CONTEXT-AWARE RESOURCE MANAGEMENT

by

Igor Crk

A Dissertation Submitted to the Faculty of the
DEPARTMENT OF COMPUTER SCIENCE
In Partial Fulfillment of the Requirements
For the Degree of
DOCTOR OF PHILOSOPHY
In the Graduate College
THE UNIVERSITY OF ARIZONA

2010
As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Igor Crk entitled Context-Aware Resource Management and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Date: 16 April 2010

Chris Gniady

Date: 16 April 2010

John Hartman

Date: 16 April 2010

Kobus Barnard

Date: 16 April 2010

Beichuan Zhang

Final approval and acceptance of this dissertation is contingent upon the candidate’s submission of the final copies of the dissertation to the Graduate College. I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Date: 16 April 2010

Dissertation Director: Chris Gniady
STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Igor Crk
Looking back on this long journey, it is clear that many have contributed to the success of this work, each in their own unique way.

First, my thanks to my advisor, Chris Gniady, whose pursuit of innovation and penchant for unconventional approaches to important problems, such as energy management, enlivened this research. Your advice and insight were invaluable to the success this work has enjoyed. Thank you for your guidance and help in reaching this important milestone.

A heartfelt thanks to my friends and fellow musicians, Christian Collberg, Michael Hammer, and Stephen Kobourov. Our musical interludes were an important outlet throughout the challenges of the recent years. As a poet once said, “Music washes away from the soul the dust of everyday life.”

Many thanks to my friends from the department of computer science: Fahd Albinali, Mingsong Bi, Alex Balderrama, Kevin Coogan, Varun Khare, Kyri Pavlou, Somu Perianayagam, Joe Roback, and Rui Zhang.

A simple thanks will never be enough to satisfy the debt of gratitude I have to my family, Vladimir, Vesna, and Tanja. Your love is a beacon that orients me even in the darkest of times.

Finally, a thanks to Libby, my patient soul-mate.
DEDICATION

To Libby
# TABLE OF CONTENTS

LIST OF FIGURES ......................................................... 8

LIST OF TABLES ......................................................... 10

ABSTRACT .................................................................. 11

CHAPTER 1 INTRODUCTION .................................................. 13
  1.1 Interactive Systems and Context ................................. 15
  1.2 Role of Interactions in User Think-Time ......................... 16
  1.3 Role of Interactions in Delay Management of Disks .......... 19
  1.4 Role of Interactions in Energy Management ...................... 20

CHAPTER 2 BACKGROUND .................................................... 23
  2.1 User Activity Monitoring Systems ............................... 24
     2.1.1 Automatic User Interaction Capture ......................... 25
  2.2 Facilitated Pointing ............................................... 27
  2.3 Disk Energy Management ........................................... 28
     2.3.1 Shutdown prediction techniques .............................. 28
     2.3.2 Reducing spin-up delays ....................................... 29
     2.3.3 Predicting spin-up time ...................................... 30
     2.3.4 Challenges in predicting spin-up time ....................... 30
  2.4 Wireless Network Card Energy Management ...................... 31

CHAPTER 3 CAPTURING INTERACTIVE CONTEXT ....................... 36
  3.1 Context Capture in Linux ........................................... 36
     3.1.1 X Window System ........................................... 37
     3.1.2 Listening with xmond ........................................ 39
     3.1.3 Logical mapping of user interfaces ....................... 40
     3.1.4 Evaluating Overheads of Context Capture ............... 43
     3.1.5 Aliasing ....................................................... 50
  3.2 Application of Context Capture in Microsoft Windows ........ 51
     3.2.1 Predictive Target Selection in Pull-Down Menus .......... 51
     3.2.2 Capturing Context in Windows .............................. 52
     3.2.3 Prediction Mechanisms ....................................... 54
     3.2.4 Evaluation .................................................... 59
     3.2.5 Acquisition Distance ....................................... 62
TABLE OF CONTENTS – Continued

CHAPTER 4 MANAGING DELAYS IN DISKS ........................................... 68
  4.1 Modified Adaptive Learning Tree ........................................... 68
  4.2 Interaction-Aware Spin-Up Prediction Design ............................. 69
    4.2.1 The Naïve Predictor .................................................. 70
    4.2.2 Monitoring & correlating I/O activity ............................... 71
    4.2.3 To predict or not to predict ....................................... 74
    4.2.4 Predicting upcoming activity ..................................... 75
    4.2.5 Multiprogramming environment ................................... 76
  4.3 Evaluation ........................................................................ 76
    4.3.1 File I/O Correlation Accuracy ..................................... 78
    4.3.2 Impact of the Buffer Cache ........................................ 81
    4.3.3 Confidence Levels ..................................................... 82
    4.3.4 Delay Reduction ....................................................... 84
    4.3.5 Energy .................................................................... 86
    4.3.6 Energy-Delay ........................................................... 88
    4.3.7 Overheads ............................................................... 89
    4.3.8 Experimental Evaluation ............................................ 90

CHAPTER 5 MANAGING ENERGY IN WNICs .................................... 93
  5.1 Design ............................................................................ 93
    5.1.1 The Naïve Predictor .................................................. 93
    5.1.2 Capturing GUI interactions ........................................ 94
    5.1.3 Advanced Prediction ................................................ 96
    5.1.4 Mode Switching Heuristics ....................................... 98
  5.2 Methodology .................................................................... 103
  5.3 Evaluation ....................................................................... 105
    5.3.1 Accuracy ................................................................. 106
    5.3.2 Delay ..................................................................... 109
    5.3.3 Energy ................................................................. 111
    5.3.4 Energy-Delay^2 ..................................................... 113
    5.3.5 Overheads ............................................................. 114

CHAPTER 6 CONCLUSIONS AND FUTURE DIRECTIONS .................... 117
  6.1 Context ......................................................................... 117
  6.2 Disk Delays ..................................................................... 118
  6.3 WNIC Energy ................................................................. 119
  6.4 Future Directions ........................................................... 120

REFERENCES ........................................................................ 122
# LIST OF FIGURES

1.1 Computer system organization and optimization levels. 14

2.1 High level interaction capture 24
2.2 Anatomy of a disk idle period 28
2.3 Illustration of prediction timeliness 34

3.1 X Window system overview 37
3.2 Basic X Window system architecture 38
3.3 Augmented X Window hierarchy 39
3.4 Simple window layout and window hierarchy in X 40
3.5 Example interactive window layout 41
3.6 Initial overheads of computing interaction identifiers 48
3.7 Cumulative overheads of computing interaction identifiers 49
3.8 Overheads of identifier calculations 50
3.9 Context capture in windows 53
3.10 Drop-down menu example 54
3.11 Comparison of history-based prediction mechanisms 56
3.12 Comparison of probabilistic prediction mechanisms 57
3.13 Prediction accuracy 61
3.14 Normalized acquisition distance 62
3.15 Average accuracies with varying history length 65
3.16 Average distances with varying history length 66

4.1 IASP architecture 71
4.2 Example of possible interaction sequences 72
4.3 IASP decision flowchart 75
4.4 IASP prediction accuracy without buffer cache 79
4.5 IASP prediction accuracy with buffer cache 79
4.6 IASP hit/miss ratio and I/O activity coverage 83
4.7 Average delays using WD2500JD 84
4.8 Average delays using 40GNX 84
4.9 Energy consumption of WD2500JD 86
4.10 Energy consumption of 40GNX 87
4.11 Energy-delay product using WD2500JD 88
4.12 Energy-delay product using 40GNX 88
4.13 Experimental setup for power measurement 90
LIST OF FIGURES – Continued

4.14 Illustration of disk power consumption .......................... 91
5.1 IAP architecture .................................................. 95
5.2 IAP decision flowchart ........................................... 98
5.3 Example of training and prediction ............................... 100
5.4 Normalized prediction accuracy. ................................. 106
5.5 Details of delays incurred by predictors. ......................... 109
5.6 Detailed breakdown of energy consumption results. ........ 111
5.7 Energy-delay$^2$ product results. ................................. 113
LIST OF TABLES

2.1  WNIC energy consumption specifications ............................ 32

3.1  Application trace details ............................................. 46
3.2  Storage and computational overheads of identifier computation ........ 47
3.3  Details of simulations with a set history length ..................... 64

4.1  Disk energy consumption specifications .............................. 77
4.2  Details of application traces used to evaluate IASP ................. 78

5.1  Summary and comparison of IAP policies ............................ 99
5.2  Details of application traces used to evaluate IAP .................. 101
5.3  Energy and delay costs for selected mechanisms .................... 113
5.4  Mechanism overheads .................................................. 115
ABSTRACT

The demand for performance and resources that is placed on the system is dictated by the application alone in non-interactive environments, and by a combination of application and user interactions in interactive environments. Understanding user interaction can provide valuable information about which resources will be needed ahead of time. This leads to performance optimizations such as better resource allocations for applications that can utilize a given resource more productively, and transitioning devices to a more appropriate energy performance state before the demand arrives. The challenge is to provide a performance/energy schedule that best matches the task at hand, since keeping the device in one performance level is not energy efficient due to the continually changing demand placed on the device. This dissertation addresses the challenge of designing energy efficient systems by examining the role of user interaction in energy consumption and in providing an energy-performance schedule that adequately accommodates user demand. It is shown that system performance can be tailored to a user’s pattern of interaction and it’s energy-performance schedule optimized.

First, a detailed design of context capture systems in Linux’s X-Window System is presented with an evaluation of the associated storage and computation overheads. Due to the overall low complexity of the application window representations, the overheads of computing interaction identifiers and storing a secondary representation of the application interface within the context capture system are likewise low. Additionally, a Microsoft Windows-based context capture system leveraging the Active Accessibility framework is discussed and applied to improving the navigation of cascading pull-down menus.

Secondly, this dissertation addresses the application of interaction capture in energy and delay management of Wireless Network Interface Controllers/Cards.
(WNICs) and hard drives. The Interaction Aware Prediction (IAP) system for WNICs is evaluated showing that the available power modes can be effectively managed to provide energy efficiency while maintaining performance. Similarly, the Interaction Aware Spin-up Prediction (IASP) uses interaction awareness to reduce or eliminate the interactive delays associated with aggressive hard disk energy management.
CHAPTER 1

INTRODUCTION

Computer system designers face two challenges: energy and performance. The spectrum of system design considerations ranges from ultra high performance on one side, in which energy consumption is not a limiting factor, all the way to ultra low power design, where performance expectations are very low. The middle of the range is occupied by systems where both power and performance play a role in system design. Portable system designers are faced with a user demand for performance, functionality, and better user interfaces along with a longer battery life. Designers of servers and desktop systems focus almost entirely on performance, since energy usually is not a constrained resource. However, this trend has started to change since researchers have realized the positive financial and environmental implications of energy conservation for stand-alone servers and server clusters [8, 9, 19, 42]. The challenge of designing energy efficient systems lies in understanding the role of user interaction in energy consumption and in providing an energy-performance schedule that adequately accommodates user demand. Furthermore, system performance can be tailored to a user’s pattern of interaction and it’s energy-performance schedule optimized.

User interaction can be as simple as launching a batch job or much more sophisticated, as in case of interactive applications, where a user may continuously interact with an application. The goal is the same in both case: meeting user demand. Users usually demand high performance; however, that does not preclude energy optimizations. Performance and energy consumption are tightly coupled, where higher performance is usually achieved at the cost of an increased power demand. However, the key observation is that higher power demand does not necessarily translate into an increase in energy consumption. For instance, hardware in a higher performance state may complete a particular task faster than the same hard-
ware operating at a lower performance state. This reduces the time during which
the entire system has to be on. On the other hand, a particular device may not be
required by all tasks and so may be operated in a low performance state without a
significant impact on the performance of the executing application. Similarly, the
low performance state of a particular device does not impact performance if the
demand placed on the device is sufficiently low.

The demand for performance and resources that is placed on the system is dic-
tated by the application alone in non-interactive environments, and by a combina-
tion of application and user interactions in interactive environments. Understand-
ing user interaction can provide valuable information about which resources will
be needed ahead of time. This leads to performance optimizations such as better
resource allocations for applications that can utilize a given resource more produc-
tively, and transitioning devices to the more appropriate energy performance state
before the demand arrives. The last optimization refers to the situation where the
device is turned off to reduce energy consumption. The challenge is to provide a
performance/energy schedule that best matches the task at hand, since keeping the
device in one performance level is not energy efficient since the demand placed on
the device varies.

Existing work that addresses performance and energy optimizations ranges from
hardware optimizations to application transformations. Figure 1.1 shows a typical
organization of computer systems and the potential for optimization at each level.
Optimizations at lower levels are usually complementary to higher-level optimiza-
tions. The lower levels also provide the interfaces for managing both energy and
performance at a higher level. The key observation is that the resulting behavior
of all system layers is a response to the application behavior in non-interactive sys-
tems and combination of user actions and application behavior in case of interactive
applications.

1.1 Interactive Systems and Context

Since their origins in the early 1980’s, Graphical User Interfaces (GUIs) have be-
come the de facto standard for interfacing with applications. Direct manipulation of
graphical elements provides an intuitive means for a user to interact with the com-
puting environment. The most common systems that provide a GUI environment
are Microsoft Windows, Mac OS X, and the X Window System. Each windowing
system provides programmers with common interface components that are the func-
tional entities which are associated with specific features of an application, such as
a button or menu option to save a file. As such, most application functionality can
be accessed through mouse-driven interaction with the displayed visual elements,
and as pointed out by Dix et al., virtually all actions take only a single click of the
mouse button [15].

GUI organization emphasizes simplicity, where each interface component exposes
a limited set of functions to the user. By interacting with the various interface com-
ponents with the mouse or keyboard, the user triggers a series of actions. However,
low-level application monitoring alone is insufficient to discern the exact intent of
the user’s interaction. For example, the monitoring of a system call occurring when
a file is read, lacks the context necessary to distinguish user-driven application ac-
tivity from automated application behavior. Similarly, global coordinates and type
of mouse event, button up/down, is easily captured, but alone gives no indication
about the interactive elements that are causing application or system activity. So, the additional context provided by simple monitoring of mouse events such as a button press or movement can provide the additional information necessary to make the distinction sought by the first example, but this is the limit of what this additional context provides.

In order to make the context provided by mouse input useful, the collected information must consist of more than simple mouse button events or movement. System-level activity that is caused by interaction with a particular element can be more accurately described and even predicted by the unique correlation of mouse activity with specific GUI elements. The Transparent User Context Monitoring (TUCM) mechanism described in Chapter 3 make the following contributions:

- Proposes a novel online mechanism for precisely extracting interaction context using existing technologies.
- Eliminates the need for application modifications for providing similar execution context.
- Shows that the addition of context-aware functionality can be accomplished without excessive computational and storage overhead.

1.2 Role of Interactions in User Think-Time

Since their origins in the early 1980's, Graphical User Interfaces (GUIs) have become the de facto standard for interfacing with applications. Today’s user-directed manipulation of the now familiar graphical elements is arguably one of the most intuitive means of interacting with the computing environment. In modern GUIs, the cascading pull-down menus have become a common interface element for many graphical interactive applications. Selecting a menu item consists of the user performing the following steps: reading the selection options, choosing the option that is to be selected, making the selection, and ascertaining the consequences [40]. Making a selection requires the user to physically navigate through the menu via an
input device such as a mouse. Subsequently, the focus of Chapter 3 is partly on the optimization of menu navigation.

Navigation of a standard cascading pull-down menu is a frequently performed task, with convenient access to commonly accessed functions. The task of navigation and selection has become more time consuming as application functionality increases. With increased functionality, the growing number of possible operations performed by an average productivity application has lead to large, multiply nested menus, sometimes with content that is dynamically changed to fit the context to which the user has transitioned during the application’s execution. Infrequently used pointing devices and the user’s surroundings can further complicate the selection of tasks (e.g. using a laptop’s built-in pointing nub, rather than the preferred USB mouse, on an airplane).

Research in improving selection and navigation of cascading pull-down menus has recognized that in the common interactive environments, the selection task is governed by a combination of Fitts’ law and the Acott-Zhai steering law. Fitts’ law is a model of human-computer interaction with respect to pointing tasks, and provides us with the average movement time ($MT_f$) taken to complete a pointing act given a distance ($D$) from the starting point to the target and a width ($W$) of the target measured along the axis of motion.

$$MT_f = a + b \cdot \log_2(1 + D/W)$$

The logarithmic term in the equation is the Index of Difficulty (ID) of the target in bits. The parameters $a$ and $b$ are the interaction start/stop time and inherent device speed, respectively. These are determined through linear regression and vary according to the interactive environment and device used. The formulation of Fitts’ law clearly shows that smaller or more distant targets will take longer to acquire. However, Fitts’ law applies only to motion in one dimension, and in this case applies to the point-and-click action of vertical selection of a menu item within a drop-down menu. To fully model the navigation of a drop-down menu, we also consider the simplified Acott-Zhai steering law, which models the time to navigate a straight
A tunnel having some length $A$ and a constant width $W$, and using the same $a$ and $b$ parameters.

$$MT_s = a + b \cdot (A/W)$$  \hfill (1.2)

Through a combination of precise user-interaction monitoring of an expert user interacting with common applications, we evaluate a number of low-overhead predictive mechanisms and heuristics that transition the pointer to target options within pull-down menus to reduce the navigable distance $A$. The goal is to reduce or eliminate pointer navigation when accessing cascading pull-down menus in existing applications. Subsequently, Chapter 3 additionally makes the following contributions:

- Implements facilitated pointing in pull-down menus in real unmodified applications;
- Proposes and evaluates several basic and advanced prediction mechanisms for use with pull-down menus;
- Presents a detailed analysis of existing and the proposed novel mechanisms through trace-driven simulations of interactive sessions.

It is important to note that the system described here does not rely on contrived environments or fanciful pointing hardware, but rather is dependent solely on the existing Microsoft Windows GUI environment. Applications themselves are unmodified, and the overheads of predictive target selection are kept to a minimum, so as not to perceptibly affect the interactive session beyond the intended pointer jumps. Hours-long traces of real user sessions with common applications are used as a basis for comparison of the various predictive mechanisms introduced in subsequent sections. Rather than evaluating the proposed mechanisms through user sessions and subjective user experience studies, the focus is instead on a direct comparison of the mechanisms themselves. Mechanisms are compared according to the distance traveled by the pointer which is the result of the proposed predictive target selec-
tion mechanisms. The minimum traveled distance will subsequently translate into minimum time for menu navigation according to the Acott-Zhai steering law.

1.3 Role of Interactions in Delay Management of Disks

Decreasing energy consumption by decreasing the performance level of a component can significantly increase interactive delays. This is particularly apparent in the case of hard disk drives, where the retrieval of data from a spun-down disk results in a significant delay when platters are spun up to operational speed and during which the system may become unresponsive. Keeping the disk spinning and ready to serve requests eliminates interactive delays, but wastes energy. Stopping or slowing the rotation of disk platters during periods of idleness, i.e. periods during which I/O requests are absent, is the most effective means of reducing the energy consumed by a hard drive. While prior research has focused on predicting the upcoming idle periods in order to place the disk in a lower power mode. Little has been done in predicting the arrival of I/O activity, especially in the arena of interactive user applications, where user-generated I/O requests alone do not generally exhibit discernible patterns.

Timeliness of power mode transitions affects not only the system’s performance and the overall system’s energy consumption but also user perception of the system’s responsiveness. Significant delays are associated with the transition to a higher performance state. For example, waiting for I/O requests to arrive before switching to a higher power level may degrade system performance, keep the system processing the task longer and as result increase overall energy consumption. Switching too early wastes energy, since the demand for high performance is not present. Therefore, timely transitions to the appropriate performance level are critical for achieving both best performance and energy efficiency. Monitoring user behavior provides not only the necessary context of execution that was previously unavailable to the predictors, but enables timely predictions before the need for high performance arrives.

Chapter 4 shows that user interactions can be easily monitored and exploited to
increase both the timeliness and accuracy of prediction mechanisms. More specifically, the Interaction-Aware Spin-up Predictor (IASP) is proposed and applied to reducing the interactive delays of hard disk power management. A set of mechanisms for capturing user actions and predicting the upcoming device state is proposed, and detailed design and implementation is discussed. The proposed mechanisms gather contextual information from user’s mouse interactions within a GUI and use it in predicting an upcoming I/O request. The idea is motivated by the observation that with a majority of common interactive applications, the user fully interacts with the application through its GUI. In this context, a simple action such as opening a file requires a sequence of mouse events. By correlating the sequence of steps to the resulting I/O, future I/O occurrences can be predicted when the user initiates the same set of operations again.

