game Quake II. You will need to play the game in the windows mode, not in the full screen mode. If you havent played this game before, we can give a short demo.

Sub-task I (5 minutes) Youre trying to find a comfortable joystick setting. When the timeout alert shows, answer these 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)**

Thanks. Then ask the proctor to start the next sub-task for you.

Sub-task II (5 minutes)

The proctor will bring up a cost bar for you, which will show the current cost of using the Windows computer. When you move the joystick, both the responsiveness of the computer and current cost will change. Do your best to find a comfortable joystick setting that is of the lowest cost. When the timeout alert shows, answer these 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, whats the cost?**

Thanks. Then ask the proctor to start the next sub-task for you.

Sub-task III (5 minutes)

This sub-task is similar to previous sub-task, except that now we will check for lowest cost through a background computer efficiency measurement, and check for your comfort through analysis of your input through mouse, keyboard and joystick. Do your best to find a comfortable joystick setting that is of the lowest cost. When the timeout alert shows, answer these 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, whats the cost?**

Thanks. This is the end of the study. Please give this document to the proctor.

Appendix D

User Study Instructions for Chapter 7

D.1 Instructions for VM scheduling game

We are trying to find out whether users can solve difficult optimization problems for computer systems. Thank you for participating in our study. At the end of the study, you will receive a receipt which you can redeem for $10.

During this study, which will take approximately 1 hour, you will simply use a Windows computer to play a game through a graphical user interface. The proctor will start each task for you. You will receive an alert when it is time to close the task. When you finish a subtask, you will answer several simple questions associated with that subtask, and then ask the proctor to start next subtask for you. If an error occurs, please ask the proctor for help.

Please print

Name:

Date:

Time:

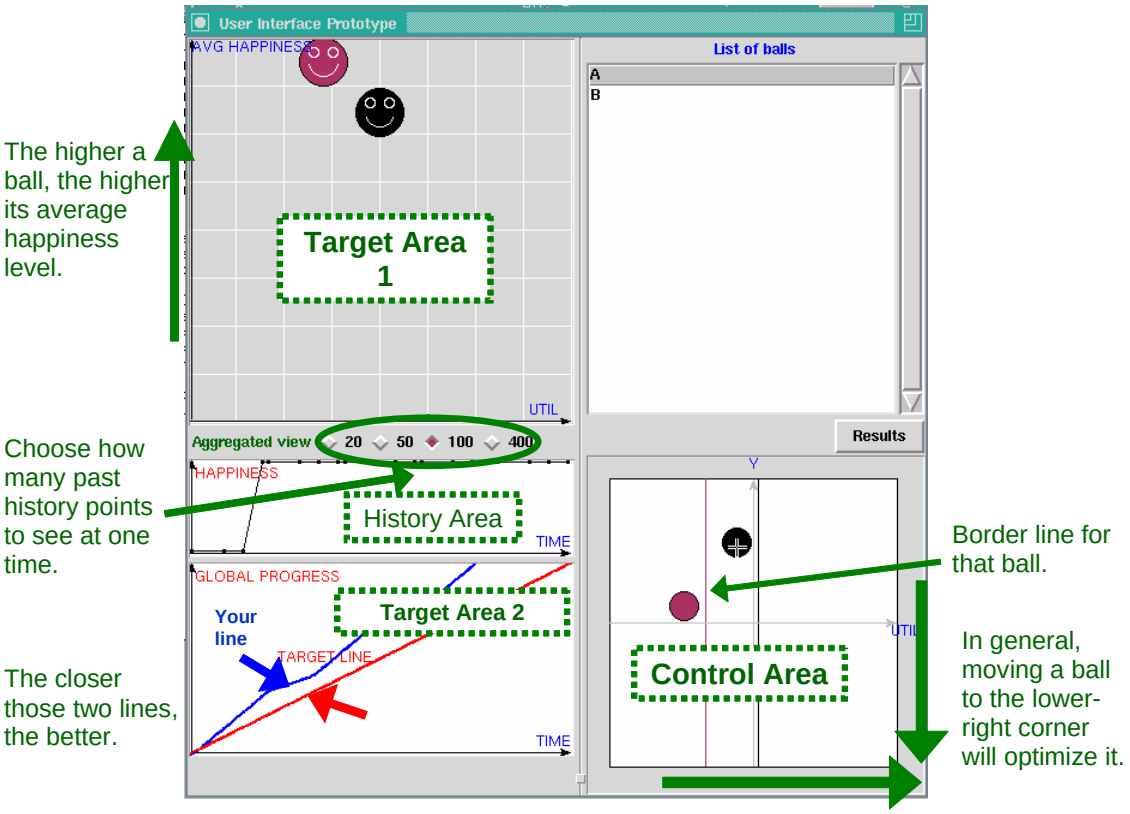


Figure D.1: Game interface.

Introduction to the game interface

Two target areas:

1. In the Target Area 1, the **average** happiness level of each ball will be dynamically displayed.

2. In the Target Area 2, both your progress line (blue) across all balls and the target line (red) will be dynamically displayed.

Goals: By moving your balls in the Control Area, you want to **maximize the happiness levels for all balls** (Target Area 1) AND **minimize the gap between your progress line and the target progress line (Target Area 2)**.

Constraints: There is one constraint line for each ball in the Control Area. The ball and its constraint line will be displayed in the same color. You will not be able to move a ball to the right of its constraint line. In addition, the constraint lines of all balls are related in various ways so that when you move one ball, the constraint lines for other balls will change.

Information: While the Target Area shows the average happiness level of a ball, in the History Area, you will see the dynamic history of that ball. When you click on a ball either in the Control Area or the Target Area 1, its happiness history will be displayed in the History Area. You can further choose how many past history points (ahead of current point of time) that you want to see at one time to help you make the next decision. Note that the happiness level of a ball in the Target area is averaged over the number of past history points that you choose. For example, 100 means that the happiness level (in the Target Area) of a ball is the average value over last 100 happiness values.

Important hints:

- If there is only one ball in the game, moving it to the lower-right will optimize it and achieve the goal of the game.
- After you move a ball to a new position, it will take some time for your action to affect the happiness levels of balls and progress line. As the number of balls increases, it may take longer for you to notice the change after each move.
- We suggest that after you make a move, observe the effect for some time before you try to move again. Be patient!
- Moving a ball will not only change its own happiness and progress but also influence other balls. Try to discover the relationship among balls.

- The difficulty of the game will increase as you proceed. A good strategy will require considering BOTH
  - Short term happiness level of single ball in the Target Area 1 & History Area.
  - Long term global progress of all balls in the Target Area 2.
  - Note: the global progress of all balls depends on their happiness levels.

The Tasks You will perform the following tasks one after the other.

1. **Adaptation Phase 1 (5 minutes)**

   Before beginning the actual tasks, we will simply let you play a simple game with only 2 balls so that you can a feel of this game. Moving a ball will change its happiness level and its progress, which will further influence other balls. In general, moving a ball to the lower-right will make it happy. However, all positions in the Control Area are valid.

   - **Do you feel that you understand the graphic interface? (Y / N)**
   - **Do you feel that you can use the interface to achieve the goal of the game? (Y / N)**

2. **Adaptation Phase 2 (5 minutes)**

   We will now let you play another simple game with only 3 balls so that t you can have a further feel of this game. Try to develop your own strategy.

   - **Do you feel that you understand the graphic interface? (Y / N)**
   - **Do you feel that you can use the interface to achieve the goal of the game? (Y / N)**

   Thanks. Then ask the proctor to start the first task for you.

3. **4 balls (8 minutes)**

   You will be given 4 balls to play with. When the timeout alert shows, answer these questions:

   - **Were you able to use the interface to achieve the goal of the game? (Y / N)**
   - **If no, why?**

4. **4 balls (8 minutes)**

   You will be given 4 balls to play with. When the timeout alert shows, answer these questions:

   - **Were you able to use the interface to achieve the goal of the game? (Y / N)**
   - **If no, why?**

5. **8 balls (8 minutes)**

   You will be given 8 balls to play with. When the timeout alert shows, answer these questions:

   - **Were you able to use the interface to achieve the goal of the game? (Y / N)**
   - **If no, why?**

