KernTune:

Self-tuning Linux Kernel Performance

Using Support Vector Machines

by

Long Yi

A thesis submitted in partial fulfillment of the requirements
for the degree of Magister of Computer Science
in the Department of Computer Science, University of the Western Cape

Supervisor: James Connan

November 15, 2006

Table of Contents

List of Tables . . . . . **v**

List of Figures . . . . . **vi**

Keywords . . . . . **vii**

Abstract . . . . . **viii**

Declaration . . . . . **ix**

Acknowledgements . . . . . **x**

Glossary . . . . . **xi**

Chapter 1 Introduction **1**

1.1 Background and Motivation 1

1.2 Research Problem 2

1.3 Research Aim 2

1.4 Approach and Methodology 2

1.5 Thesis Outline 3

1.6 Summary 3

Chapter 2 SVM Classification and Prediction **4**

2.1 Review of SVMs 4

2.1.1 Linear SVM 6

2.1.2 Non-linear SVM 8

2.2 SVM Classification 12

2.3 The SVM Tool LibSVM 13

2.3.1 Converting data 14

2.3.2 Scaling data 15

2.3.3 Selecting a kernel 15

UNIVERSITY of the
WESTERN CAPE

2.3.4 Setting kernel parameters 16
2.3.5 Training and predicting 17
2.4 Summary 17

Chapter 3 Linux Kernel Optimisation 18
3.1 Operating system optimisation and GNU/Linux 18
3.2 Overview of Linux kernel 19
3.3 Exploring `/proc` 22
3.4 Kernel Optimisation 26
3.4.1 Overview of the Kernel Parameters 26
3.4.2 Using `sysctl` 30
3.5 Summary 32

Chapter 4 Approach and Methodology 33
4.1 Understanding System Classes 33
4.2 The Monitor-Classify-Adjust Cycle 38
4.2.1 Monitoring a system 39
4.2.2 Classifying a System Class 43
4.2.3 Adjusting System Parameters 46
4.3 Related Work 50
4.4 Summary 51

Chapter 5 System Implementation 52
5.1 Design goals for KernTune 53
5.2 Overview of KernTune 54
5.3 Data Collection: *The monitor* 55
5.4 Data Processing: *The classifier* 57
5.5 System Tuning: *The adjustor* 58
5.6 Summary 60

Chapter 6 Experimental Results and Conclusions 61
6.1 Experimental Details 61

UNIVERSITY of the
WESTERN CAPE

6.2 Classification Results 66
6.3 Performance Results 67
6.4 Conclusions 70
6.5 Future Work 71
6.6 Summary 72

Bibliography **73**

Appendix A `/proc` **file-system** **76**

List of Tables

4.1	Performance factors	37
4.2	Performance objects	40
4.3	Performance counters	42
4.4	Tuned values for web servers	46
4.5	Tuned values for ftp servers	48
4.6	Tuned values for database servers	49
6.1	Machine	62
6.2	Gentoo Server	62
6.3	SUSE Workload Generator	62
6.4	Workload Simulation	63
6.5	Sample Training Set	64
6.6	Workload Statistic	65
6.7	Classification Results	66
6.8	Overhead of KernTune	66
6.9	Performance Results (10 times)	67
6.10	Web Server Results (10 times)	68
6.11	Ftp Server Results (10 times)	69
6.12	Database Server Results (10 times)	70

UNIVERSITY of the WESTERN CAPE

List of Figures

Figure 2.1 A linear / non-linear classification 5
Figure 2.2 Mapping from input space to feature space 6
Figure 2.3 A linear SVM 7
Figure 2.4 The kernel mapping 9
Figure 2.5 An classification example generated by pcSVMDemo 10
Figure 2.6 The non-linear SVM classification with linear kernel 11
Figure 2.7 The non-linear SVM classification with polynomial kernel . . . 11
Figure 2.8 The non-linear SVM classification with RBF kernel 12
Figure 2.9 The example of training data from the `heart-scale` file . . . 15

Figure 3.1 The structure of a GNU/Linux operating system 19
Figure 3.2 The components of the kernel 21
Figure 3.3 The content of `/proc` 23

Figure 4.1 One server serves users with different requirements 34
Figure 4.2 The *monitor-classify-adjust* (MCA) cycle 38
Figure 4.3 The training set used by KernTune 45

Figure 5.1 Overview of KernTune 54
Figure 5.2 The structure of monitoring 56
Figure 5.3 The structure of classifying 57
Figure 5.4 The structure of adjusting 59

Figure 6.1 KernTune Test Bed 63
Figure 6.2 Accuracy of Classification 67
Figure 6.3 Overhead of Classification 69

UNIVERSITY of the WESTERN CAPE

Keywords

Support Vector Machine
Linux Kernel
Operating System
Optimisation
Performance
Benchmark
Machine Learning
Workload
Open Source
System Profiler

Abstract

Self-tuning has been an elusive goal for operating systems and is becoming a pressing issue for modern operating systems. Well-trained system administrators are able to tune an operating system to achieve better system performance for a specific system class. Unfortunately, the system class can change when the running applications change. Our model for self-tuning operating system is based on a monitor-classify-adjust loop. The idea of this loop is to continuously monitor certain performance metrics, and whenever these change, the system determines the new system class and dynamically adjusts tuning parameters for this new class. This thesis describes KernTune, a prototype tool that identifies the system class and improves system performance automatically. A key aspect of KernTune is the notion of Artificial Intelligence (AI) oriented performance tuning. It uses a support vector machine (SVM) to identify the system class, and tunes the operating system for that specific system class. This thesis presents design and implementation details for KernTune. It shows how KernTune identifies a system class and tunes the operating system for improved performance.

UNIVERSITY of the
WESTERN CAPE

Declaration

I declare that *KernTune: Self-tuning Linux Kernel Performance Using Support Vector Machines* is my own work, that it has not been submitted for any degree or examination in any other university, and that all the sources I have used or quoted have been indicated and acknowledged by complete references.

Signed

Date

Acknowledgements

I wish to convey my greatest gratitude to my supervisor Mr. James Connan. It is he who has guided me all the way in my master's study in South Africa. Many thanks to my faculty staff Prof. IM Venter, Ms. Verna Connan and Ms. Rene Abbott for their selfless assistance. My lab colleagues and friends Pavan Rallabandi, Wilson Wu are truly worthy of my love and thanks because they give me good advice and most constructive criticism. Last, but certainly not least, thousands of hugs to my family. They never made light of my difficulties in accomplishing this study, and indeed, are endlessly patient and concerned about me and my work.

Glossary

AGP A bus specification by Intel which gives low-cost 3D graphics cards faster access to main memory on personal computers than the usual PCI bus.

AGP Accelerated Graphics Port. A bus specification by Intel which gives low-cost 3D graphics cards faster access to main memory on personal computers than the usual PCI bus.

DHCP Dynamic Host Configuration Protocol. A protocol that provides a means to dynamically allocate IP addresses to computers on a local area network.

DNS A general-purpose distributed, replicated, data query service chiefly used on Internet for translating hostnames into Internet addresses.

UNIVERSITY of the
WESTERN CAPE

FTP A client-server protocol which allows a user on one computer to transfer files to and from another computer over a TCP/IP network. Also the client program the user executes to transfer files.

I/O Communication between a computer and its users, its storage devices, other computers (via a network) or the outside world.

IDE Integrated Drive Electronics.

IPC Exchange of data between one process and another, either within the same computer or over a network. It implies a protocol that guarantees a response to a request.

IRQ Interrupt request. The name of an input found on many processors which causes the processor to suspend normal instruction execution temporarily and to start executing an interrupt handler routine.

IRQ Interrupt request. The name of an input found on many processors which causes the processor to suspend normal instruction execution temporarily and to start executing an interrupt handler routine.

PCI A standard for connecting peripherals to a personal computer, designed by Intel and released around Autumn 1993.

PCI Peripheral Component Interconnect. A standard for connecting peripherals to a personal computer, designed by Intel and released around Autumn 1993.

UNIVERSITY of the
WESTERN CAPE

SMP Symmetric Multiprocessing is a multiprocessor computer architecture where two or more identical processors are connected to a single shared main memory.

TCP Transmission Control Protocol. The most common transport layer protocol used on Ethernet and the Internet.

UDP User Datagram Protocol Internet standard network layer, transport layer and session layer protocols which provide simple but unreliable datagram services.

Chapter 1

Introduction

In this chapter, we provide the motivation and background behind the automatic optimisation of an operating system. We begin with a discussion of some of the difficulties of automatic operating system optimisation and the benefits of automatic optimisation technology which inspired our research. We then describe the research problem and aims. Thereafter, our approach and methodology are explained. Finally, the organisation of the thesis and summary are presented.

UNIVERSITY of the
WESTERN CAPE

1.1 Background and Motivation

In today's networking world, a mission-critical server requires consistently good performance [2]. To this end, almost all operating systems which run on such a critical server are managed by system administrators who should be skillful and experienced in tuning operating systems by adjusting system configuration and performance parameters of the operating system to run a specific system workload. This involves system capacity planning, performance metrics, workload characteristics, system settings, etc. Skillful system administrators are scarce and expensive. As computer hardware becomes cheaper and free critical computer software becomes more viable, e.g., Linux, Samba, Mysql, Apache, the total cost of ownership for building and maintaining a mission-critical server becomes more and more dominated by the cost of human resources. Furthermore, with the increasing number of new applications and services, a modern operating system offers more system parameters with larger ranges for more system classes than ever before. This situation serves as our motivation for a new generation of automatic optimisation technology for operating systems. The potential benefits of the automatic optimisation technology will be amplified as future applications and operating systems become more complex.

1.2 Research Problem

Operating System optimisation is often regarded as a black art [3]: adjusting system configuration settings and operational parameters of an operating system for a specific system class, in order to achieve the best possible performance. Many operating systems offer many parameters for this purpose, there are typically 5 to 50 tunable system parameters just for adjusting computer memory behaviour. However, virtually no help is provided in choosing the settings of these parameters in an intelligent way. The default settings are often meaningless for various system workloads, and product manuals are usually hard to read and do not give sufficient guidelines. Frequently, it is possible for a well-trained system administrator to tune the operating system using some so-called "rules of thumb", but these rules must by no means be applied blindly as their validity may depend critically on specific system workloads and system classes. Therefore, an automatic tool with a knowledgeable library or self-learning capacity is required for doing such optimisation. Our main question addressed in our research is: "How to automatically adjust the system parameters of an operating system for various system workloads in an intelligent way?"

UNIVERSITY of the
WESTERN CAPE

1.3 Research Aim

The aim of our research is to apply machine learning to build a performance tool that permits the automatic adjustment of system parameters with little or no intervention from system administrators or users. We have developed a *monitor-classify-adjust* loop control model to achieve this objective.

1.4 Approach and Methodology

We have developed a methodology for the automatic adjustment of system parameters by a *monitor-classify-adjust* loop control which employs support vector machine (SVM) technology [8] to determine system classes dynamically. The classifying module in the loop control automatically builds a system class classifier by learning a pre-classified set of classes. The method we propose here is inspired by recent work in Linux kernel optimisation: "Tuning the kernel with a genetic algorithm", by Jake Moilane [16]. His idea is to "generate a 'population' defined with unique strings of

'chromosomes', to test each of these chromosome strings for 'fitness', to select a subset of the chromosome strings with the best fitness and use them to create new chromosomes, to apply random mutation to a small subset, and finally to start the process all over again. Over time, all the chromosomes should 'evolve' toward having the best possible fitness, as defined by the algorithm." Our method differs from Moilane's by not evolving the solution over time. Instead, we have predefined an optimal set of system settings for each possible system class [17, 21, 23, 24, 9, 14]. The system is monitored periodically and when the SVM detects a change in system class, the approriate optimisations for that class are applied.

1.5 Thesis Outline

The thesis is structured as follows. The next chapter reviews SVMs and describes basic information and background of SVMs. Following that, in Chapter 3, we explore the `/proc` file-system and present optimisation techniques for the Linux kernel. In Chapter 4, we present our approach and methodology. Next, we describe our implementation details. In the last chapter, we present our experimental results and conclusions.

1.6 Summary

Operating system automatic optimisation aims to adjust system settings and operational system parameters to a specific system class automatically in order to achieve the best possible performance. The way to achieve this aim is studied in this thesis. This chapter mainly overviews the background, motivation, problem and approach of the research in turn. Background knowledge of support vector machines will be reviewed in the next chapter.

Chapter 2

SVM Classification and Prediction

In the previous chapter, the main concerns were outlined to clarify the research problem, methodology and aim. Literature relevant to the research on SVM classification will be reviewed in this chapter. We first briefly introduce basic ideas behind SVM classification. Then we discuss SVM classification and prediction. Next, we present LibSVM, a popular open source tool for SVM classification and regression. Finally, we present a summary.

UNIVERSITY of the
WESTERN CAPE

2.1 Review of SVMs

In the last few years, there has been a surge of interest in Support Vector Machines (SVMs), a new generation learning system based on recent advances in statistical learning theory. SVMs have empirically been shown to deliver good generalisation performance in real-world applications such as text categorisation [15], hand-written character recognition [19], image classification [18], object detection [20], speaker identification [30], etc. SVMs show a competitive performance on problems where data is sparse (few data) and noisy (many features). SVMs were first introduced in the early 1990s by Vapnik [29] as a binary classification tool and are rapidly growing in popularity due to many attractive features, and promising empirical performance.

Support Vector Machines (SVMs) are a set of related supervised learning methods used for classification and regression. Their common factor is the use of a technique known as the "kernel trick" to apply linear classification techniques to non-linear classification problems.

Suppose we want to classify some data points into two classes. We are interested in whether we can separate them by a *hyperplane*. A hyperplane is a geometrical concept. It is a generalisation of the concept of a plane. We also want to choose a hyperplane that separates the data points clearly, with maximum distance to the closest data point from both classes, this distance is called the *margin*. We desire this margin as

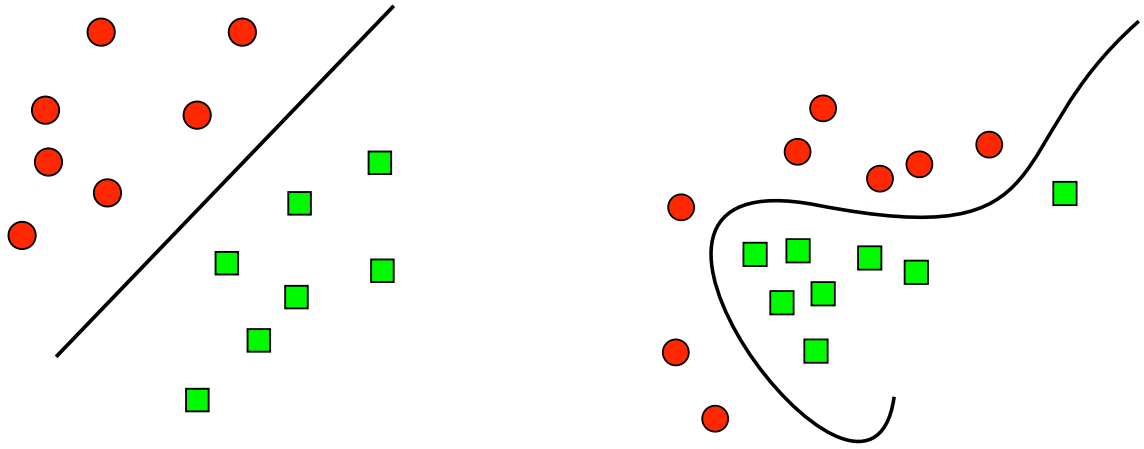

Figure 2.1: (a) A linear classification (b) A non-linear classification

large as possible since we can more accurately classify a new point when the separation between the two classes is greater. If such a hyperplane exists, the hyperplane is clearly of interest and is known as the *maximum-margin hyperplane* or the *optimal hyperplane*, as are the vectors that are closest to this hyperplane, which are called the *support vectors*. The basic idea behind SVMs is to learn a decision hyperplane that separates the data points with maximum margin. In linear classification cases, the algorithm aims to find a linear decision hyperplane that can separate the data points with maximum margin. In non-linear cases, the algorithm maps the data points into a higher dimensional space and thus finds a decision hyperplane that can separate the data points linearly. Figure 2.1 shows a classic example for linear classification.

In this example, the objects: circles and squares, belong either to class A—circles or B—squares. The separating line defines a boundary between class A and B. Any new object added to this example would be classified as class A or class B. The above is the simplest example. Unfortunately, most real-world problems are not that simple. Most problems involve non-linear separable data for which there does not exist a hyperplane that can successfully separate one class from another. More complex structures are needed to make an optimal hyperplane. This situation is illustrated by Figure 2.1 (b).

Compared to Figure 2.1 (a), it is clear that a curve not a straight line, forms the separation between circles and squares. The curve is more complex than the line. One solution of the inseparability problem is to map the data into a higher dimensional

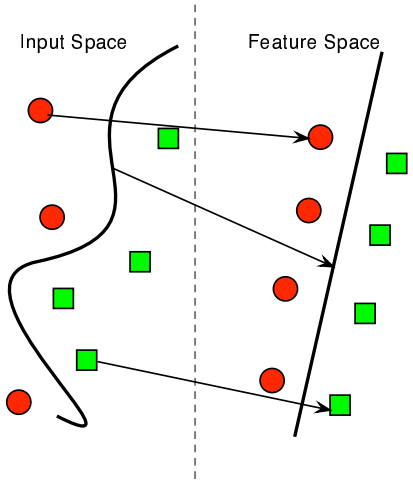

Figure 2.2: Mapping from input space to feature space

space and find a separating hyperplane there. SVM is particularly suitable to solve such problems. Rather than drawing a curve between the two classes, SVM maps the data into a higher dimensional space by using a kernel function and then draws a separating hyperplane there. Figure 2.2 shows the idea behind SVM non-linear classification.

This higher dimensional space is called the feature space and the original training set is called the input space. With an appropriately chosen feature space, any consistent input space can be separated linearly. In Figure 2.2, the original objects in the input space have been mapped into the feature space by using a set of known functions as the *kernel* which maps the input data into a different space—the feature space—where a hyperplane can be used to do the separation. Note that in the feature space, the mapped objects are linearly separable. Thus, instead of drawing a complex curve in the *input space* to separate the circles and squares, we find an optimal line in the *feature space* that separates the two classes linearly. A more detailed introductory review on SVM can be found at [29].