Chapter 4 makes the following contributions:

- Describes the application of interaction-aware prediction to spin up a hard drive (IASP).
- Applies the proposed mechanisms to predicting disk spin-ups in interactive applications.
- Details significant improvements in energy management delays exposed to the user.
- Extends Adaptive Learning Tree (ALT) mechanisms to predict the length of idle periods and spin-up the disk accordingly.

1.4 Role of Interactions in Energy Management

Higher power demand does not necessarily translate into an increase in energy consumption. For instance, hardware in a higher performance state may complete a particular task faster than the same hardware operating in a lower performance state. This reduces the time during which the entire system has to be on. On the
other hand, a particular device may not be required by all tasks and so may be operated in a low performance state without a significant impact on the performance of the executing application.

The challenge in designing efficient energy management mechanisms is to provide a energy/performance schedule that best matches the task at hand to transparently provide energy savings while satisfying the performance demand. Many energy management techniques have been proposed ranging from hardware optimizations all of the way to application transformation. However, most user interactions are still hidden from the existing approaches, which are unable to capture the context necessary for inferring what a user demands. Monitoring user interaction provides not only the necessary context of execution that was previously unavailable to the predictors, but also enables timely predictions before the need for high performance arrives. The timely transition of a device to a desired performance/energy level is critical to meet performance demand and achieve energy efficiency.

Chapter 5 shows that user interactions can be easily exploited to increase both the timeliness and accuracy of prediction mechanisms. More specifically, it shows application of user-interaction-based prediction to reduce energy consumption in Wireless Network Interface Cards (WNICs) while maintaining good performance levels.

Subsequently, Chapter 5 details the design and evaluation of a range of prediction mechanisms that balance accuracy, energy consumption, and delay to provide energy efficient management of the WNIC. Each mechanism incorporates high-level contextual information about user’s activity to predict network access patterns and provide desired energy/performance levels. The idea is motivated by observing that network traffic (for interactive applications) usually follows a specific interaction with the application interface. For instance, if the user is chatting with a friend in a webcam-enabled instant messaging application, when the webcam button is clicked it is reasonable to expect additional network traffic. Therefore, the proposed interaction-aware prediction mechanisms monitor user interactions with the application interface and correlate such interactions with resulting network activity to
predict future levels of I/O demand.

As a result, Chapter 5 makes the following contributions:

- Details low-overhead user-interaction-based predictors (IAP).
- Presents comprehensive evaluations of several proposed prediction mechanisms.
- Presents multiple prediction mechanisms to mitigate variability in user interactions.
- Shows significant improvement of WNIC energy efficiency.
CHAPTER 2

BACKGROUND

Recent research of energy management in consumer computing environments has highlighted the need for more accurate interaction context capture. Hardware components of PCs and mobile devices continue to improve their support for energy management by providing two or more performance levels that can be interactively managed to provide either high performance or lower energy consumption at the cost of lowered performance. Particular attention is paid to components that are the largest consumers of energy in a system, such as the CPU, the hard disk (HDD), and the wireless network interface card (WNIC).

Manufacturers recommend spinning down disks following a period of idleness [13, 28]. The mechanism is simple to implement and provides significant energy savings compared to a disk that is always on. However, wrong shutdowns and spin-ups are costly, both in terms of energy and delay. To improve accuracy, energy management can be relegated to programmer-inserted application hints, [18, 26, 36, 51]. To reduce the burden of manual hint insertion, Program Counter Access Predictor (PCAP) [23] proposes a mechanism for the automatic generation of application hints, however it still lacks higher level hints that monitoring user interactions can provide.

Energy and performance optimizations in WNICs face similar challenges, where the proposed mechanisms attempt to predict periods of high and low bandwidth demand. Self-Tuning Power Management (STPM) [3] explores the use of explicit application-level hints and on-line modeling of application access patterns to set network power management parameters. Automatic hint generation for managing WNICs was proposed [1]. However, the design required offline processing and was much less transparent that one may desire.
2.1 User Activity Monitoring Systems

Accurate and detailed monitoring of user activity is the basis for continuing to improve performance and energy efficiency of computing systems. Most interactive applications are driven by simple point-and-click interactions. All operating systems targeted for consumers offer a Graphical User Interface (GUI) to facilitate uniform interfaces for interactions between users and application. As a result, virtually all interactions can be accomplished through mouse clicks [15]. Users interact with an application to accomplish specific tasks, like opening or saving a file. Many tasks can be accomplished by a single point and click, but other tasks require sequences of interactions. For example, to save a new file, the user commonly clicks on the File menu then selects the Save option and is presented by a directory selection menu, once the filename is entered and the user clicks OK, the file is saved and disk I/O may be requested. It is argued that all GUI interactions resulting in disk I/O activity can be accurately captured and correlated to the activity they initiate, however, it should be noted that this does not always account for all occurring I/O activity.

One key challenge in inferring high-level notions of context is moving from low-level signals (e.g. such as mouse clicks or keyboard events) to meaningful high-
level contextual information such as the user is chatting on the Internet or user is streaming video via webcam. One of the earliest efforts in this area was the Lumiere Project [29]. Forming a foundation for Microsoft’s Office Assistant, the Lumiere project used Bayesian models to reason about the time-varying goals of computer users from observed interactions and queries. More specifically, Lumiere intercepts events and maps them into observation variables in a Bayesian network. This network is then used to provide assistance to end-users. Other promising methods for mapping low-level events to high-level contexts include Hidden Markov Models (HMMs) [41], or Bayesian classifiers [49]. Although, such methods often require specifying the type of activities to be inferred before training.

Another application of higher-level context derived from low-level events is often seen resource and energy management, where user-interactions and application behavior can provide contextual hints necessary for inferring the desirable system state. These hints can be derived from the application’s user interface events [35] or from the applications themselves [20]. However, all of these existing mechanisms do not provide detailed and accurate context. As a result, the focus of this dissertation is on transparent, non-intrusive approaches to gathering the necessary information that can be applied to both resource management and increasing the understanding of user interaction with GUI-based applications.

2.1.1 Automatic User Interaction Capture

User activity can be captured at various levels of detail. The simplest method is to capture the coordinates of mouse clicks relative to the entire display or application window. Alone, the coordinates provide the lowest level of detail, since it is unclear what visual elements the user is interacting with. Figure 2.1 shows the Instant Messenger application with clustering performed with mouse interactions. Observe that several clusters of activity correspond to the application buttons or windows. Users interact with the interface, requesting audio/video streaming, typing and sending messages, or transferring files. For example, cluster 1 in the upper left corner corresponds to file transfer and web cam interactions and cluster 2 in the lower left corner
corresponds to sending messages through clicking in the MSN Messenger text box. The key challenge of inferring these hints is moving from low-level signals (e.g. such as mouse clicks or keyboard events) to meaningful high-level contexts such as the user viewing a streaming video. Differentiating functional clusters of mouse clicks using a K-means clustering technique is a possible approach, and, as illustrated in Figure 2.1, this has the desirable effect of identifying classes of mouse clicks that correspond to mouse-driven functions of the software. Once clustering converges, new mouse clicks can be classified with respect to the identified clusters.

Clustering is necessary when the only available information about user interactions are the relative coordinates of mouse events. K-means clustering is adequate to show the potential of dynamically capturing user behavior. However, it suffers from several shortcomings that are addressed in this dissertation:

• K-means require significant offline processing to compute the clustering. The number of clusters varies from one application to the next, requiring additional processing before mouse clicks can be correlated with network I/O patterns.

• The clustering technique used to detect the layout of the user interface is an approximation. Interactions with nearby interface buttons may introduce unrelated data points to existing clusters resulting in misclassifications. The technique is also sensitive to changing window locations and sizes and, as such, may introduce inaccuracies as the clustering has to be recomputed.

• The implementation considered a single application with a relatively simple interface and a single interactive window. Other applications may have a more complicated interface that not only requires multiple sub-menu selection but also supports interaction between multiple windows.

To address the shortcomings of the coarse grained interaction capture described above, Chapter 3 explores a complete set of low overhead mechanisms that transparently integrate with Graphical User Interface (GUI) and capture user behavior exactly without any uncertainty associated with clustering. As a result, the pro-
posed system achieves a more accurate correlation and prediction. Improved capture accuracy translates to better prediction accuracies for interactive applications.

2.2 Facilitated Pointing

Virtual pointing and the optimization thereof has been the subject of research since it was shown that pointing interaction can be modeled using Fitts’ Law. Unaugmented virtual pointing has been shown to be at least equivalent to physical pointing [37, 16], and assuming that the input device enables performance that is equivalent to physical pointing, reducing D or increasing W in equation (1.1) are possible approaches for further optimization. A survey study of pointing optimizations [4] identifies three distinct categories of optimizing techniques: those that decrease the distance D, e.g. [52], those that increase the width W, e.g. [6], and those that attempt to do both simultaneously. It is also hypothesized that pointing performance enhancements that decrease the distance D ought to focus on the movement phase that covers the largest part of the distance toward the target, which is ultimately the focus of the latter part of Chapter 3.

Efforts to reduce the distance (D) to the target include interface designs that change the visual layout of the menus. Pie menus [4], for example, arrange menu items in a circle around the cursor making all items equidistant. The drag-and-pop approach [5] identifies likely targets for an item that is being dragged and generates virtual proxies of the target icons close to the pointer. A final notable distance-reduction technique is object pointing [25], where the pointer is made to move to the next selectable target, skipping the intermediary pixels not containing selectable objects.

Other attempts at facilitated pointing primarily involve increasing the size (W) of the target. Area cursors allow an activation area that is larger than the single pixel of standard pointer [54]. Target occlusion by an area cursor is resolved by using a point cursor when more than one target is present beneath a target cursor. More recently, expanding targets [21, 38], have formed the bulk of work in the attempts to
increase $W$. Of these, the most prominent are the dynamically sized widgets found in the MacOSX dock, containing icons that expand when the pointer is moved over them.

Menu reorganization is a common feature in applications with complex feature sets. On one hand, menus are reorganized by making particular options unavailable during a specific phase of execution, for example, the *File* menu may omit or contain the inactive option *Close* when the application starts but a file has not yet been opened for editing. Similarly, the *File* menu and the common *Window* menu may contain lists of recently opened files and currently active application windows, respectively.

2.3 Disk Energy Management

High performance hard drives are a significant source of energy consumption and timeout mechanisms have gained wide popularity due to the simplicity of implementation and the energy savings they provide to disks that would otherwise be spinning needlessly. Figure 2.2 shows an example of a timeout mechanism shutting down the device once a timer expires. The disk remains powered down until a new I/O request arrives and the disk has to be powered up before servicing the new request, potentially exposing several seconds of delay to the users.

2.3.1 Shutdown prediction techniques

Interestingly, spinning down the disk is not always beneficial. Accelerating the platters requires more energy than keeping them spinning while the disk is idle.
Therefore, the time during which the device is off has to be long enough to offset the extra energy needed for the shutdown and spin-up sequence. This time is commonly referred to as the *breakeven-time*, and is usually on the order of a few seconds. Eliminating wrong shutdowns that not only waste energy but also significantly delay user requests is critical to conserving energy and reducing interactive delays. Simple timeout-based mechanisms gained wide popularity but they waste energy while waiting for a timeout to expire. As a result, various selections and dynamic adjustments of the timeout value have been proposed [33, 17, 24, 27] to reduce the amount of energy consumed during the timeout period. Consequently, dynamic predictors that shut down the device much earlier than the timeout mechanisms have been proposed to address energy consumption of the timeout period [11, 30, 48]. Stochastic modeling techniques have also been applied to model the idle periods in applications and shut down the disk based on the resulting models [7, 10, 43, 47].

To improve accuracy, energy management can be delegated to programmers, since they have a better idea of what the application, and potentially the users, are doing at a given time [18, 26, 36, 51]. To reduce the burden of hint insertion on the programmer, automatic generation of application hints was proposed [23] to exploit the observation that I/O activity is caused by unique call sites within applications. Finally, operating systems can concurrently evaluate multiple predictors and select the best one for the current workload [50].

2.3.2 Reducing spin-up delays

The goal of shutdown mechanisms powering down the disk is to improve energy savings. However, every shutdown requires a corresponding spin-up to serve future requests. It is important to note that even correct shutdowns can expose spin-up delays to the application or the user as shown in Figure 2.2. There are two approaches for reducing the impact of spin-up delays. First, data can be prefetched and cached either in main memory [34, 44, 12] or an alternate storage device such as flash memory [39, 45], however disk accesses for uncached data will inevitably occur. Second, Chapter 4 proposes waking up the disk early by spinning up the
platters before the request arrives and serving the request without any delays. Both approaches are complementary since the disk will have to be spun-up at some point even if the caching techniques are very efficient.

2.3.3 Predicting spin-up time

Chapter 4 focuses on spin-up prediction, which can be achieved in two ways. First, by predicting the length of the idle period and spin up before the end of the predicted period. Second, by predicting spin-up itself by observing system events, such as user interactions. Prediction of the idle period lengths was previously proposed by Adaptive Learning Tree (ALT) [11]. The ALT approach is to predict the best current power mode based on a sequence of idle periods. Idle periods are discretized according to the time spent idling, and in relation to the number of available sleep states and device specifications. Previously observed states or sequences of states are encoded in a tree, the paths of which are matched according to newly observed sequences of discretized idle periods. Each leaf node in the tree constitutes a prediction and the most likely prediction is selected to transition the disk to the matching power state. ALT has shown significant improvement for power mode prediction in static, non-interactive applications and motivated us to adapt the design to predict the length of idle times and spin-up the disk before the predicted idle time ends.

2.3.4 Challenges in predicting spin-up time

Accurately spinning up the disk is challenging since the idle period has to be predicted very accurately. There are three possible situations that can occur following the prediction. The first and best scenario is when the prediction is accurate and the disk is powered up just before the request arrives. In this situation, there is no energy wasted in waiting for the request to arrive and also the spin-up delay is hidden from the user. The second scenario occurs when the predicted idle period is longer than actual idle period. In this case, the device is powered-up upon I/O arrival and the latency of the spin-up is exposed to the user. The last scenario occurs
when the predicted idle period is much shorter than the actual idle period. In this case, the disk is powered up and subsequently shut down without serving any disk requests. The disk is shut down to prevent it from remaining in the powered-up state for long idle periods. As a result, energy is wasted performing the unnecessary spin-up and shutdown transitions and spin-up delay is exposed when I/O requests do arrive.

Predicting the exact length of idle periods in interactive applications is difficult since it depends on the constantly changing frequency of user interactions with the application. Therefore, Chapter 4 explores the observation and use of user interactions to infer the impending arrival of I/O activity, since users are responsible for the majority of I/O activity in interactive applications. The proposed mechanisms reconstruct the user’s interaction context from mouse events directed at the application’s GUI, thereby providing the necessary hints transparently and without application modification. The captured user context results in high accuracy and prediction timeliness in the proposed IASP.

2.4 Wireless Network Card Energy Management

The challenge in designing an energy efficient system is to minimize energy consumption without sacrificing performance. Therefore, an ideal solution would provide the performance level that closely matches the bandwidth demanded by the application. The IEEE 802.11 standard [31] offers two energy management schedules: Continuous Aware Mode (CAM) and Power Saving Mode (PSM). The default operational mode for wireless network interface cards (WNICs) in portable computers is CAM, where the WNIC is continuously active and responds to a user’s requests immediately. CAM provides highest performance both in terms of lowest delay and highest effective bandwidth at a cost of high energy consumption. Alternatively, PSM periodically wakes up the network interface, switching it to a high power state at some time interval during which transmissions can occur, this is known as beaconing. Once transmissions complete, the network interface goes back to sleep. This signif-
<table>
<thead>
<tr>
<th>State</th>
<th>Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSM Idle</td>
<td>.05</td>
</tr>
<tr>
<td>PSM Receive</td>
<td>.85</td>
</tr>
<tr>
<td>PSM Transmit</td>
<td>1.05</td>
</tr>
<tr>
<td>CAM Idle</td>
<td>.73</td>
</tr>
<tr>
<td>CAM Receive</td>
<td>.91</td>
</tr>
<tr>
<td>CAM Transmit</td>
<td>1.16</td>
</tr>
<tr>
<td>PSM to CAM Switch</td>
<td>.78</td>
</tr>
<tr>
<td>CAM to PSM Switch</td>
<td>.67</td>
</tr>
<tr>
<td>Transition</td>
<td>Time (s)</td>
</tr>
<tr>
<td>PSM to CAM</td>
<td>.28</td>
</tr>
<tr>
<td>CAM to PSM</td>
<td>.28</td>
</tr>
</tbody>
</table>

Table 2.1: WNIC energy consumption specifications.

significantly reduces power consumption [32], but also increases delays and degrades the performance of applications that demand high bandwidth [3].

The delays in network transmissions due to PSM keep the rest of the system operating for longer periods, resulting in higher overall energy consumption in the system presenting a clear need for an adaptive system. Authors of STPM [3] illustrated PSM’s delay problem with the pathological case of NFS directory listings, which were 16-32x slower than when CAM was the active transmission mode. In the case of NFS, non-concurrent RPCs are used for communicating the directory listing, with two RPCs for each file in the directory. With PSM, each RPC response is delayed until the next beacon occurs. Beaconing typically occurs every 100ms, resulting in a cumulative delay that grows with the number of entries in the directory listing. The delay cost of PSM’s energy-efficient operation is the primary motivation for the proposed low-overhead, adaptive mechanisms.

Table 2.1 illustrates the energy characteristics of the Lucent Orinoco WaveLan Silver WNIC used in this study. Observe that PSM has much lower energy consumption than CAM in the idle state. Therefore, when the application does not transmit or receive any I/O or the bandwidth demand is low the WNIC should remain in the PSM mode. Systems with applications showing high bandwidth demand should keep the WNIC in CAM, since transmission power of CAM and PSM are comparable
and CAM offers much higher bandwidth. Furthermore, the delays associated with PSM can extend the running time of the application and result in higher energy consumption than CAM. However, selecting the WNIC mode is not simple, since an application may go through multiple states with varying bandwidth demand. Keeping the WNIC in a single state either wastes energy, as is the case with CAM, or increases transmission delays and potentially increases energy consumption, as in the case of PSM.

To accommodate different application behaviors, many cards provide automatic switching between CAM and PSM, based on the amount of traffic observed. The WaveLan card allows the network interface to remain in the PSM while monitoring for traffic from the access point. If more than one packet is waiting at the network interface, the WNIC is switched to CAM. The WNIC is switched to PSM once the transmission stops. This simple solution provides energy savings for low transmission rates while providing good performance for applications demanding high bandwidth. However, certain cases render this solution ineffective. First, communication patterns that exchange one packet at a time will keep the WNIC in PSM even if the applications require high bandwidth [3]. The pathological case for PSM occurs with NFS directory listings, which can result in a 16-32x slowdown as compared to CAM [3]. Second, small and bursty transmissions may unnecessarily switch the WNIC into CAM. Therefore, more sophisticated mechanisms are needed to better adapt WNIC power mode switching to reflect an application’s performance demand.

To match application demand while saving energy, the network card should be transitioned into CAM when high bandwidth is demanded and remain in PSM when the bandwidth demand is low. This observation motivated the development of Self-Tuning Power Management (STPM) [3, 2] to dynamically switch between CAM and PSM. STPM relies on programmers for inserting accurate hints about upcoming bandwidth demand into the application. As a result, STPM can be very accurate, as the programmer knows what part of the applications is executing and perhaps what kind of demand can be expected from the user. However, this
introduces power management as another optimization dimension to already difficult programming requirements that target, among other things, performance, reliability, and usability. To provide energy savings for unmodified applications, STPM profiles network traffic to anticipate future network activity [3]. Passively monitoring low-level network access provides little information about user’s current context, how the user is interacting with the application. In this case, the context of execution of an application is completely lost and heuristic profiling is not able to fully realize the potential of STPM.

Context of execution is critical to providing accurate and timely switching between power states to match the application demand. Figure 2.3 shows an example of user interactions with an application and the impact of STPM and user Interaction-Aware Prediction (IAP) mechanisms on transition timeliness. As a result of user interactions the application initiates network I/O activity. STPM evaluates initial I/O requests to verify the need for higher bandwidth and the network interface is switched to the high performance mode when the need for high bandwidth is detected. It is observed that both the evaluation period and the mode transition period impact the performance of high bandwidth transfers and may lengthen the time spent serving the I/O requests [3].