6. **8 balls (8 minutes)**

   You will be given 4 balls to play with. When the timeout alert shows, answer these questions:

   - **Were you able to use the interface to achieve the goal of the game? (Y / N)**
   - **If no, why?**

7. **16 balls (8 minutes)**

   You will be given 16 balls to play with. When the timeout alert shows, answer these questions:

   - **Were you able to use the interface to achieve the goal of the game? (Y / N)**
   - **If no, why?**

8. **16 balls (8 minutes)**

   You will be given 16 balls to play with. When the timeout alert shows, answer these questions:

   - **Were you able to use the interface to achieve the goal of the game? (Y / N)**
   - **If no, why?**

All the best!

D.2 Instructions for VM scheduling & mapping game

We are trying to find out whether users can solve difficult optimization problems for computer systems. Thank you for participating in our study. At the end of the study, you will receive a receipt which you can redeem for $12.

During this study, which will take approximately 1 hour and 15 minutes, you will simply use a Windows computer to play a game through a graphical user interface. The proctor will start each task for you. You will receive an alert when it is time to close the task. When you finish a subtask, you will answer several simple questions associated with that subtask, and then ask the proctor to start next subtask for you. If an error occurs, please ask the proctor for help.

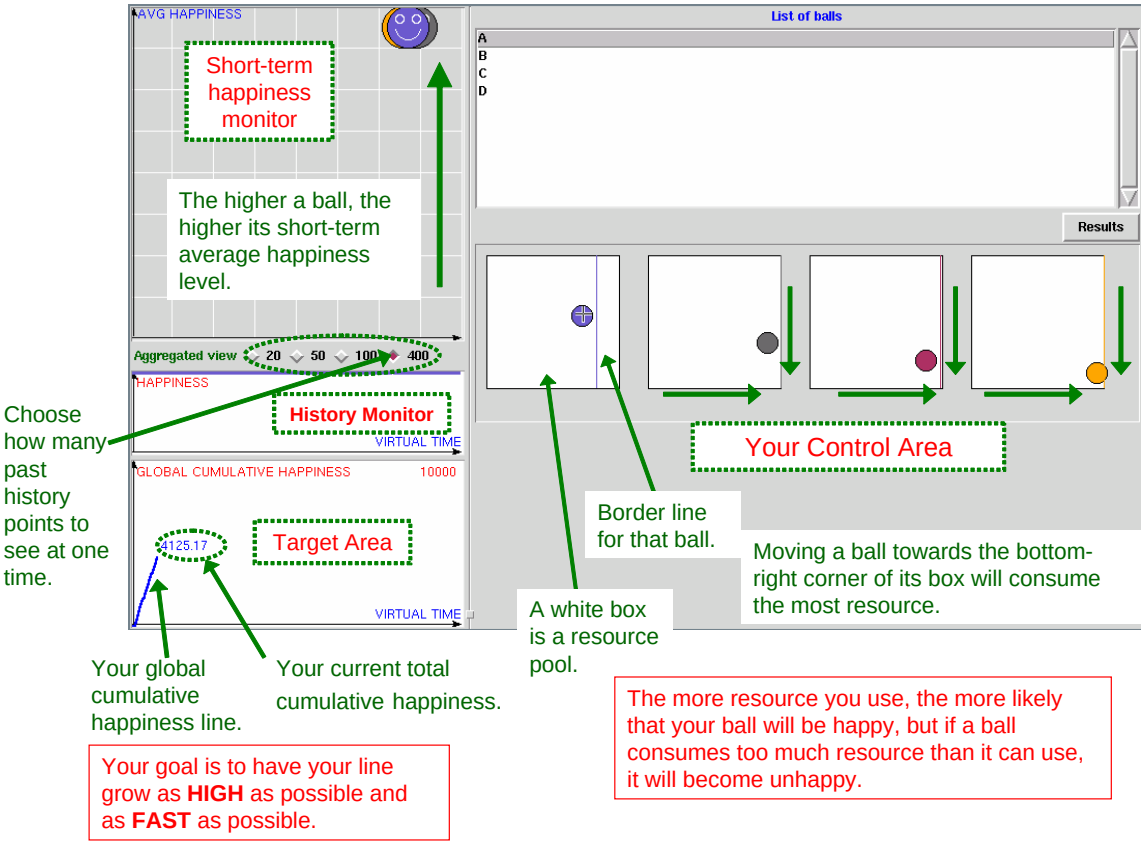


Figure D.2: Game interface.