2.1.1 Linear SVM

We begin explaining SVMs with a simple linear example. A linear SVM is illustrated in Figure 2.3. The circles and squares represent two classes in Figure 2.3. We can find many decision lines that can separate the classes in Figure 2.3. We are interested in finding one that allows us to have maximum separation between the classes. This

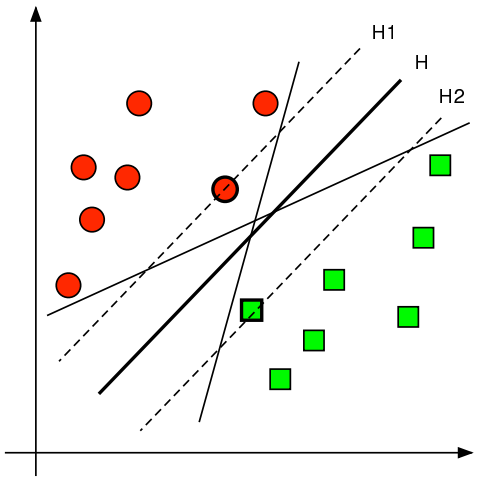

Figure 2.3: A linear SVM

separation is referred to as the *margin*. In Figure 2.3, the linear SVM will return the bold solid line, H, over other possible lines. This line is determined by first finding two parallel lines—H1 and H2—such that these lines separate the points into two classes, and pass through at least one point—a *support vector*—of one of two classes. In Figure 2.3, the goal of the linear SVMs is to find a hyperplane, i.e. a decision line with maximum margin.

Consider a linear training set with N training data points in $\mathbb{R}^2$:

$$\{x_1, y_1\}, \{x_2, y_2\}, \ldots, \{x_N, y_N\},$$

where

$$x_i \in \mathbb{R}^2 \text{ and } y_i \in \{+1, -1\}.$$

In the above, x_i represents the point to be classified and y_i is the class to which x_i belongs. We want to find one decision line H: $\mathbf{w} \cdot x - b = 0$, the bold solid line, which separates the two classes maximally and two lines $H1$: $\mathbf{w} \cdot x - b = +1$ and $H2$: $\mathbf{w} \cdot x - b = -1$, the dashed lines, parallel to H with equal distances. Furthermore, we want the margin, the distance between $H1$ and $H2$, to be a maximal margin and there to be no data points between H1 and H2. For any decision line H and the corresponding $H1$ and $H2$, there always exists $\mathbf{w}$ to make H: $\mathbf{w} \cdot x - b = 0$, $H1$: $\mathbf{w} \cdot x - b = +1$ and $H2$: $\mathbf{w} \cdot x - b = -1$. Recall that the distance from a point (x_0, y_0) to a line $Ax + By + C = 0$ is

$$\frac{Ax_0 + By_0 + C}{\sqrt{A^2 + B^2}}.$$

So the distance from a point on H to $H1$ is $\frac{\mathbf{w}\cdot x-b}{||\mathbf{w}||} = \frac{1}{||\mathbf{w}||}$ and the distance between $H1$ and $H2$ is $\frac{2}{||\mathbf{w}||}$. In order to get a maximal margin, we should minimise $||\mathbf{w}|| = \mathbf{w}^T\mathbf{w}$ with the condition that there are no data points between $H1$ and $H2$: $\mathbf{w}\cdot x_i - b \geq +1$, for positive data $y_i = 1$; $\mathbf{w}\cdot x_i - b \leq -1$, for negative data $y_i = -1$. These two conditions can be combined as $y_i(\mathbf{w}\cdot x_i - b) \geq 1$. So our problem can be formulated as:

$$\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w},$$

subject to

$$y_i(\mathbf{w}\cdot x_i - b) \geq 1.$$

This is a convex, quadratic programming problem with constraints $y_i(\mathbf{w}\cdot x_i - b) \geq 1$ for positive data and $y_i(\mathbf{w}\cdot x_i - b) \leq -1$ for negative data. Moreover, the margin is only decided by the support vectors and other data points can be removed or moved around as long as they do not cross the area between $H1$ and $H2$. This problem can be dealt with by introducing Lagrange multipliers $a_1, a_2, \ldots, a_N \geq 0$ and the following Lagrangian:

UNIVERSITY of the
WESTERN CAPE

$$L(\mathbf{w}, b, a) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} a_i \cdot (y_i \cdot (\mathbf{w}\cdot x_i + b) - 1).$$

2.1.2 Non-linear SVM

But what if the decision boundary which separates the classes is not linear? In Figure 2.4, we can map the data points into a higher dimensional space and thus the data points can be separated linearly. In Figure 2.4, the function ψ should map the data into a higher dimensional space so that B1 and B2 can be separated linearly. We call the function $K(x_i, x_j) = \psi(x_i)^T \cdot \psi(x_j)$ the kernel. The kernel function transforms the data into a higher dimensional space and enables it to be separated linearly. The goal of the non-linear SVM is to find the separating hyperplane in a higher dimensional space. There are four basic kernel functions where the γ, r and d are kernel parameters:

- $K(x_i, x_j) = (x_i)^T \cdot (x_j)$
- $K(x_i, x_j) = (\gamma(x_i)^T \cdot (x_j) + r)^d,\ \gamma > 0$
- $K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2),\ \gamma > 0$

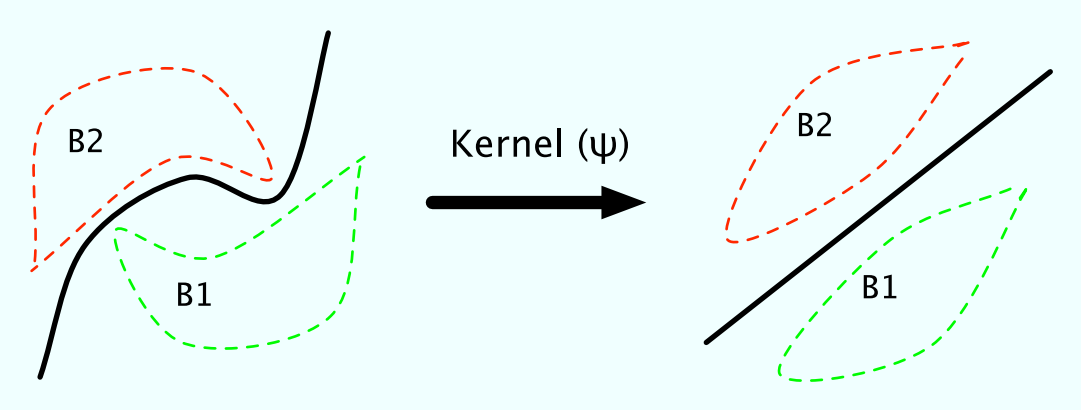

Figure 2.4: The kernel mapping

- $K(x_i, x_j) = \tanh(\gamma(x_i)^T \cdot (x_j) + r),\ \gamma > 0$

Many kernel functions can be used in SVM mapping, but only a few of them have been found that work well in a wide variety of applications. Choosing an appropriate kernel function is important, since the kernel function defines the feature space in which the training set will be mapped. The performance of the SVM is very closely tied to the choice of the optimal kernel functions [13]. There has been much research over the last few years on algorithms to help choose the best kernel for a given problem with a certain set of features [31]. Most of these methods are based on simple heuristic knowledge of the input data and there is no standardised method to obtain the best kernel. Hence, the choice of the optimal kernel is a trial and error procedure in most scenarios [8]. The default and recommended kernel function is the Radial Basis Function (RBF) [13]. It is by far the most popular choice of kernels used in SVMs. This is mainly because of their localised and finite responses across the entire range of the real X-axis. The RBF kernel unlike the linear kernel, can handle non-linear cases. The reasons we choose the RBF as our recommended kernel over others are:

1. The RBF kernel can handle non-linear and linear cases. With certain parameters, the RBF kernel can behave like the linear kernel and the sigmoid kernel;

2. The number of kernel parameters influences the complexity of SVM model selection. The RBF kernel has fewer kernel parameters. More kernel parameters will bring more complexity into SVM model selection;

3. Another advantage of the RBF kernel is less numerical difficulties. The computing of the polynomial kernel function could go to infinity.

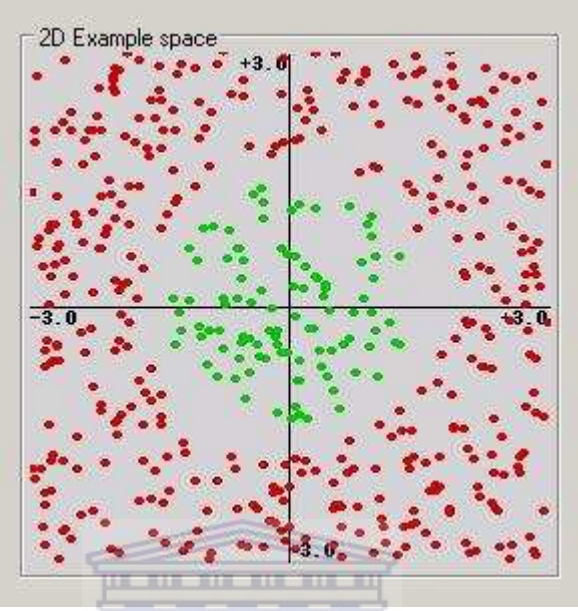

Figure 2.5: An classification example generated by pcSVMDemo

UNIVERSITY of the
WESTERN CAPE

We use one of the most common kernel functions, Gaussian RBF kernel as our kernel, which is also the default RBF kernel supported by LibSVM.

Figure 2.5 is an example automatically generated by pcSVMDemo using the circle generator. The ideal hyperplane between the two classes, green and red, should be a circle.

Figures 2.6–2.8 show the results with the SVM tested under different kernel types on the same example. We start with the simple linear kernel. Figure 2.6 shows a linear classification between the two classes when applying the linear kernel to data that is not linearly separable. It fails to apply the linear kernel correctly. We notice that the linear kernel performs quite well for data that is linearly separable. But, it does not cope with non-linear data giving many errors. Figure 2.7 shows another successful classification with a few support vectors. Figure 2.8 also shows a successful classification, but with more support vectors. Comparing Figure 2.7 with Figure 2.8, we notice that the performance of the SVM with the RBF kernel is not as good as the SVM with the polynomial kernel—the hyperplane in Figure 2.7 is more like a circle. This indicates that the polynomial kernel is more suitable for the example in this case.

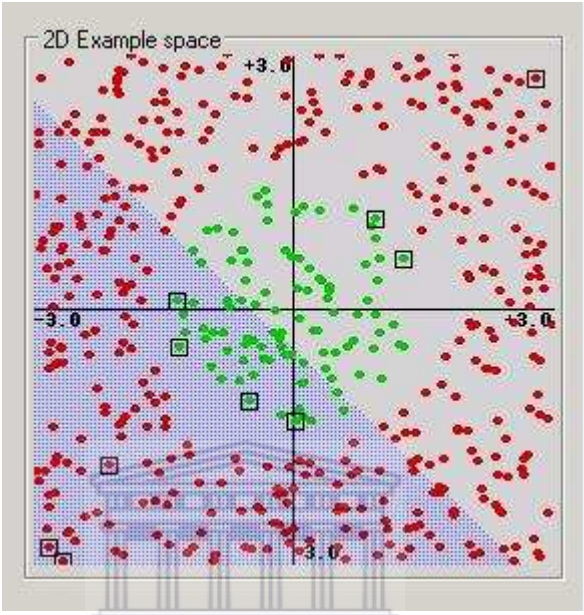

Figure 2.6: The non-linear SVM classification with linear kernel

UNIVERSITY of the
WESTERN CAPE

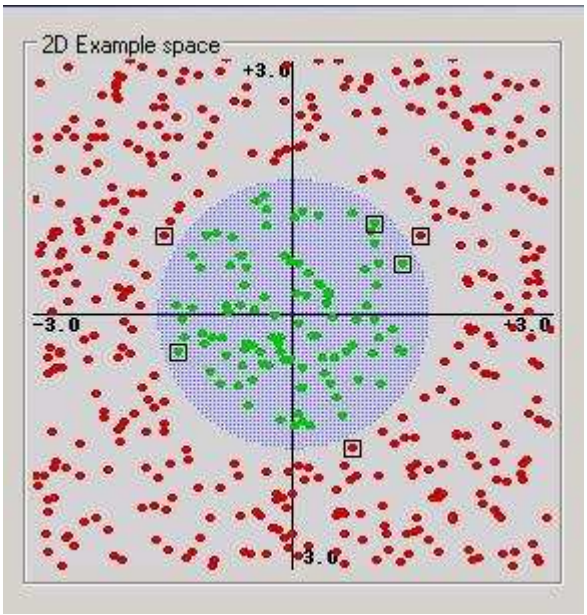

Figure 2.7: The non-linear SVM classification with polynomial kernel

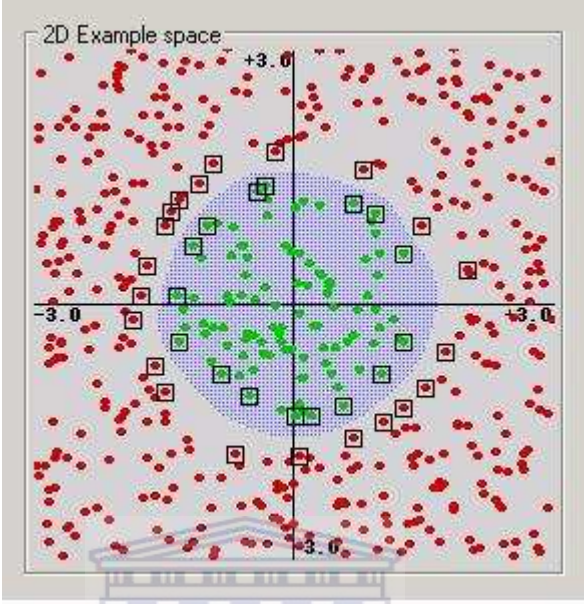

Figure 2.8: The non-linear SVM classification with RBF kernel

UNIVERSITY of the
WESTERN CAPE

We compared the performance for three of the four basic types of kernels considered: linear, polynomial and RBF. We notice from the results that the SVM with the polynomial kernel performs the best classification compared to the other two kernels. When we compare Figure 2.7 and Figure 2.8 it is also clear that the SVM with polynomial kernel uses the least number of support vectors.

2.2 SVM Classification

The SVM is a supervised classification technique for creating a *classifier* from training data. The training data consist of pairs of input objects typically *vectors*, and *desired outputs*. The output can be a continuous value—called *regression*, or can be class labels of the input object—called *classification*. Since the primary interest of our research is SVM classification, we will ignore SVM regression and focus on SVM classification.

SVM classification usually involves two kinds of data, training data and testing data. The training data includes some training examples. Each of those examples contains one *label* and several *attributes*. The testing data includes testing examples which each of them contains several *attributes*. Using the training set, the SVM classifier would distinguish between the members and non-members of a given class.

Having learned the features of the class, the SVM can predict new objects as members or as non-members of the class. The goal of SVM classification is to predict the *label* of testing instances which are only given by *attributes* in the testing set by learning from the training instances in the training set. To achieve this, the SVM learns a separating hyperplane from the training data by using the kernel function to map the data point into a higher dimensional space.

2.3 The SVM Tool LibSVM

LibSVM (Library for Support Vector Machines) [5] is a simple and easy-to-use open source implementation for SVM. It was developed by Chang and Li and contains C-support vector classification (C-SVC) [7], v-support vector classification (v-SVC) [19], e-support vector regression (e-SVR) [19], and v-support vector regression (v-SVR) [28]. It supports multi-class classification, weighted SVM for unbalanced data, cross-validation and automatic model selection. It has interfaces for Python, R, Splus, MATLAB, Perl, Ruby, and LabVIEW. Users can easily link it with their own programs. It includes four kernels: Linear, Polynomial, Radial Basis Function, and Sigmoid. The goal of LibSVM is to help users to easily use SVM as a tool.

UNIVERSITY of the
WESTERN CAPE

Recommended Procedure for using LibSVM

This is a recommended procedure of using LibSVM:

1. Convert the data to the format that LibSVM requires
2. Scale the data
3. Select a kernel
4. Find the optimal parameter values for the kernel
5. Train on the training data and predict the testing data

We will discuss this procedure in detail in the following sections.

2.3.1 Converting data

LibSVM requires that each data instance in the training set and the testing set are represented as real numbers. So we must convert textitlabel and *attribute* into numeric data format. For example, we have three classes Web Servers, FTP Servers and Database Servers and each of them has three attributes (the programs' name):

Web Servers: `apache`, `tomcat`, `netscape-enterprise`

FTP Servers: `proftp`, `pureftp`, `vsftp`

Database Servers: `mysql`, `postgres`, `oracle`

The data above can be represented as:

```
1: 610.0,  648.0, 1985.0
2: 667.0, 774.0, 563.0
3: 566.0, 887.0, 630.0
```

We are using (`1`, `2`, `3`) to represent (Web Servers, FTP Servers, Database Servers) three different classes. The rest of numbers above are the sum of the corresponding ASCII codes. (e.g, apache = '`a`' + '`p`' + '`a`' + '`c`' + '`h`' + '`e`' = 97 + 112 + 97 + 99 + 104 + 101 = 610)

The format of training and testing set file is:

(label 1) (index 1):(value 1) (index 2):(value 2) ... (index N): (value N)

(label 2) (index 1):(value 1) (index 2):(value 2) ... (index N): (value N)

...

