2006

Modeling operating system crash behavior through multifractal analysis, long range dependence and mining of memory usage patterns

Vijai Gandikota
West Virginia University

Modeling Operating System Crash Behavior through Multifractal Analysis, Long Range Dependence and Mining of Memory Usage Patterns

Vijai Gandikota

A Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in Computer Science

Bojan Cukic Ph.D., Chair
Katerina Goseva-Popstojanova, Ph.D.
John Atkins, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2006

ABSTRACT

Modeling Operating System Crash Behavior through Multifractal Analysis, Long Range Dependence and Mining of Memory Usage Patterns

Vijai Gandikota

Software Aging is a phenomenon where the state of the operating systems degrades over a period of time due to transient errors. These transient errors can result in resource exhaustion and operating system hangups or crashes.

Three different techniques from fractal geometry are studied using the same datasets for operating system crash modeling and prediction. Holder Exponent is an indicator of how chaotic a signal is. M5 Prime is a nominal classification algorithm that allows prediction of a numerical quantity such as time to crash based on current and previous data. Hurst exponent measures the self similarity and long range dependence or memory of a process or data set and has been used to predict river flows and network usage.

For each of these techniques, a thorough investigation was conducted using crash, hangup and nominal operating system monitoring data. All three approaches demonstrated a promising ability to identify software aging and predict upcoming operating system crashes. This thesis describes the experiments, reports the best candidate techniques and identifies the topics for further investigation.

To Prasanthi,

and Kishore.

ACKNOWLEDGEMENTS

I would like to express my deep respect and profound thanks to Dr. Bojan Cukic. Any words cannot truly express my gratitude for the opportunities, encouragement and knowledge that you have bestowed upon me and the skills that I have gathered through my interaction with him. I have learnt a lot from you both professionally and personally. I was very excited to work with you and I shall cherish the fun times with you and the team outside of work. Dr. Tim Menzies it was great to work with you on the Data Mining part of this work. Dr. Mark Shereshevsky, thank you for your guidance on the Fractal Analysis part of this work. John Atkins, you are one of the best teachers I ever had and I shall always be grateful to you for the high standards and expectations you set very early on that has set me up do well ever since. I also would like to thank Dr. Katerina for her advice on this work.

My colleagues Yan Liu, Martin Mladenovsky, Sampath Yerramalla, Srikanth Gururajan, Dejan Desovski, Paola Bracchi and Yue Jiang deserve a big thanks as well. Srikanth thanks for all the thesis discussions and being a great friend. Yan thanks for being a great friend and colleague and for all your interesting insights during the course of this work. Sam, it was a whole lot of fun to work on setting up HASRC together. Your dedication to your work will always be an inspiration to me. Martin and Dejan thanks for the umpteen number of things I learnt from you guys and for your friendship. Yue it was great to work with you.

I am fortunate for the life and guidance that my parents have given me. Their support was invaluable when I formed my desire to pursue this line of study. None of this would have been possible but for their sacrifices. I also express my deepest gratitude to my sister Prasanthi Ganti and brother in-law Kishore Ganti for their undying support and understanding during the past few years. You both will always inspire me to do better than my best in work and life.

Vijai Gandikota

March 2006

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

This thesis is a presentation of the investigations of techniques to identify, model and predict software aging in operating systems and the results of these investigations. Software Aging is a phenomena where the state of the operating system degrades over time due to transient errors. The focus of this work is the analysis of resource usage using fractals and data mining algorithms. This chapter introduces the goals of this research, background concepts and organization of the thesis.

Fractal Geometry is a relatively new field having mostly been developed during the latter part of the 20th century. However a technique being studied here was used over a century ago in the study of Nile river flows in Africa. Fractals have been found to be very useful in modelling the complex and chaotic phenomena of many events in nature. It has been shown that stock market behavior, cellular division etc. exhibit fractal behavior. About a decade ago, ground breaking research conducted at Bellcore (now Telcordia Inc.) showed that internet traffic exhibits a high degree of fractal behavior and is not poisson in nature as was previously assumed. Internet traffic was shown to also exhibit a high degree of self similarity. Self Similarity is a very important property of fractals. This knowledge has led to development of many effective models and routing strategies for internet traffic[58][26][58].

Since the subject of analysis is resource usage data, looking at appropriate data mining algorithms that allow classification and prediction of future data is one of the natural investigation choices.

If the data collected while studying the resource usages of operating system shows underlying patterns of fractal behavior or if the data can be mined for information about the state of the system, then this information or hidden characteristics can be marshalled to demonstrate the aging of the operating systems. This knowledge can be further exploited to predict optimal times and strategies for software rejuvenation.

1.1 Computer Crashes

Most industries that depend on large servers to process and run applications face a big risk to their revenues and the ability to retain customers if these servers crash due to any number of possible reasons. In most cases large volumes of important commercial transactions are processed by these servers. Most computer users these days are sure to have experienced either a system crash or "hang-up". Most of the time the reasons for the problem seem unclear. It is imperative to try to keep the system running because a downed system is revenue lost. Unplanned downtime could result in loss of data and service causing further loss in revenue.