Observe that hints are needed ahead of time in order to transition a device to a higher energy state before I/O requests arrive. These hints are difficult to obtain.
at the operating system level, so the need for a higher context of execution is clear. Since users are responsible for the majority of I/O activity in interactive applications, the natural approach is to observe user interactions and infer from interactions when the increase in performance demand will arrive. Therefore, Chapter 5 examines several IAP mechanisms and explores in detail the monitoring and prediction of user activity in order to improve prediction timeliness and accuracy due to the added context of execution. The IAP mechanisms continuously monitor user activity and predict the need to transition the WNIC device to a higher power level in time to meet the increased performance demand, as shown in Figure 2.3.
CHAPTER 3

CAPTURING INTERACTIVE CONTEXT

The key to the design of the Transparent User Context Monitoring (TUCM) system is that the graphics management subsystem in modern operating systems can be leveraged to transparently monitor, record, and utilize user interactions to improve system performance or energy efficiency. Subsequently, TUCM design faces several requirements:

• User interactions have to be captured transparently without any modification to the application.

• Monitoring and capture should be efficient to prevent excessive energy consumed by the CPU, which would reduce system energy efficiency.

• The system should handle multiple applications in a graphically rich environment.

• User behavior correlation and classifications should be performed on-line and without direct user involvement.

All of those items have to be successfully addressed to provide efficient user context monitoring and management that can be applied for energy or performance management in the emerging computer systems.

3.1 Context Capture in Linux

Recent interface design trends emphasize simplicity, where designers break complex tasks into simpler ones that are represented by autonomous interface components (e.g. icons, menus, and buttons). Each interface component exposes a limited set of functions to the end-user. The user invokes these functions by interacting with
different interface objects using input devices (e.g. mouse, keyboard) which in turn trigger a series of actions. GUI toolkits and widget libraries provide programmers with abstracted building blocks that hide the specifics of physical interaction devices, manage the details of interaction for the application by abstracting it to callbacks, and provide encapsulation for the GUI appearance and behavior [14].

On Unix-like systems, the X-Window System provides a common display protocol built on the client-server model [53]. The X Server is responsible for accepting graphical output requests from clients and reporting user input to clients. The stream of data from the client to the server contains low-level information regarding the window layout, such as window size and parent and children windows, while the data sent from the server to the client applications contains information about user interactions, such as mouse button events, the windows they occurred in, and relative coordinates. Section 3.1.1 addresses the implementation of context capture within the X Window System in a Linux environment. Section 3.2 details an alternative implementation of context capture within Microsoft Windows and presents an immediate application for the HCI domain.

3.1.1 X Window System

The basic architecture of an X application is shown in Figure 3.1. As shown, the application defines the widgets that are to be used to make up the user interface. A widget is an interactive interface element, such as a window, text box, button, etc. The application is also responsible for receiving and processing user input via
callbacks. The user can only interact with the application by use of the graphical output drawn by the widgets and input device events, such as mouse button presses, processed by the widgets.

Additionally, the basic architecture of the X Window System is shown in Figure 3.2. For now, consider the Xlib layer, which is an X protocol client library containing functions for interacting with an X Server. Applications may interact directly with this abstraction layer, but generally they employ widget toolkit libraries which then interface with Xlib.

Xlib itself provides five types of functions: connection operations (XOpenDisplay, XCloseDisplay, etc.), server requests for operations (XCreateWindow, etc.), server requests for information (XGetWindowProperty, etc.), operations on the event queue (XNextEvent, XPeekEvent, etc.), operations on local data (XCreateImage, XSetRegion, etc.). The most important aspect of the relationship between Xlib and the X Server is that objects such as windows and color maps are managed entirely by the server. The client cannot directly access these objects, but must instead request that the server perform an operation on them using identifiers. An object identifier is unique in an X Server and may be used by multiple clients to refer to the same object. Since no displayed object is directly manipulated by the client, the client-server communication stream must therefore contain any and all manipulations that the user requests through interaction with the on-screen elements.
By adding an intermediary layer, as shown in Figure 3.3, between the server and attached clients (more specifically, the Xlib-driven protocol), the exact sequence of requests and events is observed. This layer provides a means of transparently monitoring user behavior. No modification of applications is necessary. Furthermore, user interactions are captured exactly, eliminating both the excessive computational overhead of computing a clustering and the inaccuracies associated with the clustering present in the previously described solution that relied on K-means clustering. Since the need for cluster formation and behavior detection is eliminated, the offline processing needed in the clustering approach is eliminated, fully allowing for detection, correlation, and prediction to be performed online.

3.1.2 Listening with xmond

In order to accomplish the basic tasks of monitoring the protocol traffic, the intermediary layer is comprised of a modified xmon, an interactive X protocol monitor. Xmon recognizes all core X protocol requests, events, errors, and replies that comprise the client-server communication. As shown by Figure 3.3, in order to successfully monitor the protocol stream, clients now attach to xmon instead of the X Server directly. However, as it is placed transparently between the clients and the server, the clients see it simply as an X Server, and the X Server sees xmon as a number of clients. The basic xmon monitoring functionality is extended and modified to transparently provide contextual hints about the user’s activity.
3.1.3 Logical mapping of user interfaces

The X Server generates unique identifiers for each window a client application creates, regardless of its visibility. These identifiers are unique to a particular window for the duration of the application’s execution, but may or may not correspond to the identifiers generated in future invocations of the same application. Since these identifiers are not persistent, instead a logical representation of the windowed interface is used.

A unique ID of each interactive region is computed using a logical representation of the windows and their respective components. The generated IDs are persistent across application executions, so the interactions in future invocations of the application will have the same logical representation allowing for persistent contextual information.

The feature that allows us to uniquely identify the windows across multiple executions of the application is the order of window creation. Each time an application is started and its GUI about to be displayed, all of the sub-windows comprising the displayed interface are created in the same order and arranged in a tree, as shown in Figure 3.4. A sub-window’s location within the tree uniquely identifies the window in which the interaction is occurring and the location is persistent across multiple executions of the application.

A more concrete example is shown in Figure 3.5. This is an actual interactive window from the memprof application, seen when the Run button is pressed. On
Figure 3.5: A simple interactive window with a relatively flat layout.

The left, see the exploded view of the component windows contained in the interface presented to the user. As in Figure 3.4, the visible interface is composed of a hierarchy of windows. The window containing the title bar, and comprising the members of the first level of the tree, L1, as shown on the left in Figure 3.5, is the parent of the window containing the application’s GUI, L2. The GUI for this particular window is composed of 5 interaction elements: the text input box, the expand dropdown button, browse, cancel, and execute. All of the interactive elements are children of the window in L2, so they comprise the members of L3 in the window hierarchy. In this interface’s case, each interaction element is contained within its own window. This section provides a mechanism for labeling each interaction element uniquely across different executions of an application. On the right-hand side, see the interface that the user is presented with. IDs 1 through 4 are placeholders for the actual IDs generated by TUCM. The actual IDs do not explicitly state which element has been interacted with, for example TUCM output does not state that the user has clicked on the *Execute* button. The actual IDs are integers that can be used to correlate system behavior to user interaction.

There are two possible approaches for computing the logical sub-window IDs. In the first approach, the window tree may be constructed when the application window is constructed, and retrieve the logical ID from the tree upon each click. However, this approach introduces the overhead of storing the tree-based representation of each new window. Additional computational overhead is incurred upon subsequent
invocations of previously seen applications, since the newly constructed window
trees must be compared with all other stored window trees in order to find the one
matching the logical layout of the current application and update the tree’s node
IDs with the newly generated window IDs from the X Server.

Alternatively, an ID can be computed by walking the tree and assigning node
IDs recursively upon visiting each node in the tree. Since the application interface’s
logical structure is persistent across executions, tree traversal is performed in the
same order each time, and each node is visited only once per traversal, window IDs
are likewise unique and persistent across executions. The resulting IDs can identify
not only the windows, but also the interactions with multiple elements within each
window.

To minimize the storage and computational overheads, the alternative approach
is favored here, dynamically calculating a unique and persistent ID for each event.
The example in Figure 3.4 describes the approach. When a mouse click occurs
inside of the visible element 6, an event for that window is generated by the server
and passed to the client. After capturing the window ID, in this case 6, the event
information is passed to the client which executes the action associated with the
event. Meanwhile, the X Monitor uses the captured window ID to query the X
Server for additional information regarding this particular window ID. Traversal of
the example tree yields the sequence shown in Figure 3.4.

A basic non-recursive version of the window-tree labeling algorithm starts when
the client is informed of a user interaction by the X protocol event ButtonPressEvent,
captured by the X Monitor, which also provides the current internal ID of the
window within which the interaction occurred. Using this temporary ID and calls
to XQueryTree, which returns the parent and child nodes of the input tree node, the
root window can quickly be found, which is the ancestor of every displayed window.
Reaching the root window is the stop condition of the algorithm. Starting at the
interaction window, progressively move up the tree, using the child number and tree
level number to incrementally calculate the unique window ID. This approach is
adequate when GUI construction is simple and each interactive element, i.e. button
or text box, is contained in its own window.

Creation of application GUIs is aided by widget toolkits that may manage multiple buttons within a single window. This complicates the approach since each window may contain more than one interaction region. Using the simple method of assigning unique ID to windows results in all buttons within a single window having the same ID, which would prevent any meaningful correlation between user activity and the I/O behavior. Therefore, it is necessary to differentiate separate regions within a single sub-window to infer interface features. To do so, additional information exchanged between X Server and the clients is obtained. More specifically, moving the mouse pointer over a button causes the button to react in some way, signaling the user that a click here would cause the associated action to occur. The visual cue is accomplished by redrawing the area occupied by the button with a new image similar to the original, but distinct in appearance from the inactive buttons. The actions associated with the image swap can be seen in the exchange of messages between the client application and the X Server. The most relevant of these is the `CopyArea` graphics request from the application, which includes, among other things, the relative coordinates and size of the bitmapped image that becomes the button a user sees. The request is passed by the application and includes the coordinates of the area to be drawn relative to the window that is to contain the image as well as the size of the image. While an element’s size may be the same as the other elements contained in the window, the placement coordinates of the drawn area are differ. By adding the newly drawn area’s x and y coordinates, which are relative to the sub-window in which the graphic is displayed to the ID generated during tree traversal, a new ID is generated that is unique to the interactive element contained in the window.

3.1.4 Evaluating Overheads of Context Capture

The implementation of the TUCM has two flavors, depending on system restrictions or desired run-time performance. The mechanism can be run to achieve either low storage overheads or to reduce the computational overhead. The computation of the
unique IDs is achieved in part with repeated calls to \textit{XQueryTree} which reveals part of the window tree structure of the application in which the interaction occurs. The overhead of making this function call is proportional to the depth of the window tree. Consider the straightforward approach of capturing window IDs. Parents and children of currently examined window are calling \textit{XQueryTree}. However, with this approach, there is no storage overhead, at the cost of the already small overhead of computing the unique ID for every mouse event. This approach is therefore named NSO (for No Storage Overhead).

The alternative, low computation overhead (LCO), approach is to make the tree traversal exhaustive and do so only once, recording the structure of the application’s window tree upon first interaction, to be reused upon each subsequent interaction. In this approach, the initial overhead far exceeds both the overhead of subsequent ID calculations on the same tree and ID calculation by NSO. However, ID calculation for interactions other than the first are much less costly than NSO, which does not change regardless of the number of encountered interactions with the same window, and thus the same window tree. The initial overhead of LCO consists of building the window tree through calls to \textit{XQueryTree} and labeling the nodes of the tree. The storage overhead is encountered when storing the application’s window trees. Ideally, the window tree is built only once, the first time an application is encountered. Upon building the window tree, a comparison is made with all other stored trees so that the same tree is not stored several times. The overhead of comparing the structure of the stored trees with a newly generated one can be mitigated by additionally storing a string representation of the structure of each tree and using the string for comparisons rather than the entire tree.

In order to evaluate the efficacy of context capture, the user’s interaction with several applications was traced. The captured traces contain simply the type of mouse event which occurred, timestamp, and the corresponding unique ID. The applications considered are commonly used in the Linux environment, they include: \textit{Firefox}, \textit{Thunderbird}, \textit{Gaim}, \textit{GFTP}, \textit{Pan}, \textit{Dia}, and \textit{Gimp}.

- \textit{Firefox} is a widely used web browser. Through its GUI, the users may navigate
the web, reading website content, downloading files, etc. Some page browsing activities invoke external plug-ins to access additional content. Except for entering a target URL, the user’s interaction is generally confined to the mouse.

- Thunderbird is an email application where the user interacts with the application in order to send, receive, read, and compose email messages. With the exception of entering the text comprising an outgoing email, the application provides all intended functionality through a mouse-interactive GUI.

- Gaim is an Internet messaging application used with a variety of messaging protocols. The user’s actions are simply sending and receiving of messages and files. Again, the interaction is fully mouse driven, save for the entry of outgoing text. In the traced version of this application, the top-level interactive elements allow the user to click to send the message, and provide buttons for sending files and changing the format of the message text fonts.

- GFTP is a file transfer client used to upload and download files to and from an FTP server. Once the server address is input, all interaction with this application occurs through mouse events. The visible interface consists of server information text input fields, connect/disconnect buttons, directory listings in dropdown menus and navigable selection boxes, and buttons to initiate a client-to-server and server-to-client transfer.

- Pan is a newsreader application, where the user activity mainly consists of navigating newsgroups, browsing post headers, reading posted messages and downloading files, and posting messages. The interface is relatively complex, but allows a significant amount of functionality to be accessed through top-level interaction. For example, a message may be posted to a newsgroup, or it may be posted as a follow-up, or new headers can be pulled from subscribed or selected newsgroups, and selected articles can be downloaded. This abundance of interactive elements allows for fully mouse-driven interaction, except for message composition.
<table>
<thead>
<tr>
<th>Application</th>
<th>Total Clicks</th>
<th>Unique Click IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox</td>
<td>3113</td>
<td>128</td>
</tr>
<tr>
<td>Thunderbird</td>
<td>2175</td>
<td>32</td>
</tr>
<tr>
<td>Gaim</td>
<td>2992</td>
<td>59</td>
</tr>
<tr>
<td>GFTP</td>
<td>2253</td>
<td>34</td>
</tr>
<tr>
<td>Pan</td>
<td>7316</td>
<td>50</td>
</tr>
<tr>
<td>Dia</td>
<td>46864</td>
<td>118</td>
</tr>
<tr>
<td>Gimp</td>
<td>8465</td>
<td>157</td>
</tr>
</tbody>
</table>

Table 3.1: Application trace details.

- *Dia* is an application used for drawing various types of diagrams. Both the selection of diagram elements and creation of diagram layout are top-level interactions. Keyboard input is only necessary when using the text input tool.

- *Gimp* is an image manipulation program used to create and edit graphics, resize/crop photos, alter an image’s appearance and colors through various effects and filters, and to convert between various image formats. Due to the nature of the intended functionality of this application, the user seldom has to resort to keyboard input.

**Trace Details**

Each interactive application described above was traced under common usage scenarios in order to capture a realistic variety of interactions. Details of traces are shown in Table 5.2. The table shows that despite a large number of mouse button clicks, as in the case of *Impress* and *Dia* where the number of clicks respectively exceed 25000 and 46000, the number of unique click IDs is relatively low. No application shows an excess of 200 unique click IDs. It should be noted that the unique click IDs shown in the table account for all types of interactions such as drop-down menus, pop-up dialog boxes, and external plug-in interactions. The *Firefox* example shows a large number of unique click IDs as compared to the number of interactive elements. The reason for this is the nature of the application, where interactions
Table 3.2: Storage and computational overheads of NSO and LCO.

<table>
<thead>
<tr>
<th>Application</th>
<th>Number of Tree Nodes</th>
<th>Max. Tree Depth</th>
<th>Per-Click Overhead NSO (ms)</th>
<th>Per-Click Overhead LCO (ms)</th>
<th>Initial Overhead LCO (ms)</th>
<th>Storage Overhead LCO (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox</td>
<td>25</td>
<td>19</td>
<td>.3</td>
<td>.005</td>
<td>11</td>
<td>300</td>
</tr>
<tr>
<td>Thunderbird</td>
<td>39</td>
<td>14</td>
<td>.29</td>
<td>.007</td>
<td>14.9</td>
<td>468</td>
</tr>
<tr>
<td>Gaim</td>
<td>8</td>
<td>3</td>
<td>.1</td>
<td>.004</td>
<td>2.5</td>
<td>96</td>
</tr>
<tr>
<td>GFTP</td>
<td>88</td>
<td>7</td>
<td>.15</td>
<td>.01</td>
<td>22.3</td>
<td>1056</td>
</tr>
<tr>
<td>Pan</td>
<td>82</td>
<td>4</td>
<td>.12</td>
<td>.008</td>
<td>19.3</td>
<td>984</td>
</tr>
<tr>
<td>Dia</td>
<td>31</td>
<td>5</td>
<td>.12</td>
<td>.006</td>
<td>8</td>
<td>372</td>
</tr>
<tr>
<td>Gimp</td>
<td>34</td>
<td>3</td>
<td>.14</td>
<td>.005</td>
<td>10.1</td>
<td>408</td>
</tr>
</tbody>
</table>

are not confined to the application GUI itself, but usually extend to the content of the window containing the loaded web page. When the loaded web page contains additional elements requiring the use of external plug-ins for displaying the content, the structure of the window tree will change to accommodate the addition of these elements resulting in new unique click IDs for interactions within the content window.

**Overheads**

Table 3.2 shows a summary of overheads for both NSO and LCO, both in terms of computation and storage as gathered from several X Monitor runs on a 2.6GHz Pentium 4 laptop. Regarding computational overhead, NSO incurs no initial computational overhead, but rather exhibits higher per-click overheads than LCO, due to the need to recompute the unique IDs as each click occurs. Disregarding the initial computational overhead of LCO, the average of LCO’s unique ID retrieval takes 97% less time than NSO’s ID computation. However, the initial overhead of LCO is not present in NSO, and in this case, the average NSO ID computation time is 99% lower than LCO’s initial overhead.

The storage overhead of LCO is also shown in Table 3.2. While NSO incurs no storage overhead, the overhead of storing the trees in LCO is proportional to the number of nodes in each application’s tree. Storing a pointer to a node’s parent, pointers to its children and the unique unsigned integer ID will require at least the storage shown. Additional optimizations, such as storing the string representation
of each tree for reducing tree storage overheads by tree comparison and reuse means that additional storage would be necessary. In any case, the overheads of LCO and NSO are far below the overheads of a clustering approach, where data about each click must be stored, and clustering periodically recomputed.

Figures 3.6 and 3.7 show the implications of LCO and NSO mechanisms’ computational overheads. Figure 3.6 compares the initial overheads of the two mechanisms. Each tick represents the addition of a tree level and the number of nodes shown on the x-axis. For NSO, the addition of tree levels results in a linear increase of computational overheads, while LCO’s overhead increases exponentially as the size of the tree depth and number of nodes increases. In order to label each interaction window, NSO makes a number of calls to $XQueryTree$ that is equivalent to the interaction node’s level in the tree. On the other hand, the overhead of LCO’s labeling of all the nodes in the tree grows both with the number of nodes contained in the tree and the depth of the tree, since to label a leaf node, LCO must visit it’s ancestors. The computational overhead is due to the call being made to query X Server’s internal window data. Server-side queuing and processing takes up the majority of the time.

Figure 3.6: The initial overheads of NSO and LCO as the size of the tree is increased. Starting with two nodes, each x-axis tick represents an additional tree level as well as the shown number of additional nodes.
Figure 3.7: Setting the tree size at 8 levels and 510 nodes, the graph shows the computation time overhead as mouse clicks arrive and the mechanisms compute window IDs. Eventually, the cumulative overhead of computing IDs with NSO exceeds the cumulative overhead of LCO.

However, as shown in Figure 3.7, the low-overhead ID lookups in LCO result in NSO’s computational overhead eventually increasing beyond the cumulative overhead of LCO. In this case, the tree size is fixed to 8 levels and 510 nodes and show the cumulative delay associated with each mechanism in milliseconds as the number of incoming clicks that have occurred is increased, and therefore the number of times an ID has to be computed or looked up is increased. The cumulative overhead of computing IDs with NSO eventually exceeds the overhead of LCO, since the tree size is fixed, with each new observed mouse click LCO performs and ID lookup with an overhead that is orders of magnitude lower than the ID computation in NSO.