(label N) (index 1):(value 1) (index 2):(value 2) ... (index N): (value N)

"label" is the target value of the training data. For classification, it should be an integer which identifies a class. The labels in the testing data files are only used to calculate accuracy or errors. "index" is an integer starting from 1. The indices must be in ascending order. "value" is a real number. Thus, the example above should be represented like this:

```
1 1:610.0 2:648.0 3:1985.0

2 1:667.0 2:774.0 3:563.0

3 1:566.0 2:887.0 3:630.0
```

```
+1 1:0.583333 2:1 3:1 4:-0.509434 5:-0.52968 6:-1 7:1 8:-0.114504 9:1 10:-0.16129
-1 1:-0.416667 2:1 3:1 4:-0.603774 5:-0.191781 6:-1 7:-1 8:0.679389 9:-1 10:-0.612903
-1 1:-0.291667 2:-1 3:1 4:-0.169811 5:-0.465753 6:-1 7:1 8:0.236641 9:1 10:-1
+1 1:0.25 2:1 3:1 4:0.433962 5:-0.086758 6:-1 7:1 8:0.0534351 9:1 10:0.0967742
+1 1:-0.125 2:1 3:1 4:-0.0566038 5:-0.6621 6:-1 7:1 8:-0.160305 9:1 10:-0.709677
+1 1:-0.208333 2:1 3:1 4:-0.320755 5:-0.406393 6:1 7:1 8:0.206107 9:1 10:-1
+1 1:0.25 2:1 3:-1 4:0.245283 5:-0.328767 6:-1 7:1 8:-0.175573 9:-1 10:-1
-1 1:-0.208333 2:1 3:1 4:-0.471698 5:-0.561644 6:-1 7:1 8:0.755725 9:-1 10:-1
+1 1:-0.541667 2:1 3:1 4:0.0943396 5:-0.557078 6:-1 7:-1 8:0.679389 9:-1 10:-11
-1 1:0.375 2:-1 3:1 4:-0.433962 5:-0.621005 6:-1 7:-1 8:0.40458 9:-1 10:-1
+1 1:-0.375 2:1 3:1 4:-0.698113 5:-0.675799 6:-1 7:1 8:0.618321 9:-1 10:-1
+1 1:0.541667 2:1 3:-0.333333 4:0.245283 5:-0.452055 6:-1 7:-1 8:-0.251908 9:1 10:-1
+1 1:0.5 2:-1 3:1 4:0.0566038 5:-0.547945 6:-1 7:1 8:-0.343511 9:-1 10:-0.677419
+1 1:-0.458333 2:1 3:1 4:-0.207547 5:-0.136986 6:-1 7:-1 8:-0.175573 9:1 10:-0.419355
+1 1:0.25 2:-1 3:1 4:0.509434 5:-0.438356 6:-1 7:-1 8:0.0992366 9:1 10:-1 12:-1 13:-1
+1 1:-0.0833333 2:-1 3:1 4:-0.320755 5:-0.182648 6:-1 7:-1 8:0.0839695 9:1
```

Figure 2.9: The example of training data from the `heart-scale` file

Figure 2.9 is an example of the training data format from the `heart-scale` file in the LibSVM-2.71 software package.

UNIVERSITY of the
WESTERN CAPE

2.3.2 Scaling data

Scaling data is very important before applying LibSVM. [25] explains why we scale data while using Neural Networks, and most of the considerations also apply to SVM [13]. The benefit of scaling data is first, avoid large numeric ranges by scaling all attributes in a smaller range, usually it is between 0 and 1, or -1 and 1. And second, we can also avoid large numerical calculation and speed the LibSVM processing. Third, large attribute values might cause numerical problems in the linear kernel and the polynomial kernel. The testing data and training data should be scaled in the same way before applying LibSVM. For example, if we scaled the first attribute of training data from $[-10, +10]$ to $[-1, +1]$ and if the first attribute of testing data is in $[-11, +8]$, we must scale the testing data to $[-1.1, +0.8]$.

2.3.3 Selecting a kernel

LibSVM handles four basic kernel functions: Linear, Polynomial, Radial Basis Function (RBF), and Sigmoid. Since the performance of the SVMs is very closely tied to the kernel functions, the selection of an appropriate kernel function is important. A standardised method for obtaining the appropriate kernel does not exist. The choice

of the kernel is a trial and error procedure in most cases. The LibSVM default recommended kernel function is the RBF kernel. Section 2.1.2 explained the reasons why we choose the RBF kernel as the recommended kernel.

2.3.4 Setting kernel parameters

There are a few parameters in the four basic kernel functions. See Section 2.1.2. Each of the kernel functions has different parameters. The accuracy of the SVM is largely dependent on the selection of the kernel parameters. After choosing a kernel function, we should find optimal parameter values for the kernel parameters. Therefore, a common way is to:

1. Separate training data into several random training sets;

2. Set values of the parameters to train the separated subsets and predict other subsets to calculate the accuracy;

3. Change the values and calculate the accuracy until getting good accuracy;

4. Apply the optimal parameter values to the kernel function;

This whole process is called *cross validation* [5].

UNIVERSITY of the
WESTERN CAPE

There are many possible values for each of the parameters. Cross validation is a direct way to try different values for parameters. We usually try the values by increasing or decreasing in exponential order until we get the best cross validation accuracy. This process is called *grid search*. For example, there are two parameters in the RBF kernel: C *and* γ. We apply a grid-search on C *and* γ using cross validation. Basically pairs of (C, γ) are tried and the one with the best cross-validation accuracy is picked. The method of trying exponentially growing sequences of C *and* γ is a practical procedure to find the optimal parameters ($C = 2^{-5}, 2^{-3}, \ldots, 2^{15}$, $\gamma = 2^{1}, 2^{3}, \ldots, 2^{15}$).

The procedure of grid search is computationally expensive because the model must be evaluated at many points within the grid for each of the parameters. For example, if a grid search is used with 10 search intervals in a RBF kernel function (two parameters), then the model must be evaluated at $10 \times 10 = 100$ grid points. There are several methods that can save computational cost. However, there are two reasons that LibSVM prefers the simple grid search approach. One is that psychologically we may not feel safe to use methods which avoid doing an exhaustive parameter search

by approximations or heuristics. The other reason is that the computational time of seeking optimal parameters is not much more than that of other methods since there are only two parameters [13].

2.3.5 Training and predicting

There are three important executable files in the LibSVM software package: `svmtrain`, `svmscale` and `svmpredict`. They are for training on the training data, scaling the training and testing data, and predicting the testing data respectively. We scale both the training data and the testing data using svmtrain, train on the training data using svmtrain and then use svmpredict to predict the testing data.

The file `heart-scale` is provided as an example for classification in the LibSVM-2.71 software package. Type `svm-train heart-scale`, and the program will read the training data file `heart-scale` and output the model file `heart-scale.model`. If you have a testing data file called `heart-scale.t`, the command is: `svm-predict heart-scale.t heart-scale.model output` to see the prediction accuracy on the testing data. The `output` file contains the predicted class label [13]. The class label is the result of the SVM classification.

UNIVERSITY of the WESTERN CAPE

2.4 Summary

A loop control called *monitor-classify-adjust* is built into the KernTune. The classifying module in the loop learns the system type using SVM technology. In this chapter, the basic idea of SVMs has been reviewed. The goal of SVM is to learn a separating hyperplane between classes by mapping the data points into a higher dimensional space. The adjusting module in the loop adjusts operating system settings and operational system parameters to a specific workload in order to achieve the best possible performance. In the next chapter, the skills and knowledge of operating system optimisation for performance will be presented.

Chapter 3

Linux Kernel Optimisation

KernTune provides a *monitor-classify-adjust* loop control for automatic collection and management of system profile information. In the previous chapter, we reviewed the SVM technology used in the classifying module of the loop. In this chapter, we focus on the GNU/Linux operating system optimisation knowledge that the adjusting module of the loop requires.

3.1 Operating system optimisation and GNU/Linux

Operating system optimisation is the manipulation of system parameters in order to achieve maximal possible performance, or reach a target workload for an acceptable cost within given constraints. Operating system optimisation commonly refers to the collection of system information, handling and management of system parameters on a running operating system.

In our research, we have chosen GNU/Linux, a free, popular UNIX clone as our experimental operating system. It is licensed under the GNU Public License (GPL). The source code for the Linux kernel is freely available to anyone. The GPL is a widely used license from the Free Software Foundation (FSF). If one modifies the source code which is protected under GPL, one is obliged to release your modified code. The complete definition of GPL can be found at [11].

Linux is the name given to the Linux kernel by its creator, Linus Torvalds and it was initially created in 1991. In common usage, Linux refers to the entire operating system which is based on the Linux kernel and GNU software. The development of the kernel has evolved rapidly with the assistance of developers around the world since it was first released [33]. Linux is the *kernel*, an essential part of an operating system, which refers to the low-level system software that provides an abstraction layer between hardware and software. It enables the operating system to manage all the hardware and resources such as: hardware devices—CPU, memory, disk, display

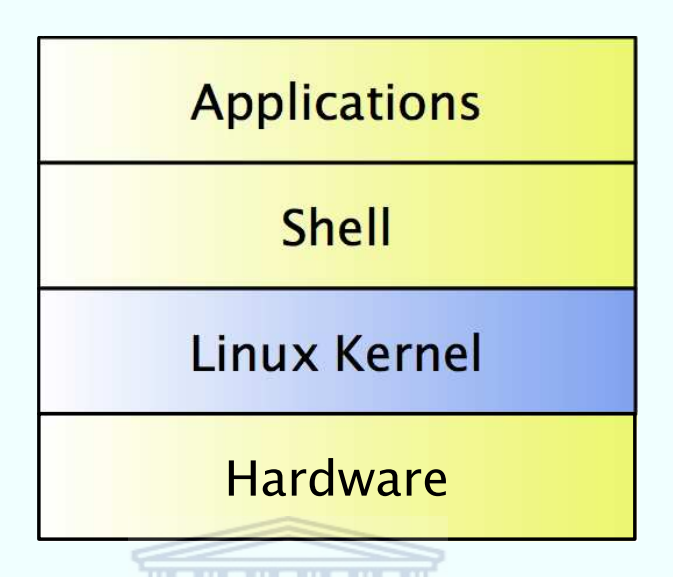

Figure 3.1: The structure of a GNU/Linux operating system

cards, etc., memory assignment, data store, processes, workload balance, networking, running applications and security. A complete Linux operating system is composed of GNU software and the Linux kernel. The GNU project is lead by the FSF and its goal is to create free software and also develop the GNU operating system. Because all Linux distributions rely on a lot of important software from the GNU project, like GNU Emacs, the GNU Debugger, the GNU Compiler Collection, the GNU C library, GNOME desktop environment and so on, the FSF ask that the Linux system to be referred as "GNU/Linux" [33].

The KernTune program uses the GNU/Linux operating system as our research base because it is open source, efficient, well-documented and this makes it ideal for research and experimentation [4].

3.2 Overview of Linux kernel

The kernel is the core part of a GNU/Linux operating system. The other parts of the system cannot run on a machine if the kernel does not participate as part of the entire system [1]. As such, it makes sense to discuss the kernel in the context of an entire system. Figure 3.1 shows the structured overview of a GNU/Linux operating system. The GNU/Linux operating system is composed of four major subsystems:

1. *Applications*. The set of user software in use on a GNU/Linux system. Applications allow users to perform one or more specific tasks. Typical applications include word processors, spreadsheets, business programs, databases and games. The applications installed on the GNU/Linux depend on what the machine is used for [33].

2. *Shell*. It is typically considered part of a GNU/Linux system and it provides an operating interface through which users can interact with the system and issue commands. It is the utility that processes user requests [33].

3. *Kernel*. A single executable program, loaded into memory at boot time and remaining there until the system is powered down. The kernel abstracts hardware devices and acts as an intermediary between the hardware and shell and provides hardware devices, memory and processor management functions, e.g. handling interrupt requests from hardware devices, sharing memory and using the processor(s) among processes, etc. User applications can interface with the kernel through system calls [33].

UNIVERSITY of the
WESTERN CAPE

4. *Hardware*. All the possible physical devices in a computer, for example, processor, memory, hard disk, network cards, etc [33].

Each of the layers in Figure 3.1 can only communicate with adjacent layers. In addition, the dependencies between layers are from top to bottom: higher layers depend on lower layers. Since the primary interest of this research is the kernel, we will ignore the application, shell and hardware layers and focus on the kernel layer only.

The kernel is the heart of a GNU/Linux operating system. It has two main tasks:

1. To serve low level hardware requirements;

2. To provide an environment for running programs.

The function of the kernel is to present a virtual machine for user processes and hide the underlying hardware [26]. The kernel (1) *controls and mediates access to hardware*, (2) *implements and supports fundamental abstractions*—processes, files, devices etc., (3) *allocates and releases system resources*—memory, CPU, disk, descriptors, etc., (4) *enforces security and protection*, and (5) *responds to user requests for service*—system calls. In addition, the kernel (6) *supports a multi-tasking computing environment* that is transparent to user processes. Each process acts as if it is

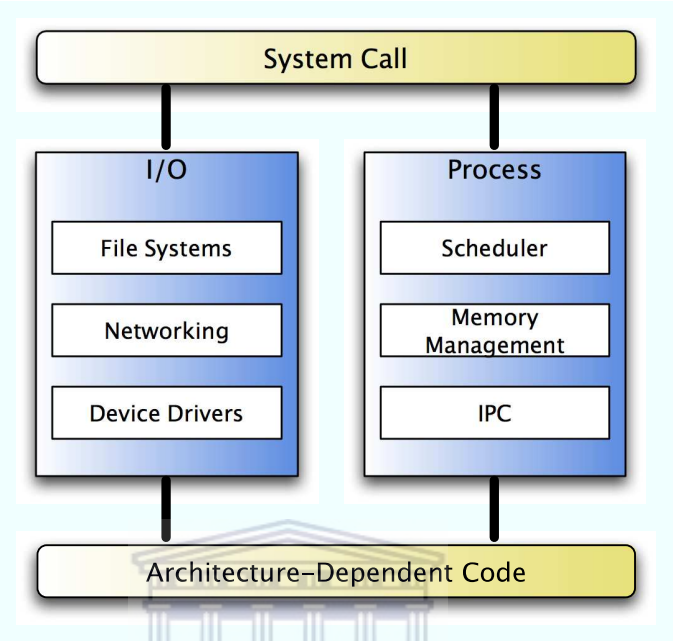

Figure 3.2: The components of the kernel

UNIVERSITY of the
WESTERN CAPE

the only process running on the computer, with exclusive use of memory and other hardware resources. The kernel actually runs multiple processes concurrently and rapidly switches back and forth among the processes. Figure 3.2 shows the divided components of the kernel.

Ivan Bowman [1] divides the Linux kernel into five logically divided components. We regard *device drivers* as an essential component of the kernel and regard its components to be:

1. *Process scheduler.* It controls the way of processes access the processor. The scheduler applies a policy to ensure that each process on the computer gets its fair share of the processor[23].

2. *Memory manager.* It keeps track of the whole memory map, allocates and de-allocates memory to processes, and manages swapping between main memory and disk. It permits multiple processes to share the main memory simultaneously [23].

3. *Interprocess communication* (IPC). It allows one process to communicate with another process. It is required in all modern operating systems [23].

4. *File System.* Its job is to keep track of the whole disk map, to allocate and de-allocate sectors to files, and to manage files and directories. In addition, it abstracts the various hardware devices' details by presenting a common file interface [23].

5. *Device Drivers.* They control different hardware devices. They issue commands to the devices, interact with them and provide interfaces to the devices that are simple and easy to access.

6. *Networking.* It provides access to network function using several networking protocols [23].

We will discuss the optimisation parameters of the various kernel components in Section 3.4.

3.3 Exploring `/proc`

The `/proc` file-system is a real-time reflection of the running system kept in memory and represented in a hierarchal manner [10]. The job of the `/proc` file-system is to track the information of the system and running processes and to provide an easy way to view such information. The `/proc` is a pseudo file-system residing in the virtual memory and it does not exist on any physical media. The premise behind the `/proc` file-system is to view the state of the system and information of currently running processes by easily reading from related files in the `/proc` instead of having difficult to understand system calls. For security reasons, the content of the `/proc` can only be read or written to users with the appropriate authority. The `/proc` file-system can provide information such as:

UNIVERSITY of the
WESTERN CAPE

- Viewing hardware information
- Viewing kernel runtime status and memory/disk/network statistical information

And can modify information such as:

- Modifying kernel runtime parameters
- Modifying memory, disk, network parameters

Figure 3.3 shows the content of the `/proc` file-system with the Linux kernel 2.4.29 on a GNU/Linux system. The following list [10, 17] briefly describes the files and directories in the `/proc` file-system seen in Figure 3.3.

```
[lyee@cent ~]$ ls /proc
1       2980  3237  4474          fs           mounts
1371    2990  3351  5             ide          mtrr
190     3     3446  acpi          interrupts   net
19085   3010  3564  asound        iomem        partitions
2       3018  3658  buddyinfo     ioports      pci
2049    3029  3659  bus           irq          self
23      3060  3829  cmdline       kallsyms     slabinfo
2360    3061  3847  cpuinfo       kcore        stat
24      3071  4     crypto        keys         swaps
2752    308   41    devices       key-users    sys
2756    3096  4123  diskstats     kmsg         sysrq-trigger
2778    3115  42    dma           loadavg      sysvipc
2798    3134  43    dri           locks        tty
2826    3144  44    driver        mdstat       uptime
2896    3155  4443  execdomains   meminfo      version
2906    316   4445  fb            misc         vmstat
2965    3165  4446  filesystems   modules
```

UNIVERSITY of the WESTERN CAPE

Figure 3.3: The content of `/proc`

- `loadavg`

Example of `cat loadavg`

```
0.00 0.00 0.00 3/57 28964
```

This file contains system load information about the kernel. The first three entries represent the average number of active tasks on the system that were actually running over the last 1, 5, and 15 minutes. The next entry is the number of currently runnable processes that are scheduled to run rather than being blocked in the kernel and the total number of processes on the system. The final entry is the process ID of the process that ran most recently.

- `uptime`

UNIVERSITY of the
WESTERN CAPE

Example of `cat /proc/uptime`

```
1588073.71 1009804.77
```

This file contains the time in seconds since the system was booted and the total time used by processes rather than the idle time of the system. Both of these values are given as floating point values, in seconds.

- `meminfo`

Example of `cat /proc/meminfo`

```
        total:    used:    free:  shared: buffers:  cached:
Mem:  262873088 68243456 194629632        0  1445888 35373056
Swap: 518180864 37249024 480931840
MemTotal:       256712 kB
MemFree:        190068 kB
MemShared:           0 kB
Buffers:          1412 kB
```

```
Cached:          26620 kB
SwapCached:       7924 kB
Active:          21780 kB
Inactive:        34780 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:       256712 kB
LowFree:        190068 kB
SwapTotal:      506036 kB
SwapFree:       469660 kB
```

MemTotal: Total usable physical memory. Physical memory minus the memory reserved for the kernel

UNIVERSITY of the WESTERN CAPE

MemFree: Sum of LowFree + HighFree

MemShared: always zero

Buffers: Memory in buffer cache

Cached: Memory in the pagecache (diskcache) minus SwapCache

SwapCache: Memory that was swapped out, is now swapped back in but is still also in the swap file

Active: Memory that has been used recently and usually will not be released unless absolutely necessary.

HighTotal: The total amount of memory in the high region.

LowTotal: The total amount of low memory region.

LowFree: The amount of free memory of the low memory region. This is the memory that the kernel can address directly.

SwapTotal: The total amount of swap memory.

SwapFree: The total amount of free swap memory.

- See Appendix A

The contents of the /proc file-system is used by the monitoring module of KernTune which grabs the data from the particular files in the /proc directories and formats it as the input for the classifying module of KernTune. The monitoring module collects information from /proc/stat, /proc/loadavg, /proc/meminfo, /proc/net/dev, /proc/net/tcp and all the numbered directories.