1.2 Crash Anticipation and Early Warning System

Computer Servers in major industries process large volumes of data and also run large number of applications. Faults in either the systems or the applications or the sheer volume of data processing can cause systems to be brought down. Due to the highly sensitive information that resides on today's computer systems in various industries there is the possibility of malicious attacks like Denial of Service Attacks which result in system crashes. An Early warning system that detects the degradation and raise a red flag well in advance would allow the system administrators to try and invoke a disaster prevention process and guide the system away from the road to a crash.

1.3 Motivation for this research

This research was motivated by a need to develop software aging detection and rejuvenation systems for mission critical applications where the system is at such a remote location that it would would be virtually too difficult or would take too long to communicate with the remote system to debug any problems or command it to rejuvenate itself. Examples of such system include the deep space probes and Mars rovers sent out by NASA. Such systems travel to distances so large from earth that a signal can take a very long time to reach its destination. Manually fixing problems from the ground are very difficult. A multitude of errors either known (like flipped bits) or unknown can occur in deep space due to a

number of factors like high radiation as well as strong gravitational or electromagnetic fields in outer space. In such systems it is imperative to have an on board module that can detect software failures, especially operating system aging. Such a module can then safely transfer control to an alternate or secondary system while issuing commands to rejuvenate the primary system. Many such projects use real time operating systems. This research focusses on more widely used operating systems in order to develop techniques and test them. Common operating systems like Windows and UNIX have a wide range of uses and many mission critical systems in the industry rely on these operating systems. Failures due to aging have serious ramifications. Study of aging in these operating systems therefore has a very high value in being able to ensure a more reliable environment for critical systems. Aging studies on UNIX have been reported in [19][61] and [53] and the previous work done in this project and reported in [19] and [31] involved UNIX as well as Windows NT. In the research work presented in this thesis detailed crash testing, analysis and investigation of aging prediction techniques was done on Windows 2000 Server operating system. This is due to the fact that Windows 2000 is widely used and is a successor to Windows NT. Conclusions drawn from the work done on Windows NT were used in the current work on Windows 2000. This research was supported in part by NASA IV & V research center and by the Lane Department of Computer Science and Electrical Engineering at West Virginia University.

1.4 Operating Systems

Windows NT and 2000 to be effective with any useful software like SQL Server would need about 128 MB RAM or more. This family of operating systems do not allow direct hardware access due to security issues. The NT family uses hardware memory protection and do not allow errant tasks from destroying the operating system. The Windows NT family uses preemptive multitasking for all applications and conducts self tuning. The Windows NT operating system architecture is shown in Figure 1. Windows NT family supports Win 32 API and has a virtual memory manager as well as a registry [39].