Figure 3.8 shows the expected overhead of each mechanism when it is assumed that 1000 mouse clicks have been observed. With a sufficiently large tree, depth of 13 and over 12000 nodes, NSO’s cumulative delay will eventually be surpassed by LCO, due to the complexity and overhead of initially labeling the full window tree. However, trees of this size have not been observed in any of the traced applications,
so this situation is not expected to occur.

Therefore, precomputing the window IDs results in low computational overhead for LCO, at the cost of very modest storage overhead. Eliminating the storage overhead, on the other hand, introduces more delays, as the IDs are computed at every click, rather than merely looked up. In any case, the perceived responsiveness of interactive applications is unaffected by both NSO and LCO. To a user, interactive response times between 50-100ms are acceptable [46]. Under realistic conditions, both mechanisms deliver window IDs with delays far below the 50ms low end and can therefore be considered unobtrusive and transparent to the user.

3.1.5 Aliasing

Aliasing is defined here as the generation of equivalent IDs from different applications. When considering a multiprogramming environment, the concurrent execution of multiple interactive applications is inevitable. Similarly, interactive applications often rely on dropdown menus and dialog boxes to capture input. When these
additional input windows are displayed, new window trees are created, resulting in
the generation of additional IDs. Auxiliary input windows created by the application
generally have simpler structures than the primary application display. Aliasing is
more likely in the cases where GUIs with simple layouts, and therefore simple trees,
are more common than complex interfaces. As shown in Table 3.2, most applications
used in this evaluation have shallow trees with relatively few nodes. However, as
described in the design, additional information, such as graphical element location
and size within the interactive windows is used alongside window IDs to produce
the final ID, significantly lowering the opportunity for aliasing. In experiments,
aliasing amongst the windows of a single application did not occur. Furthermore,
any possibility of inter-application ID aliasing can be eliminated by the additional
collection of application-related data.

3.2 Application of Context Capture in Microsoft Windows

Since their origins in the early 1980’s, Graphical User Interfaces (GUIs) have be-
come the de facto standard for interfacing with applications. Direct user-directed
manipulation of the now familiar graphical elements is arguably one of the most
intuitive means of interacting with the computing environment. In modern GUIs,
the cascading pull-down menus have become a common interface element for many
graphical interactive applications. Selecting a menu item consists of the user per-
forming the following steps: reading the selection options, choosing the option that is
to be selected, making the selection, and ascertaining the consequences [40]. Making
a selection requires the user to physically navigate through the menu via an input
device such as a mouse. Optimizing menu navigation is the focus of this section.

3.2.1 Predictive Target Selection in Pull-Down Menus

The key observation here is that the common organization of drop-down menus
by functional categories lends itself to prediction. For example, the *File* menu
is generally present in interactive applications and contains the items relevant to
opening, closing, and saving files, printing, and quitting the application. While this menu may contain various other items, from experience it can be intuited that the user’s interaction will generally be constrained to opening and saving a file. Quitting the application is performed once for each execution of an application, and for most users it is likely an item that is less frequently accessed than, for example, saving a file. Similarly, the selection of one item may be dependent on the prior selection of another item. For example, when a file-editing application is started but no file is loaded when the application starts, the Close File item in the File menu will not be available, but is a likely candidate for selection once the Open File item has been selected. To provide the optimizations in the mentioned examples, context of interactions must first be captured and prediction mechanism constructed that utilizes the captured context to optimize menu navigation.

3.2.2 Capturing Context in Windows

The approach of Predictive Target Selection (PTS) that is favored here involves existing unmodified interactive applications in the Microsoft Windows environment. The choice of windowing environment is a result of the authors’ familiarity with the MS Active Accessibility (MSAA) API. Similarly featured mechanisms can easily be created in either the Linux or OS X windowing environments where similar accessibility frameworks exist enabling the use of screen readers and other assistive technologies. Accessibility APIs are useful here since they are intended for use with assistive technology products that require interaction with the user interface elements of an application. As a result, assistive technologies are able to access, identify, and manipulate the user interface elements whose information is provided by the MSAA servers, where the MSAA servers are the applications themselves.

Applications expose the tree structure of the user interface through the IAccessible COM interface. Each element in the tree contains a set of properties that can be accessed and used to identify the element. For our purposes, it will suffice to identify the elements that have the role of menuitem, which identifies them as items in drop-down menus. Pointers to the parent and children nodes in
the same tree allow the PTS system to navigate through the user interface tree to collect information about the items contained in a menu before the user begins the menu navigation and item selection. The path in the tree leading to the selectable item is used to identify each item, starting with the root node, the node containing the all-encompassing application window, and ending with the selectable item node, a leaf in the tree unless the item leads to the expansion of a sub-menu. The information that is necessarily stored by the predictor includes the size and location of each selectable menu item, and the full item name (including the names of the containing menu and application).

Each time a menu is accessed, meaning that the user has selected the menu heading from the toolbar and clicked on it, a prediction is made regarding the next item to be accessed, however, a menu may be configured with several sets of selectable items depending on the application state or context. Therefore, it is necessary to traverse the child nodes (items) of a menu whenever it is accessed and determine each item’s state before a prediction can be made with respect to the menu’s state.

The actual distance to the target is measured with the use of a windows mouse hook. Once the hook is set, mouse events, such as movement and button clicks, can be captured. Actual distance measurements contained in the unmodified application usage traces form the empirical basis of comparison for the menu traversal simulated by the proposed mechanisms. The windows mouse hook enables the monitoring of
Figure 3.10: Example of two drop-down menus.

clicks and the distance traveled between each click. A separate module, informed by the data captured by the dynamically linked library containing the hook and requisite interfaces, tracks the cursor location with respect to the interface elements revealed by the MSAA API. Figure 3.9 shows the organization of these elements in the PTS system architecture. The monitoring and prediction module also computes the location of the likely next accessed item and communicates it to the mouse hook DLL, which then accordingly sets the new mouse position. The evaluation of the subjective user response and effect of immediate mouse cursor position changes may be pursued in future work, but it is anticipated that coupling the mouse position setting with an indication of the new mouse position through visual cues (e.g. momentarily enlarging the cursor) should provide a measure of perceived continuity, minimizing the impact of PTS on the user’s interactive experience.

3.2.3 Prediction Mechanisms

In order to comprehensively evaluate the effectiveness of PTS, several prediction mechanisms are proposed here that can be utilized for prediction and subsequent cursor jump to the predicted item.

**Simple Prediction** The simplest mechanisms for moving the mouse pointer require only monitoring of the application context. First proposed is the move-to-middle (MtM) mechanism. The MtM mechanism computes the height of the drop-down menu, when the menu is expanded following a click on the corresponding
toolbar item, and moves the pointer to the center of the item in the vertical center of the menu. The MtM mechanism does not require any prior information, resulting in lowest storage overheads. Furthermore, assuming that the items within a menu are all visited with similar frequency, the MtM heuristic should reduce cursor movement by half. However, uniform accesses are rare and users are using certain items in the menus more frequently dictated by some temporal user tasks.

The move-to-last-item (MtL) mechanism is proposed here to address some temporal menu-access behaviors that users may exhibit. The MtL mechanism records the last visited item for each menu in a given application in a prediction table. Each time a specific menu is visited, MtL looks up the last visited item in the prediction table using the given menu and the pointer is moved to that item. When the application is opened for the first time, or the menu has not been visited before some training is required since the MtL mechanisms will not have any prior knowledge. In that case, no prediction is generated and the pointer is not moved. Alternatively, the stateless MtM can be utilized and the pointer moved to the menu center. However, poor performance of MtM (as shown by our experiments), reasonably quick training, and prediction table reuse in the future application execution make MtM undesirable as a backup predictor. Small size of the prediction table allows it to be kept in the memory between the application executions, and almost eliminates the impact of training on prediction. Subsequently, all remaining predictors maintain prediction tables between application executions and do not make any predictions or move the mouse pointer during training.

**Frequency Prediction** In some scenarios, the popularity of some menu item may be better described by the access frequency than temporal access information, hence the proposed frequency-based predictor (FP). FP records the number of visits to the menu items for each menu in the prediction table. Once FP detects an interaction with a given menu, it looks up the most often visited item for the given menu and moves the mouse pointer to the predicted item. In the cases where two or more items are just as likely to be selected, the jump is made to the mean of the heights
of the item’s centers within the menu to minimize the cursor movement that is to follow. The drawback of FP is that prior selection frequency alone is not always the best indicator of the user’s future intent, e.g. when in menus contain multiple frequently-accessed items, or when the next selection is an item depending on the current sequence of user interactions.

**History-Based Prediction**  To improve upon the FP and MtL, we aggregate a larger temporal menu access context and propose two history-based predictors. The first one collects item-level selection histories locally for each menu (HPl). The second one considers global histories, across all menus, for each interaction with the application (HPg). The histories that are collected are essentially lists of observed item selections. Given a particular history of items selected within a particular menu, HPl uses the current history of interactions to lookup the prediction table and the menu item that followed a given history in the past is predicted as an upcoming interaction. In case of misprediction the prediction table is updated with the current interaction. In certain interaction, menu selection may depend on user interacting with other menus in the application and therefore global information in HPg may prove to be a better indication of current context of execution and subsequently result in more accurate predictions.

To clarify the operation of HPl and HPg, consider the two drop down menus

<table>
<thead>
<tr>
<th>HPl</th>
<th>HPg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq.</td>
<td>Menu</td>
</tr>
<tr>
<td>a1, a2</td>
<td>B</td>
</tr>
<tr>
<td>b1, b2</td>
<td>A</td>
</tr>
<tr>
<td>a2, b1</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>B</td>
</tr>
<tr>
<td>b1, a3</td>
<td>B</td>
</tr>
<tr>
<td>b1, b2</td>
<td>B</td>
</tr>
<tr>
<td>a3, b2</td>
<td>B</td>
</tr>
</tbody>
</table>

Figure 3.11: Comparison of history-based prediction mechanisms’ prediction hash table states, using history window length 2, and following the interaction sequence {a1, a2, b1, a3, b2, b3}. Light gray shading indicates the predicted value.
Figure 3.12: Comparison of probabilistic prediction mechanisms’ prediction hash table states, using history window length 2, and following the interaction sequence \{a1,a2,a1,a2,a1,a2,b1,a3\}. Light gray shading indicates the predicted value, while no shading in the predicted value column indicates that either the item will not be predicted or that the prediction is ambiguous.

Table: PPL vs PPg

<table>
<thead>
<tr>
<th>Seq.</th>
<th>Pred.</th>
<th>Cnt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1, a2</td>
<td>a1</td>
<td>2</td>
</tr>
<tr>
<td>a3</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Seq.</th>
<th>Menu</th>
<th>Pred.</th>
<th>Cnt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1, a2</td>
<td>A</td>
<td>a1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>b1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>a3</td>
<td>1</td>
</tr>
<tr>
<td>a2, a1</td>
<td>A</td>
<td>a2</td>
<td>1</td>
</tr>
<tr>
<td>a2, b1</td>
<td>B</td>
<td>a3</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 3.11 shows the state of the two mechanisms’ prediction tables following this interaction sequence: \{a1, a2, b1, a3, b2, b3\}. Using Figure 3.10 as a reference, menu A contains the items \{a1, a2, a3\}, while menu B contains the items \{b1, b2, b3\}. In this example, each table entry contains the history of two interactions and the predicted menu item. Note that, following interaction b3, HPg records a significantly richer history of interactions than HPl, and HPl contains only a small subset of HPg’s table. Additionally, HPg tracks alternate predictions on a per-menu basis, allowing, for example, \{b1\} to be predicted when sequence \{a1, a2\} is observed prior to the user opening menu B, instead of A.

**Probabilistic Prediction** Further improved prediction mechanisms proposed here are ones that take advantage of history-driven and probabilistic decision making. As the history-based predictors, the probabilistic predictors utilize history lists. However, each history list here is preserved in order to determine the likeliest candidate for the user’s next action out of a set of like histories. Each history list matching the observed sequence of menu interactions is updated when the match is found, the list additionally containing the selected item is further updated to indicate that the item has been visited again. When a menu is expanded, the preceding sequence of user actions including finally the menu selection in the toolbar
is matched to all like sequences and the most likely next action is chosen among the matched histories. As was the case with HPl and HPg, there are two variants here as well: PPl and PPg. PPl collects histories organized by menu, while PPg collects interaction histories globally per application.

Table 3.12 shows the state of prediction tables of PPl and PPg for a history window of length 2 following the interaction sequence: \{a1,a2,a1,a2,a1,a2,b1,a3\}. Note that for PPl and PPg a count of visits is also kept, in order to find the most likely item to be visited following a particular history of interactions. PPl and PPg encode a richer history of interactions than HPl and HPg. Whereas HPl and HPg record only the last menu item visited following an interaction sequence, PPl and PPg record both the mispredicted item and the newly selected item, keeping instead a count of visits. Again, keeping Figure 3.10 in mind, the interesting sequence, in the case of PPg, is \{a1, a2\}, since it generates three outcomes: a1, a3, and b1. When the user expands menu A, the predicted item is a1, and when menu B is expanded, the predicted item is b1, however after a3 is encountered, PPg may only incur a single miss on subsequent selections of a1, given that it is the more frequently accessed item.

**Menu Reorganization**  Menu reorganization is considered here as an alternative to pointer jumps. The analysis in this chapter revolves around pointer jumps due to the ease of implementation. However, with simple modifications, applications can be made to produce menus that reserve the top portion of the menu to the items most likely to be accessed or copies of the same in order to minimize the distance that must be traveled to access these items. The decision of which items should be placed in this area can be made using the same mechanisms proposed here. Limiting this area to contain a single item produces equivalent results to the pointer jumps described here. Expanding the area to include additional items will marginally increase the distance to acquire the desired target, but will significantly increase the hit rate of the predictors, i.e. if the area contains the top predicted target item as well as the last predicted item, mispredictions from alternating item
choices will be reduced.

3.2.4 Evaluation

We have collected hours-long usage traces of common windows applications and used to directly compare the performance of each mechanism. Here, trace-based analysis, rather than the performance of the mechanisms during live user sessions, is used to eliminate the variability naturally encountered during the live sessions. The trace-based simulation also allows us to individually evaluate each optimization and directly compare the given optimization to others proposed mechanisms.

Tracing and Simulation

The applications chosen for tracing are common windows applications: Word, Excel, Powerpoint, Visual Studio, and Textpad. The chosen applications contain complex interactive systems with multiply nested cascading drop-down menus, as well as context-sensitive menu contents. Word, Excel, and Powerpoint are applications from the Microsoft Office suite of applications. The versions used for tracing predate the Office Fluent UI’s ribbon, and as such reflect the use of the traditional cascading menus. Visual Studio’s menus are particularly complex, with extensive use of context-sensitive menu changes. Transitioning from editing to debugging causes significant reconfiguration of available menus and menu options. Finally, Textpad is a general-purpose editor for plain text files.

A detailed analysis that allows for a direct comparison is made possible through the use of a simulator that replays each applications interaction trace through each of the mechanisms. Each trace contains the raw usage data, such as the distance traversed by the mouse from the time a pull-down menu is expanded until the target option is acquired, i.e. clicked on, as well as the time that elapsed from the menu expansion until target acquisition. The traces also record the state of the expanded menus, containing a list of the options in the menu, the size of each options interactive area, and the state of each option: visible, hidden, active, inactive. Each
option may be visible and active, visible and inactive, or hidden and inactive. The collected size of visible options' interactive areas provides the total height of the expanded menu. The total height is used to determine the midpoint for MtM, the sum of heights of the visible options between the predicted target and the top of the menu are used to transition the mouse pointer for the remaining mechanisms that predict a menu item. In the case of mispredictions, the distance between the erroneously predicted target option and the desired target is calculated from the heights of the visible menu options and used for traveled distance comparison among predictors. In the cases where the predicted target is not visible or is inactive, the predictive mechanisms will choose a target from the set of visible and active options, provided that the data to make such a prediction is available.

The time to traverse the distance from the top of the expanded menu to the desired option or from an erroneously predicted target option to the actual target can determined by the application of the Acott-Zhai steering law, using the distance from either the top of the expanded menu to the middle point of the target option or the middle points of the erroneously predicted target to the desired target option. Acquisition time is proportional to the traveled distance and therefore, only traveled distance is used to compare the studied mechanisms.

**Accuracy**

The accuracy of each mechanism depends on the number of hits – the number of times the pointer was transitioned to the option that the user selects, and the number of misses – the number of times the pointer is transitioned to an option other than the one that is selected by the user, or the pointer was not moved when the prediction was not available during training. Details regarding the numbers of prediction hits and misses in the collected traces are contained in Table 3.3. Accuracy is the ratio obtained by dividing the number of hits by the total number of predictions.

\[
Accuracy = \frac{Hits}{Hits + Misses}
\] (3.1)
As is shown in Figure 3.15, the highest overall accuracy is obtained by using a history length of 3. This history length strikes the best balance between fast training and the context of current execution. Furthermore, longer histories will increase the probability that uncorrelated menu interactions are part of history and used for predictions.

Figure 3.13 shows the average accuracies for each application and mechanism. According to the above discussion, history length is set to 3 for all history based mechanisms. The highest accuracy mechanisms are HPg and PPg. Both of the highly accurate mechanisms record interaction histories globally per application as well as per each menu in an application. With the accuracies of 56%-73%, they each outperform the simple MtL mechanism by almost 30%.

In terms of specific applications, as shown by Figure 3.13, the highest accuracy is obtained by PPg using the Textpad traces, where the accuracy is as high as 73%. The biggest improvement occurs in the cases of Textpad, Excel and Development Studio usage traces, where the improvement over MtL is 67% for Textpad and 32% for both of the other two cases.
Figure 3.14: Average normalized distance studied applications and mechanisms.

3.2.5 Acquisition Distance

The target acquisition distance is defined to be the distance from the middle of the top edge of the menu to center of the target option in the case of the base traces. The raw movement distances are recomputed for the base case in order to create a direct comparison, so while a user’s mouse path may wander, the actual distance used is considered to be the straight vertical line to the center of the selected option. For the proposed mechanisms, the acquisition distance is the straight line distance between the centers of the predicted option and the desired target option, in the case where the facilitated movement selects one specific option. In the case where multiple predicted options are equally likely, the initial facilitated movement places the mouse cursor at the mean of the distances of the equally likely options, and the acquisition distance is computed from this point to the center of the target option. Equivalently, one of the equally likely options can be chosen at random. When the desired option is selected by the facilitated pointing mechanism, the acquisition distance is 0.

As shown in Figure 3.16, the shortest acquisition distances are obtained when the
history window length 3, 4, and in the case of Word, 5. The distances in this figure are shown as percentages of the movement distance in the unmodified pointing environment, shown as \textit{RAW} in Table 3.3. The improved acquisition distances at short window lengths indicate that despite the highest accuracies occurring at window length 3, at times the shortest average target acquisition distance may occur at slightly longer windows due to the proximity of the predicted option to the desired option. This is likely caused either by an effective grouping of options with closely matched functionality within a particular menu or by a grouping of commonly accessed options. For example, the \textit{File} menu commonly groups the file access options together, such as \textit{Open File} or \textit{Save File}, while infrequently accessed options, such as \textit{Exit}, are placed at the bottom edge of the menu.

Figure 3.14 shows the average performance of each mechanism with respect to acquisition distance and for each application, given an interaction sequence with length 3. Across the board, PPg performs best and reduces the average acquisition distance by 60\% from the base case and by 36\% from the simple MtL. In terms of specific applications, PPg achieves as much as 83\% reduction in acquisition distance from the base case, as in the case of Textpad, and even in the least improved case, Microsoft Development Studio, PPg cuts the average acquisition distance by half. On the other hand, the MtM mechanism, which, when visits to all options in an application’s menus are visited with uniform frequency, would cut the acquisition distance by half over the base case, performs the worst, even increasing the distance over that traversed in the base case. This is the result of commonly accessed options, as contained in the traces, appearing at the top of their respective menus, causing the user to traverse a greater distance to reach those options using MtM that would normally be most quickly accessed from the top of the menu.