3.4 Kernel Optimisation

Another important part of the /proc file-system is the directory /proc/sys. This directory not only provides information of the running kernel, it also allows administrators to modify the value of the parameters that the kernel uses to determine behaviour. The files in /proc/sys can be used to monitor and to optimise general and miscellaneous operation of the Linux kernel. Unlike most other directories in the /proc file-system, /proc/sys variables are typically wirable, and are used to adjusting the running kernel rather than simply monitoring currently running processes and system information. To change the value of a parameter, simply apply the new value to the related file using the echo or the sysctl command. We will focus on /proc/sys/kernel, /proc/sys/fs, /proc/sys/vm, and /proc/sys/net, which are used to tune the kernel, file-system, virtual memory and disk buffers, and networking respectively. The 2.4.29 kernel has many parameters that can be used for improving performance [22]. In the remainder of this section, we present several areas of sysctl that can result in large performance improvements.

UNIVERSITY of the
WESTERN CAPE

3.4.1 Overview of the Kernel Parameters

The following definitions of the kernel parameters are taken from Linux kernel 2.4.29 documents which can be found in the kernel source package [34]. The list below shows the kernel parameters that are most relevant to performance [32, 21].

- /proc/sys/net/ipv4/inet_peer_gc_maxtime

  Minimum interval between garbage collection passes. This interval is in effect under low or absent memory pressure on the pool. Measured in *jiffies*[1]

[1]A jiffy is a unit of time that depends on the speed of the processor.

- `/proc/sys/net/ipv4/inet_peer_gc_mintime`

  Minimum interval between garbage collection passes. This interval is in effect under high memory pressure on the pool. Default is 10, measured in jiffies.

- `/proc/sys/net/ipv4/inet_peer_maxttl`

  The maximum time-to-live for the inet peer entries. New entries will expire after this period of time. Default is 600, measured in jiffies.

- `/proc/sys/net/ipv4/inet_peer_minttl`

  The minimum time-to-live for inet peer entries. Set to a high-enough value to cover fragment time to live in the reassembling side of fragmented packets. This minimum time must be smaller than `net.ipv4.inet_peer_threshold`. Default is 120, measured in jiffies.

- `/proc/sys/net/ipv4/inet_peer_threshold`

  The approximate size of the storage. Starting from this threshold entries will be threwn aggressively. This threshold also determines entries' time-to-live and time intervals between garbage collection passes. More entries, less time-to-live, less GC interval.

- `/proc/sys/vm/hugetlb_pool`

  The `hugetlb` feature works the same way as `bigpages`, but after `hugetlb` allocates memory, the physical memory only can be access by `hugetlb` or `shm` allocated with `SHM_HUGETLB`. It is normally used with databases such as Oracle or DB2. The default is 0.

- `/proc/sys/vm/inactive_clean_percent`

  Designates the percent of inactive memory that should be cleaned. The default is 5%.

- `/proc/sys/vm/pagetable_cache`

  The kernel keeps a number of page tables in per-processor caches. `pagetable_cache` bounds the number of pages in each cache. This helps a lot on SMP systems. The cache size for each processor lies between the low and the high value. On

a low-memory, single CPU system one can safely set both these values to 0 so that memory is never wasted by caching. On SMP systems it assists the system to do fast pagetable allocations without having to acquire the kernel memory lock. For large and normal systems, the default settings are reasonable. For small systems, with a memory smaller than 16MB ram, it is advantageous to set both values to 0 so that there is no cache. The default values are: 25 50.

- `/proc/sys/fs/file-nr`

  The kernel allocates file handles dynamically, but as yet it doesn't free them again. The three values in `file-nr` denote the number of allocated file handles, the number of used file handles and the maximum number of file handles. When the allocated file handles come close to the maximum, but the number of actually used ones is far behind, you've encountered a peak in your usage of file handles and you don't need to increase the maximum.

UNIVERSITY of the
WESTERN CAPE

- `/proc/sys/fs/file-max`

  The value in `file-max` denotes the maximum number of file-handles that the Linux kernel will allocate. When you get lots of error messages about running out of file handles, you might want to increase this limit.

- `/proc/sys/vm/bdflush`

  This file controls the operation of the `bdflush` kernel daemon. The source code of this `struct` can be found in `fs/buffer.c`. It currently contains 9 integer values, of which 6 are actually used by the kernel: `nfract`, `ndirty`, `dummy2`, `dummy3`, `interval`, `age_buffer`, `sync`, `stop_bdflush`, `dummy5`. The default are: `30 500 0 0 500 3000 60 20 0` for 100 HZ.

- `/proc/sys/vm/kswapd`

  The kernel swapout daemon, `kswapd`, is that piece of the kernel that frees memory when it gets fragmented or full. Since every system is different, you'll probably want some control over this piece of the system. The numbers in this page correspond to the numbers in the `struct pager_daemon`—`tries_base`, `tries_min, swap_cluster`. The `tries_base` and `swap_cluster` probably have the largest influence on system performance.

- `/proc/sys/net/ipv4/tcp_max_syn_backlog`

  Maximal number of remembered connection requests, which still did not receive an acknowledgement from connecting client. Default value is 1024 for systems with more than 128Mb of memory, and 128 for low memory machines. If server suffers of overload, try to increase this number.

- `/proc/sys/net/ipv4/ip_local_port_range`

  Defines the local port range that is used by TCP and `UDP` to choose the local port. The first number is the first, the second the last local port number. The default values depend on the amount of memory available on the system: larger than 128Mb 32768–61000, smaller than 128Mb 1024–4999 or even less. This number defines the number of active connections, which this system can issue simultaneously to systems not supporting TCP extensions (timestamps). With `tcp_tw_recycle` enabled, i.e. by default, range 1024–4999 is enough to issue up to 2000 connections per second to systems supporting timestamps.

- `/proc/sys/net/ipv4/tcp_wmem`

  min: Amount of memory reserved for send buffers for TCP socket. Default: 4K.

  default: Amount of memory allowed for send buffers for TCP socket by default. Default: 16K.

  max: Maximal amount of memory allowed for automatically selected send buffers for TCP socket. Default: 128K.

- `/proc/sys/net/ipv4/tcp_rmem`

  min: Minimal size of receive buffer used by TCP sockets. It is guaranteed to each TCP socket, even under moderate memory pressure. Default: 8K.

  default: default size of receive buffer used by TCP sockets. Default: 87380 bytes.

  max: maximal size of receive buffer allowed for automatically selected receiver buffers for TCP socket. Default: 87380 $\times 2$ bytes.

- `/proc/sys/net/ipv4/tcp_keepalive_time`

  How often TCP sends out keepalive messages when keepalive is enabled. Default: 2 hours.

- `/proc/sys/net/core/wmem_max`

  Maximum buffer size for the send queue for any protocol, including IPv4.

- `/proc/sys/net/core/rmem_max`

  Maximum buffer size for the read queue for any protocol, including IPv4.

The following lists the kernel parameters that are relevant but not typically used in performance tuning [32, 21].

- `/proc/sys/kernel/panic`

  The value in this file represents the number of seconds the kernel waits before rebooting on a panic. When you use the software watchdog, the recommended setting is 60.

- `/proc/sys/kernel/pid_max`

  Determines the maximum pid that a process can allocate. Default is 32768.

- `/proc/sys/net/ipv4/tcp_tw_recycle`

  Enable fast recycling `TIME-WAIT` sockets. Default value is 0. It should not be changed without the advice/request of technical experts.

- `/proc/sys/vm/overcommit_ratio`

  Percentage of memory that is allowed for over commit. Default is 50%.

3.4.2 Using `sysctl`

`sysctl` is an interface that exports the ability to tune kernel parameters in a running Linux system. It is easy to make changes to the kernel parameters by issuing the `sysctl` command. For example, to modify the `file-max` kernel parameter, *root* can alter the value of the parameter with two different commands:

- using `echo` command

```
cat /proc/sys/fs/file-max
26188
echo 30000 > /proc/sys/fs/file-max
cat /proc/sys/fs/file-max
30000
```

- using the `sysctl` command

```
sysctl fs.file-max
fs.file-max = 26188
sysctl -w fs.file-max=30000
fs.file-max = 30000
sysctl fs.file-max
fs.file-max = 30000
```

UNIVERSITY of the WESTERN CAPE

Notice that using the `echo` command can easily introduce errors, so we prefer using `sysctl` because it checks the consistency of the parameter before it makes change. The system call `sysctl` to programmers is available. It is an alternative way to change parameters rather than modifying parameters by using read/write system calls. The advantage of `sysctl` is that it is faster, as no fork is executed nor any directory look-up. KernTune uses the `sysctl` instead of operating parameter files directly. To use `sysctl` in a C program, the header file `linux/sysctl.h` must be included. The declaration is like this:

```
int sysctl (int *name, int nlen, void *oldval, size_t *oldlenp, void
*newval, size_t newlen);
```

- `name` is a integer pointer that points to an array of integers: each of the elements of the array identifies a sysctl item.

- `nlen` is a integer indicates the number of elements which are stored in the array.

- `oldval` is an integer pointer that points to a data buffer where the old value of the parameter must be stored.
- `oldlenp` is a `size_t` pointer that points to the length of the oldval.
- `newval` is an integer pointer that points to a data buffer that hosts the new value of the parameter.
- `newlen` an integer indicates the length of `newval`.

3.5 Summary

UNIVERSITY of the
WESTERN CAPE

The adjusting module in the *monitor-classify-adjust* loop which is built into KernTune adjusts operating system settings and operational system parameters for best possible performance. In this chapter, we explored the `/proc` file-system and explained how to optimise general and miscellaneous operations of the kernel using `sysctl`. Then we presented several kernel parameters that can result in large performance improvements, which will be used in KernTune, and we described `sysctl`. In the next chapter, our approach and methodology of automatic system performance tuning technology will be presented.

Chapter 4

Approach and Methodology

In this chapter, we present our approach and methodology for the automatic adjustment of system settings. The optimisation strategy used by KernTune is dependent on the functions performed by the server. Thus, the monitoring tool needs to collect the appropriate system information to allow us to identify the class of system that needs to be optimised. We then describe how this information is used by the SVM to classify the system class. Next, we discuss the kernel parameters KernTune will use to improve system performance and the recommended values of these parameters for various system classes. Finally, we present a summary of this chapter.

UNIVERSITY of the
WESTERN CAPE

4.1 Understanding System Classes

To optimise system performance, it is important to first understand the intended use of the system and the performance constraints of the particular system. Once the system classes have been identified, particular attention can be focused on the specific performance characteristics of each class. Figure 4.1 shows one server delivering services to users with different requirements.

Below is a list of common services:

- *Ftp servers*

  The role of the ftp server is to receive requests from users, store data or send the requested data back. The basic requests include upload, download and update data, list files, change directory, etc. Therefore, the critical performance issue of the ftp server is the speed of data transfer. The networking subsystems of the kernel handles the data transfer. The memory subsystem handles tasks such as network buffers and disk I/O caching which also greatly influence performance. When a user requests a file, the ftp server must initially locate, then read and send the file back to the user. Therefore the disk subsystem will potentially

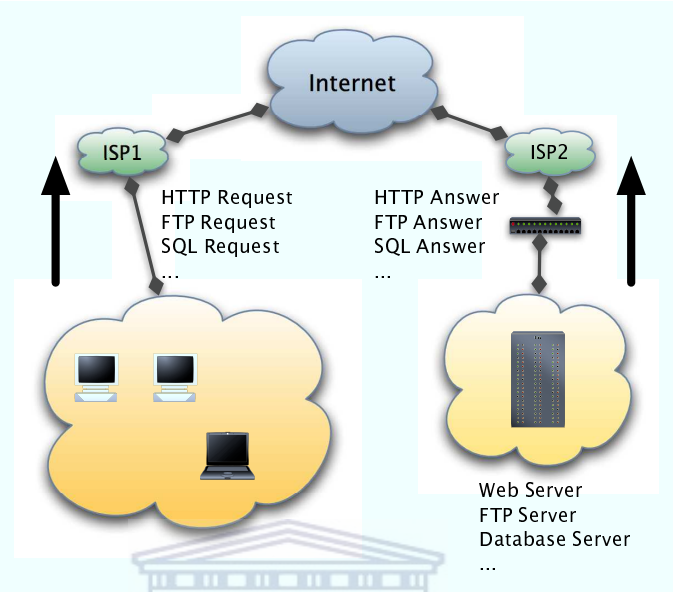

Figure 4.1: One server serves users with different requirements

UNIVERSITY of the
WESTERN CAPE

influence performance. Processor speed or quantity typically has little impact on ftp server performance [33].

The subsystem that has the most impact on ftp server performance is the network subsystem [32].

- *Print servers*

  The role of the print server is to manage a large number of print requests and queues, and send the data to the appropriate printer. The print server keeps data in memory and waits for slower printers to produce output. Therefore, the critical performance issues are the speed of data transfer and memory capacity. The printer server spools print jobs via a disk and the spool directory is located on a physical drive. Therefore the disk subsystem will also impact performance [33].

  The subsystem that has the most impact on print server performance is the memory subsystem [32].

- *Database servers*

  The role of the database server is to store, search, retrieve and update data from disk. The database server handles a large number of random disk I/O requests

and computational activities. Paging will occur between disk and memory if the server does not have sufficient memory, which will result in large amounts of disk I/O. Therefore, the critical performance issues of the database server are memory capacity and disk I/O. The database queries, insert/remove/update operations and replication require intensive CPU time. Therefore, the CPU processing capacity is another important factor for database servers [33].

The subsystem that has the most impact on database servers performance is the memory subsystem [32].

- *E-mail servers*

  The role of the e-mail server is to store and distribute electronic mail messages. E-mail servers use memory to keep database buffers. Paging will occur if there is not enough memory. Therefore, the disk subsystem is important to the server's performance. As the e-mail server also has to deliver mail, check e-mail address in the list and route the message accordingly, the processor's capacity is also important. High speed and stable network between e-mail servers and their clients are required when the clients are receiving large e-mails [33].

  The most important subsystem for e-mail servers is the memory subsystem [32].

- *Web servers*

  The role of the web server is to host web pages and run web applications. The performance issues of the web server depends on the web site's content. There are two kinds of web site. A *dynamic* site allows Internet users to register, log on and query from its database. A *static* site only accepts user requests and returns the requested web pages. A dynamic site requires more CPU cycles to process user requests and produce dynamic pages. Memory is very important in dynamic sites because the web server must have adequate memory for caching and for processing dynamic pages. Because of high hit ratio and transferral of large objects, like pictures, music, flash pages, etc., the network is also very important in both types of web sites [33].

  If web site content is static, the subsystem that has the most impact on the web servers' performance is the network subsystem. If the web server is not

static, the subsystem that has the most impact on web servers' performance is the memory subsystem [32].

- *Groupware servers*

  The role of the groupware server is to cooperage, share and exchange information in a user community. The groupware server generally supports folder/file access, scheduling, calendaring and work flow applications. These functions usually require significant CPU power similar to an e-mail server. Memory is used for caching database buffers and a special memory cache is designed to increase the data access rate. Therefore, the groupware server should have enough memory to reduce paging between memory and disk. Groupware servers are client/server database applications and their performance characteristics are similar to database servers. Therefore, the disk subsystem is also an important performance factor [33].

  The most important subsystem is the memory subsystem [32].

- *Multimedia servers*

  The role of the multimedia server is to provide concurrent video streams to multiple users through intranet or the Internet. The multimedia server requires high-speed networking and fast disk I/O because of the large data transfers. If compression/decompression of the data is required, then CPU power and memory capacity are important factors as well [33].

  The most important subsystem is the network subsystem [32].

- *Terminal servers*

  The role of the terminal server is to enable multiple users to access a server and run applications on the server through a network. The terminal server executes the user requested application and sends screen updates and results back to the user. As the application and all the processing are run on the server, the terminal server requires powerful CPUs and sufficient memory. The terminal server processes concurrent users with large numbers of requests through the network, therefore, the network is another important subsystem [33].

The subsystem that has the most significant influence on performance is the memory subsystem [32].

- *DNS servers*

  The role of the Domain Name System (DNS) server is to translate between domain names and IP addresses. The DNS server utilises memory to cache files and resolve names to IP addresses or other information associated with the names. A high-speed network between the DNS server and other DNS servers is required when the server fails in order to find the name in its database [33].

  The most important subsystem is the network subsystem [32].

- *DHCP servers*

  The role of the Dynamic Host Configuration Protocol (DHCP) server is to manage and administer IP addresses and other related configurations. The DHCP server also provides a mechanism for allocating IP addresses to client hosts. The DHCP server responds and distributes the configuration information, such as a valid IP address, to clients. A high-speed network is required for transferring the configuration and related information. The disk subsystem is another important factor which impacts performance [33].

  The most important subsystem is the network subsystem [32].

UNIVERSITY of the
WESTERN CAPE

Table 4.1 lists all the important subsystems which most significantly influence performance in the above system classes.

Table 4.1: Performance factors

System Class	**Performance Factor**
Ftp server	Network, memory and disk
Print server	Memory and disk
Database server	Memory and processor
E-mail server	Memory, processor and network
Web server	Network and memory

continued on next page

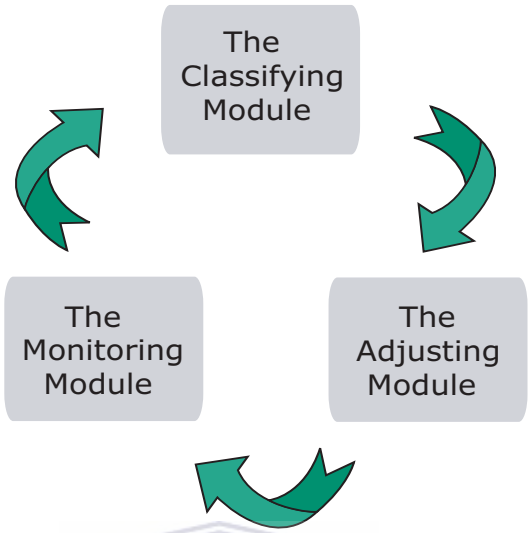

Figure 4.2: The *monitor-classify-adjust* (MCA) cycle

continued from previous page

System Class	**Performance Factor**
Groupware server	Memory, processor and disk
Multimedia server	Network, disk and processor
Terminal server	Memory, processor and network
DNS server	Network and memory
DHCP server	Network and disk

4.2 The Monitor-Classify-Adjust Cycle

The main principle that we pursue in the KernTune project is the concept of an on-line control cycle called *monitor-classify-adjust*. The system continuously monitors certain performance data and analyses the necessary data using SVM. Whenever change is detected the system dynamically adjusts kernel parameters to improve performance. While this principle seems very simple, applying it to real systems is problematic. It is unclear which kernel parameters should be considered for each system class. Furthermore, system load also has to be taken into account to adjust the parameters. To address these problems, we apply SVM technology and divide the control cycle into three modules: monitor, classify, and adjust, the MCA cycle for short. Figure 4.2 shows this cycle:

- Keep observing the status of the running system (monitor);
- Employing SVM techniques to classify the system class (classify);
- Adjusting system parameters for the particular system class (adjust).