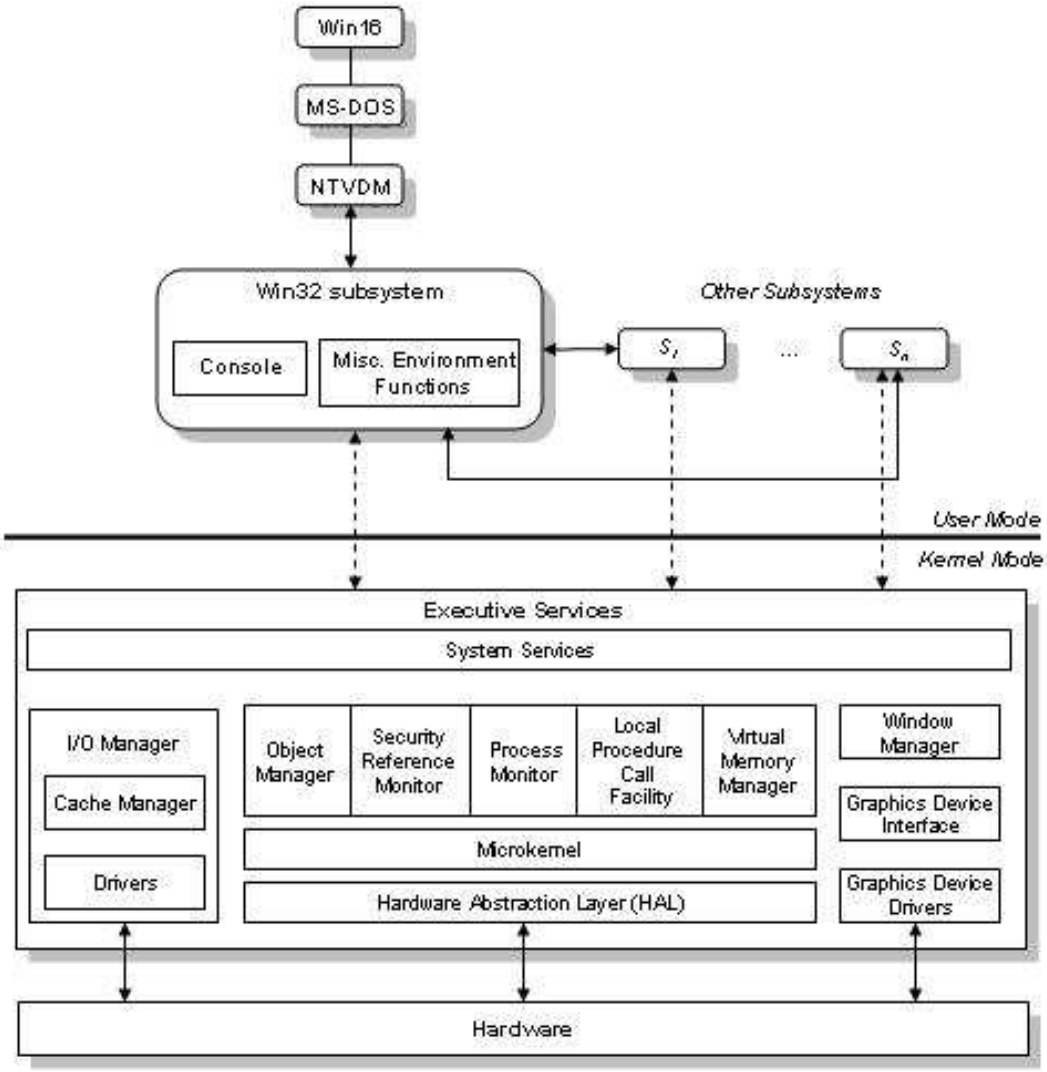

Figure 1: The Windows NT 4.0 Operating System Architecture

1.4.1 Processes

Processes in Windows NT and 2000 are programs loaded into memory, prepared for execution and include all the resources that the program needs like code, files, data, files, pipes and synchronization objects. Each process has its own private virtual address space that is shared with all the threads of the process. When there are multiple processes the operating system implements multitasking. Each new process that is created is created with a primary thread initially[39].

1.4.2 Threads

Within a process a thread is a unit of execution. It is threads that execute and not processes. All the necessary parameters to control execution including an entry point, an area for register saves used during context switching and control parameters such as base page limits are contained within a process. In a process the first thread is called the primary thread[39].

1.4.3 Scheduling

In Windows NT family of operating systems thread scheduling is done on the basis of priority classes and not based on thread priority. The thread priority determines the amount of CPU time that a thread receives relative to another thread within the same process or in other processes [32]. The priority class determines the base priority of the thread. Priority classes for *normal scheduling* include *high, normal foreground, normal background and low or idle classes.* Initially threads are selected based on the priority class and then the thread priority is considered. In the case of the five priority classes mentioned above the "normal" priority is called the *base priority.* The base priority within a priority class is relative zero. In absolute terms this base priority may have a different value. The priority of a thread is a combination of the base priority and the values of the other relative priorities. There are five relative priority levels (-2, +2)[39].

The *Kernel* is responsible for thread scheduling and object management and makes policy decisions about when a process ought to be removed from memory. The implementation

of the policy is however done by the *Executive*. Threads are dispatched by the Kernel and are controlled by a Process object which consists of the virtual address space and control information necessary for thread execution. Information contained in a process includes list of ready threads, total accumulated thread execution time, a base priority and a thread affinity[39].

As mentioned above the thread execution policies are implemented by the Executive and it consists of various managers like the Object Manager, Process manager, I/O Manager and Virtual Memory Manager. Among other things the Object Manager tracks the creation and use of an object by a process. Here objects can be *Directory objects, Symbolic Link objects, Process and Thread objects, Port objects, File Systems objects, Object-type objects, Semaphore and Event objects, Section and Segment objects, Device objects* or *File objects*. The Process Manager is responsible for the creation and deletion of the process objects, policy imposition and process hierarchy or grouping rules for processes. It tracks the Process and Thread objects. Interprocess protection is imposed by the Process Manager using a *security access token* assigned by the Security Manager[39].

CPU Scheduling leads to a requirement to share memory and message passing. *Virtual memory* is a means for the operating system to allocate more memory than physically available. A page in the virtual memory is 512 bytes long and has states *Free, Reserved, Committed and Not Present*. Page faults occur when the referenced page is not in memory. In virtual memory management, pages are mapped to to a linear address space. A *working set* is the collection of memory pages currently in use. The pages in a working set are called committed pages. Newly referenced pages which are not in the working set are earned by the tasks by causing a page fault. The newly loaded page is mapped to a physical address that may be different from its virtual address. Newly referenced pages are obtained from the page pool in memory or from the page file on the disk. If the desired page is not in memory older pages are flushed to the disk to make space for new pages. This process is also known as *demand paging.* Performance degradation can be observed during operations like file copy because the virtual memory manager swaps code pages of a task to disk to make space for file copy pages. When an application needs those code pages it has to re-initiate

their loading through page faults[39].

Memory management selection scheme depends on the hardware design of a system. Each memory management algorithm needs its own type of hardware support.

1.4.4 Windows 2000 Memory Model

In Windows 2000 operating system, the kernel heavily uses the protected-mode virtual memory management mechanisms of the Intel i386 CPU class. This operating system has been designed for the Pentium CPUs. Despite this these CPUs rely on the memory management model of Intel's 80386[45].

1.4.4.1 The 80386 memory model

Among the many innovative features of the 80386 CPU architecture like 32-bit linear addresses, twice the size of the CPU's data registers from 80286 it also introduced the concept of demand paging where a section of physical memory's contents were swapped out to disk to allow the CPU to access more memory than was physically available. This is not fast but allows a CPU to process large amounts of data that would exceed the available memory[45].

1.4.4.2 The Pentium and 80486 memory model

The Pentium and 80486 CPUs use the same segmentation and paging scheme of the 80386. Other addressing features like Physical Address Extension(PAE) of Pentium Pro, higher clock frequencies, dual instruction pipeline and other optimizations made this set of CPUs a major advancement in the hardware on which to run the Windows operating system[45].