Please print Name:

Date:

Time:

Introduction to the game interface

Target Area: Your global cumulative happiness line (blue) will be dynamically displayed over time. You may think of the happiness line as the sum of the happiness levels across **ALL** balls.

Goal:

By moving your balls in the Control Area, you want to push your global cumulative

happiness line to as HIGH as possible. The growth speed of your line can be judged from the slope of line. You want the slope to be as steep as possible. Note that the ceiling of the Target Area may or may not be reachable.

History Area:

In this Area, the happiness history of your **SELECTED** ball will be displayed. You can select a ball in the Control Area, Ball List or Short-term Happiness Monitor. Note that you need to select how many past history points up to the current time that you want to watch at one time to help you make the next decision.

Short-term Happiness Monitor:

The **average** happiness level (height) of each ball will be dynamically updated here. The average is calculated over the number of past history points that your select in the History Area. So if you select 10 in the History Area, the height of the balls in the monitor is equal to the average happiness over last 10 happiness levels.

Constraint Line:

There is one constraint line for each ball in the Control Area. The ball and its constraint line will be displayed in the same color. You will not be able to move a ball to the right of its constraint line. In addition, the constraint lines of all balls are related in various ways so that when you move one ball, the constraint lines for other balls will change. The constraint line reflects the resource usage constraint for that ball.

Control Area:

You have two ways of controlling a ball, **move it within the same resource box or migrate it to a new resource box**.

How to move a ball within the same box:

Left-click on the ball that you want to move **until a surrounding black circle appears**. Then move the ball. The circle is an indicator of your current position and will disappear after you move the ball to a new position and release the mouse button.

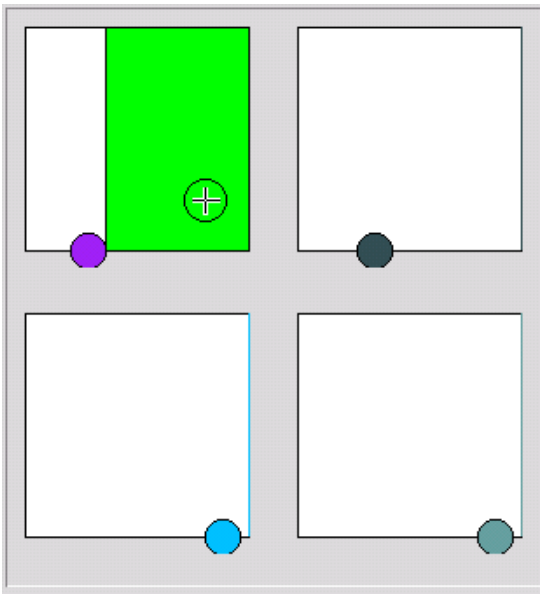


Figure D.3: Migration.

How to migrate a ball to a different resource box (2 ways):

1. If you try to move a ball outside of its box, or across its constraint line, you may see some green areas appear to indicate potential new resource boxes for this ball. Hold the left-button and move the mouse to one green area, release the button, the ball will be migrated to the new box. You need to make sure the center of the mouse cross fall within the green area in order to migrate a ball.

2. If you **hold the left SHIFT key**, select and move a ball, release the mouse button and then **release the left SHIFT key**, the position of the ball will remain UNCHANGED no matter where you move your ball within the same box. You will notice that the ball will jump back to its old position, after you release the SHIFT key. However, if you keep holding the left SHIFT key and move the ball outside of its box, or across its constraint line to a green area, the ball will be migrated.

3. The difference between 1) and 2), in terms of migration, is that in 2) you will be able to maintain the same position (i.e. the same resource usage level) of the ball while moving it to a new resource box. In 1), you need to first of all move a ball to the border of its current box or across its constraint line, which changes its resource usage, before you are able to migrate it.