\textbf{Determining the History Size}

The optimal history window is determined experimentally, by simulating the operation of each of the proposed mechanisms and varying the length of the history window from 2 to 20 for each replayed trace. Each execution incurs the training
<table>
<thead>
<tr>
<th>Application</th>
<th>RAW</th>
<th>MtM</th>
<th>MtL</th>
<th>FP</th>
<th>HPl</th>
<th>HPg</th>
<th>PPl</th>
<th>PPg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excel</td>
<td>62</td>
<td>68</td>
<td>52</td>
<td>39</td>
<td>47</td>
<td>21</td>
<td>52</td>
<td>17</td>
</tr>
<tr>
<td>Total Distance</td>
<td>222860</td>
<td>243232</td>
<td>189486</td>
<td>142604</td>
<td>169686</td>
<td>75878</td>
<td>190916</td>
<td>62590</td>
</tr>
<tr>
<td>Hits</td>
<td>N/A</td>
<td>57</td>
<td>587</td>
<td>800</td>
<td>605</td>
<td>1149</td>
<td>568</td>
<td>1170</td>
</tr>
<tr>
<td>Misses</td>
<td>N/A</td>
<td>1769</td>
<td>1239</td>
<td>1026</td>
<td>1221</td>
<td>677</td>
<td>1258</td>
<td>656</td>
</tr>
<tr>
<td>PowerPoint</td>
<td>47</td>
<td>55</td>
<td>40</td>
<td>30</td>
<td>39</td>
<td>26</td>
<td>39</td>
<td>24</td>
</tr>
<tr>
<td>Total Distance</td>
<td>152955</td>
<td>178123</td>
<td>130537</td>
<td>128084</td>
<td>87329</td>
<td>129437</td>
<td>80102</td>
<td></td>
</tr>
<tr>
<td>Hits</td>
<td>N/A</td>
<td>0</td>
<td>327</td>
<td>630</td>
<td>390</td>
<td>774</td>
<td>369</td>
<td>822</td>
</tr>
<tr>
<td>Misses</td>
<td>N/A</td>
<td>1556</td>
<td>1229</td>
<td>926</td>
<td>1166</td>
<td>782</td>
<td>1187</td>
<td>734</td>
</tr>
<tr>
<td>MSDev</td>
<td>74</td>
<td>46</td>
<td>47</td>
<td>39</td>
<td>33</td>
<td>34</td>
<td>47</td>
<td>27</td>
</tr>
<tr>
<td>Total Distance</td>
<td>323422</td>
<td>202598</td>
<td>206030</td>
<td>170610</td>
<td>144650</td>
<td>152262</td>
<td>208208</td>
<td>120252</td>
</tr>
<tr>
<td>Hits</td>
<td>N/A</td>
<td>707</td>
<td>927</td>
<td>826</td>
<td>942</td>
<td>679</td>
<td>1167</td>
<td></td>
</tr>
<tr>
<td>Misses</td>
<td>N/A</td>
<td>2086</td>
<td>1379</td>
<td>1159</td>
<td>1260</td>
<td>1144</td>
<td>1407</td>
<td>919</td>
</tr>
<tr>
<td>Word</td>
<td>44</td>
<td>62</td>
<td>32</td>
<td>24</td>
<td>35</td>
<td>22</td>
<td>32</td>
<td>19</td>
</tr>
<tr>
<td>Total Distance</td>
<td>145420</td>
<td>204930</td>
<td>106700</td>
<td>80014</td>
<td>116864</td>
<td>72842</td>
<td>108262</td>
<td>62480</td>
</tr>
<tr>
<td>Hits</td>
<td>N/A</td>
<td>0</td>
<td>704</td>
<td>928</td>
<td>585</td>
<td>999</td>
<td>689</td>
<td>1055</td>
</tr>
<tr>
<td>Misses</td>
<td>N/A</td>
<td>1545</td>
<td>841</td>
<td>617</td>
<td>960</td>
<td>546</td>
<td>856</td>
<td>490</td>
</tr>
<tr>
<td>Textpad</td>
<td>133</td>
<td>111</td>
<td>146</td>
<td>140</td>
<td>69</td>
<td>26</td>
<td>145</td>
<td>22</td>
</tr>
<tr>
<td>Total Distance</td>
<td>428441</td>
<td>357605</td>
<td>469205</td>
<td>451697</td>
<td>223359</td>
<td>86366</td>
<td>468218</td>
<td>72918</td>
</tr>
<tr>
<td>Hits</td>
<td>N/A</td>
<td>0</td>
<td>173</td>
<td>935</td>
<td>1113</td>
<td>2056</td>
<td>169</td>
<td>2182</td>
</tr>
<tr>
<td>Misses</td>
<td>N/A</td>
<td>3009</td>
<td>2836</td>
<td>2074</td>
<td>1896</td>
<td>953</td>
<td>2840</td>
<td>827</td>
</tr>
</tbody>
</table>

Table 3.3: Details of simulations using a history window length 3.

cost, while the prediction table becomes populated. Intuitively, the number of interactions that must be observed before each mechanism’s prediction table is primed to produce useful predictions increases as the length of the history window is increased. However, intuition also suggests that the organization of menu options and the relatively small subset of menu options that are commonly used favor a small window size.

To test our intuition, consider Figures 3.15 and 3.16, which show the average accuracies and average distances traversed by the mouse for each of the prediction mechanisms. In Figure 3.16, the average distance traversed from menu expansion to option selection is normalized by the average distance traversed in the interaction traces, meaning the unoptimized menu navigation. In each case, the history window was varied from a history of 2 interactions up to 20. The MtM, MtL, and FP mechanisms do not depend on interaction history, so their accuracy and the distances to the predicted options do not vary. The MtM mechanism records a hit only when the chosen option is at the center of the expanded menu, resulting in a very low hit rate over all the applications. Further, if the selected options are equally distributed
across the menu, then MtL would intuitively halve the distance traversed, however, the overall average distance being greater than the raw distance would indicate that commonly accessed options generally appear near the top of the menu. MtL is a simple heuristic that shows a moderate improvement. 28% of selections within each menu involve the option that was the immediately prior selection. Choosing the most frequent option, FP, results in 45% accuracy, while the best predictor, PPg, resulted in 65% accuracy when the history window was set to 3 or 4. The accuracy drops below FP when the history window is increased to 7 and below MtL when the history window is increased to 9. This indicates that short sequences, for example: open a file, save changes, and exit the program, are the common interaction scenarios. Due to this observation, the remainder of this chapter will analyze the proposed mechanisms’ performance with the recorded interaction sequence length of 3.

Overheads

The accuracies obtained with the proposed mechanisms will be sufficient to considerably reduce the target acquisition distance, and, as seen above, the acquisition distance can be completely eliminated in almost 70% of pull-down menu interactions in
particular applications. It is expected that with more sophisticated prediction mechanisms, this accuracy can be further improved and the target acquisition distance further reduced. However, the use of more complex machine learning mechanisms carries the associated computational overheads. In order to maintain interactive responsiveness and transparency to the user, the proposed mechanisms and any that may follow must ensure that the impact on the interactive environment remains minimal. At the very least, the combined overheads of interaction capture, predictive mechanism logistics and execution must remain below the human perception threshold (50ms) to keep from introducing any unwanted interactive stutter.

Low computational overheads that ensure real-time interactive continuity and preserve the interactive experience for the user are key to the described transparent system. Determining the states of the selected menu’s options, and prediction table lookup and update do not incur more than 3ms computation time. In terms of the human perception threshold, this length of time is negligible and any interactive stutter will easily be unnoticed. It is important to note that the prediction table lookups and updates occur once per interaction, if the interaction is determined to be with a menu within an application. The overheads of operating the mouse hook
DLL responsible for reporting and setting the mouse position are negligible, since
the DLL does not perform the computation itself, but rather relies on a separate
monitoring and prediction process that manages the prediction tables. However, the
latency of the DLL is dependent largely on the monitoring and prediction module,
which is responsible for computing the new mouse position based on current menu
state received from the MSAA server.

The storage overheads vary between mechanisms, where the amount of storage
required for the prediction table grows linearly with table size and is dependent
on the type of histories encoded in the prediction table. For each interaction, the
owning application and the enclosing menu must be encoded along with the option
name, size, and location. The prediction table segment associated with any one
application is maximally 182KB, while the overhead of storing the global prediction
table, i.e. the histories and predictions of all applications, is at most 615KB. The
small table size indicates that a global prediction table can be kept in memory and
reused between executions to reduce the future need for training.
CHAPTER 4

MANAGING DELAYS IN DISKS

The key idea here is that the correlation between user interactions and disk I/O activity can be transparently exploited to predict I/O activity ahead of time and perform a timely disk spin-up to serve the request. This chapter details the design and evaluation of the modified Adaptive Learning Tree (ALT+), and the Interaction-Aware Spinup Prediction (IASP), mechanisms for the disk.

4.1 Modified Adaptive Learning Tree

The Adaptive Learning Tree (ALT) approach is to predict the best current power mode based on a sequence of idle periods. Idle periods are discretized according to the time spent idling, and in relation to the number of available sleep states and device specifications. Previously observed states or sequences of states are encoded in a tree, the paths of which are matched according to newly observed sequences of discretized idle periods. Each leaf node in the tree constitutes a prediction and the most likely prediction is selected to transition the disk to the matching power state. ALT has shown significant improvement for power mode prediction in static, non-interactive applications and motivated us to adapt the design to predict the length of idle times and spin-up the disk before the predicted idle time ends.

ALT’s discretization of idle periods depends on the number of available power states of the disk, and the prediction of the period lengths allows transition into shallower sleep states, where the disk’s RPMs are reduced, but not halted, and the time to ready the disk is lessened. These periods are on the order of seconds and correspond to the breakeven time of each state. The design of ALT is, therefore, extended to predict longer idle periods; for the purpose of differentiation the modified design is referred to as ALT+. The discretization of idle periods in ALT+ results not
in the prediction for the best power mode, but rather for the duration of the current idle period. In order to spin up the disk in ALT+, thresholds are used in multiples of the disk’s breakeven time, where the multiple is given by the discretization and encoded in the same tree structure as found in ALT. With this modification, ALT+ generates a likely time to ready the disk in anticipation of upcoming I/O activity, allowing the disk to be spun up on time to service the request.

4.2 Interaction-Aware Spin-Up Prediction Design

IASP is designed with the following requirements:

- User interactions have to be captured transparently without modification of applications.
- Capture and prediction should be efficient to prevent excessive energy consumption by the CPU to train and generate predictions.
- The system should handle multiple applications in a graphically rich environment.
- User behavior correlation and classifications should be performed online and without direct user involvement.

The first three items are addressed by the novel implementation described in this chapter. The last item is addressed by the proposed context capture design.
4.2.1 The Naïve Predictor

The observation that user interactions are responsible for the majority of disk I/O, in the interactive applications, leads us to a proposal of a simple mechanism that spins up the disk upon mouse clicks. The intuition dictates that if the user actively interacts with an application, which may require disk I/O, the disk should stay on to satisfy user requests. If the user is not actively interacting with the application the likelihood that the disk will be needed drops and the disk can be shut down. Therefore, the naïve All-Click Spin-Up mechanism (ACSU) spins the disk up upon each mouse click and keeps it spinning as long as the user is interacting with the application. Once the user stops interacting, the disk shuts down after a timeout period.

ACSU mechanisms act on all mouse clicks and spin up the disk as soon as possible, with the downside of unnecessary spin-ups for clicks that are not followed by any disk I/O. It is important to note that user interactions that require disk I/O are a small subset of all user interactions with the application. Therefore, ACSU mechanisms have the greatest potential for reducing spin-up delays at the expense of energy consumption caused by unnecessarily spinning the disk up and keeping the disk spun up without serving any disk I/Os. ACSU mechanisms set a lower bound on the spin-up delays for the proposed IASP and also illustrate the need for more intelligent prediction schemes that decide when the disk should be spun up to improve energy efficiency.

On Unix-like systems, the X Window System is the common display protocol built on the client-server model. It is responsible for accepting graphical output requests from and reporting user input to clients. The stream of data from the client to the server contains the information about the window layout, while the data sent from the server to the client applications contains the information about user interactions. The additional intermediary layer, as shown in Figure 4.1, between the server and its clients, enables the observation of the exact sequence of requests and events. This layer allows for transparent monitoring of user behavior. No mod-
taxonomy of applications is necessary. Furthermore, user interactions are captured exactly, eliminating both the excessive computational overhead of computing a clustering and the inaccuracies associated with the clustering present in the previously described solution [1]. Since the need for cluster formation and behavior detection is eliminated, the offline processing needed in the clustering approach is eliminated, fully allowing for detection, correlation, and prediction to be performed online.

4.2.2 Monitoring & correlating I/O activity

Each application is monitored individually for mouse clicks and file I/O. This allows a more accurate correlation of file I/O activity to user interactions with an application. Two levels of correlation allow file I/O and disk I/O to be distinguished. First, the application’s file I/O activity is captured by the kernel in the modified I/O system call functions that check for file I/Os. For example, a modified `sys_read` checks if the I/O call that entered `sys_read` is indeed file I/O since `sys_read` can be used for many types of I/O. This stage does not consider buffer cache effects since file I/O activity is captured before the buffer cache. As a result, a more accurate correlation between file I/O and mouse interactions is obtained. Second, once potential file I/O activity is detected, the call is followed to determine if it resulted in an actual disk I/O or it was satisfied by the buffer cache. This information is used to correlate the user interactions that invoke file I/O to the actual disk I/O. The access patterns in the buffer cache will also correlate to the user interactions, since
user behavior is repetitive. Hence, it will be shown that IASP is able to predict actual disk I/Os with a high degree of accuracy.

**Correlating file I/O activity**

Correlation statistics are recorded in the prediction table that is organized as a hash table indexed by the hash calculated using the mouse event IDs. Figure 4.1 shows the prediction table organization and the content of the table entry that is maintained by the IASP daemon. The click IDs are unique to the window organization and therefore do not result in aliasing between different applications and windows as explained earlier. The data stored by the prediction table contains only the unique event ID, the number of times the event was observed, and the number of times I/O activity followed. The counts are a simple, but efficient means of computing an empirical probability for future predictions. The table resides globally in a daemon and is shared among processes to allow table reuse across multiple or concurrent executions of the application. Furthermore, the table can be easily retained in the kernel across multiple executions of the application due to its small size.

In addition to the global prediction table, IASP records the history of recent click activity for each process in the system. Consider a typical usage scenario shown in Figure 4.2 where a user is editing a file in a word processor. After a while, the user clicks through a file menu to change properties of the edited file. The recorded history of clicks is C1, C2, C3, C4. At this point, the user decides to work on the file
again. If the time is long enough the clicks are considered to be uncorrelated and the history of clicks is cleared. Alternatively, the user may immediately proceed to open a new file with click sequence of C1, C5, C6, C7. In this case, the history is also reset when the user clicks on C1. Since all menus are organized as trees in the application, clicking on C1 signifies return to the root of the File menu tree. Therefore, when IASP detect a repeated click ID in the history, the history is restarted with the current click. It is still possible to record uncorrelated clicks in the history. For example, user interacts with the Edit menu and subsequently opens a file. In this case, the history will contain clicks for edit menu interactions and the file open interactions. However, the uncorrelated clicks will have low probability and will eventually be made insignificant to the predictor with further training.

IASP uses very simple training where the observed count is updated every time a particular click is detected. In order to correlate file I/O activity, the history of clicks that lead to file I/O is traversed and the I/O count for every click present in the history is incremented. Ratio of both counts gives us the probability of file I/O following the particular click.

**Correlating disk I/O activity**

File I/Os issued by the application can be satisfied by the buffer cache and as a result may not require any disk I/O and the disk can remain in a power saving state. Since not all file I/Os result in disk I/O, an additional correlation step is necessary to relate the mouse click to the disk I/O. A history of file I/Os generated by the particular click IDs is used to predict the future disk I/O generated by the particular click. An additional 2-bit history table with a 2-bit saturating counter records the history of file I/Os that resulted in disk I/O after a given mouse click was observed. The prediction table is updated using the history of file I/Os and the resulting disk I/O and the current outcome of the file I/O. Combination of the file I/O probability and the resulting I/O prediction results in a final decision about disk spin-up.

The proposed mechanisms rely on file I/O prediction and disk I/O prediction
to separate application behavior from the file cache behavior. By considering both the probability of a particular click being followed by file I/O and the behavior of the resulting I/O in the buffer cache, IASP can accurately discern whether that click will result in actual disk I/O. Separating the predictor’s training into file I/O and buffer cache behavior allows accurate correlation of clicks to the application’s file I/O, which is the fundamental goal of this chapter. Mouse interactions with the application’s GUI are strongly correlated to file I/O, so, intuitively, the goal of the described implementation is to filter all uncorrelated clicks first before the buffer cache impact on disk I/O is considered. Finally, a simple 2-bit history is used to predict buffer cache behavior, which provides sufficient accuracy. However, more sophisticated buffer-cache behavior prediction can be potentially employed to further improve IASP accuracies.

4.2.3 To predict or not to predict

The critical issues that are addressed in this design are timeliness and accuracy, which turn out to be competing optimizations. Many application functions can be invoked with just a single click, however certain operation may require several steps. In case of multiple clicks, the last click initiates a system action that is a response to the user’s interaction. More specifically, if only the last click just before disk I/O occurred is used and correlated to the particular disk behavior, it results in high accuracy. Note that while this approach is very accurate, it is not very timely.

Correlating disk I/O to the last click occurring before the I/O request was observed does not provide adequate time before the I/O arrives to offset a significant portion of the spin-up latency, and so has a negligible impact on reducing the associated interactive delays. This scenario is illustrated in Figure 4.2. Clicking on C7, which is the final Open button in file open sequence, will result in an I/O system call. However, the click is immediately followed by I/O and waiting for prediction until last click will provide little benefit in reducing delays exposed to users. Spinning up the disk upon C1 click provides sufficient time to reduce delays; however, it may result in erroneous spin-ups, since the user may perform other operations that
do not lead to file I/O.

4.2.4 Predicting upcoming activity

Figure 4.3 shows the decision-making process. Upon each mouse click, the ID of the current click is used to calculate the prediction table hash index. The daemon performs a prediction table lookup with three possible outcomes:

1. The entry is found indicating that the interaction leads to file I/O with probability above the threshold.

2. The corresponding entry is found but contains a low probability of upcoming file I/O.

3. The entry is not found.

Events which are not found in the table are added and updated accordingly to the training routine described earlier. Once the entry is found that satisfies the desired probability threshold, the current history of disk I/Os for the given mouse click is used and a final lookup into history prediction table for the selected click is made. The prediction from the history table dictates the outcome of the disk state and the disk is transitioned to the predicted state.

The experimental implementation considers only two possible states for the disk: standby and sleep. Therefore, the decision to spin-up the disk is binary, i.e. the
determination is made that either I/O activity is forthcoming following the mouse event or it is not. However, the binary decision predictor can easily be extended to devices with multiple sleep states. The only required modification is the discretization of the probabilities computed from the prediction table. Consider the case of 2 sleep states, one having the platters fully spun-down, and the other having the platters spinning at a reduced RPM. The third possible state is full idle. This scenario is easily handled by adding a second threshold. With two thresholds, probability values falling beneath the lower one generate a prediction favoring the halted platters sleep state. Probability values falling between the two thresholds would cause the disk to enter the reduced RPM sleep state. Finally, any probability values above the higher threshold would fully spin-up the disk. Clearly this modification can be extended to an arbitrary number of sleep states.

4.2.5 Multiprogramming environment

As described, these mechanisms work in a multiprocess environment since training and prediction are made independently of other processes. The described monitoring mechanism allow the system to uniquely identify windows from multiple processes and allow for accurate correlation without any aliasing from other applications. The prediction performed by IASP is also easily integrated into a multiprocess environment since as soon as IASP predict spin-up for a single process the disk is spun up without considering other processes. This is opposite of the shutdown mechanisms which have to consider other processes that are currently running and may need the disk. In case of spin-up, once the disk is needed it has to be spun-up.

4.3 Evaluation

The performance of ACSU and IASP mechanisms is evaluated and compared to ALT+. A trace-based simulator is used to fully evaluate the effectiveness of the proposed mechanisms. An implementation on an actual system, replaying the traces in real-time on a disk was used to validate the simulation results. Since the focus is
<table>
<thead>
<tr>
<th>State</th>
<th>WD2500JD</th>
<th>40GNX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write Power</td>
<td>10.6W</td>
<td>2.5W</td>
</tr>
<tr>
<td>Seek Power</td>
<td>13.25W</td>
<td>2.6W</td>
</tr>
<tr>
<td>Idle Power</td>
<td>10W</td>
<td>1.3W</td>
</tr>
<tr>
<td>Standby Power</td>
<td>1.8W</td>
<td>0.25W</td>
</tr>
<tr>
<td>Spin-up Energy</td>
<td>148.5J</td>
<td>17.1J</td>
</tr>
<tr>
<td>Shutdown Energy</td>
<td>6.4J</td>
<td>1.08J</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State Transition</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Spin-up time</td>
<td>9 sec.</td>
</tr>
<tr>
<td>Shutdown time</td>
<td>4 sec.</td>
</tr>
</tbody>
</table>

Table 4.1: Disk energy consumption specifications.

on predicting spin-ups and as a result, the shutdown mechanism is a simple timeout set to 20 seconds which is comparable to the breakeven time of both disks. This means that the disk is shut down after 20 seconds of idleness. This also applies to erroneous spin-ups, where when the disk is spun up, it waits for the timeout to expire before subsequently shutting down.