1.4.4.3 Memory Segmentation of Windows 2000

The Windows NT and 2000 operating systems have a memory segmentation scheme in which 4-GB process address space is divided in two equal parts. Low privileged code and application data resides in the lower half and the upper half is used by the system. Low privileged code cannot execute certain instructions and cannot access restricted regions of memory[45].

1.4.4.4 Tracking memory usage

While good design and programming practices go a long way in creating software that behaves well it is also important to track memory usage for understanding how an application is using the memory of the system it runs on. [46] lists a way to track resource exhaustion by implementing memory tracking. When a block of memory is allocated on the heap but not properly released then memory leaks occur. This article presents a library of routines called Mtlib to implement memory tracking in code. The library of routines help identify a leak and which memory block is leaked. One has to then set breakpoints and trace the code that allocated that memory. This can be quite time consuming so Mtlib2 also has features to identify exactly where a leaked memory block is allocated.

1.4.5 Crashes

As mentioned in 1.4.4.3 the process address space is divided into upper and lower halves. What happens when a crash occurs? If a user-mode application accesses memory in the upper (system) memory a memory exception is thrown and the application is terminated. Kernel-mode drivers have the highest privilege for running on the system and accessing all memory locations. However if a virtual memory address is not backed up by physical or page file memory and if the kernel-mode driver accesses such an address it results in the *Blue Screen of Death*. Unlike in the case of an application memory exception, where only the application is terminated, in the case of a driver exception, the entire system is stopped and a reboot is forced. As stated in [39] a faulty GDI driver or a faulty printer driver can bring a whole Windows operating system to a crash.[45] discusses details of the Windows 2000 memory management and presents examples of building a driver that reads the memory and interacts with a user application to display the contents.

Table 1 shows the test resulting in a crash. This test was started on October 11 2002 and the crash occurred on October 12, 2002. The type of stress tests collection used from the Microsoft Stress Test Kit was *default*. Please refer to Appendix A.3 for the contents of the *default* set of stress tests. The selection method for adding processes was randomized using the SctrlRandConsole.exe file. Please refer to Appending A.1 on how to

Table 1: Blue Screen Death crashes recorded on October 31, 2002

The message displayed on a dual processor 2.2 GHz Pentium 4 server with 1GB RAM
*** STOP: 0x000000001E (0xC0000005, 0x8046749B, 0x00000000,0x00000000) KMODE_EXCEPTION_NOT_HANDLED *** Address 8046749B base at 80400000, DateStamp 384d9b17 -ntoskrnl.exe Beginning dump of physical memory Physical memory dump complete. Contact your system administrator or technical support group.
The message displayed on a 200Mhz GHz Pentium PC with 256MB RAM
On test machine Blue Screen *** STOP: 0x0000000A (0x00000000, 0x00000002,0x00000000,0x80432196) IRQL_NOT_LESS_OR_EQUAL *** Address 80432196 base at 80400000, DateStamp 384d9b17 - ntoskrnl.exe Beginning dump of physical memory Physical memory dump complete. Contact your system administrator or technical support group.

run SctrlRandConsole.exe.

We shall now take an introductory look at some of the techniques used in this work to model operating system behavior during the software aging process.

1.5 Fractals

Mandelbrot offered the following definition for the term fractal[30]: *A fractal is by definition a set for which the Hausdorff-Besicovitch dimension strictly exceeds the topological dimension.*

The topological dimension (D_T) is always an integer. This is also known as the Euclidean dimension. This definition is however too restrictive. Mandelbrot also says that *a fractal is a shape made of parts similar to the whole in some way* [30]. This being a better definition of fractal, means that at any scale a fractal looks the same.

For non-fractal sets of points both the topological dimension D_T and the Hausdorff-Besicovitch dimension D are the same. For example for a set of points that form a line, $D = 1 = D_T$. For a surface $D = 2 = D_T$. For a sphere $D = 3 = D_T$.

There also another term called the *similarity dimension* defined as in Section 2.5 of

[30] as $D_S = -lnN/lnr(N)$. In the case where the fractal is self-similar the Hausdorff-Besicovitch dimension D equals the similarity dimension.

Falconer in [9] and Crowell in [6] present the five properties of fractals as fine structure, irregularity, self-similarity, fractional dimension and recursive definition.

1.6 Self Similarity and Hurst Exponent

Self Similarity occurs in network traffic, memory reference traces, cartography, biology, and medicine. Geometric figures are similar if they have the same shape (not necessarily the same size). Self similarity can be observed when one zooms into a figure and sees the same pattern repeated. A simple example is the Sierpinski Triangle where many triangles of different sizes are enclosed in a bigger triangle. In three dimensions this can be observed with the Sierpinski Pyramid. Each of the enclosed triangle is similar to the largest one displaying self-similarity. In self-similar shapes there is a pattern that occurs repeatedly at each level of scaling applied to the original shape. This may continue to many levels of scaling(even infinite).

Hurst invented a statistical method called the rescaled range analysis method or R/S analysis method during his studies of water reservoir construction sizes. He used this to estimate the size of reservoir sizes required to collect water from the Nile river in Africa. The Hurst exponent has been shown to be useful in measuring the self-similarity or long range dependence exhibited by a process or data set. In this thesis we look at the results of the search for long range dependence in operating system memory parameters.