Penalty of moving a ball to a different box:

The ball will be frozen during the migration for a certain amount of time. During that time, a red circle will appear surrounding the ball. **You will not be able to move that ball until the red circle disappears.** In addition, during the migration, the ball being migrated will not contribute to the global happiness line, and its own happiness level will not be updated. The happiness level of other balls may be affected due to the temporary absence of the frozen ball.

Important hints for you to beat the game:

- In general moving a ball downward and/or rightward will consume more resources. However all positions in the Control Area are valid.

- The more resources you assign to a ball, the more likely that your ball will become happier, BUT if a ball consumes more resources than it can use, it will become unhappy. In addition, all balls share a global resource pool, so if one ball gets more resources, the other balls will get less. Try to discover the relationship among balls will help you select the correct resource usage level.

- After you move a ball to a new position, it will take some time for your action to affect the happiness levels of balls and the cumulative global happiness line. As the number of balls increases, it may take longer for you to notice the change after each move. We suggest that after you make a move, observe the impact for a while before making a new move again. Be patient!

- If all balls are happy most of the time in the Short-term Happiness Monitor, your cumulative global line will certainly grow fast and high, but keep in mind that your ONLY goal is to get the global happiness line as high as possible. You may observe a certain degree of fluctuation of the happiness of balls. If the fluctuation is temporary and short, it does not necessarily mean that you need to adjust the balls. However, if the fluctuation is persistent and/or displays a certain pattern, it is then necessary for you to make some adjustment.

- In the end of each game, you will see a similar summary as shown in Figure D.4. Your final score is calculated based on the global cumulative happiness averaged across all balls and all history points. The closer your score to the highest possible score, the better you perform. You will be asked to evaluate yourself in the end of each game.

- Note that although we show the highest score in theory, we do not know the highest achievable score in practice. That is why we bring you here to help us find it out!

The Tasks

You will perform the following tasks one after the other.

1. **Adaptation Phase 1 (4 minutes)** Before beginning the actual tasks, we will simply let you play a simple game with only 2 balls so that you can a feel of this game. You can try out different controls.

    - Do you feel that you understand the graphic interface? (Y / N)
    - What is your final score?
    - How do you evaluate your performance in this game session? (1 (bad) to 10 (outstanding))

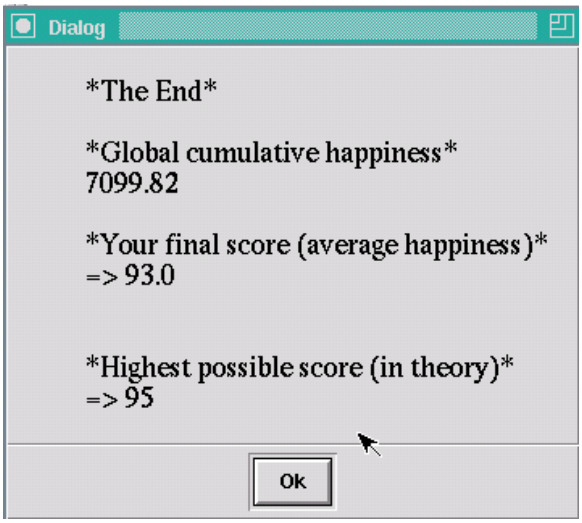


Figure D.4: final screen.

2. **Adaptation Phase 2 (4 minutes)** We will now let you play another simple game with only 3 balls so that t you can have a further feel of this game. Try to develop your own strategy.

   - Do you feel that you understand the graphic interface? (Y / N)
   - What is your final score?
   - How do you evaluate your performance in this game session? (1 (bad) to 10 (outstanding))

3. **4 balls (7 minutes)**

   You will be given 4 balls to play with. When the timeout alert shows, answer these questions:

   - What is your final score?
   - How do you evaluate your performance in this game session? (1 (bad) to 10 (outstanding))

- If bad, why?

4. **4 balls (7 minutes)**

   You will be given 4 balls to play with. When the timeout alert shows, answer these questions: What is your final score?

   - What is your final score?
   - How do you evaluate your performance in this game session? (1 (bad) to 10 (outstanding))
   - If bad, why?

5. **4 balls (7 minutes)**

   You will be given 4 balls to play with. When the timeout alert shows, answer these questions: What is your final score?