The monitoring module observes performance metrics and the system workload that can be viewed as indicators for a shift of system class. In order to implement this module, we first need to identify which parameters to observe. While disk throughput and CPU times are objects of system metrics, they do not give enough clues to classify a system. In addition to these observation, we investigated which performance metrics should be used in order to identify the system class. Furthermore, the data could also guide us in choosing the appropriate values for the kernel parameters. The purpose of the classifying module is to assess the performance objects of the various system classes. The classifying module needs a mathematical model for classification of the system classes. Performance tuning should be adjusted only when we identify a system class successfully. SVM classification has been used in the classifying module as the mathematical model. The critical role of this module is to interpret information. It takes the output from the monitoring module and provides input for the adjusting module. The last module in the three-module cycle is the adjusting module. When the classifying module returns a system class, we can determine which parameters to be adjusted and by how much. The adjusting module adjusts the kernel settings based on known tuning rules or practical experiments.

4.2.1 Monitoring a system

Through monitoring, KernTune obtains performance data that is useful in classifying system classes and in tuning these systems. The monitoring module in the *monitor-classify-adjust* cycle ensures that KernTune always has up-to-date information about how the computer is operating. The performance data provides KernTune information about the behaviour of system components at runtime, which is useful for determining the system class. When KernTune has performance data for the system over a range of activities and loads, KernTune can determine the system class. The system runtime data represents a system class. In addition, this system class provides a reference point for KernTune to tune the system. The following subsections describe the scope and

type of performance data collected, the design of the performance data architecture, and the methods of data collection used by the monitoring module.

Scope and Type of Performance Data

In general, performance monitoring illustrates how the system, including the operating system and any applications or services uses the resources of the system.

KernTune collects data about system resources, such as disks, memory, processors, and network components. In addition, KernTune also collects information about applications and services that might be running on the system, such as application or service name and ports used by services. The monitoring module obtains performance data, system resources and application information using the `/proc` file-system. When the monitoring module read the `/proc` file-system for performance data, the module collects the data from the appropriate directories and files in the `/proc` file-system, such as the `loadavg`, `uptime`, `meminfo`, `/proc/net/`, `stat`, numbered directories, etc. As an option, the Linux kernel supports collecting performance data using the `sysctl` system call. KernTune prefers reading the data directly from the `/proc` file-system.

Performance Objects and Counters

By default, operating systems have numerous performance objects corresponding to hardware or other resources in the system. Table 4.2 shows the default performance objects in a Linux system [12].

Table 4.2: Performance objects

Object Name	**Description**
Cache	Reports activity for the file-system cache, an area of physical memory that holds recently used data.
Memory	Reports usage of random access memory (RAM), the part of computer hardware that used to store code and data.

continued on next page

continued from previous page

Object Name	**Description**
Network	Reports rates at which bytes and packets are sent and received by the network adapters.
Swap	Reports usage of the swap file-system, swaps data between memory and disk.
Disk	Reports usage of hard disks, the part of computer hardware that stores programs and data.
Processor	Reports activity of the processor, the part of computer hardware that executes program instructions.
System	Reports statistics for system-wide counters that track processor time, file operations, process information and so on.
TCP	Reports the rates at which Transmission Control Protocol (TCP) segments are sent and received using the TCP.
UDP	Reports the rates at which User Datagram Protocol (UDP) datagrams are sent and received using UDP.

UNIVERSITY of the WESTERN CAPE

Each object has counters that are used to measure various aspects of the system's performance, such as read/write rates of disk, the usage of memory, transfer rates of the network interface, or the utilisation of the processor(s). Depending on how a counter is defined, its values might be described in one of the following ways:

1. Instantaneous counters, display the most recent measurement;

2. Averaging counters, measure a value over time and display the average of the last two measurements.

Table 4.3 describes performance counters and their descriptions [12, 24].

Table 4.3: Performance counters

Performance Counter	**Description**
Avg. Disk Bytes Read/sec	The rate at which bytes are read
Avg. Disk Bytes Write/sec	The rate at which bytes are written
Avg. Disk IO Read Operation/sec	The rate of disk I/O reads requests
Avg. Disk IO Write Operation/sec	The rate of disk I/O writes requests
Pages in/sec	The rate at which the disk was read to resolve hard page faults
Pages out/sec	The rate at which pages are written to disk to free up space in physical memory
Memory free(MB)	Free physical memory in megabytes
Swap pages in/sec	The rate at which pages are read from swap
Swap pages out/sec	The rate at which pages are written to swap
Swap free(MB)	Free swap space in megabytes
% User Time	Represents the time spent in the CPU by the user's application
Bytes Received/sec	The rate at which bytes are read from network adapters
Bytes Sent/sec	The rate at which bytes are sent to network adapters
Packets Received/sec	The rate at which packets are read from network adapters
Packets Sent/sec	The rate at which packets are sent to network adapters
TCP Bytes Received/sec	The rate at which bytes are read from the network by TCP
TCP Bytes Sent/sec	The rate at which bytes are sent to the network by TCP
TCP Packets Received/sec	The rate at which packets read from the network by TCP

continued on next page

UNIVERSITY of the WESTERN CAPE

continued from previous page

Performance Counters	Description
TCP Packets Sent/sec	The rate at which packets are sent to the network by TCP
UDP Bytes Received/sec	The rate at which bytes are read from the network by UDP
UDP Bytes Sent/sec	The rate at which bytes are sent to the network by UDP
UDP Packets Received/sec	The rate at which packets are read from the network by UDP
UDP Packets Sent/sec	The rate at which packets are sent to the network by UDP

KernTune *samples* data periodically rather than collecting it continuously. This collection method has the advantage of keeping overhead low, but it might occasionally overestimate or underestimate values when activity falls outside the sampling interval. The objective of KernTune is to improve system performance with minimal additional overhead. This objective is very important for KernTune.

4.2.2 Classifying a System Class

Classifying system classes is the core part of the *monitor-classify-adjust* cycle for KernTune. KernTune uses performance data collected by the monitoring module as input to the classifying module. The classifying module in the *monitor-classify-adjust* cycle automatically builds a system class classifier by learning a pre-classified system class set using the SVM technology. This system class is used as input to the adjusting module described in the next section. A system class can change when any new application or service starts running. The classifying module employs a SVM and the pre-classified set to develop a system class classifier that can classify the new classes. SVMs are particularly well suited to this type of classification. See Sections 2.1 and 2.2.

This section describes how to identify a system class using SVM technology, and shows how the classifying module uses the data from the monitoring module for

classification. The following subsections include the pre-classified system class set, the input of the classifying module, and the method of classification used by the classifying module.

Training Pre-classified System Class Sets

A pre-classified system class set is a set of pre-classified examples which contains one *system class* and several *system attributes*. The use of SVM classification begins with training on the set of pre-classified examples. Figure 4.3 shows an acceptable training set for KernTune.

Each line of the training set describes an instance of a system class. The data presented in each of the columns are: the identification number of the system class, the name of the active program, the port number of the active service, the processor usage, the physical memory usage, the swap usage, the number of blocks read from the disk, the number of blocks written to the disk, the number of packets received by the network interface, the number of packets transmitted by the network interface.

The pre-classified system class set is constructed manually and used as training set for the classifying module. The collection is a set of 1000 items (class:attributes), that have been pre-classified manually into 10 categories. These 10 categories are: ftp servers, printer servers, database servers, e-mail servers, web servers, groupware servers, multimedia servers, terminal servers, DNS servers and DHCP servers. The set is stored in text format and can be used as the testing set in later experiments and trial tests.

Input of The Classifying Module

The SVM in the classifying module uses the pre-classified examples to determine the set of parameters required for proper discrimination. The SVM then encodes these parameters into a model called a system class classifier. Once an effective classifier is developed, it is used to classify new classes into the same predefined classes. We modified the `libsvm` source code to accept the input of the training set so that the SVM can interpret each sample of the set in the context in which it was generated.

```
1 1:660 2:80 3:0.00 4:0.68 5:0.11 6:0 7:0 8:100 9:100
1 1:660 2:80 3:0.00 4:0.68 5:0.11 6:0 7:0 8:100 9:100
1 1:660 2:80 3:0.00 4:0.68 5:0.11 6:0 7:0 8:100 9:100
1 1:660 2:80 3:0.00 4:0.68 5:0.11 6:0 7:0 8:101 9:100
1 1:660 2:80 3:0.00 4:0.68 5:0.11 6:0 7:0 8:100 9:100
1 1:660 2:80 3:0.00 4:0.68 5:0.11 6:0 7:0 8:110 9:110
2 1:767 2:21 3:0.00 4:0.57 5:0.00 6:0 7:0 8:2 9:2
2 1:767 2:21 3:0.00 4:0.57 5:0.00 6:0 7:0 8:7 9:7
2 1:767 2:21 3:0.00 4:0.57 5:0.00 6:0 7:0 8:2 9:2
2 1:767 2:21 3:0.00 4:0.57 5:0.00 6:0 7:0 8:4 9:0
2 1:767 2:21 3:0.00 4:0.57 5:0.00 6:0 7:0 8:0 9:0
2 1:767 2:21 3:0.00 4:0.57 5:0.00 6:0 7:0 8:8 9:2
2 1:767 2:21 3:0.00 4:0.57 5:0.00 6:0 7:0 8:0 9:0
3 1:666 2:3306 3:100.00 4:0.42 5:0.00 6:0 7:0 8:4 9:0
3 1:666 2:3306 3:100.00 4:0.42 5:0.00 6:0 7:0 8:0 9:0
3 1:666 2:3306 3:100.00 4:0.42 5:0.00 6:0 7:0 8:3 9:0
3 1:666 2:3306 3:100.00 4:0.41 5:0.00 6:0 7:0 8:0 9:0
3 1:666 2:3306 3:100.00 4:0.41 5:0.00 6:0 7:0 8:0 9:0
3 1:666 2:3306 3:100.00 4:0.41 5:0.00 6:0 7:0 8:1 9:0
3 1:666 2:3306 3:100.00 4:0.40 5:0.00 6:104 7:32 8:0 9:0
4 1:660 2:80 3:0.00 4:0.56 5:0.00 6:0 7:0 8:0 9:0
4 1:660 2:80 3:0.00 4:0.56 5:0.00 6:0 7:0 8:1 9:0
4 1:767 2:21 3:0.00 4:0.56 5:0.00 6:0 7:0 8:3 9:0
4 1:767 2:21 3:0.00 4:0.56 5:0.00 6:0 7:0 8:0 9:0
4 1:767 2:21 3:0.00 4:0.56 5:0.00 6:0 7:248 8:1 9:0
4 1:767 2:21 3:0.00 4:0.56 5:0.00 6:0 7:0 8:0 9:0
4 1:767 2:21 3:0.00 4:0.56 5:0.00 6:0 7:0 8:1 9:0
```

Figure 4.3: The training set used by KernTune

4.2.3 Adjusting System Parameters

The Linux kernel offers a variety of parameters to system administrators for tweaking the performance of a system. Adjusting these parameters for specific system usage is difficult, because the effect on the system and the applications (or services running on it needs to be extensively tested. There are no hard and best rules on adjusting these parameters as they are very dependent on the system class. Our approach of finding the 'right' value of the parameters is simply to experiment. We wrote a script to adjust the value periodically and measured its effect until we found the 'right' value for the target system class. These values are examined on our test machine. This section describes the parameters that can achieve the most improvement in performance and their best possible values for three system classes: web server class, ftp server class and database server class.

UNIVERSITY of the
WESTERN CAPE

Suggested Values for Web Servers

Table 4.4 shows the best possible values of the kernel parameters for the web server system class. The suggested values are come from [9, 14, 24, 34, 32].

Table 4.4: Tuned values for web servers

Parameter	**Suggested Value**
net.ipv4.inet_peer_gc_maxtime	240
net.ipv4.inet_peer_gc_mintime	80
net.ipv4.inet_peer_maxttl	500
net.ipv4.inet_peer_minttl	80
net.ipv4.inet_peer_threshold	65644
vm.hugetlb_pool	4608
vm.inactive_clean_percent	30
vm.pagecache	50 100
vm.bdflush	30 500 0 0 500 3000 80 20 0
vm.kswapd	1024 32 64
net.ipv4.tcp_max_syn_backlog	8192

continued on next page

continued from previous page

Parameter	Suggested Value
net.ipv4.ip_local_port_range	16384 65536
net.ipv4.tcp_wmem	4096 131072 262144
net.ipv4.tcp_rmem	4096 87380 174760
net.ipv4.tcp_keepalive_time	1800
net.core.wmem_max	262144
net.core.rmem_max	103424
net.ipv4.ip_forward	0
net.ipv4.conf.all.rp_filter	1
net.ipv4.conf.lo.rp_filter	1
net.ipv4.conf.eth0.rp_filter	1
net.ipv4.conf.default.rp_filter	1
net.ipv4.conf.all.accept_redirects	0
net.ipv4.conf.lo.accept_redirects	0
net.ipv4.conf.eth0.accept_redirects	0
net.ipv4.conf.default.accept_redirects	0
net.ipv4.conf.all.accept_redirects	0
net.ipv4.conf.lo.accept_redirects	0
net.ipv4.conf.eth0.accept_redirects	0
net.ipv4.conf.default.accept_redirects	0
net.ipv4.tcp_fin_timeout	15
net.ipv4.tcp_window_scaling	0
net.ipv4.tcp_sack	0
net.ipv4.tcp_timestamps	0
net.ipv4.icmp_echo_ignore_broadcasts	1
net.ipv4.icmp_ignore_bogus_error_responses	1
net.ipv4.conf.all.log_martians	1
net.ipv4.tcp_max_tw_buckets	1440000
net.ipv4.tcp_sack	0

continued on next page

continued from previous page

Parameter	Suggested Value
net.ipv4.tcp_timestamps	0

Suggested Values for Ftp Servers

Table 4.5 shows the best possible values of the kernel parameters for the ftp server system class. The suggested values come from [9, 14, 24, 34, 32].

Table 4.5: Tuned values for ftp servers

Parameter	Suggested Value
net.ipv4.inet_peer_gc_maxtime	240
net.ipv4.inet_peer_gc_mintime	80
net.ipv4.inet_peer_maxttl	500
net.ipv4.inet_peer_minttl	80
net.ipv4.inet_peer_threshold	65644
vm.hugetlb_pool	4608
vm.inactive_clean_percent	30
vm.pagecache	50 100
vm.bdflush	30 500 0 0 500 3000 80 20 0
vm.kswapd	1024 32 64
net.ipv4.tcp_max_syn_backlog	1024
net.ipv4.ip_local_port_range	32768 61000
net.ipv4.tcp_wmem	4096 87380 174760
net.ipv4.tcp_rmem	4096 131072 262144
net.ipv4.tcp_keepalive_time	1800
net.core.wmem_max	103424
net.core.rmem_max	262144
net.ipv4.tcp_sack	0
net.ipv4.tcp_timestamps	0

UNIVERSITY of the WESTERN CAPE

Suggested Values for Database Servers

Table 4.6 shows the best possible values of the kernel parameters for the database server system class. The suggested values are come from [9, 14, 24, 34, 32].

Table 4.6: Tuned values for database servers

Parameter	Suggested Value
net.ipv4.inet_peer_gc_maxtime	240
net.ipv4.inet_peer_gc_mintime	80
net.ipv4.inet_peer_maxttl	500
net.ipv4.inet_peer_minttl	80
net.ipv4.inet_peer_threshold	65644
vm.hugetlb_pool	4608
vm.inactive_clean_percent	30
vm.pagecache	50 100
vm.bdflush	30 500 0 0 500 3000 60 20 0
vm.kswapd	1024 32 64
net.ipv4.tcp_max_syn_backlog	1024
net.ipv4.ip_local_port_range	1024 4999
net.ipv4.tcp_wmem	4096 87380 8388608
net.ipv4.tcp_rmem	4096 87380 8388608
net.ipv4.tcp_keepalive_time	7200
net.core.wmem_max	8388608
net.core.rmem_max	8388608
kernel.shmmax	268435456
kernel.msgmni	1024
fs.file-max	8192
kernel.sem	250 32000 32 1024
kernel.sem	500 512000 64 2048
kernel.msgmni	2048
kernel.msgmax	64000

continued on next page

UNIVERSITY of the WESTERN CAPE

continued from previous page

Parameters	Values
net.ipv4.tcp_tw_reuse	1
net.ipv4.tcp_tw_recycle	1

4.3 Related Work

There are a number of tools in automatic Linux kernel optimisation. In this section we review some relevant tools and explain how they differ from KernTune.

KernTune enables systems to evolve with changes in usage. This functionality is currently not provided by any other tool. KernTune builds on technology originally developed for classification. Several existing tools adjust kernel parameters for the purpose of improving performance. These tools differ from KernTune in two important ways. First, though these tools are based on the designer's knowledge and experience of adjusting the kernel parameters for improving performance, the users of the tools need to know the purpose of the system (the system class). In KernTune, we employ SVM technology to classify the system class. A further difference between KernTune and earlier tuning tools is that KernTune is designed specifically for automatic tuning and optimisation. It runs as a daemon in the background and keeps track of usage changes.

Powertweak-Linux [27] is a kernel tuning tool that improves Linux system performance. It uses the `/proc` file-system, and `hdparm` tool to tweak systems. Like KernTune, this tool also performs optimisations on systems by known tuning rules. Powertweak-Linux was designed for Linux systems and can either apply manual optimisation or tuning rules. KernTune uses machine learning, parameter-driven and system-class-specific optimisations to improve system performance.

The most recent tool for tuning the Linux kernel is the kernel patches developed by Jake Moilanen [16]. He implements an automatic optimisation module in the Linux kernel. It supports continuous tuning of the kernel by applying a Genetic Algorithm. There are substantial similarities between Moilanen's patches and KernTune. Both Moilanen's patches and KernTune use a Machine Learning method to the tune systems while causing very little overhead. The primary difference between the two tools

is how they are applied. In KernTune we have focused on classification, whereas the emphasis of Moilanen's patch has been on chromosomes. Another difference is that Moilanen's patch works in the kernel as a module and KernTune runs out of the kernel as a daemon.