Detailed traces of user-interactive sessions for each application were obtained by a modified `strace` utility over a number of days. The modified `strace` utility allows us to obtain the PID, access type, time, file descriptor, as well as the amount of data that is fetched for each I/O operation. The specifications of the simulated disks belong to Western Digital Caviar WD2500JD and Hitachi Travelstar 40GNX hard drives and are shown in Table 4.1. The WD2500JD has a spin-up time of about 9 seconds from the sleep state, the surprising duration of which appears to be remarkably common for high-speed commodity drives. The 40GNX is designed for portable systems and as such has much lower energy consumption and spin-up time than the WD2500JD.

Table 4.2 shows six popular desktop applications chosen for evaluation: Firefox, Writer, Impress, Calc, Gimp, and Dia. Firefox is a web browser with which a user spends time reading page content and following links. In this case, I/O behavior depends on the content of the page and user behavior. Impress (presentation editor), Writer (word processor), and Calc (spreadsheet editor), are part of the Open Office
Table 4.2: Details of application traces used to evaluate IASP.

<table>
<thead>
<tr>
<th>Appl.</th>
<th>Number of I/O Periods</th>
<th>Read (MB) Without Cache</th>
<th>Write (MB) Without Cache</th>
<th>Read (MB) With Cache</th>
<th>Write (MB) With Cache</th>
<th>Number of Clicks</th>
<th>Number of IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox</td>
<td>814</td>
<td>1903.35</td>
<td>350.8</td>
<td>851.91</td>
<td>120.39</td>
<td>857</td>
<td>130</td>
</tr>
<tr>
<td>Writer</td>
<td>1385</td>
<td>2043.62</td>
<td>2186.28</td>
<td>1434.05</td>
<td>2120.47</td>
<td>5755</td>
<td>195</td>
</tr>
<tr>
<td>Impress</td>
<td>1485</td>
<td>1230.42</td>
<td>263.6</td>
<td>517.06</td>
<td>60.44</td>
<td>25375</td>
<td>194</td>
</tr>
<tr>
<td>Calc</td>
<td>2846</td>
<td>1840.4</td>
<td>116.7</td>
<td>1280.7</td>
<td>59.67</td>
<td>46864</td>
<td>118</td>
</tr>
<tr>
<td>Gimp</td>
<td>844</td>
<td>1443.32</td>
<td>957.3</td>
<td>796.9</td>
<td>936.54</td>
<td>8465</td>
<td>157</td>
</tr>
<tr>
<td>Dia</td>
<td>6362</td>
<td>174.31</td>
<td>65.3</td>
<td>123.64</td>
<td>10.28</td>
<td>46864</td>
<td>118</td>
</tr>
</tbody>
</table>

suite of applications. All three are interactive applications with both user driven I/O and periodic automated I/O, i.e. autosaves. Gimp is an image manipulation program used to prepare and edit figures, graphs, and photos. Finally, Dia is an application used for drawing diagrams for papers and presentations.

Table 4.2 also lists the total number of idle periods for which a potential shut-down and a corresponding spin-up are required, the total amount of read and write activity, a total number of mouse click interactions and the number of unique click interactions encountered in the studied applications. Also shown are the statistics for I/O requests that are generated by the application (shown as 'Without Cache') and I/O requests that are not filtered by the buffer cache and are send to the disk (shown as 'With Cache'). The experiments assume an LRU managed buffer cache of size 512MB, which is representative of current systems’ capabilities.

4.3.1 File I/O Correlation Accuracy

Accurate prediction ensures that the disk is not spun-up needlessly, when no activity is forthcoming. Firstly considered is the accuracy of correlating mouse clicks to file I/O at the application level, before it is filtered by the buffer cache. Figure 4.4 shows the breakdown of correct and incorrect spin-ups, i.e. hits and misses, for ACSU, IASP and ALT+ that result from predicting file I/O when the system does not employ buffer caching. Hits are counted when the prediction to spin-up the disk is made and it is followed by file I/O. Misses are those spin-ups which were not followed by any I/O, and Missed Opportunities are periods for which the mechanism failed to provide a prediction, but a spin-up was needed. Each missed opportunity results
in the disk being spun up on demand, essentially spinning up when an I/O request arrives. ACSU mechanisms keep the disk powered up while a user is interacting with the application, minimizing the interactive delays. While it provides an upper bound for the number of I/O periods that may be predicted by clicks (i.e. the number of periods covered by IASP can equal but not exceed the number of periods covered by ACSU), it naïvely spins-up the disk for all clicks, resulting in excess misses. Low coverage and high inaccuracies in ALT+ illustrate the behavior of the mechanisms that solely rely on observing system events without considering user interactions.

ACSU on average covers 81% of all file I/O periods, while IASP correctly covers an average of 79% of periods. The lack of contextual information and the random
nature of idle period duration results in ALT+ correctly covering an average of 7% of periods. ACSU shows the greatest number of misses for all applications, 52% of spin-ups are misses. This miss rate reflects the number of existing mouse clicks that do not correlate to any I/O. When the disk is spun-up in ACSU, it will remain spinning as long as new clicks are observed and the idle threshold is not reached between any two clicks. IASP consistently results in the fewest misses, averaging 2%, which mostly occur while the predictor is warming up.

Whereas ALT+ considers solely I/O patterns when generating predictions, coverage by the ACSU and IASP mechanisms is contingent on the availability of mouse events preceding I/O activity. Firefox, Writer, and Calc show the greatest number of misses and missed opportunities for both ACSU and IASP, meaning that there is a good deal more ambiguity in the mouse events available for prediction generated by these applications than the others. In the case of Firefox, most mouse activity occurs within the window displaying the visited web pages. As such, the constantly changing structure of the window increases the number of mouse IDs that are encountered resulting in a high misprediction rate for ACSU. IASP, on the other hand, does not spin-up the disk for these clicks, since their IDs are not observed as often as those that belong to the static part of the GUI. In the case of Writer and Calc, the relatively low coverage by both ACSU and IASP mechanisms is caused by lower availability of clicks preceding I/O. While most or all functionality of these applications is accessible through the GUI, the interaction is made simpler through the use of keyboard shortcuts. As only mouse events are considered, any I/O that occurs in response to a keyboard shortcut is not predicted by the proposed mechanisms. The applicability of keyboard events to I/O prediction will be explored in future research.

The applications for which both ACSU and IASP perform best are Impress, Dia and Gimp. These applications have more complex GUIs for which extensive keyboard shortcuts are not intuitive to an average user. This is the case with Dia and especially Gimp. All three of these applications also depend heavily upon the mouse, due to the graphical nature of their content and usage. Manipulating images,
graphs, and figures is done most easily with the mouse and in the traces the user depended more heavily on the mouse for all interactions with these applications.

4.3.2 Impact of the Buffer Cache

High file I/O prediction accuracy shown in Figure 4.4 represents the strong correlation between mouse clicks and file I/O. However, file I/O may also be satisfied by a buffer cache access, making a disk spin-up unnecessary. Hence, the impact of the buffer cache on prediction accuracy must be considered. From this point on, all figures show the mechanism with the buffer cache enabled. The buffer cache size is set to 512MB, which is representative of current systems’ capabilities. The buffer cache can satisfy many file I/Os resulting in fewer required disk accesses. In addition to introducing additional randomness into file I/O patterns, the buffer cache also increases the training time of prediction mechanisms due to both inclusion of the access history and and fewer spin-ups encountered in the system. Similarly to Figure 4.4, Figure 4.5 shows hits, misses, and missed opportunities, but those metrics now realistically reflect the actual required disk I/O.

The aggressiveness of the ACSU mechanism again makes it a top performer when the amount of periods covered with correct spin-ups is considered. This behavior can be expected since the mechanism just keeps the disk on no matter if the buffer cache satisfies the request or not. Therefore, this mechanism’s behavior is not impacted by the presence of the buffer cache. The different fraction of period misses, and hits, as compared to Figure 4.4 are due to a change in the periods’ composition since there are fewer and longer periods due to filtering of I/O by the buffer cache. In this case, ACSU is able to spin up the disk ahead of time for 66% of required periods, while incurring misprediction rates as high as 54% with the average of 52%. The ALT+ mechanism is also not impacted much by the buffer cache since the randomness observed by the file I/O in the interactive application is already large rendering this mechanism not very useful in either case. The coverage is as low as 3% with an average of 7%, incurring a high misprediction rate of 24% on average.

The impact is more pronounced in the case of IASP, since IASP uses contextual
prediction selectively to predict what user activity will result in disk I/O. Therefore, the introduction of any randomness by the buffer cache affects the accuracy of the history-based IASP disk I/O prediction. IASP remains the most accurate mechanism, resulting in only 2% mispredictions while achieving 65% of correct spin-ups, on average. Low misprediction rate indicates that the randomness introduced by the buffer cache is insignificant and the history-based prediction is able to capture correctly the behavior of the disk I/O. Lower coverage indicates that the fewer I/O periods increase the fraction of learning time. It is worth noting that the significance of learning time decreases the longer the system stays on.

In the case of all applications except for Dia, the lower coverage of the IASP mechanism as compared to the coverage of the uncached I/O is due to learning, since predictions followed by an absence of disk I/O due to caching result in fewer learning opportunities. Interaction with GUI elements results in the requisite file data being stored in the cache. In the absence of a cache, even the infrequently used elements would generate disk I/O, but not so with the cache. In general, IASP greatly reduces the number of unnecessary spin-ups that are present in ACSU, at the cost of lower coverage, due to more energy-efficient spin-up policies.

In the case of Dia, the type of interactions encountered during tracing were limited to very simple actions, such as opening, creating and saving a number of files containing various simple figures, meaning that the availability or absence of I/O was quickly learned by IASP. Creating even the simplest diagrams may require a large number of clicks. ACSU therefore exhibits a large number of mispredictions in this case, while IASP easily filters out the events that cause the program to, for example, draw a triangle rather than open a file.

4.3.3 Confidence Levels

Confidence levels are dynamically set thresholds for prediction within IASP. Recall that confidence levels associated with the mouse events represent the ratio of how many times the event was observed and the number of times the event was followed by file I/O. A given confidence level dictates the amount of predictions made and
the prediction accuracy as illustrated in Figure 4.6. Confidence of 1 means that the click is always followed by I/O activity, confidence of .9 means that the click is followed by I/O activity 90% of the time, and so on. If the confidence level is set too low, the predictor may spin-up the disk early in response to events that rarely lead to I/O activity. For example, the user clicked on File menu but interacted with options that did not involve disk I/O. However, early spin-ups hide more latency if the interaction leads to disk I/O. Setting the confidence level too high, however, may delay the disk spin-up and potentially expose the entire spin-up latency to the users. It is therefore important to set confidence levels such that the energy consumption caused by early-erroneous spin-ups and the delay reduction offered by the early spin-ups are in balance.

Figure 4.6 illustrates impact of confidence on Hit/Miss ratios and file I/O period coverages. Coverage is defined as the fraction of correctly predicted spin-ups in the applications. The ratio of hits to misses shows the average accuracy over all applications in predicting upcoming I/O activity. Due to optimistic prediction, the hit/miss ratio declines slightly as the acceptable confidence level increases past 0.5, but overall remains steady at just over 80%. The sharp increase and steady behavior in the hit/miss ratio indicates quick convergence during training and stable behavior for each mouse event. Increasing confidence level past 0.5 results in longer training
Figure 4.7: Average delay in seconds, WD2500JD.

Figure 4.8: Average delay in seconds, 40GNX.

and the predictor’s coverage drops sharply, since it attempts to predict fewer and fewer disk spin-ups.

4.3.4 Delay Reduction

Figures 4.7 and 4.8 show the average spin-up delays that are exposed to the user in the case of each of the two disks. Each missed opportunity seen in Figure 4.5 results in delay equal to the average spin-up time, 9 seconds in the case of WD2500JD and 4.5 seconds in the case of 40GNX. On the other hand, hits described in Figure 4.5 are predictions that result in the disk spinning up correctly and may arrive either early enough to allow the disk to spin-up before the I/O arrives, resulting in no delay, or late, where the disk is in the process of spinning up when I/O arrives. The
demand based spin-up exposes full spin-up delay to the application, during every 
spin-up, and therefore the average delay in demand based system is full 9 seconds 
in the case of WD2500JD and 4.5 seconds in the case of 40GNX.

ACSU is very aggressive in reducing spin-up delays at the expense of increased 
energy consumption. High coverage of file I/O periods in Figure 4.5 results in an 
average spin-up delay reduction from 9 seconds to 3.3, which is only 37% of the 
spin-up delay exposed by the demand-based spin-up for the WD2500JD. In the case 
of the 40GNX disk, shown in Figure 4.8, the average spin-up delay is reduced to 1.36 
seconds, which is 30% of demand-based spin-up delay. As expected from Figure 4.5 
ALT+ performs poorly exposing high delays of 8.58 seconds and 4.35 seconds for the 
WD2500JD and 40GNX, respectively. The exposed delays in ALT+ are comparable 
to demand based spin-up delays, since most periods require on-demand spin-up.

IASP is able to shorten interactive delays exposed to the users down to 5.89 sec-
onds and 2.67 seconds for WD2500JD and 40GNX, respectively, while maintaining 
high accuracy and low energy consumption. IASP exposes 2.6 seconds and 1.3 sec-
onds more delay than ACSU for WD2500JD and 40GNX, respectively. ACSU sets 
the lower bound on the spin-up delay for mechanisms that utilize mouse interaction 
since it spins up or keeps the disk on for all mouse interaction as shown by the higher 
coverage in Figure 4.5. ACSU not only captures more I/O periods, but also does 
so earlier than IASP, since it is not governed by the confidence requirement set in 
IASP to prevent erroneous spin-ups. ACSU is therefore most effective in situations 
where low delay is desired, assuming that of course energy-efficiency is also a desired 
attribute, but to a lesser extent. The higher accuracy of IASP makes it the most 
desirable choice when energy efficiency is important and users are willing to tolerate 
slightly higher delays than ACSU provides, which are still much lower than delays 
exposed by the demand-based spin-ups.

Highest delay reduction is present in Dia, where the delay is reduced by 93% 
and 85% by ACSU and IASP for 40GNX, and 96% and 85% for the WD2500JD, 
indicating that there is plenty of user think time to overlap spin-up delays. On 
the other hand, Writer shows the lowest reduction in spin-up delays. The most
significant factor contributing to the low reduction in spin-up delays in case of Writer is single button interaction with toolbars, which results in I/O activity. For example, if the user clicks on the spell-check button in the toolbar rather than finding spell-check in the Tools menu, the resulting activity arrives quickly following the single mouse event that predicted it.

Reduction in delay is generally accompanied by increase in energy consumption, since the disk must remain on in order to minimize the delay. For example, if the disk is allowed to remain spinning for the entirety of an application’s run, the interactive delays are eliminated, but at the cost of vastly increased disk energy consumption. On the other hand, simple demand-based mechanisms are often the lowest energy solution, due to the fact that they do not have extraneous spin-ups, but they incur delays each time the disk is spun up. What has been shown is that delay can be significantly improved using the proposed ACSU and IASP mechanisms.

4.3.5 Energy

Figures 4.9 and 4.10 show the details of energy consumption of the two disks. The energy consumption is divided into I/O serving energy, power-cycle energy, and idle energy. I/O serving energy is consumed by the disk while reading, writing, and seeking data. I/O serving energy is the same for all mechanisms, since the amount of I/O served is the same. Power-cycle energy is consumed by the disk
during spin-up and shutdown and is directly related to number of spin-ups which also include erroneous spin-ups. Finally, idle energy is the energy consumed by the disk while it is spinning but not serving any I/Os. Idle energy is dependent on the number of I/O periods and the timeout before the disk is shutdown after an I/O period, additional idle energy consumption occurs in ACSU, IASP, and ALT+ due to mispredictions, during which the disk idles before shutting down when I/O does not arrive. In addition, early spin-ups result in additional energy being consumed by the disk between the time when the disk is ready to serve data and the arrival of the first I/O, which is most prevalent in ACSU.

Due to a large number of mispredictions, ACSU consumes significantly more idle and power-cycle energy than IASP. On average, IASP consumes 30% less energy idling than ACSU, and 40% less energy cycling power modes when using WD2500JD. In 40GNX’s case, IASP consumes 27% less idle energy than ACSU, and 25% less cycling energy. On average, with the WD2500JD, IASP consumes 6% more energy than the on-demand mechanism due to waiting after early spin-ups and the few mispredictions that result in the consumption of energy not present in the on-demand mechanism. Similarly, in the 40GNX case, IASP consumes 7% more energy than the on-demand mechanism. Keeping the disk always on has the effect of increasing idle energy consumption to levels that are prohibitively large for energy constrained systems. Overall, the energy consumed by WD2500JD using ACSU is 49% lower than

Figure 4.10: Energy consumption, 40GNX.
keeping the disk always on, 70% lower in case of IASP, and 65% lower for ALT+.
The energy consumed by 40GNX when using IASP is 64% lower, 60% lower for ALT+. Differences in relative energy consumption result from the different power profiles of the two disks in question.

4.3.6 Energy-Delay

The energy-delay product is an established metric that captures the tradeoff between energy consumption and incurred delay. Reducing delays is important, but it has to be done in an energy-efficient manner. Here, the energy-delay product is obtained
by multiplying the overall delay incurred by each energy management mechanism with the energy consumed due to the use of the same mechanism for the length of each trace.

Figure 4.11 and Figure 4.12 show the energy-delay products for ACSU, IASP, and ALT+, normalized to the energy-delay of the demand-based spin-up mechanism with both the WD2500JD and 40GNX disks. The high improvement of delay by ACSU results in an average 63% improvement of energy-delay over the demand-based spin-up for WD2500JD and 55% for the 40GNX, despite the increased energy consumption due to numerous wrongful spin-ups. IASP improves the energy-delay of the two disks by 44% and 41%. ALT+ does not on average improve the energy-delay product of the WD2500JD and has a 16% worse energy-delay product in the case of 40GNX. Due to the low coverage of ALT+, its behavior tends to resemble the demand-based predictor. However, when the interactive delay is not improved significantly, such as in the case with ALT+, the additional energy spent in mispredictions actually harms the energy-delay product.

4.3.7 Overheads

The computational and storage overheads of any power management mechanism have to be taken into consideration, since improving the energy consumption of one device while equally increasing that of another does not result in energy-efficiency. Therefore, it is critical to keep computational requirements to minimum to avoid the excess energy consumption in the processor. Additionally, the storage overheads of a power management mechanism’s data should be low enough to be considered insignificant, since storing a large amount of data could potentially impact the execution of interactive applications by polluting data caches in the processor.

Considering those requirements, ACSU has a clear advantage since it does not have to store or compute anything. It simply spins the disk up when the disk is shutdown and a click arrives or resets the timeout variable when the disk is active. IASP, on the other hand, computes IDs that uniquely identify user mouse interactions and store the interaction predictions in a prediction table. Due to the
efficiency of hash tables, the only measurable computational overhead is incurred when the unique window ID is computed. *Firefox* has the deepest tree of 27 levels in the studied applications. Therefore, the average overhead of traversing 27 levels of tree hierarchy was experimentally measured and found to be negligible. Furthermore, this overhead can be almost eliminated by modifying the X-Window Server to automatically generate mouse click IDs as it itself builds the window tree, rather than building a separate representation as shown in Figure 4.1.

The storage overhead is likewise relatively low in IASP. For each unique mouse event a table stores its ID (32 bits), the number of times the event was observed (32 bits), the number of times it was followed by I/O activity (32 bits), and the two-bit history table (8 bits) with two bit saturating counters for the prediction outcome (8 bits). The resulting table entry is 14 bytes. The number of unique click IDs in the studied applications ranged widely from 35 in *Calc* to 195 in *Writer*. Therefore, in the worst case *Writer* requires 2.67KB to store 195 entries. An 11.3KB table would suffice for storing all entries from every one of the six studied applications.