1.7 Data Mining and Nominal Classifiers

Machine learning is the process of learning a "concept" from a set of data which can be applied to understanding or predicting future data. This learning is carried out by computers using Machine Learning algorithms which provide the technical ability to learn. Data mining is the process of extracting or mining previously unknown and implicit information that is potentially useful and can be "learnt" and applied to similar and subsequent data. *Information* is the term used to refer to the underlying pattern found in a dataset. On the

other hand *Data* is the hard recorded facts obtained empirically from various sources in different fields of study.

In the case of empirical learning there are large data sets with a large number of exemplary cases and there are machine learning algorithms that build and revise models. In many cases many of these learners are classification learners which learn a concept and use it to classify every new instance. The data applied to the classification learners contains a single class column which is predicted. The class column contains only a discrete set of values. However there are cases when it is required that the learned model should be able to predict a numerical value associated with each instance rather than a class. Such classifiers are called nominal or numerical classifiers.

Classically continuous, numerical or nominal prediction is done by writing the predicted value as a linear sum of the parameters each multiplied by an appropriate weight. The weight determination process is called *regression* the linear prediction equation is called a *regression equation*. *Regression* refers to the process of computing an expression that predicts a numeric quantity. Their disadvantage is that most regression methods are not useful in mining nonlinear relationships. Such situations that cannot be represented well by a linear model can be modelled much better by *decision trees*.

Decision trees can be used for prediction of numerical quantities (i.e numerical classifiers). In this case the leaf nodes would contain a numeric value. This numeric value is the average of all the training set values that the leaf applies to. Such trees are called regression trees. When comparing regression equations and regression trees, the regression trees show lower values of the average absolute errors between actual and predicted values. These trees however tend to be very large and cumbersome to use and manipulate. The advantages of both approaches can be combined by replacing the average numerical values at the leaf nodes with a linear expression (regression equation). Such a tree is called a *model tree*. M5 and M5' (pronounced as M5 prime) are examples of model tree based nominal classifiers. Such trees are smaller, more comprehensible than regression trees and also have lower average error values on the training data.

1.8 Overview of the thesis

This thesis is organized as follows. Chapter two presents a review of previous work done in software aging, prediction and rejuvenation. Chapter three presents the development of a theory for software aging prediction using concepts from fractals and data mining. Chapter 4 discusses the experiments and applications developed. Chapter 5 shows results and discusses their implications. Chapter 6 presents conclusions of this research work.

CHAPTER II

REVIEW OF PREVIOUS WORK

2.1 Software Aging and Rejuvenation

2.1.1 Software Aging

Software Aging refers to the degradation of a software system, such as an operating system, over a period of hours to months as errors accumulate while the system is running and the system is not able to recover from it. Such systems are designed to run over long periods of time providing uninterrupted service, for example a server for client-server applications in an e-commerce environment. While advances are being made in better design of the operating systems the complexity of the systems and the variety of the applications as well as the resources they consume far outpaces these modifications. It is therefore very important to have a second line of defense where the system is monitored and proactive measures are taken to provide pre-disaster alerts. This is so that data can be saved, ongoing work transferred to another server and the system brought down in a controlled way, cleaned and restarted in a rejuvenated state. This process is called *Software Rejuvenation.*

As a note of caution it is necessary to point out that the software aging referred to here is not what is discussed by Parnas in [37]. In that paper Parnas discusses Software Aging from a Software Engineering standpoint. Software aging there is regarded as the inability of a software product to meet the changing needs of its users and the negative effects of changes that are made during the course of maintenance of a software product. Software obsoleteness, its prevention, costs associated with the loss of usability of the software are discussed. Recommendations for good software engineering practices are made to improve the long term viability of software. Among others these include designing for change through object orientation, designing for changes in operating systems and hardware, estimation of the probability of changes to be better prepared for them, separation of design and implementation, proper documentation, reviews, retroactive documentation and modularization

of code. His paper briefly mentions some of the issues discussed in this thesis but states that that is not what is meant as software aging in that paper[37].

2.1.2 Related Literature Software Aging

Trivedi in [49] describes a fault model for software components and the ways in which it can be handled. Software faults can be classified into *Bohrbugs*, *Heisenbugs* and *Aging related bugs*.

1. The class of bugs called *Bohrbugs* can be prevented by debugging the application. If these bugs escape detection or correction before deployment of the application then the approach to tolerating these faults is through creation of multiple applications with differing designs and implementation. This provides redundancy and masks the faults. These bugs can be reproduced and removed very easily.

2. *Heisenbugs* are a class of bugs that may not be detected except under a certain set of conditions. These conditions can be the sequence or time of occurrence of events. These transient faults may not necessarily occur when repeated. Rejuvenation can be carried out by retrying the failed operation or restarting the process that has crashed.

3. *Aging related bugs* occur due to faults that appear when the system resources approach or reach the point of exhaustion. Lack of proper programming practices resulting in memory leaks or not releasing all the resources results in exhaustion of swap space, free memory etc. A consequence of this is system hang-ups (soft failure) or a memory dump (crash) due to access of privileged memory by an un-privileged code.