SarCheck [6] is another performance analysis and tuning tool for Linux systems. It produces recommendations and explanations for tuning the kernel with supporting graphs and tables. It uses the resulting information to drive administrators schedule optimisation. Although KernTune design does not produce recommendations for tuning the kernel, it is not required to do so. The goal of KernTune is to tune real systems. Though the goal of SarCheck is different from KernTune, the two tools share many similarities: both tools continuously collect system information and use the information to guide further operations. The main differences between the two tools are that SarCheck produces recommendations and suggestions for tuning the kernel, it collects system information, and produces the optimisation recommendations, rather than applying the optimisation as in KernTune. KernTune also supports automatic optimisation.

UNIVERSITY of the
WESTERN CAPE

4.4 Summary

Understanding system classes is important for parameter-driven, system-class-specific optimisation. In this chapter, we first introduced the importance of understanding system classes and discussed the different system classes. Next, we presented the *monitor-classify-adjust* cycle which is based on SVM technology and built into KernTune. The purpose of the cycle is to identify the system class. Once the system class has been identified, KernTune performs specific optimisation by applying class specific rules to tune the system performance. Last we discussed related work on Linux kernel optimisation. In the next chapter, we will discuss the design goals for KernTune and its implementation details.

Chapter 5

System Implementation

KernTune is a combination of performance tuning and SVM technology that provides a practical tool for operating system optimisation needed to support continued improvements in system performance. A major obstacle to operating system optimisation is obtaining high-quality system information that represents the current system usage. Ideally, such information should be representative of how an operating system is used on a machine. Given the current operating system architecture, it is not practical for end-users to obtain and use such information, as typical users do not know how to generate the information and would not be able to optimise their systems if they did. The KernTune tool addresses this problem, using the *monitor-classify-adjust* cycle to collect, process, and apply information to optimise a system automatically.

UNIVERSITY of the
WESTERN CAPE

This chapter describes our implementation of KernTune for GNU/Linux systems. Our implementation is composed of three components: the *Monitor*, the *Classifier* and the *Adjustor*. *The monitor* is a monitoring component that implements continuous, low-overhead collection and system information monitoring. *The classifier* is a classifying component that translates system information and classifies the system class, including computing the best possible system class from the classification results. *The adjustor* is an adjusting component that implements optimisation, choosing kernel parameters and adjusting system settings to the specific system class. In this chapter we focus on the first two components, *the monitor* and *the classifier*. Although some discussion of *the adjustor* is needed to provide context for the rest of the work, a detailed description of specific system optimisations is outside the scope of this thesis. In Section 5.1, we introduce the design goals of KernTune and describe how they are incorporated into a complete system for automatic system optimisation. Section 5.2 gives an overview of KernTune. The subsections that follow focus

on the main components of KernTune, namely *the monitor* for performance data collection (Section 5.3), *the classifier* that provides data processing (Section 5.4), and *the adjustor* that implements system optimisations (Section 5.5). Finally, we give our summary.

5.1 Design goals for KernTune

We designed our KernTune tool to meet a number of requirements that we believe are necessary for mainstream systems:

- Optimisation should happen on the machine where the application or service is running. For complex systems, there can be substantial variation in how the operating system is used for different requirements. Given these variations between system classes, automatic optimisation will be most effective if it can process information specific to the system and the machine.

UNIVERSITY of the
WESTERN CAPE

- Optimisation should not happen only once. In a real computing environment, system usage changes according to various requirements. The application or service running in the morning might not be used by end-users in the evening. The end-users could require another service instead of the morning one. Although re-optimisation for this requirement is reasonable, it is not practical for a dynamically changing system. To enable optimisation continuously, we assume that the system information represents the class of system and base all optimisations on this system class. In practice, a user-level daemon is usually used for this.

- Optimisation must be transparent to the user. We assume that the people who use computers do not have much background in the field of computers or system tuning. Optimisation should be transparent in that the user does not need to understand or participate in the optimisation process. Transparency also implies that the optimisations must achieve consistent performance benefits with no negative performance impact.

The KernTune tool is designed to address all these requirements. It automatically collects and analyses system information and uses that to optimise the kernel. Our

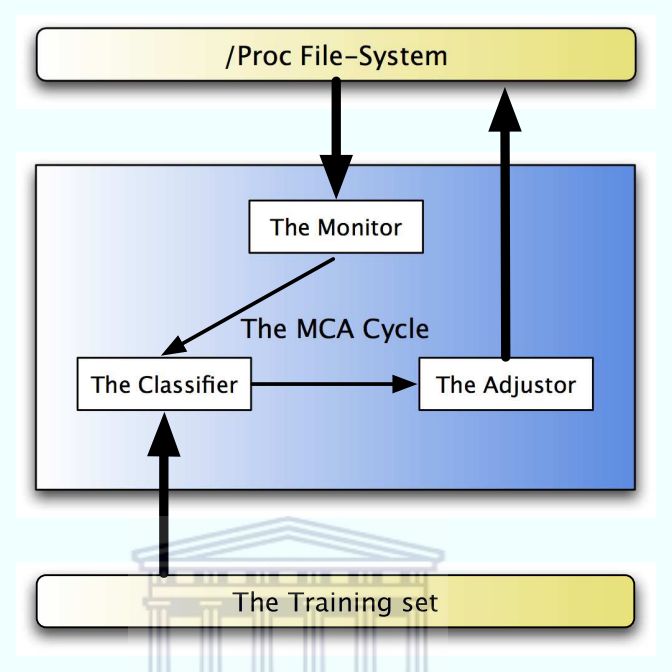

Figure 5.1: Overview of KernTune

UNIVERSITY of the
WESTERN CAPE

prototype implementation of KernTune supports automatic optimisation for Intel x86-based computers running the Linux kernel. We built several custom system tools to collect performance data, test results and made small modifications to LibSVM to support automatic collection and classification. The main component of KernTune is based on the Library for Support Vector Machines (LibSVM) [5]. We have extended this library to support system-specific optimisations. Practical considerations led us to choose a GNU/Linux operating systems for our initial implementation.

5.2 Overview of KernTune

Our initial design targets a single server environment. KernTune provides a *monitor-classify-adjust* cycle control for SVM-based and system-specific optimisation. An important feature of KernTune is that optimisation occurs on the system where the applications are running. The optimisation occurs only when the applications or services have already changed.

The major components of KernTune and their relationships are represented schematically in Figure 5.1. Optimisations in KernTune are applied without the use of Linux kernel source code. To achieve this goal we explore the `/proc` file-system that presents

the state of a running GNU/Linux system. Prior work on Linux kernel tuning was implemented by Jake Moilane [16]. His goals were to modify the kernel source and applying a Genetic Algorithm (GA) to kernel optimisation. An important aspect of our design is that the *monitor-classify-adjust* cycle is machine and operating system independent. Though we are aware of prior work on automatic kernel optimisation, we have chosen not to modify the Linux kernel, focusing instead on the problem of using SVM-based optimisations to improve system performance. The traditional method for providing a viable tool for self-optimisation is to define a series of optimisation rules that build a knowledge-based database to support various system classes. Our method is to apply the SVM technology to build a system tool (KernTune) that permits automatic adjustment of system settings.

KernTune has three major software components: *the monitor*, *the classifier* and *the adjustor*. *the monitor* collects system information for use by *the classifier*. *The monitor* collects system information with very low overhead, leaving subsequent processing to *the classifier*. *The classifier* is a user-level daemon that provides system settings management as well as information analysis to classify system classes and decide when to invoke *the adjustor*. *The adjustor* implements the optimisations provided by KernTune. The current version of our Adjustor supports three different system-class optimisations: web server class, ftp server class and database server class.

UNIVERSITY of the
WESTERN CAPE

5.3 Data Collection: *The monitor*

All the optimisations require system-specific information, including the active application information and performance counters. *The monitor* automatically collects this information. The data collection is implemented in *the monitor*, the output of *the monitor* is the input of *the classifier*. In this section we describe the design of *the monitor* and some of the interesting problems we encountered in its implementation. Figure 5.2 shows the structure of the monitoring procedure.

One key goal of *the monitor* is low system overhead. Our monitor achieves this goal by using a subset of the performance counters and the active application information, rather than collecting all performance and application data. A straightforward approach for system data collection is real-time collection. Although this approach

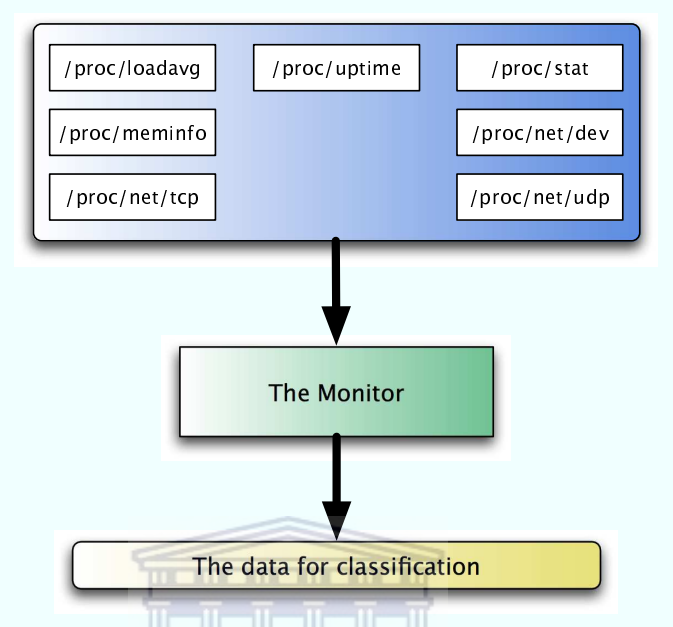

Figure 5.2: The structure of monitoring

UNIVERSITY of the
WESTERN CAPE

has the advantage that it is well understood, the system overhead incurred during *the monitor* execution would be substantial. If too little data is collected, it would be difficult to ensure that the resulting data was representative of the system class. By collecting performance counters and the active application information on the system continuously, we assure that the data accurately reflects how the system is used.

After collecting the system data from `/proc` file-system, we combine the individual performance counters from each performance object to generate a single basic output file. *The monitor* sums the counters from each performance object for each kernel module to create an aggregate file. This file includes the active application name, its open port number, and other system performance counters such as processor time, memory usage, packets sent/sec, etc. See Section 4.3. The selection of performance counters is a general problem for performance tools and is not unique to our methodology. In conventional performance tools, complete system information is collected. It is a large data collection and results in system overhead. By contrast KernTune only collects a very small amount of important performance counters and the active application information. We believe that our method is appropriate, even though complete data collection can give better classification results.

The overhead of our current monitor satisfies our main subjective performance

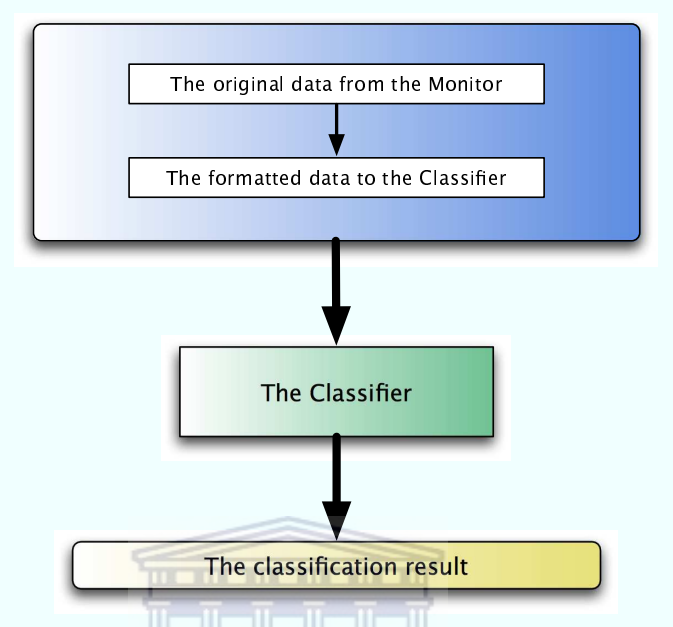

Figure 5.3: The structure of classifying

UNIVERSITY of the

goal: it is unnoticeable. Measurements of our current Monitor implementation show the overhead of system profiling is typically below 1% of CPU usage. Given this level of performance, we have chosen to defer further work on reducing monitor overhead and focus on the other functionality required for automatic optimisation.

5.4 Data Processing: *The classifier*

The classifier is implemented as a user-level daemon that periodically reads and processes the raw data samples generated by *the monitor*. Whereas *the monitor* is responsible for data collection, *the classifier* performs data translation and processing. The goals of this processing are twofold: to transform the raw data samples into a compact form that can be used directly by LibSVM and to decide when to invoke an optimisation. In our work to date, we have primarily focused on transforming raw data for use by LibSVM. Figure 5.3 shows the structure of the classifying procedure.

The classifier translates *the monitor* output file into the LibSVM file format and identifies the system class in the translated file by applying LibSVM. In the conversion from raw data to the LibSVM file format, each sample has to be translated into numeric values that the LibSVM library can recognise. The translated data provides

the information needed by LibSVM to identify the system class. LibSVM maps the translated data to an intermediate form that the LibSVM engine can process. *The monitor* output file must be scaled. To understand the need for scaling, see Section 2.3.3. Scaling the data by invoking LibSVM interface gives more accurate classification and speeds up LibSVM processing.

From the optimisation point of view, operating systems are logically composed of multiple modules, typically with CPU, memory, file-system, disk I/O and networking. The number of performance counters used, results in increasing the classification accuracy. We use the term sample set to refer to the output from *the monitor*. When processing the sample set, *the classifier* invoke the `svm_predict` interface of LibSVM and outputs a result file containing the class of system being recognised. A sample set may not include enough information to perform an accurate classification. This makes it desirable for *the classifier* to process multiple sample sets from *the monitor* and combine the results into a single, more accurate, result for *the adjustor*. The method of combining the results is simple, as each result of the classification is represented as one system class. *The classifier* sums all the results and concludes the best possible system class.

UNIVERSITY of the
WESTERN CAPE

5.5 System Tuning: *The adjustor*

The adjustor applies optimisations using the system class from *the classifier*. For this thesis, *the adjustor* implemented three different system-class optimisations, namely the Web Server, Ftp Server and Database Server classes. The input to *the adjustor* is the system class which has already been identified by *the classifier*. Using a set of suggested kernel values, *the adjustor* performed optimisations through the `sysctl` system call. The rest of this section provides a brief description of the optimisations implemented by *the adjustor*. Figure 5.4 shows the structure of the adjusting procedure.

In our implementation *the adjustor* is the simplest of the three components: when *the classifier* gives us a clear recommendation on which parameter should be adjusted and by how much, we merely have to change the parameter. However, this simple adjustment may cause technical problems: dynamically changing parameters could cause system instability while the system is performing a high workload. To address

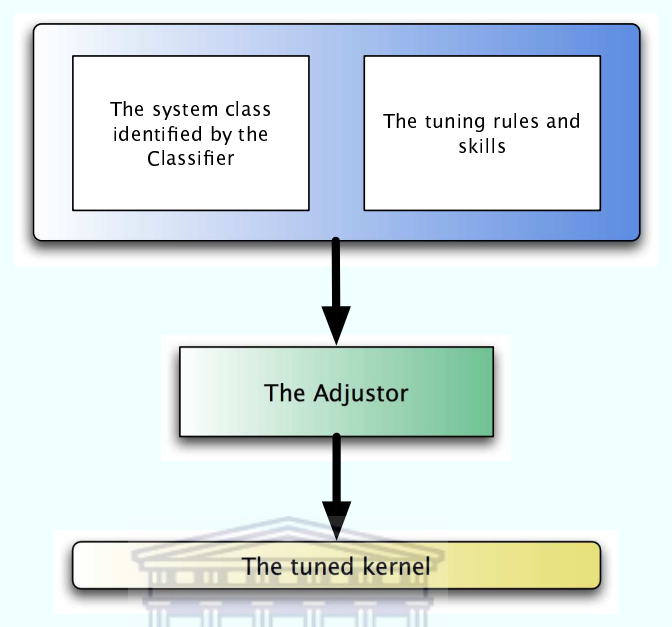

Figure 5.4: The structure of adjusting

UNIVERSITY of the
WESTERN CAPE

this problem, an adaptable mechanism is necessary for the high-workload condition. We provide an extra workload check before adjusting the parameters, to make sure the adjustment is under a low-workload. The optimisations do not have to include all tuning parameters since the parameters are used only to direct optimisation toward the most important parts of the system. Our KernTune Adjustor is currently implemented as a composition of tuning rules which optimise the most important kernel components for the system class. Considering system overhead and complexity, we have chosen to adjust the most important parameters that impact on the performance rather than adjusting every parameter in the kernel.

Since these optimisations make some system components run faster at the expense of others, a potential pitfall of the optimisation is that system might perform worse when running more interactive applications on the system. This problem could be avoided by introducing more system characteristics which represent more system classes into the training set, for example, adding the port numbers of applications or services to the sample set. This scheme is used by *the monitor* and *the classifier*.

5.6 Summary

KernTune makes it possible to optimise systems by applying SVM technology to a small number of samples. Our implementation shows that SVMs are an effective tool for system-class classification. The overhead of *the monitor* is our main consideration of the implementation because KernTune is supposed to be a performance tuning tool. If KernTune slowed down the system, the main purpose of this thesis would be useless. The 1% overhead of CPU usage satisfies our performance goal, and we will present our overhead test details in the next chapter.

UNIVERSITY of the
WESTERN CAPE

Chapter 6

Experimental Results and Conclusions

This chapter describes our experiments to evaluate automatic SVM-based optimisation with KernTune. After describing the experimental system and workloads, we present two sets of results. One set of experiments—Section 6.2—documents the correctness of classification in KernTune. The second set of experiments—Section 6.3—quantifies the effectiveness of optimisation in the KernTune Adjustor. We report the results, both in terms of correctness of classification and effectiveness of optimisation. Our experimental results show that the correctness of classification using SVM is as high 90% and the optimisations applied by KernTune achieve substantial performance improvements for our test workloads. The main criteria upon which KernTune should be evaluated is its correctness for SVM classification. This chapter also discusses issues relating to making KernTune more practical. Then our conclusions and future work will be presented.