4.3.8 Experimental Evaluation

Experiments were conducted using a setup of two desktop machines with dual-core 3.0GHz processors and 2GB of memory. As shown in Figure 4.13, a multi-channel data acquisition board (DAQ) from NI was connected to the power cable.
of a WD2500JD hard drive dedicated to replaying the traces. To measure the power consumed, a 0.1 ohms resistor was placed in series with the hard disk power supply and the voltage drop across the resistor was fed to the DAQ. The second machine, running Windows XP and the DAQ drivers, ran the LabView setup sampling measurements at 1000Hz from the DAQ. The simulated trace-driven prediction mechanisms were ported to a driver that replays the traces on the measured hard drive.

Figure 4.14 shows a selected portion of the Dia trace, as replayed on the hardware and captured through the experimental setup. Shown are several activity periods for on-demand spin-up, ACSU, and IASP. ALT+ is omitted due to its low accuracy. Figure 4.14 shows several Lines C, E, and G which show the spin-ups that were initiated on-demand when the I/O arrived. The period of disk activity beginning at A, as initiated by ACSU, illustrates ACSU’s aggressive spin-up policy. At A the user begins a series of interactions which cause the disk to spin up and remain spinning until the interaction ends and I/O is served. The I/O arrived at C and the disk was spun up on-demand to serve the request. IASP, on the other hand, spun-up early at B, in response to a user interaction that predicted that I/O will follow.

Similarly, observe that the next interaction at D, caused both IASP and ACSU to spin-up the disk ahead of E, where the disk was spun up on-demand. Another example occurs at G, with on-demand spin-up shortly following the IASP and ACSU.
spin-ups. Matching up the trace replay to the simulator output, it has been verified that this behavior is indeed expected.
Similarly to Chapter 4, the key assumption here is that there exists a strong correlation between user interaction with the application and resulting network activity. Further, this correlation can be transparently exploited to manage the power states of wireless interfaces. This chapter discusses the design and evaluation of the Interaction-Aware Prediction, IAP, mechanisms for wireless network interface cards, WNICs.

5.1 Design

IAP mechanisms for the WNIC are designed with the following requirements:

- User interactions have to be captured transparently without modification to the application.

- User interaction correlation and classifications should be performed online without any user involvement.

- Capture and prediction have to be efficient to prevent excessive energy consumed by the CPU to train and generate predictions.

- The system has to be able to handle multiple applications in a graphically rich environment.

5.1.1 The Naïve Predictor

The observation that within interactive applications, network activity is caused by the user’s interactions leads us to a proposal of a simple mechanisms that switches the WNIC to CAM for every mouse click. This approach was previously explored
for managing the power states of a processor [35]. The intuition that motivates the naïve mechanism is that if the user invoked some I/O that followed a mouse click it is probably important and should be served with least amount of the delay. On the other hand, network activity that is not correlated with, or immediately preceded by, mouse clicks is probably less important and can incur some delays without degrading the applications interactive performance. Therefore, the naïve all-click (AC) switching mechanism transitions the WNIC to CAM upon each click and transmits all network I/O that follows the last mouse click within some time interval in CAM, thereby minimizing delays by already being in the high-power mode before I/O arrives and serving data with higher throughput than was measured for PSM. Network I/O that is not preceded by mouse activity is served in PSM, saving energy. Once the network I/O is served, the WNIC is transitioned to PSM to conserve energy during idle periods.

Clearly, this mechanism has the effect of capturing and serving all I/O activity periods preceded by clicks in CAM, with the downside of transitioning the WNIC unnecessarily for clicks that are not followed by network I/O. It is important to note that the total number of clicks may greatly exceed the total number of I/O activity periods, since most of the clicks do not invoke network I/O. In addition, even if there is network activity following a click, it may be small and best served in PSM without the overheads of transition to CAM. The energy consumed in CAM following each switch, the energy required to make each switch, and the energy consumed during unnecessary switches make AC one of the least energy efficient mechanisms. The AC mechanism illustrates the need for more intelligent prediction schemes that decide when the WNIC should stay in PSM or transition to CAM for the upcoming I/O activity period.

5.1.2 Capturing GUI interactions

To address the shortcomings of the naïve predictor, which does not distinguish between different types of potential interactions with the application, a more sophisticated Interaction-Aware Predictor (IAP) approach is proposed, utilizing a detailed
context of user interactions to accurately predict WNIC transmission modes. Accurate and detailed monitoring of user activity forms the basis for IAP’s design. Virtually all interactions with common interactive applications in a GUI environment can be accomplished through mouse clicks [15]. While the capture of mouse click data, such as absolute screen coordinates of the event or type of click, is relatively trivial, capturing application-specific context using mouse clicks is more problematic. In order to uniquely identify the components of application GUIs that the user is interacting with, the monitoring layer between the X Window Server and applications in Linux that utilizes the GUI window structure is used to uniquely identify interactive components such as buttons and menu selections with an integer ID. Figure 5.1 shows the monitoring layer and the kernel structure used to record interaction IDs. This layer provides us with transparent user interaction monitoring, meaning that no modification of applications is necessary. Furthermore, all interaction IDs are obtained while the user is interacting with the applications, allowing for detection, correlation, and prediction to be performed without any offline processing. High prediction accuracy is achieved by use of the hierarchical trees of visible and non-visible windows that fully describe an application’s GUI. The structure of the window tree is the same across executions and is used to uniquely identify a particular event.
5.1.3 Advanced Prediction

Accurate and detailed description of the user interactions allows the IAP to distinguish between different types of user interactions with an application that generate various amounts of network I/O. The central part of the IAP is a prediction table that is organized as a hash table indexed by the unique click IDs. The prediction table stores a 2-bit counter that indicates the most appropriate WNIC mode for a given click ID. The table resides globally in the IAP daemon and is shared among processes to allow table reuse across multiple executions of the same application as well as concurrent executions of the same application. The table can be easily retained in the kernel across multiple executions of the application due to its small size [23, 22], and will reduce training in future invocations of an application.

A prediction table lookup is performed for every click, as shown in the Figure 5.2. The lookup results in three possible outcomes:

1. the entry is found and the 2-bit saturating counter indicates that the interaction leads to high levels of network activity (best served in CAM).

2. The entry is found and the 2-bit saturating counter indicates that the interaction leads to low or no network activity (best served by PSM).

3. The entry is not found, in which case a placeholder in the prediction table is created.

Based on the lookup outcome the IAP daemon switches the network interface to the appropriate state. The IAP daemon continues to monitor the network activity recording the number of bytes transferred and the time of the I/O activity for each process as shown in Figure 5.1. Arrival of a new click during the monitoring indicates that there is a new interaction with the application and the prediction table entry for the previous click has to be updated. Detection of a longer idle period indicates that the network activity initiated by the current click ceased, the WNIC can be transitioned from CAM to PSM and the prediction table entry for the given click can be updated.
The information about the given activity collected by the IAP daemon is used to update the 2-bit counter in the prediction table and recorded in the process table as shown in Figure 5.1. The IAP daemon calculates the relative time and energy cost of transmitting the observed network I/O in PSM and CAM relying on equations proposed in [2]. The energy cost of each mode includes:

1. The WNIC energy to transmit the data in a given mode.

2. The energy consumed in the idle state waiting for the end of the activity signaled by the arrival of the click or detection of a longer idle period.

3. The energy consumption of the overall system, since it has to remain on when the request is served.

The time cost of serving the observed activity in CAM includes the transfer time and the time to transition the WNIC from PSM to CAM. The time cost of serving the observed activity in PSM includes the transfer and the initial latency of receiving data, which is on average one-half of the beacon frequency. Transfer time in PSM is higher than in CAM because throughput is lower in PSM mode. Ignoring the switching latency, a request served in CAM will always complete earlier than if it were served in PSM.

The relative costs of time and energy for every predicted I/O period can be weighted for emphasis on either timely transitions or energy-efficient completion of the requests. This design considers a balanced approach of minimizing delays and energy for completing each activity period to minimize the energy-delay² product. The mode with the lowest energy-delay² product is selected as the best choice for completing the request and the 2-bit counter in the corresponding prediction table entry is updated. Energy-delay² emphasizes the delay reduction in interactive applications.

Finally, the necessary energy profiles for both the WNIC and the system were obtained prior to implementation, but it is envisioned that eventually, since energy management continues to increase in importance, the devices themselves will
provide energy profiles. In addition, the focus was on the PSM to CAM transitions, leaving the CAM to PSM transition to a simple timeout, making it much easier to directly compare all the proposed mechanisms with traditional and state-of-the-art approaches.

5.1.4 Mode Switching Heuristics

There are two decisions that IAP has to make upon retrieving the prediction from the table. First, what to do when there is no prediction for a given click. This occurs during training and becomes less important as time passes since there is only a limited number of ways the user can interact with the application. The second decision is when the prediction should be acted upon. IAP can immediately switch to CAM, if predicted to do so, or delay the switch till the network I/O arrives. Both of those questions trade aggressiveness of prediction with higher reduction in delay for higher accuracy with the potential for higher energy savings. As a result, several variations of the IAP approach are evaluated:

- IAP with Delay optimization (IAPD).
- IAPD with deferred switching/Waiting to switch (IAPDW).
The basic mechanisms for context capture and prediction are shared across all heuristics, but the actions taken upon prediction vary significantly from one to the next. An overview of each mechanism is contained in Table 5.1. Table 5.1 is a brief summary of switching policies, stating when the switch occurs, and the relative benefit for energy and delay reduction. Checkmarks in energy and delay savings columns compare the relative impact on energy and delay for each of the proposed mechanisms.
The goal of IAPD and IAPDW mechanisms is the reduction of transmission delays by serving network I/O in CAM by default and in PSM only when the prediction dictates to do so. IAPD and IAPDW switch the WNIC to CAM for:

- Mouse click IDs that have not been seen before;
- Mouse clicks for which IAPD is still training;
- Mouse clicks that result in CAM prediction;
- Any activity not preceded by mouse clicks.

The main difference between IAPD and IAPDW is that with IAPD transitions from PSM to CAM occur immediately upon prediction from a given mouse click, while IAPDW defers the same transition until network activity arrives following the click. IAPD therefore has the additional benefit of reducing switching delays. The WNIC
remains in PSM during idle periods and will remain in PSM after a click when these mechanisms predict that the I/O activity following the click is small and best served in PSM.

In the case of IAPD, delays are reduced at the cost of higher energy consumption, since IAPD switches for all clicks where prediction is uncertain. Both mechanisms incur additional energy consumption due to switching for unpredictable periods, which may be efficiently served in PSM. Figure 5.3 illustrates the switching differences for IAPD and IAPDW. In the example, IAPD switches the WNIC to CAM when click with ID 1 is seen the first time, but remains in PSM when ID 1 is subsequently seen again, since it was not previously followed by network I/O. When click with ID 2 arrives, the mechanism transitions the WNIC to CAM correctly, subsequently using the prediction to again transition the WNIC to CAM correctly when ID 2 is seen again. IAPDW waits for the network I/O to arrive and does not switch for clicks with ID 1 since they are not followed by the I/O but switches for the network activity following the click with ID 2. Furthermore, Figure 5.3 shows that IAPDW slightly delays the network transmissions due to switching delay.

**IAPE & IAPEW: Maximizing Energy Savings**

IAPE and IAPEW mechanisms differ from IAPD and IAPDW in how the predictions are utilized in transitioning the WNIC power modes. Specifically, where IAPD and IAPDW favor CAM for unpredicted or uncertain network I/O, IAPE and IAPEW default to PSM unless the prediction is that the upcoming network I/O is best served in CAM. Therefore, IAPE and IAPEW eliminate unnecessary transitions to CAM.
during unpredictable periods of low network I/O traffic that can be served efficiently in PSM. The resulting behavior maximizes energy savings at the cost of higher delays. As in IAPD, IAPE transitions the WNIC from PSM to CAM immediately upon prediction following a mouse click in order to reduce switching delays. Similarly to IAPDW and its relationship to IAPD, IAPEW is identical to its counterpart, IAPE, but the predicted transition from PSM to CAM is deferred until I/O activity arrives. IAPEW reduces unnecessary switches due to erroneous predictions and reduces the time WNIC is waiting in CAM for incoming I/O, making it the most energy conscious predictor. However, this is done at the cost of introducing switching delays for each predicted PSM to CAM transitions.

Figure 5.3 serves to further illustrate the differences between the proposed mechanisms. IAPE and IAPEW default to PSM when a prediction is lacking or uncertain. Therefore, there are no switches for the click with ID 1 and the first I/O period, since click with ID 2 has not been encountered previously. The next time ID 3 is seen, the prediction to switch to CAM is made correctly. Additionally, IAPEW, like IAPDW, does not immediately request that the WNIC transition to CAM, but does so only when the first network I/O is encountered following a prediction.

**IAPED: Minimizing Energy-Delay**

The challenge of designing energy efficient predictors is the balance between reducing delays and reducing energy consumption, which can be accomplished by minimizing the energy-delay\(^2\) product. A detailed evaluation of the mechanisms proposed in the previous sections found that IAPE and IAPEW consistently performed better in this respect. This implies that utilizing the CAM mode during training and transmissions not preceded by mouse clicks is less significant than minimizing delays and wrongful switches. As a result, an additional IAP mechanism is proposed minimizing the Energy-Delay\(^2\) (ED) product (IAPED). IAPE and IAPEW, the top performers in Figure 5.7, are included in the design of the IAPED predictor that dynamically selects between the combined predictors in an attempt to minimize the energy-delay\(^2\) product. Training and predictions are done according to the diagram shown
in Figure 5.2. The only additional step needed is deciding if waiting for the I/O or switching ahead of time is more beneficial for the energy-delay$^2$ product, given the activity period that has just ended. Energy-delay$^2$ calculations from each prediction mechanism are compared, and if switching ahead of time is more beneficial, an additional 4-bit saturating counter is incremented for each click where this is the case, otherwise it is decremented. Once the prediction to switch to CAM is made, the IAPED consults the 4-bit saturating counter for each click ID and decides if the switch should occur immediately or when I/O activity arrives.

The saturating counter is biased to IAPEW performance during training, since, as it will be shown in subsequent sections, IAPEW outperforms IAPE in terms of the energy-delay$^2$ product for all applications except Firefox. As a result, the initial value of the saturating counter is set to 0 and will require two incorrect choices for each individual mouse click ID before IAPED starts performing like IAPE since this can potentially reduce the energy-delay$^2$ improvement for applications where IAPEW is the best performer.

5.2 Methodology

A trace-driven simulation is used to accurately compare the various energy management techniques for WNICs. The following mechanisms were implemented and used for analysis and comparison:

1. Standard power modes: PSM and CAM;

2. Interaction-aware mechanisms: AC, IAPD, IAPDW, IAPE, IAPEW, and IAPED;

3. Existing state-of-the-art STPM mechanisms, according to the authors’ specifications [3];

4. Optimal power management mechanism that minimizes the energy-delay$^2$ product based on future knowledge (OPT);
5. Another reference predictor that switches the WNIC to CAM for each network I/O period (AN).

The AN mechanism shows one extreme of the spectrum where all I/O is served in CAM to minimize transmission delays. This mechanism is similar to the hardware PSM to CAM switching mechanism that switches the card to CAM if there is more than one packet waiting at the network interface. Through experimentation, it was observed that a timeout of 4 seconds used to delay the transition of the WNIC from CAM to PSM results in the best energy-delay$^2$ product for studied mechanisms.

Application traces are composed of two components: the mouse event trace, and the network activity trace. Mouse events are traced using the X monitoring layer. The trace includes the mouse interaction type, a timestamp, and the unique ID that identifies the component with which the user has interacted. Network activity traces are collected using a modified `strace` that captures the system call information from `send` and `recv` interfaces, such as type of request, timestamp, bytes send, and bytes received. The simulator relies on timestamps from each trace to order the incoming events.

Six applications commonly executed on desktop or portable systems are used for analysis:

- **Firefox** is a widely used web browser and represents the behavior of users surfing the web, reading news articles or downloading files among other activities. Browsing web pages generally results in a low network activity while downloading files will generate high bandwidth demand. Some page browsing activities may invoke external plug-ins to access media content within a page such as flash animations or movies that also may require high bandwidth.

- **Thunderbird** is an example of a mail application where users interact with the application to perform tasks such as sending, receiving, reading and composing email messages. These interactions generate varying amounts of network traffic due to sending or receiving emails with or without attachments. Transferring
attachments will usually require much higher bandwidth than transferring a plain text messages.

- **Gaim** is an Internet messaging application. It generates low bandwidth network traffic when users are sending text messages, while occasional file transfers performed by users require high bandwidth.

- **GFTP** is a file transfer client used to upload and download files between the client and a server. The traffic consists of low overhead directory listings and high overhead file transfers.

- **Pan** is a newsreader application, where user interactions result in low network traffic generated by connecting to a news server, getting lists of groups and message headers, getting messages, and posting messages.

- **DJGame** is an interactive online game that initiates network activity for each action within the game, with some actions leading to the transfer of information regarding user actions and others leading to the larger transfers of game-state related information.

Trace details such as trace duration, the amount of data transferred, and the number of network I/O activity periods representing opportunities for switching the WNIC power mode are included in Table 5.2. Also included are the numbers of user interactions, represented by mouse clicks. Unique click IDs serve as a measure of GUI complexity for each application. Applications with relatively simple static GUIs, such as Thunderbird or GFTP, have few unique click IDs, while those with rich interfaces have a somewhat higher number of unique IDs. Correlated click IDs are a subset of unique click IDs that belong to clicks that precede network activity and can be correlation to the network I/O.

5.3 Evaluation

The efficacy of the proposed mechanisms is evaluated along several dimensions. First, the prediction performance of the proposed predictors is considered. Second,
the delays and energy consumption of all mechanisms are compared in order to identify the various sources of performance degradation and energy consumption. Third, the relative energy-delay\(^2\) performance of each mechanism is considered, showing a combined view of predictor efficacy. Finally, the overheads of IAP mechanisms are compared to the state-of-the-art STPM mechanism.

5.3.1 Accuracy

Figure 5.4 shows a breakdown of prediction outcomes for the studied mechanisms and combines two related metrics: one for the correct CAM periods and the other for wrong switches, both normalized to AN. The Correct portion of the bar represents the number of correctly predicted CAM periods, while Missed Opportunity represents the number of CAM periods that were missed by the predictor and were subsequently served in PSM. Incorrect switches are divided into Inefficient and Unnecessary switches. The Unnecessary switches occur when the WNIC was switched to CAM upon a prediction but the WNIC did not serve any I/O and the card was transitioned back to PSM. Unnecessary switches waste energy without providing any benefit in reducing delays. Inefficient switches, on the other hand, occur when the WNIC switches to CAM and serves some I/O but the I/O activity is insufficient to justify serving it in CAM. Inefficient switches are less energy efficient but reduce
some delays by serving I/O in CAM. Therefore, they are less damaging to the resulting energy-delay\textsuperscript{2} product than Unnecessary switches. AN is used as the base for Figure 5.4 since it shows maximum potential for serving I/O periods in CAM and also the maximum amount of network I/Os that can be served incorrectly in CAM if more sophisticated prediction mechanisms are not employed. The term coverage here indicates the percentage of the correctly predicted CAM periods to the total CAM periods.

As shown in Figure 5.4, AC covers 87% of CAM periods on average, with the best case being 95% in GFTP. CAM period coverage is on average 77% for IAPD and IAPDW; and 74% for IAPE and IAPEW. There is no difference in the converges for the IAPDW and IAPEW as compared to their base cases since the prediction are the same, only the timing of switching is different. The average improvement of wrongful switches for IAP mechanisms over AN is on average 77% for IAPD, 79% for IAPE, 82% for IAPDW, 84% for IAPEW, and 82% for IAPED. The lower wrong switches in IAPDW and IAPEW mechanism are due to delaying switches after prediction. Slightly higher coverages for IAPD mechanisms are due to handling network I/O in CAM during initial training of the predicate when the mouse clicks are observed for the first time.

The naïve AC mechanism switches for all clicks but does not switch for the I/O not preceded by click. Switching for all clicks explains the higher coverage in Firefox, GFTP, and Pan, at a cost of much higher incorrect switches. I/O periods that were not immediately preceded by clicks are served in PSM, since according to the intuition behind AC design they are not latency sensitive. This explains slightly lower coverage in case of Gaim and DJGame for AC. The periods that are not immediately preceded by clicks are found in Thunderbird, Gaim and DJGame and are due to delays generated by the server providing the data. Those delays can result in the fragmentation of long network I/O periods into several smaller network I/O periods. This delay results in the CAM to PSM transition timeout to expire and AC serves the remaining network I/O in PSM. IAP mechanisms generate persistent predictions that will serve any I/O in the last predicted state until a new
prediction is generated at the next mouse click. In this way, fragmented periods are served in a correct WNIC mode, albeit incurring a small amount of switching delay for each switch occurring at the arrival of a new I/O period fragment, if predicted mode of the WNIC is CAM. This policy is responsible for better coverage by IAP mechanisms by an average of 6% in Gaim and 5% in DJGame.