Garg, Moorsel, Vaidyanathan and Trivedi in [41] proposed a methodology for detection and estimation of software aging in the UNIX operating system. *Operating system resource usage and system activity data* was collected periodically using an SNMP based distributed monitoring tool. Their analysis consisted of application of *statistical trend detection techniques* to detect resource exhaustion and is quantified through a metric that they called "estimated time to exhaustion". This was calculated using slope detection techniques. They

state that software aging is the accumulation of errors with time and also with computational load during the execution of a software eventually leading to either a memory dump of the operating system commonly referred to as a crash or an unusable state commonly referred to as a hangup. Performance degradation due to these errors is also an expected phenomenon in software aging. The kinds of errors that can cause the degradation of a software's performance include exhaustion of operating system resources, data corruption, numerical rounding errors, unreleased file locks, memory bloating and leakage and fragmentation of storage space. Despite advances in design of all kinds of operating systems Software Aging is observed in all of them. [41] lists previous work done in dependability evaluation. The data used for dependability evaluation was collected for failure events only. Garg et. al.[41] however focussed on periodically monitoring the software. The operating system object values were collected every fifteen minutes. A timeout interval of two minutes was set. If there was a timeout or an error a *"No Response"* value was recorded.

Their analysis techniques looked into the presence of a long term trend of software aging, exhibition of any periodic behavior and if there was any relation of failures to observed data values. Visual cues were used to identify if "periodicities" were observed justifying a harmonic analysis or a trend was observed necessitating a trend estimation. In the presence of both of these effects, a de-trending is needed to perform a harmonic analysis or remove the cyclic variations to do a trend analysis.

Linear dependency was studied through the plotting of correlograms. Periodicity was studied using harmonic analysis by fitting a known periodic function or a certain frequency to the data. *Smoothing of data by robust locally weighted regression* is used for detecting the trend. The true slope of the trend is estimated by a least squares estimation for prediction purposes using a non parametric linear regression methods. The seasonal Kendall test was used to test the null hypothesis of the non-existence of a trend versus the alternative hypothesis of the existence of an upward or downward trend. This was regarded as a suitable test also due to the reason that their data consisted of missing values. On detecting a trend and the slope an Estimated Time to Exhaustion is calculated.

Resource exhaustion due to a particular resource only is calculated. No study was made into the interaction and correlation of other resources that can lend themselves to a reliable prediction. So this paper proposed a methodology, a metric and suggests the usage of this metric to identify resources to be monitored but does not proposed a definite list of such resources to be monitored. Also as pointed out in a later paper [53] which is discussed next, this method of detection and estimation of Software Aging does not take into account system workload.

Vaidyanathan and Trivedi, in [53] and [43] list the causes of software aging. These causes which are called Heisenbugs are difficult to detect during system test due to their non-deterministic character and also depend on the actual run time situations. Since these bugs are difficult to detect and predict directly, other ways of indirect prediction are proposed. Vaidyanathan and Trivedi propose a semi-Markov reward model based on system workload and resource usage to estimate the time of failure of a system [53].This semi-Markov reward model is built using the workload and resource usage data obtained from UNIX machines. The semi-Markov model consists of states where transitions do not depend on states before the state that the system was in before it got to the current state. Workload was characterized and modelled with the help of various CPU and file system I/O parameters and a set of workload states was identified using the Hartigan's k-means clustering algorithm. These states were then selectively merged into a smaller set of states. To include the effect of workload on the resources a reward function was assigned to the model which corresponded to each resource considered. This reward rate r_{WR} for each workload state W and resource R is the slope of the resource value in each state. A non-parametric procedure was used to estimate the slope of a resource at each workload state. A correspondence between workload and resource exhaustion was demonstrated and a time to exhaustion calculated. However this estimator cannot be taken as an actual estimation of the failure times for the machines. The measurements were taken quite far apart in time. Relationships between multiple parameters and combinations of such parameters for analysis, was not considered.

2.1.3 Software Rejuvenation

Huang, Kintala, Kolettis and Fulton in [62] discuss approaches and costs in implementing software rejuvenation for continuously running applications as well as client server systems where the server has to be continuously available. Cost estimation models, software implementation to implement rejuvenation and experiences with implementing software rejuvenation in telecommunication systems are described in that paper.

Trivedi, Vaidyanathan, Goseva-Popstojanova in [61] present an evaluation of stochastic models for proactive fault management in operational software. They present an analytical approach to calculate optimal rejuvenation times. Two policies to implement preventive maintenance (PM) are proposed. Behavior of a system in a state under both the policies is modelled through state diagrams.

In [1] Pfening et. al. discuss how to determine an optimal time for software rejuvenation and why it is necessary to rejuvenate software even in the event of "soft failures". This makes the study of predictability of operating system crashes as well as hang-ups and slowdowns more necessary. This is because as in the case of telecommunication systems even if the system continues to function with an unacceptably high rate of packet loss, it is virtually useless or highly inefficient resulting in financial loss. Faced with such a situation, it becomes imperative to conduct software rejuvenation through a reboot of a computer or through other activities such as file system cleanup, buffer queue flushing, internal kernel table re-initialization and garbage collection.

Pfening et. al. state that a large percentage of failures are such that if a program is re-executed the failure may not necessarily occur. This is a reality even in the environment of outer space where deep space probes may not necessarily encounter the same kind of situations in which an error occurs. It is therefore essential to conduct randomized crash studies where a system is made to crash with a random set of conditions. This is the approach taken in this thesis.

Many techniques that enable the tolerance of software failures are reactive in nature. This however is not useful when faced with a prospect of losing a space mission. This thesis investigates into techniques to present a proactive approach to operating system failure

detection.