6.1 Experimental Details

Our KernTune prototype tool used the Linux kernel 2.4.29. We ran our experiments on two Intel x86-based PCs. One machine represented a server from one of our chosen system classes. The other machine was a workload generator to simulate a real computing environment. The two systems both include a 512 KB second-level cache and 256 MB of main memory and they have exactly the same hardware configuration. System information was collected by *the monitor* every 10 seconds, producing 1 sample set every 10 seconds. The target experimental system is based on Gentoo Linux version 2004.2. The workload-generator system is based on SUSE Linux 10.0. Table 6.1 lists our hardware environment for the two experimented machines. Table 6.2 and Table 6.3 list our software environment for the experiment.

Table 6.1: Machine

Hardware	**Description**
Processor	Intel P4 1.6GHz, 512KB L2 cache
Memory	256M
Hard Disk	IDE 20GB, 2048KB Cache
Network Adapter	Intel PRO/100, 100 Mbps Full duplex

Table 6.2: Gentoo Server

Software	**Description**
Operating System	Gentoo Linux version 2004.2
Linux Kernel	Version 2.4.29 with `/proc` support
Compiler	gcc 3.3.3
File System	Linux Ext2
Performance Tool	perfmon 1.0, KernTune 1.0
Web server	Apache 2.0.52
Ftp Server	ProFtpd 1.2.10
Database Server	Mysql 4.0.24
Benchmark Tool	unixbench-4.1.0

UNIVERSITY of the
WESTERN CAPE

Table 6.3: SUSE Workload Generator

Software	**Description**
Operating System	SUSE Linux 10.0
Linux Kernel	Version 2.6.13-15 with `/proc` support
Compiler	gcc 4.0.2
File System	Linux ReiserFS
Workload Generator	http_load, dkftpbench-0.45, sql-bench
Benchmark Tool	time, httperf-0.8, dkftpbench-0.45, sql-bench

We used three workloads a web server workload, an ftp server workload and a

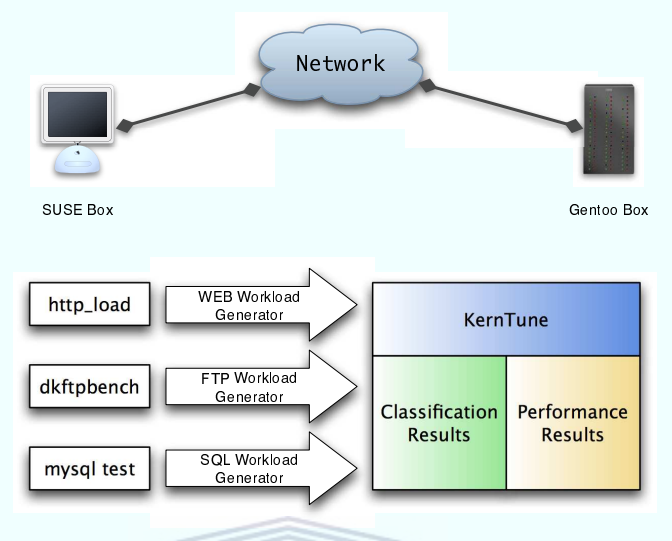

Figure 6.1: KernTune Test Bed

database workload for the three system classes in this study. Figure 6.1 shows our experimental environment for KernTune.

Table 6.4 gives a description of each workload generator, and Table 6.5 describes some of training samples we used. Table 6.6 gives summary statistics for the workloads. All experiments for this thesis were run in single-system-class mode. Two factors limited our choice of workloads. First, the system workload must be easily simulated by existing tools. As a research performance tool, KernTune does not support a full set of system classes and workloads. Second, a benchmark tool for the workload must be readily available. As a result we were unable to simulate workloads for many popular system classes. We also excluded similar training samples within a class, as they will not effect the SVM processing in *the classifier* significantly.

Table 6.4: Workload Simulation

System Class	**Workload Generator**
	continued on next page

continued from previous page

System Class	Workload Generator
web server	`http_load` runs multiple http fetches in parallel, to simulate a web server workload. It can also be considered a web server benchmark tool. Example: `./http_load -rate 10 -seconds 30 urls`
ftp server	`dkftpbench` is an ftp benchmark tool. It can be used as a ftp server workload generator. Example: `./dkftpbench -n1 -hxxx.xxx.xxx.xxx -t30 -v -uftp -pftp -fbigfile`
database server	The `mysql test suit` is a database benchmark suite for mysql. It includes test_insert, test_select, test_connect and test_create tools. They can be found in the mysql package. They can also be used as sql simulation generators to simulate database workloads. Example: `./test_insert;./test_select`

Table 6.5: Sample Training Set

Class	Name	Port	CPU	Mem	Swap	Read	Write	Received	Sent
1	660	80	0.00	0.68	0.11	0	0	100	100
1	660	80	0.00	0.68	0.11	0	0	106	100
1	660	80	0.00	0.68	0.11	0	0	100	100
1	660	80	0.00	0.68	0.11	0	0	101	100
1	660	80	0.00	0.68	0.11	0	0	100	100
1	660	80	0.00	0.68	0.11	0	0	111	110
1	660	80	0.00	0.68	0.11	0	0	100	100

continued on next page

continued from previous page

Class	Name	Port	CPU	Mem	Swap	Read	Write	Received	Sent
1	660	80	0.00	0.68	0.11	0	0	100	100
1	660	80	0.00	0.68	0.11	0	0	101	100
1	660	80	0.00	0.68	0.11	0	0	100	100
2	767	21	0.00	0.57	0.00	0	0	8	2
2	767	21	0.00	0.57	0.00	0	0	0	0
2	767	21	0.00	0.57	0.00	0	0	2	2
2	767	21	0.00	0.57	0.00	0	0	4	4
2	767	21	0.00	0.57	0.00	0	0	5	3
2	767	21	0.00	0.57	0.00	0	0	2	2
2	767	21	0.00	0.57	0.00	0	0	3	0
2	767	21	0.00	0.57	0.00	0	0	2	2
2	767	21	0.00	0.57	0.00	0	0	11	7
3	666	3306	100.00	0.46	0.00	0	0	7	0
3	666	3306	100.00	0.45	0.00	0	0	0	0
3	666	3306	100.00	0.85	0.00	0	0	6	0
3	666	3306	100.00	0.45	0.00	0	0	9	10
3	666	3306	100.00	0.85	0.00	0	0	9	10
3	666	3306	100.00	0.44	0.00	0	0	0	0
3	666	3306	100.00	0.44	0.00	0	256	7	0
3	666	3306	100.00	0.44	0.00	0	0	0	0
3	666	3306	100.00	0.43	0.00	0	0	2	0
3	666	3306	100.00	0.40	0.00	104	32	0	0

Table 6.6: Workload Statistic

Workload Generator	Number of Users	Total Time
http_load	1	1000
dkftpbench	1	1000
mysql test	1	1000

6.2 Classification Results

Table 6.7 shows our classification experimental results for the three system-class workloads. It gives the number of testing attempts and the system classes identified by *the classifier*. The accuracy is the ratio of correct classification against the total number of test attempts. There are two accuracy results in the table. One is calculated from a training set including only 40 samples. The other is calculated from a bigger training set with 400 samples. Our comparative experiment results demonstrate that increasing effective training samples improves the classification results of the SVM significantly. Classification using SVM technology achieves a high accuracy, depending on system class. However, more training samples will need more CPU time to analyse and calculate, this would lead to a high overhead of the system. Our experience with SVM suggests that an effective and small training set is needed to obtain better accuracy of classification. To determine the trade-off between the appropriate number of training samples and system overhead is important. Our experiments show that using up to 2000 samples causes tolerable overhead. Table 6.8 lists the overhead in the 400-sample case. Figure 6.2 plots the accuracy of classification for each class as the number of samples are varied. Figure 6.3 plots the overhead as the number of samples are varied. We developed `perfmon` for monitoring system performance and it is also a part of *the monitor*.

UNIVERSITY of the
WESTERN CAPE

Table 6.7: Classification Results

System Class	**Test Tool**	**Attempts**	**40 Samples**	**400 Samples**
Web Server	http_load	1000	57.8	89.8
Ftp Server	dkftpbench	1000	66.7	96.2
Database Server	test_insert	1000	77.8	97.9

Table 6.8: Overhead of KernTune

System Class	**KernTune-off (CPU%)**	**KernTune (CPU%)**	**Overhead%**
Web Server	0.00	0.99	0.99%
Ftp Server	1.98	2.97	0.99%

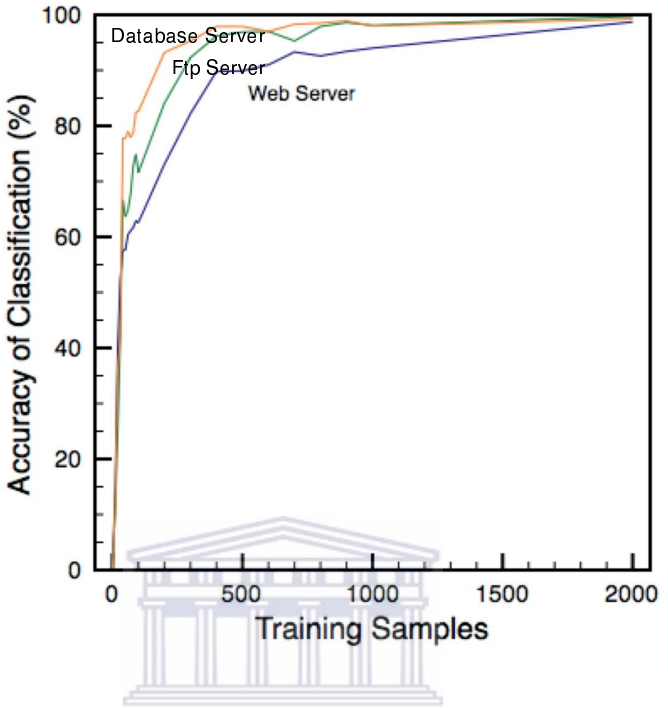

Figure 6.2: Accuracy of Classification

UNIVERSITY of the
WESTERN CAPE

System Class	KernTune-off (CPU%)	KernTune (CPU%)	Overhead%
Database Server	85.15	87.13	1.98%

6.3 Performance Results

KernTune optimised the system, and then we benchmarked the optimised system. Table 6.9 shows optimisation results of the three system classes. Each optimisation improvement given in tables 6.9 are computed from the average of ten tests in 6.10, 6.11 and 6.12. Table 6.9 shows that in each case the optimised system performs better that the original system. The optimisation results show that the benefits of SVM-based and parameter-driven optimisation are significant.

Table 6.9: Performance Results

System Class	Test Tool	Improvement
Web server	time, httperf	6.16%
Ftp server	time, dkftpbench	2.19%

System Class	Test Tool	Improvement
Database server	time, test-insert	8.04%

Table 6.10 shows the optimisation results of web server class. The test tool is `httperf`. The test command below:

```
time httperf --wsess=1000,50,0 --rate=200 --server=xxx.xxx.xxx.xxx \
     --uri=/index.html
```

means that 200 sessions are generated in one second and 1000 session will be created. Each session has 50 requests and each request is sent to fetch `index.html` file which located on the ftp server xxx.xxx.xxx.xxx. The `time` program is used to compute the CPU time. The result is the time of CPU costs, smaller is better.

Table 6.10: Web Server Results

Test	Tool	KernTune-Off	KernTune-On	Improvement
1	`time`, `httperf`	50.956	47.283	7.2%
2	`time`, `httperf`	50.993	47.043	7.7%
3	`time`, `httperf`	50.453	47.843	5.2%
4	`time`, `httperf`	50.879	47.409	6.0%
5	`time`, `httperf`	51.334	48.943	4.7%
6	`time`, `httperf`	50.339	47.463	5.7%
7	`time`, `httperf`	50.421	47.579	5.6%
8	`time`, `httperf`	50.842	47.085	7.4%
9	`time`, `httperf`	50.563	47.467	6.1%
10	`time`, `httperf`	50.675	47.636	6.0%

Table 6.11 shows the optimisation results of ftp server class. Our test tool is `dkftpbench`. The test command below:

```
time dkftpbench -hxxx.xxx.xxx.xxx -uftp -pftp -n500 -t60 -fx1000k.dat
```

means that a 1000 KB file named `x1000k.dat` is fetched by 500 users in 60 seconds. The `time` program is used to compute the CPU time. The result is the time of CPU costs, smaller is better.

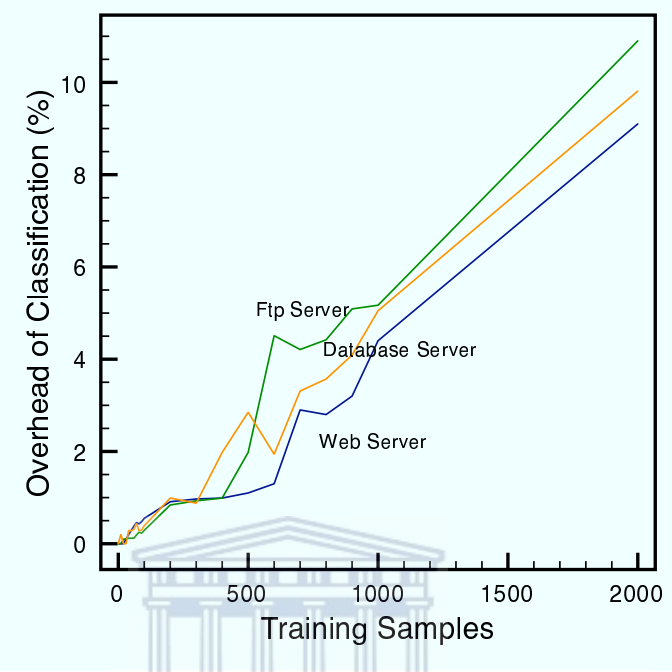

Figure 6.3: Overhead of Classification

UNIVERSITY of the
WESTERN CAPE

Table 6.11: Ftp Server Results

Test	Tool	KernTune-Off	KernTune-On	Improvement
1	`time`, `dkftpbench`	62.070	60.093	3.2%
2	`time`, `dkftpbench`	61.278	60.061	2.0%
3	`time`, `dkftpbench`	61.630	60.092	2.5%
4	`time`, `dkftpbench`	60.929	60.034	1.5%
5	`time`, `dkftpbench`	61.422	60.053	2.2%
6	`time`, `dkftpbench`	60.708	60.016	1.1%
7	`time`, `dkftpbench`	60.776	60.011	1.3%
8	`time`, `dkftpbench`	61.653	60.082	2.5%
9	`time`, `dkftpbench`	61.756	60.025	2.8%
10	`time`, `dkftpbench`	61.755	60.021	2.8%

Table 6.12 shows the optimisation results of database server. We used the test tool `test-insert`. The test command is:

Command: `time test-insert`

The `test-insert` is an SQL test script that can be found at the installation package of `mysql`. The `time` program is used to compute the CPU time. The result is the time of CPU costs, smaller is better.

Table 6.12: Database Server Results

Test	Tool	KernTune-Off	KernTune-On	Improvement
1	`time, test-insert`	36.263	33.656	7.2%
2	`time, test-insert`	36.871	33.491	9.2%
3	`time, test-insert`	36.123	33.131	8.3%
4	`time, test-insert`	36.443	33.154	9.0%
5	`time, test-insert`	36.754	33.212	9.6%
6	`time, test-insert`	36.278	33.512	7.6%
7	`time, test-insert`	36.218	33.625	7.2%
8	`time, test-insert`	36.389	33.643	7.5%
9	`time, test-insert`	36.261	33.712	7.0%
10	`time, test-insert`	36.411	33.589	7.8%

UNIVERSITY of the
WESTERN CAPE

6.4 Conclusions

Our study has demonstrated that automatic optimisation based on SVM technology is both efficient and effective. In our experience of developing KernTune, we found some issues that need to be resolved to make KernTune more practical. Our current KernTune uses a training set to train the SVM classifier. Finding and testing a good training set with minimal samples for all different system classes is very difficult, especially in a complex networking environment. The samples must be able to represent the different system usage concisely to reduce the time and overhead of SVM processing. As more system classes evolve in the future, this would lead to even more samples. It is not possible to collect those training samples manually. A tool should be developed to collect the samples automatically. Our current KernTune uses the SVM method for classifying system classes. The classification results will decide when to optimise the system. We achieved high classification accuracy with a small training set. We did not optimise the SVM classification procedure and the training set in this study. We believe that providing optimisation for the SVM process with carefully tested training samples can achieve more accurate classification. Optimising SVM also includes choosing better feature vectors for training sets, scaling training

data, choose a better kernel function, etc. As more system classes should be considered in the real computing world, this would bring a bigger training set and more complex training samples to the SVM processing. The continuous collection of system information and SVM processing can lead to overhead issues with KernTune. This issue must be fully resolved if the overhead approaches the performance improvement, since KernTune is a tuning tool to make a system run faster, and not to slow down the system. KernTune must achieve a balance between good classification accuracy and the load caused by the SVM processing. The KernTune tool has a major practical drawback, however, as it requires new training samples when applying it to a new system class. Another possibility is to develop a tool to generate samples for new classes.

This thesis describes a tool for continuous low-overhead monitoring, classifying and adjusting to meet the requirements of automatic optimisation. We achieve these low overheads through SVM-based statistical sampling and by deferring SVM processing to available idle CPU cycles. Our results show that continuous optimisation can be achieved with very low overhead and that the resulting optimisations are effective. As modern operating systems become increasingly complex, the importance of these automatic optimisations will increase. The research combines aspects of both operating systems and machine learning. Our KernTune demonstrates how a practical tool for automatic optimisation can be implemented for GNU/Linux systems. It brings a new application to the SVM area and a new approach to operating system automatic optimisation.

UNIVERSITY of the
WESTERN CAPE

6.5 Future Work

The following is a list of future work:

- Develop a tool to collect the training samples automatically.
- Tune the SVM classification and make the SVM processing faster.
- Scale and tune training sets to train SVM.
- Introduce more system classes to KernTune and generate data for the classes.
- Discover and incorporate more system optimisation techniques into KernTune.

- Optimise the SVM library—LIBSVM—built into KernTune or replace it with a better one.
- Learn system loads and collect samples in a real and complex computing environment.
- Improve test skills by introducing industry-standard test suits.
- Add the Linux kernel-2.6.x support to KernTune and bring KernTune to other operating systems.
- Optimise KernTune itself and reduce its overhead to the system.

6.6 Summary

We described details of our experiments, and gave our experimental results in this chapter. Results in this chapter are computed from the average of ten experiments. We achieved a very high classification accuracy in the first set. The second set shows our optimisation is significant. Overall, the results demonstrate that the SVM-based optimisation of KernTune is both efficient and effective. There are some issues that still need to be resolved. One of the most important issues is the trade-off between the accuracy of classification and the number of training samples. More training samples are needed to improve the accuracy, but increasing the samples will lead to system overhead. We then discussed some other issues that must be resolved before making KernTune practical. These issues are left as future work. This new research introduces opportunities to apply machine learning to operating systems.