Lower coverages in Firefox, GFTP and Pan are due to variability in transfers following certain mouse clicks. Those applications are dominated by the PSM periods where most interactions are efficiently handled in PSM. IAP mechanisms are able to correctly handle PSM periods by having very small fraction of periods that should be handled in PSM but are instead handled in CAM. Occasional interactions that usually result in low bandwidth demand may require higher bandwidth to accomplish, resulting in the predictors missing an occasional CAM switch. Those scenarios are observed in Firefox where fetching pages usually requires low bandwidth, while some pages contain rich graphical content with high resolution graphics occasionally requiring higher bandwidth to load them. Similarly, GFTP requires high bandwidth for file transfers and usually low bandwidth for retrieving directory listings. However, some large directories will require higher bandwidth to transmit a very long list of file names with corresponding information. Finally, Pan retrieves a majority of messages without any attachments, generally requiring low bandwidth that is accurately handled in PSM. Occasional downloads of the message with an attachment will result in lower coverage for the CAM periods.

The goal of the IAP design is to handle I/O periods accurately to minimize energy-delay$^2$ product. Much higher accuracy is achieved at the cost of reduced CAM coverage, but the ultimate benefit to reduction of energy-delay$^2$ product is achieved. This is illustrated in case of AN and AC with both mechanisms having higher coverage of CAM periods than IAP mechanisms. However, both AN and AC mechanism are plagued by incorrect handling of PSM periods, where IAP mechanisms on average improve wrongful switches by 80%. 
5.3.2 Delay

Figure 5.5 breaks down the various delays incurred by each mechanism. The three types of delay shown are Transfer, Beacon, and Switch delay. Since CAM incurs minimum delays, the delay of each mechanism is computed relative to CAM. Transfer delay occurs whenever data is served in PSM and is due to the lower throughput rates of PSM. Beacon delays are caused by the delays that occur due to the WNIC downtime between transmitting beacon frames, where serving data following a beacon is delayed by the time remaining before the next beacon arrives. Switch delay is the delay incurred by transitioning the WNIC power modes and is equal to the time to transition the card from PSM to CAM or vice versa. All results are normalized to the PSM delay, showing the deficiencies of the existing hardware mechanisms.

The delay that results in maximum energy efficiency is shown by the energy-delay\(^2\) optimal mechanism, shown as OPT in Figure 5.5. OPT does not have any switching delay since it transitions the WNIC to CAM prior to the arrival of I/O activity. The delay incurred by OPT is mainly due to serving periods in PSM, that minimize energy-delay\(^2\) product. Similarly to OPT, AC eliminates switching delay by switching upon a mouse click, ahead of network I/O. However, lower than optimal transfer delay indicate that AC sacrifices energy efficiency at the cost of
lower delay. AN eliminates transfer delay altogether since it is serving all I/O in CAM. However, it switches from PSM to CAM when I/O arrives, encountering switching delays for the I/O periods that could be served more efficiently in PSM without a switch. Delays, due to the decision-making process in STPM, result in the network interface serving more I/O in PSM, encountering transmission delays even for the periods that are later switched to be served in CAM. Regardless, STPM improves delay by 59% with respect to PSM. On average IAP mechanisms further improve delay over PSM by 71% for IAPD, 69% for IAPE, 62% for IAPDW, 61% for IAPEW, and 63% for IAPED. It is important to note that the optimization of energy-delay² product by each successive variation of the IAP mechanisms results in increasing delay as this is the cost of reducing the energy consumed by incorrect or early switches. IAPED shows the result of the interplay of the two mechanisms that have been combined.

The IAPDW and IAPEW mechanisms were proposed to reduce the energy consumed by unnecessary switches and also reduce the time spent in CAM waiting for the I/O to arrive. Therefore, the only difference for those mechanisms is visible in switching delay, since both IAPDW and IAPEW wait for the I/O to arrive before switching the WNIC mode. Firefox shows a significant difference in delay between IAPD and IAPDW, as well as between IAPE and IAPEW, that is driven by switching delay. Switching the WNIC to CAM when the prediction is made, as in the case of IAPD and IAPE, results in a lower switching delay in situations where the transition prior to network activity does not time out before I/O arrives. The switching delay that is present in the case of IAPD and IAPE is due to persistent predictions transitioning the WNIC to CAM once I/O activity arrives after the idle time threshold and the card is switched again from PSM to CAM. For example, the deferred switches comprise about 16% of all switches in Firefox for IAPD and IAPE.

Finally, variations in transfer, switching, and beacon delays across the applications are due to the composition of an application’s I/O activity. Applications with a large amount of low bandwidth activity will incur higher transmission delays as
compared to CAM, since those periods are more efficiently served in PSM. The same applications will also incur beaconing delays due to the PSM mode. The example of this behavior in Figure 5.5 is Pan with 96% of all periods that are best served in PSM. Applications with relatively high number of CAM periods or activity that is hard to predict may incur a higher fraction of switching delay, with Firefox serving as an example.

5.3.3 Energy

Energy consumption in Figure 5.6 is divided into four components: PSM idle, CAM idle, switching, and transfer energy. PSM idle energy is consumed by the WNIC when it is in PSM and there is no I/O transfer occurring. It includes energy consumed during beaconing as well as the base PSM energy. CAM idle energy is the energy consumed when the WNIC is in CAM but not transferring data. Switching energy is the energy consumed by the WNIC to make the transition from CAM to PSM and vice versa. Finally, transfer energy is the energy consumed while transferring data, either while in CAM or PSM. Similarly to Figure 5.5, all results are normalized to the PSM energy consumption.

PSM on average consumes the least amount of energy, at the cost of increased delays as shown in Figure 5.5. Energy consumption below the optimal level again
suggests that the mechanism is not efficient in terms of the energy-delay\textsuperscript{2} product. The most energy-hungry mechanism is either AN or AC, depending on the application. The behavior of AN and AC, shown in Figure 5.6, mirrors their behavior in Figure 5.4, showing that the large number of incorrect switches is responsible for the increased energy consumption due to switching, and subsequently the energy consumed waiting for the timeout to expire before switching back to PSM. Therefore, accuracy of the prediction is critical to energy consumption. Higher switching accuracy of STPM, which only switches when it encounters network I/O with sufficiently high bandwidth demand, results in much better energy efficiency than the naïve AC and AN mechanisms. Higher accuracy in IAP mechanisms also translates to better energy efficiency. On average, STPM consumes 9% more energy than OPT, IAPD 16%, IAPE 13%, IAPDW 4%, and IAPEW and IAPED only 3% more.

Energy consumption due to data transfers is similar across mechanisms for a given application since all mechanisms have to transfer the same amount of data and transfers in CAM and PSM use approximately the same power, as shown in Table 2.1. The largest difference is observed in GFTP, which can be attributed to large transfers which when performed in PSM take more time and as a result consume more energy. This difference stands out most in GFTP since, on average 71% of energy consumed in GFTP was due to transferring large amounts of data, which is four times more than the average of the remaining applications. The smallest energy consumption during transfers occurred in Firefox with an average of 10% of total energy, while PSM idle makes up the bulk of energy consumption at an average of 61%. As a result, those differences in transfer delays are less prominent in other applications.

The largest differences among energy management mechanism are visible in switching energy and CAM idle energy. Every erroneous switch increases switching energy consumption and idle CAM energy consumption while waiting for I/O to arrive and idle timeout to expire. Furthermore, even correct switches can contribute to idle energy consumption in the case when they have to wait for I/O to arrive. IAPE consumes less energy than IAPD, since it has fewer erroneous switches as
shown in Figure 5.4. The energy consumed while waiting for I/O to arrive is further reduced in IAPDW and IAPEW since those mechanisms wait for I/O to arrive before switching to CAM. The only energy consumed in idle CAM state in IAPDW and IAPEW is due to timeout interval after an I/O has been served and before the WNIC was transitioned to PSM.

5.3.4 Energy-Delay$^2$

The final metric in evaluating each of the energy management mechanisms is the energy-delay$^2$ product. The energy-delay$^2$ product is an established metric that takes into account both performance degradation and energy consumption, with an emphasis on performance degradation, due to the interactive applications’ requirements for low-delay operation. Energy-delay$^2$ provides a single metric and therefore is useful for visualizing the tradeoff between energy consumption and performance.
Figure 5.7 shows the energy-delay\(^2\) product for each mechanism as normalized to OPT. OPT energy-delay\(^2\) product is at 1 and the increases in energy-delay\(^2\) product as compared to OPT are above 1. Computing the energy-delay\(^2\) product uses the total WNIC energy consumption for each of the traced applications using each of the described mechanisms and total transmission time, which includes all delays and time to transmit data, but does not include the idling time, which may vary according to user behavior. For additional reference, see Table 5.3, which shows the transmission time and energy, in seconds and Joules, respectively, for OPT, PSM, STPM, and IAPED.

The IAPED mechanism outperforms AC and AN by an average of 75% and 60%, respectively, and is 11% lower than STPM. Of the IAP mechanisms, IAPD and IAPE are generally the weaker performers, with an average energy-delay\(^2\) product that is 31% and 26% higher than OPT, respectively, due to larger number of erroneous switches and resulting higher energy consumption. The exception is Firefox where the trend is reversed, where IAPD and IAPE perform better than their energy-optimizing counterparts do. Firefox overall exhibits a larger amount of activity than the other applications. In this case, the switching delay incurred by IAPEW and IAPDW is largely eliminated by the early switching of IAPE and IAPD. In Thunderbird and DJGame the pronounced differences between IAPD and IAPDW, as well as IAPE and IAPEW, are due to long delays between correlated transmissions. While the energy consumed by IAPE is significantly higher, by 17% over IAPEW, for example, the savings in delay by IAPE are not significant due to a relatively small number of total switches in these applications. Finally, Pan is a PSM dominant application, therefore the performance of IAP mechanisms is comparable to PSM.

5.3.5 Overheads

A large amount of computational or storage overhead can reduce or eliminate the gains made by implementing energy management for a single system component. For example, high computational overhead results in more energy being used by the
Table 5.4: Mechanism overheads in seconds.

<table>
<thead>
<tr>
<th>Apps</th>
<th>STPM</th>
<th>IAPED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox</td>
<td>22</td>
<td>.08</td>
</tr>
<tr>
<td>Thunderbird</td>
<td>8</td>
<td>.05</td>
</tr>
<tr>
<td>GFTP</td>
<td>3</td>
<td>.23</td>
</tr>
<tr>
<td>Gaim</td>
<td>3</td>
<td>.02</td>
</tr>
<tr>
<td>Pan</td>
<td>4</td>
<td>.02</td>
</tr>
<tr>
<td>DJGame</td>
<td>5</td>
<td>.03</td>
</tr>
</tbody>
</table>

CPU. This additional energy that is not normally consumed by the CPU can result in more energy being consumed to run the predictor than is saved by the predictor in the managed component. STPM is a relatively high cost prediction mechanism, due to having to compute probabilities for upcoming activity from histograms describing prior activity. In simulations, STPM is implemented as described by its authors, including a 10-minute interval between recomputing of probability lookup tables. The update of probability tables used to determine the likelihood of upcoming activity is a high-cost operation which, when performed for each run, results in additional computational overhead without significantly benefiting the resulting mechanism accuracy.

The overheads shown in Table 5.4 can be decreased for STPM at the cost of lower prediction accuracy. The computational overhead shown in Table 5.4 is captured by a timed execution of the simulation of each mechanism as it evaluates the application traces. Due to the predictions being made once for each mouse event rather than once for each incoming I/O request, the overheads for the IAPED mechanism are obviously lower. Compared to STPM, the IAPED overhead is between 99% and 98% less.

The computational overhead of all IAP mechanisms comes from the computation of unique IDs used to index into the prediction table. Firefox has the deepest tree of 27 levels in the studied applications. One experiment measured the average overhead of traversing 27 level of tree hierarchy that is present in Firefox. The overhead of calculating the ID in this case is less than .8ms. Currently, each mouse click results in a communication overhead as the application window tree is built.
through use of X Server requests, so this overhead can be reduced by use of persistent tree representations of the application’s GUI. Furthermore, ID calculations and subsequent prediction table lookup are performed once per click, meaning that they need to be performed as infrequently as the user interactions occur.

Finally, the storage overhead is relatively small in IAP mechanisms. The IAP daemon has to maintain prediction entries, which ranged from 32 in Thunderbird to 128 in Firefox. Each table entry is composed of five fields containing a single unsigned integer each. Therefore, in the worst case Firefox requires 2.5KB to store 128 entries. A 6.8KB table would suffice for storing all entries from every one of the six traced applications. This relatively small overhead can be further reduced, since only a small number of the entries contain information about clicks that lead to network I/O. As a result, less useful entries can be aged and removed using a simple replacement mechanism such as LRU.
CHAPTER 6

CONCLUSIONS AND FUTURE DIRECTIONS

6.1 Context

Chapter 3 described the implementation of a low-overhead context capture tool the X Window System in Linux. Compared to the state of the art interaction capture mechanism employing k-means clustering and nearest neighbor calculations, the described mechanisms result in exact identification of the visible interactive elements. Additionally, the approach is transparent to the user, requires no offline processing, and has minimal overheads, requiring storage in the range of several kilobytes and exhibiting computational delays well below the 50ms interactive delay regarded as acceptable to the user.

Applying context capture in Windows demonstrated that predictive target selection (PTS) can be performed transparently in existing windowing environments and with unmodified applications. Facilitated pointing in the common cascading pull-down menus is shown to be possible with low overheads, but remarkably accurate, mechanisms. The distance to acquire targets, i.e. options contained in pull-down menus, is reduced or, in almost 70% of menu interactions, eliminated by predictive mouse cursor jumps. The section proposed a range of PTS mechanisms, ranging from simple heuristics to probabilistic predictors, and presented their respective benefits and drawbacks. The proposed mechanisms coupled with cursor jumps are capable of reducing the average menu traversal distance by as much as 83%. Applying the same methods to menu reorganization can yield similar improvements to menu navigation time.
6.2 Disk Delays

Chapter 4 proposed two disk spin-up mechanisms: ACSU that simply keeps the disk powered when users are interacting with the application and IASP that accurately and efficiently monitors user behavior. Both mechanisms reduce interactive delays exposed to the users due to energy management in hard disk drives. Hard disks contribute significantly to the overall energy consumption in computer systems. Therefore, aggressive energy management techniques attempt to maintain the hard drive in a low power state as much as possible exposing long latency spin-up delays to the users. Reducing the spin-up delays provides twofold benefit. First, the users are less irritated by constant lags in the responsiveness of the system due to disk spin-ups. Second, shorter delays allow the system to accomplish tasks quicker resulting in less energy being consumed by other components that have to wait for the disk spin-up.

The key observation used in the design is that users are responsible for the demand placed on the system through interactive applications. Therefore, monitoring user interaction patterns with applications provides the opportunity for predicting I/O requests that follow the interactions and use this to spin-up the disk ahead of time, reducing delays. Evaluation of the proposed ACSU and IASP shows significant improvement over ALT+ mechanisms in terms of predicting upcoming disk I/O activity and thereby shortening the interactive delay associated with energy management. ALT+ mechanisms are not able to accurately predict I/O activity in interactive applications resulting in an average misprediction rate of 25% that increases energy consumption in the system without providing any benefit of reducing delays with only 7% of periods correctly predicted spin-ups, on average. ACSU mechanisms are very aggressive and achieve 81% of accurate predictions that reduce delays at a cost of 52% misprediction rate. As a result, ACSU is able to reduce spin-up delays on average by over 60% (over 5 seconds), albeit at the cost high energy consumption. Finally, IASP is much more accurate since it is monitors user interactions. IASP on average achieves 79% of accurate predictions that reduce delays
with only 2% of mispredictions. As a result, IASP is able to reduce spin-up delays on average by 35% (over 3 seconds), while maintaining low energy consumption.

The primary goal of Chapter 4 was to reduce interactive delays due to disk spin-up exposed to the users, while maintaining the energy efficiency of the shutdown mechanism. Spin-up mechanisms do not reduce energy consumption of the individual device, however they have a side effect of making the system more energy efficient by accomplishing tasks quicker and reducing the energy consumed by the system waiting for the disk to spin-up.

6.3 WNIC Energy

Chapter 5 proposed a new direction for designing network resource management predictors. While current predictors monitor low-level application behavior such as network activity or sometimes reach as high as monitoring the application systems calls, proposed is the next step in monitoring by leveraging the key component responsible for the behavior of interactive applications, the user. Hence, Chapter 5 proposed and successfully implemented several user-interaction-aware energy management mechanisms that dynamically learn the context of user interactions with respect to network activity in interactive applications.

Other recent resource management solutions could potentially perform better with hints from modified applications, but for unmodified applications they rely on monitoring low-level activity. Application modifications are impractical due to the additional burden that is placed on programmers, therefore the proposed mechanisms provide a transparent solution that provides high energy efficiency without the need for application modifications. More importantly, the proposed mechanisms are readily implementable in existing systems due to: (1) the simplicity of the proposed mechanism which monitors and correlates user behavior with system activity, (2) quantifiable, low computational and storage overheads, (3) on-line monitoring and prediction that does not require application modification or offline processing for the analysis of user interactions.
6.4 Future Directions

Virtualized environments are becoming commonplace on all platforms, ranging from high-end server systems to smart phones. In virtualized environments, a hypervisor runs either on bare hardware or on top of an operating system, providing an abstraction of the underlying hardware to one or more operating systems coexisting on the same computer. The advantages of virtual machines are strong isolation between each coexisting operating system, an instruction set architecture that may differ from the underlying hardware, and ease of maintenance, provisioning, and recovery. However, the primary disadvantage of running operating systems within a virtualized environment is that access to the real hardware is made inefficient. In particular, scheduling hardware performance/energy modes becomes difficult due to the isolation between the coexisting operating systems. For example, when one running operating system requests that the disk be shut down due to a long period of idleness, it may affect other systems that may be making disk I/O requests or the disk may not end up in the desired power mode due to activity from other coexisting operating systems. Future work will examine the applicability of context-aware energy management mechanisms in virtualized environments without compromising the essential isolation of guest operating systems and the underlying hardware and while maintaining the above-listed advantages.

Severely energy-constrained environments, i.e. smart phones, nevertheless feature PC-like functionality, combining 3G, wi-fi and bluetooth connectivity, a powerful processor, and significant storage capacity. However, due to their small form-factor, interaction types are limited, potentially lending the system to highly accurate correlation between user interactions and device-level activity. Future work will explore the opportunities for energy-efficient performance in these constrained platforms. This is an important direction due to the increasing reliance on smart phones for emerging applications, such as health care, where the ease of implementation of patient monitoring systems on fully-featured smart phones still outweighs the cost of creating single-purpose hardware for each monitoring application. Considering
that patient monitoring involves collecting either vital function data or information about the patient’s interaction with the physical surroundings, maintaining an awareness of the type of activity, e.g. continuous vs. sporadic or high-throughput vs. low-throughput, is useful contextual information that may be used for managing the power states of the on-board devices used for communicating or processing the collected data.

A further direction is in the area of user interface design. The proposed interaction capture system could be extended to provide detailed usage information that can help identify design inefficiencies, such as overly-nested yet commonly accessed functionality, and aid in the refinement of interface design. Furthermore, similarly to the proposed facilitated cascading pull-down menu navigation, the simplified menus in the smart phone interfaces can benefit from predictive menu navigation.

Further facilitated pointing improvements are possible with the use of more advanced prediction mechanisms. Facilitated pointing in unmodified environments is an area that can benefit from the application of advanced learning algorithms that can move beyond simply monitoring menu activity and generate predictions about likely future actions based on application-wide activity. Current exploration involves the application of facilitated pointing prediction in reconfigurable menus, determining the optimal size for the reconfigurable menu region and applying both the mechanisms described here as well as more advanced techniques to populating the reconfigurable region.

Additionally, since the operation of the interaction capture systems described in this dissertation does not depend on modified applications, the applications of the system go beyond the described facilitated pointing. The ability to monitor interactions in great detail and without application modifications lends the system to applications in UI testing, determining the distance traversed by the user under a common usage scenario. Such monitoring can yield menus that are preconfigured to reduce the time spent in menu navigation and determine the logical menu configurations that improve the interactive experience.
REFERENCES