Pfening discusses two approaches for the development of a theoretical MDP model. He also discusses cost estimation for the two approaches with the simple rules as well as a numerical example.[1]

Castelli et. al. in [51] discuss software rejuvenation, prediction of software aging, prediction of time remaining for complete resource exhaustion and implementation of these strategies in IBM Xserver series.

2.2 Software Aging Prediction by Multi-fractal Analysis of Memory Usage Patterns

Crowell, Shereshevsky and Cukic [19] suggest a promising approach based on fractal analysis of patterns of resource use in a software system and try to identify a consistent pattern of behavior in the time series data of three system parameters in order to predict that the system is about to crash. The extent to which the values of these series display fractal and multi-fractal characteristics was elucidated through the calculation of the Holder exponent. Operating systems have a variety of parameters that can be measured. However not all are suitable for fractal analysis. The Holder exponent calculation is only suited to time-series of non-additive data. And each data point has to be an actual measurement of the internal state of the system at a particular point in time as opposed to some other parameters called "measures" which are a function of the interval of time and are additive. Memory use parameters are a right fit for the criteria necessary for fractal analysis because they are not regarded as a "measure" [19].

2.2.1 Three system parameters

Available Bytes is the number bytes of physical memory that is available to running processes. This consists of all the space on the "Zeroed", "Free" and "Standby" memory lists. Zeroed memory is that section of memory with pages that have been filled with zeros to hide the data used by a previous process, from processes that subsequently use this section of the memory. Free memory is memory that is available immediately for use. Standby memory is the chunk of memory that has been removed

from a process' physical memory or working set but it is still recallable if needed. Available Bytes is a last observed aggregate quantity and is neither an average nor a per unit time quantity.

Pool Paged Allocs represents the number of system calls made by the operating system for space allocation in the paged pool. The system memory consists of an area that is used for storing objects that can be swapped to a hard disk when unused. This represents the total number of calls made irrespective of the amount of space requested and like the Available Bytes it is also a last observed aggregate quantity and is neither an average nor a per unit time quantity.

System Cache Resident Bytes This parameter represents the size, in bytes, of the operating system code that is pageable (currently residing physical pages and not virtual memory pages) in the file system cache. This parameter represents the last observed value only and is not an average or a per unit time quantity.

2.2.2 Hölder Exponent

The Hölder exponent measures the fluctuation of a function at a certain point and indicates the fractality of the function at that point. Since a function can have changes in the amount of local fractality that it displays, it can have different values of Hölder exponents at different points. A continuous function $f : R \rightarrow R$ is said to be Hölder function with exponent α if

$$|f(x) - f(y)| \leq c\ |x\ -\ y|^{\alpha} \tag{1}$$

where c is some constant. If at a certain point a function is highly irregular and chaotic, it will have a lower Hölder exponent. At a smoother point the function will have a higher value. A fractal function is characterized by the fact that the Hölder exponent stays within the range of (0,1).

The Hölder exponent at a certain point in a function can be calculated as [19][6]

$$H_f(t) = \liminf_{h \rightarrow 0} \frac{log(|f(t+h) - f(t)|)}{log(|h|)} \tag{2}$$

2.2.3 Hölder Exponent Computation Algorithm

Crowell in [6] gives an algorithm to calculate the Hölder exponent for a time-series of discrete data points. This algorithm was developed by Dr. Mark Shereshevsky at West Virginia University. This algorithm is described here.

Let Y_i represent a time series of n values, $\{y_0, y_1, ..., y_n\}$ measured at uniform intervals. The value of the Hölder exponent at a point y_i is is a function of a certain number s data points that are before and after its location in the series. s is called the *Window Width* and is user specified. Thus there are $2s$ data points on either side of y_i that are taken to compute the Hölder exponent at that point. Each of these $2s$ points do not however have the same level of influence in the Hölder exponent calculation. This influence or "weight" is specified by a parameter λ called the *Weighted Regression Coefficient*. λ falls in the range (0,1) its value is adjusted to dictate how much of an influence the data points far away from the point y_i have on the Hölder exponent calculation. The values of s and λ have been chosen as 10 and 0.5 respectively for the calculations here.

1. For each integer value k ($\neq 0$) such that $-s \leq k \leq s$, a value $R_{i,k}$ is computed as

$$R_{i,k} = \frac{log|y_{i+k} - y_i|}{log(\frac{|k|}{n})} \tag{3}$$

   where $0 \leq i+k \leq n$. If on the ends of the time series sufficient values are not available on either side of y_i, s is shrunk appropriately.

2. A value $h_{i,j}$ is computed to correspond to *lim inf* computation in equation 3, where $1 \leq j \leq k$ as

$$h_{i,j} = min\{R_{i,k} : |k| \leq j\} \tag{4}$$

3. The estimate of the Hölder exponent H_i is calculated as the weighted average of the approximations h_i as

$$H_i = \frac{1-\lambda}{1-\lambda^k}(h_{i,1} + \lambda h_{i,2} + ... + \lambda^{k-1} h_{i,k}) \tag{5}$$

   As can be seen from equation 5 more weight is given to those terms that have a smaller value of k.

The veracity of this algorithm has been shown in [31].