   - What is your final score?
   - How do you evaluate your performance in this game session? (1 (bad) to 10 (outstanding))
   - If bad, why?

6. **8 balls (7 minutes)**

   You will be given 8 balls to play with. When the timeout alert shows, answer these questions: What is your final score?

   - What is your final score?
   - How do you evaluate your performance in this game session? (1 (bad) to 10 (outstanding))

- If bad, why?

7. **8 balls (7 minutes)**

   You will be given 8 balls to play with. When the timeout alert shows, answer these questions: What is your final score?

   - What is your final score?
   - How do you evaluate your performance in this game session? (1 (bad) to 10 (outstanding))
   - If bad, why?

8. **8 balls (7 minutes)**

   You will be given 8 balls to play with. When the timeout alert shows, answer these questions: What is your final score?

   - What is your final score?
   - How do you evaluate your performance in this game session? (1 (bad) to 10 (outstanding))
   - If bad, why?

9. **10 balls (7 minutes)**

   You will be given 10 balls to play with. When the timeout alert shows, answer these questions: What is your final score?

   - What is your final score?
   - How do you evaluate your performance in this game session? (1 (bad) to 10 (outstanding))

- If bad, why?

10. **10 balls (7 minutes)**

    You will be given 10 balls to play with. When the timeout alert shows, answer these questions: What is your final score?

    - What is your final score?
    - How do you evaluate your performance in this game session? (1 (bad) to 10 (outstanding))
    - If bad, why?

11. **10 balls (7 minutes)**

    You will be given 10 balls to play with. When the timeout alert shows, answer these questions: What is your final score?

    - What is your final score?
    - How do you evaluate your performance in this game session? (1 (bad) to 10 (outstanding))
    - If bad, why?

Appendix E

User Study Instructions for Chapter 8

We are trying to understand how different users get irritated due to the slow performance of their computers. Thank you for participating in our study. At the end of the study, you will receive a receipt which you can redeem for $15.

During this study, which will take approximately 1 hour, you will simply use a Windows computer for six tasks while we make it slower and faster in different ways. The proctor will start each task for you. You will receive an alert when it is time to close the task and proceed to the next one. If an error occurs, please ask the proctor for help.

Whenever you feel discomforted by the unexpected slowness of the computer, you should express your irritation using the "I am Irritated" button (F11) located on top of the keyboard or by right clicking the irritation icon (Figure E.1) in the system tray. You will hear a beep as feedback each time you press the button:

Each time you press the button, the speed of the computer will change. You may need to express irritation several times during your tasks. Please press the button whenever you

Figure E.1: Tray interface.

feel discomforted by the slowness of the computer. The proctor will give you a short demo before you start.

The Tasks

You will perform the following tasks one after the other.

1. **Adaptation Phase**

   Before beginning the actual tasks, we will simply let you use the computer for 5 minutes. You are recommended to have a feel of these tasks by using the applications for couple of minutes and getting a feel for the normal speed of the PC.

2. **Presentation Software (4 minutes)**

   You will listen to music and use Microsoft PowerPoint to draw some diagrams using the drawing features of PowerPoint. You should work at a comfortable pace for you. Please try to duplicate the diagram as closely as possible including fonts and positions etc. You can save all the files in a directory by your name on the desktop of Windows.

3. **Watching 3D animation using Web browser (4 minutes)**

   You will watch a 3D animation using Microsoft Internet Explorer. You can use num key 0-9 to change the cameras of the animation.

4. **Playing a Game (8 minutes)**

   You will play a Windows game. If you havent played this game before, we can give a short demo.

5. **Presentation Software (4 minutes)**

   You will listen to music and use Microsoft PowerPoint to draw some diagrams using the drawing features of PowerPoint. You should work at a comfortable pace for

you. Please try to duplicate the diagram as closely as possible including fonts and positions etc. You can save all the files in a directory by your name on the desktop of Windows.

6. **Watching 3D animation using Web browser (4 minutes)**

   You will watch a 3D animation using Microsoft Internet Explorer. You can use num key 0-9 to change the cameras of the animation.

7. **Playing a Game (8 minutes)**

   You will play a Windows game. If you havent played this game before, we can give a short demo.

All the best!