Bibliography

[1] Ivan Bowman. Conceptual architecture of the linux kernel, 1998. Available at http://plg.uwaterloo.ca/~itbowman/CS746G/a1.

[2] H. Briceno. Design techniques for building fast servers, 1996. Available at http://citeseer.ist.psu.edu/briceno96design.html.

[3] Bill Calkins. *Inside Solaris 9*. NRG–Voices. QUE, Nov 2002.

[4] Philip Carinhas. Linux fundamentals, 2001. Available at ftp://fortuitous.com/pub/training/lf_08.26.2001.ps.gz.

[5] C. Chang and C. Lin. LibSVM: a library for support vector machines, 2001. Available at http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.ps.gz.

[6] Aptitune Corporation. *SarCheck*, 1996. Available at http://www.sarcheck.com.

[7] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. Available at http://citeseer.ist.psu.edu/cortes95supportvector.html.

[8] Nello Cristianini. Support vector and kernel machines, 2001. Available at http://www.support-vector.net/icml-tutorial.pdf.

[9] Phillip Ezolt. *Optimizing Linux Performance*. O'Reilly, 2005.

[10] Jay Fink. *An Overview of the Proc Filesystem*, 1999. Available at http://csislabs.palomar.edu/Student/csis227/ AnOverviewoftheProc Filesystem.doc.

[11] FSF. *GNU Public License*, 2006. Available at http://www.gnu.org/licenses/gpl.html.

[12] Mike Loukides Gian-Paolo D. Musumeci. *System Performance Tuning, 2nd Edition*. O'Reilly Media, 2002.

[13] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification.
Available at http://www.csie.ntu .edu.tw/~cjlin/papers/guide/guide.pdf.

[14] Matthew D. Sherer Jason R Fink. *Linux Performance Tuning and Capacity Planning*. Sams, 2001.

UNIVERSITY of the WESTERN CAPE

[15] Thorsten Joachims. Transductive inference for text classification using support vector machines. In Ivan Bratko and Saso Dzeroski, editors, *Proceedings of ICML-99, 16th International Conference on Machine Learning*, pages 200–209, Bled, SL, 1999. Morgan Kaufmann Publishers, San Francisco, US. Available at `http://citeseer.ist.psu.edu/joachims99transductive.html`.

[16] Jake Moilanen. Linux: Tuning the kernel with a genetic algorithm, Jan 2005. Available at `http://kerneltrap.org/node/4493`.

[17] Terry Dawson Olaf Kirch. *Linux Network Administrator's Guide, Second Edition*. O'Reilly, 2000.

[18] E. Osuna, R. Freund, and F. Girosi. Training support vector machines:an application to face detection, 1997. Available at `http://citeseer.ist.psu.edu/osuna97training.html`.

[19] B. Scholkopf S. Kah-Kay C. J. Burges F. Girosi P. Niyogi T. Poggio and V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE Trans*, 45:2758–2765, Nov 1997.

[20] Massimiliano Pontil and Alessandro Verri. Support vector machines for 3d object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(6):637–646, 1998. Available at `http://citeseer.ist.psu.edu/pontil98support.html`.

[21] Dustin Puryear. *Linux Kernel Tuning Using System Control*, November 2003. Available at `http://www.samag.com/documents/s=8920/sam0311a/0311a.htm`.

[22] Alessandro Rubini. *The* `sysctl` *Interface*, 1997. Available at `http://www.linuxjournal.com/article/2365`.

[23] David A Rusling. The linux kernel, 1999. Available at `http://tldp.org/LDP/tlk/tlk.html`.

[24] Badari Pulavarty Sandra K. Johnson, Gerrit Huizenga. *Performance Tuning for Linux(R) Servers*. IBM Press, 2005.

[25] Warren S. Sarle. Neural network FAQr. Available at `ftp://ftp.sas.com/pub/neural/FAQ.html`.

[26] Andrew Tanenbaum. *Modern Operating Systems*. Prentice Hall, 2 edition, Feb 2001.

[27] Arjan van de Ven. *Powertweak Linux*, 2003. Available at `http://powertweak.sourceforge.net`.

[28] Vladimir N. Vapnik. *Statistical Learning Theory*. Adaptive and Learning Systems for Sigal Processing. Wiley and Sons, Sep 1998.

UNIVERSITY of the WESTERN CAPE

[29] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, Nov 1999.

[30] V. Wan and W. Campbell. Support vector machines for speaker verification and identification, 2000. Available at `http://citeseer.ist.psu.edu/wan00support.html`.

[31] Huan-Jun Liu Yao-Nan Wang and Xiao-Fen Lu. A method to choose kernel function and its parameters for support vector machines. *IEEE*, 7:4277–4280, 2005.

[32] David Watts, Martha Centeno, Raymond Phillips, and Luciano Magalhes Tome. *Tuning Red Hat Enterprise Linux on IBM Eserver xSeries Servers*. IBM Corp., 2004. Available at `http://www.redbooks.ibm.com/redpapers/pdfs/redp3861.pdf`.

[33] Wikipedia. *Wikipedia*, 2006. Available at `http://en.wikipedia.org/wiki/keyword`.

[34] Linus Torvalds with the assistance of developers around the world. *linux-2.4.29 source*, 2005. Available at `http://www.kernel.org/pub/linux/kernel/v2.4/linux-2.4.29.tar.bz2`.

UNIVERSITY of the
WESTERN CAPE

Appendix A

/proc file-system

- kmsg

  This file is used by klogd to log kernel information and it includes hardware and kernel information.

- version

  Example of cat /proc/version

  ```
  Linux version 2.4.29 (root@long) (gcc version 3.3.3 20040412 (Gentoo
  Linux 3.3.3-r6, ssp-3.3.2-2, pie-8.7.6)) #8 Sun Jul 3 10:38:34 GMT
  2005
  ```

  UNIVERSITY of the
  WESTERN CAPE

  This file contains the kernel version. It also includes information about the build of the kernel: the user name of the user who compiled the kernel, the host name of the machine on which the kernel was compiled, the date that the kernel was compiled , and the compiler version that was used for compiling the kernel.

- cpuinfo

  Example of cat /proc/cpuinfo

  ```
  processor       : 0
  vendor\_id       : GenuineIntel
  cpu family      : 15
  model           : 2
  model name      : Intel(R) Pentium(R) 4 CPU 1.60GHz
  stepping        : 4
  ```

```
cpu MHz         : 1594.855
cache size      : 512 KB

fdiv\_bug        : no
hlt\_bug         : no
f00f\_bug        : no
coma\_bug        : no
fpu             : yes
fpu\_exception   : yes
cpuid level     : 2
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cm
bogomips        : 3139.17
```

UNIVERSITY of the
WESTERN CAPE

This file contains information about the system's processor(s). It has information about the processor number, the vendor, CPU family, model, stepping, and flags etc.

- `pci`

Example of `cat /proc/pci`

```
PCI devices found:
  Bus  0, device   0, function  0:
    Host bridge: Intel Corp. 82845 845 (Brookdale) Chipset Host Bridge (rev 3).
      Prefetchable 32 bit memory at 0xf8000000 [0xfbffffff].
  Bus  0, device   1, function  0:
    PCI bridge: Intel Corp. 82845 845 (Brookdale) Chipset AGP Bridge (rev 3).
      Master Capable.  Latency=32.  Min Gnt=10.
  Bus  0, device  30, function  0:
    PCI bridge: Intel Corp. 82801BA/CA/DB/EB PCI Bridge (rev 18).
      Master Capable.  No bursts.  Min Gnt=2.
  Bus  0, device  31, function  0:
    ISA bridge: Intel Corp. 82801BA ISA Bridge (LPC) (rev 18).
  Bus  0, device  31, function  1:
    IDE interface: Intel Corp. 82801BA IDE U100 (rev 18).
      I/O at 0xffa0 [0xfffaf].
  Bus  0, device  31, function  2:
    USB Controller: Intel Corp. 82801BA/BAM USB (Hub \#1) (rev 18).
      IRQ 11.
```

```
    I/O at 0xef40 [0xef5f].
Bus  0, device  31, function  3:
  SMBus: Intel Corp. 82801BA/BAM SMBus (rev 18).
    IRQ 9.
    I/O at 0xefa0 [0xefaf].
Bus  0, device  31, function  4:
  USB Controller: Intel Corp. 82801BA/BAM USB (Hub \#2) (rev 18).
    IRQ 10.
    I/O at 0xef80 [0xef9f].
Bus  0, device  31, function  5:
  Multimedia audio controller: Intel Corp. 82801BA/BAM AC'97 Audio (rev 18).
    IRQ 9.
    I/O at 0xe800 [0xe8ff].
    I/O at 0xef00 [0xef3f].
Bus  1, device   0, function  0:
  VGA compatible controller: nVidia Corporation NV5M64 [RIVA TNT2 Model 64/Model 64 Pro] (rev 21).
    IRQ 11.
    Master Capable.  Latency=32.  Min Gnt=5.Max Lat=1.
    Non-prefetchable 32 bit memory at 0xfd000000 [0xfdffffff].
    Prefetchable 32 bit memory at 0xf2000000 [0xf3ffffff].
Bus  2, device   8, function  0:
  Ethernet controller: Intel Corp. 82801BA/BAM/CA/CAM Ethernet Controller (rev 3).
    IRQ 11.
    Master Capable.  Latency=32.  Min Gnt=8.Max Lat=56.
    Non-prefetchable 32 bit memory at 0xfeaff000 [0xfeafffff].
    I/O at 0xdf00 [0xdf3f].
```

UNIVERSITY of the
WESTERN CAPE

This file lists the devices attached to the PCI bus or buses. These are actual PCI expansion cards and may also include devices built into the system's motherboard, plus AGP graphics cards. The file includes the device type, the device and vendor ID, the device name, IRQ, etc. It also includes features offered by the device and the resources used by the device.

- `self/`

  Example of `ls /proc/self/`

  ```
  cmdline cwd environ exe fd maps mem mounts root stat statm status
  ```

  This file is a symbolic link to the `/proc` numbered directory corresponding to the current running process.

- `net/`

  Descriptions about the network layer(s).

- `scsi/`

Contains files with information on individual scsi devices

- `mount`

Example of `cat /proc/mounts/`

```
rootfs / rootfs rw 0 0
/dev/root / ext2 rw,noatime 0 0
none /proc proc rw 0 0
none /dev/shm tmpfs rw 0 0
```

This file lists all the file-systems which have been mounted on the system. Each line lists the mounted device, the mount point, the file-system class and the flags of accessing the file-systems.

UNIVERSITY of the
WESTERN CAPE

- `kcore`

This is a core dump file (memory snapshot) for the kernel.

- `modules`

This file lists the information about the kernel modules loaded in the kernel.

- `stat`

Example of `cat /proc/stat`

```
cpu  51340588 5003346 1782103 101438756
cpu0 51340588 5003346 1782103 101438756
page 5260525 5889524
swap 138248 251312
intr 170538937 159564793 17311 0 0 0 0 0 0 0 0 19 9860204 793978 0 302630 2
disk\_io: (3,0):(303494,152939,10521050,150555,11779048)
ctxt 2293995924
btime 1147706605
processes 29261
```

---

cpu: The total of all separate CPU statistics. The four numbers following `cpu` are: user, nice, system and idle usage. These are given as in jiffers.

cpu0: If the system only have on CPU. The four numbers following `cpu` are: user, nice, system and idle usage. These are given as in jiffers.

page: The number of pages read and the number of pages written

swap: The number of pages read from swap and the number of pages written to swap

intr: The number of interrupts. The first number is the total number of interrupts for all IRQs. The following numbers are the number of interrupts for each IRQ.

disk_io: The transfer data and I/O operations for each disk. The first pair is the (majoy, minor) disk number. The remaining are the total number of I/O operations, read I/O operations, read I/O sectors, write I/O operations and write I/O sectors.

ctxt: The number of context switches.

btime: The up time in seconds of the system.

processes: The number of processes that have run since the system was booted.

- `devices`

Example of `cat /proc/devices`

```
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 ttyS
  5 cua
  7 vcs
 10 misc
```

```
 14 sound
128 ptm
136 pts
162 raw
180 usb
226 drm

Block devices:
  3 ide0
 22 ide1
```

This file lists major device numbers of the character and block devices on the system.

UNIVERSITY of the
WESTERN CAPE

- `interrupts`

Example of `cat /proc/interrupts`

```
           CPU0
  0:  159693886          XT-PIC  timer
  1:      17311          XT-PIC  keyboard
  2:          0          XT-PIC  cascade
  9:          0          XT-PIC  Intel ICH2
 10:         19          XT-PIC  usb-uhci
 11:    9875179          XT-PIC  usb-uhci, eth0
 12:     793978          XT-PIC  PS/2 Mouse
 14:     302644          XT-PIC  ide0
 15:          2          XT-PIC  ide1
NMI:          0
ERR:          0
```

This file lists the interrupts assigned to devices.

- `filesystems`

Example of `cat /proc/filesystems`

```
nodev   rootfs
nodev   bdev
nodev   proc
nodev   sockfs
nodev   tmpfs
nodev   shm
nodev   pipefs
        ext2
nodev   ramfs
        vfat
        iso9660
nodev   devpts
```

This file lists all the file-system classes known to the current kernel. Note that the contents of the file list only file-system classes that are currently loaded into the kernel.

- `/ide`

Example of `ls /proc/ide`

```
drivers hda hdc ide0 ide1 piix
```

This directory contains ide0 and ide1 which corresponds to the primary and secondary IDE controllers on the system. The subdirectories of ide0 and ide1 contains physical devices which are attached to the controllers.

- `ksyms`

This file lists kernel symbols exported by the kernel.

- `dma`

  This file lists the DMA channels occupied by the system.

- `ioports`

  This file lists all the i/o ports used by the system.

- `smp`

  This file lists information about each CPU on the system.

- `cmdline`

Example of `cat /proc/cmdline`

```
BOOT_IMAGE=kernel-2.4.29 ro root=303
```

UNIVERSITY of the
WESTERN CAPE

This file describes the command line parameters passed to the kernel at boot time.

- `sys/`

  Important kernel and network information

- `locks`

  This file describes all the files currently locked in the system.

- numbered directories

Example of `ls /proc/1518`

```
cmdline cwd environ exe fd maps mem mounts root stat statm status
```

These directories contain the information of running processes on the system. The name of each directory is the process ID of the corresponding process. These directories exist in the file-system only if the corresponding process exist

on the system. Each directory contains several files providing information about the corresponding process. Each process directory contains these entries:

`cmdline`: the command to start the process and the arguments given to it.

`cwd`: a symbolic link that points to the working directory of the current process.

`environ`: the environment of the process.

`exe`: a symbolic link that points to the program image of the process.

`fd`: a directory that contains the file descriptors opened by the process.

`maps`: information about the files which mapped into the process's address.

root: a symbolic link that points to the root directory.

`stat`: statistical information about the process.

`statm`: information about the memory used by the process.

`status`: statistical information about the process in a readable format.

UNIVERSITY of the
WESTERN CAPE

- `net/dev`

Example of `cat /proc/net/dev`

```
Inter-|   Receive                                                |  Transmit
 face |bytes    packets errs drop fifo frame compressed multicast|bytes    packets errs drop fifo colls carrier compressed
    lo:   41407     560    0    0    0     0          0         0    41407     560    0    0    0     0       0          0
  eth0:1516507022 10072918    0    0    0     0          0         0 204142743 5083396    0    0    0     0       0          0
dummy0:       0       0    0    0    0     0          0         0        0       0    0    0    0     0       0          0
```

This file contains details of data received and transmitted through the configured network interfaces. Each line of the file describes a logical network interface. The meanings of each of the columns are:

`bytes`: The total number of bytes transmitted or received by the network interface.

`packets`: The total number of packets transmitted or received by the network interface.

`errs`: The total number of errors detected by the network device driver.

`drop`: The total number of packets dropped by the network device driver.

`fifo`: The number of FIFO (First In First Out) buffer errors.

`frame`: The number of packet framing errors.

`colls`: The number of collisions detected by the network interface.

`compressed`: The number of compressed packets transmitted or received by the network device driver.

`carrier`: The number of carrier losses detected by the network device driver.

`multicast`: The number of multicast frames transmitted or received by the network device driver.

- `net/tcp`

Example of `cat /proc/net/tcp`

UNIVERSITY of the

```
 sl  local\_address rem\_address   st tx\_queue rx\_queue tr tm\_when retrnsmt   uid  timeout inode
   0: 0100007F:0CEA 00000000:0000 0A 00000000:00000000 00:00000000 00000000     0        0 1418 1 cfd4eb80 300 0 0 2 -1
   1: 00000000:0050 00000000:0000 0A 00000000:00000000 00:00000000 00000000     0        0 1431 1 cf087880 300 0 0 2 -1
   2: 00000000:0015 00000000:0000 0A 00000000:00000000 00:00000000 00000000    21        0 1592 1 ce142080 300 0 0 2 -1
   3: 00000000:0016 00000000:0000 0A 00000000:00000000 00:00000000 00000000     0        0 1414 1 cfd4e800 300 0 0 2 -1
   4: 00000000:01BB 00000000:0000 0A 00000000:00000000 00:00000000 00000000     0        0 1428 1 cf087500 300 0 0 2 -1
   5: 102710AC:0016 4A9810AC:05FB 01 00000000:00000000 02:000852ED 00000000     0        0 1272171 3 c9e9a880 24 4 11 2 2
```

This file contains TCP socket information of the kernel. Each line represents one open socket. The meanings of each of the columns are:

`sl`: The numeric labels start from 0.

`local_address`: The local IP address and port number for the socket. They are represented as hexadecimal numbers.

`rem_address`: The remote IP address and port number for the socket. They are represented the same as for the local_address.

`st`: The status of the socket.

`tx_queue:rx_queue`: The size of the transmit and receive queues.

`tr:tm_when`: Whether a timer is active for this socket and the time remaining before timeout occurs.

`retrnsmt`: Unused.

`uid`: The user ID that owns the socket.

`time-out`: Unused.

`inode`: A number that can be identified by the Linux virtual file-system.

- `net/udp`

  This file contains UDP socket information of the kernel. The format of the file is the same as tcp.

- `net/raw`

  This file contains RAW socket information of the kernel. The format of the file is the same as tcp.

UNIVERSITY of the
WESTERN CAPE