2.2.4 Multidimensional Hölder Exponent

When more than one parameter is involved in a data set and the fractality of each parameter might change the behavior of all the parameters. The fractality of the function can be studied by computing the multidimensional Hölder exponent. Consider a data set consisting of p parameters. An observation at a time t can be represented as a point in a p-dimensional space, R^p. The distance D between two points, x=$(x_1, x_2, ..., x_p)$ and y = $(y_1, y_2, ..., y_p)$ in R^p can be computed as

$$D = ||x - y|| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_p - y_p)^2} \tag{6}$$

where "|| ||" is used to represent distance between two vectors. Thus the Hölder exponent for a function $f : R \rightarrow R^p$ is given as

$$H_f(t) = \liminf_{h \rightarrow 0} \frac{log(||f(t+h) - f(t)||)}{log(|h|)} \tag{7}$$

Crowell in [6] considered signals rather than measures for the Hölder Exponent analysis. The Hölder exponent computation equation in equation 5 assumes that the parameter being analyzed is a signal. Unlike in the case of internet traffic data analysis where the granularity of data was very fine, the measurements here are one second apart and are considered too coarse for a reliable measures based multidimensional Hölder exponent analysis. Data of memory related parameters was normalized into the range of (0,1) for a more meaningful comparison between the various parameters as the actual values of the parameters as well as the fluctuations observed differed by several orders of magnitude.

The normalization was performed by taking the maximum and minimum values of the data for each parameter and adjusting it as

$$f(x) = \frac{f(x) - min}{max - min} \tag{8}$$

This normalization preserves information from the original data set like shape of the graph and the fractality of the data set.

When the Holder exponent of a function $f(t)$ at a certain point t is greater than or equal to 1, the function is differentiable or locally smooth at that point t. That means that the magnitude of oscillations of the function near t decreases much faster than the distance to the point t. In the case of a function that is locally constant, it behaves similar to a linear function. In the case of linear functions the Hölder exponent is 1. Therefore for constant functions(i.e when the values of a string of data points in a time series did not change) the Hölder exponent was adopted as 1. This is because, the Hölder exponent calculation involves taking the logarithm of the absolute value of the difference between the values of two points. When the function is constant this difference is zero and the logarithm is undefined.

CHAPTER III

A DISCUSSION OF THE THEORY FOR THIS RESEARCH

3.1 Data Mining : Model Trees and M5' Algorithm

In this section we look into some promising continuous class prediction algorithms for software aging prediction and propose some future work in this area. The algorithm chosen for this work is M5' [57].

3.1.1 Data Mining and Machine Learning applied to Crash Data Analysis

[6] and [53] are two of some of the different approaches that are being taken in studying the crash data on various systems. However the approach taken in the work reported in this part of the thesis tries to investigate the effectiveness of applying Machine Learning Algorithms that have been published and which perform numerical prediction, to this problem. Various examples given in [40] and [8] demonstrate the effectiveness of applying model tree learners to different types of numerical data like CPU Performance, Car Prices and a variety of data sets available at the UCI Collection [50]. The CPU Performance example in [40] gives a very good example of predicting CPU Performance using M5 Model Tree Learner for data sets with numeric attributes and provided. This suggests that application of model tree learners would be a very promising approach to the study of crash data analysis. If the results of this analysis displays the ability to predict a degradation in the system it gives a very simple yet sound approach to solving this problem using commonly available machine learning algorithms.

3.1.2 Model Trees

3.1.2.1 M5 Model Tree Learner

It is a system for learning models that predict numerical values. It builds a tree based model but unlike regression trees where values are present at the leaves the trees obtained from M5 called Model Trees are binary decision trees that can have linear regression equations at the leaves. Model trees represent a piecewise linear regression approximation to an unknown function [40], [8].

Consider a set of T training instances each consisting of a fixed set of attributes and an associated numeric target value. M5 aims to build a model that associates the target values of the training instances with the values of the various attributes. These attribute values can be either numeric or discrete[40].

To build the tree M5 computes the standard deviation of the target values and the set of T training instances is split on the outcomes of a test. If T_i is the subset of cases that have the ith outcome of a potential test, the standard deviation $sd(T_i)$ of the target values of cases in T_i is treated as a measure of the error and the expected reduction in error is then represented as [40]

$$sdr = sd(T) - \sum_i (\frac{|Ti|}{|T|}) \times sd(T_i) \tag{9}$$

After examining all possible tests the test that maximizes (9) is chosen. Once the initial tree has been grown M5 determines the average residual error on the set of training instances. The residual error is the absolute difference between the actual and predicted value. This underestimates the error on new instances. In order to increase the estimated error of models with small number of training instances and a large number of parameters M5 multiplies this residual error with $\frac{(n+v)}{(n-v)}$ where n is the number of training instances and v the number of parameters in the model [40].

Next a linear model with multiple variables is constructed at each node for the training instances at the node and this model is restricted to only those attributes that are referenced somewhere in the subtree at this node. This linear model is then further simplified by

eliminating parameters to minimize the estimated error further. A greedy search tries to eliminate variables and in some cases may return only a numeric value. Next pruning is performed on the tree starting at the bottom set of non-leaf nodes. If the simplified linear model gives a smaller estimated error, the subtree maybe replaced by a leaf with the linear model. To improve the prediction accuracy of the tree-based models a smoothing process is performed at each leaf by adjusting the value given at that leaf to better reflect the predicted values at the nodes above the leaf [40].

$$P_s = \frac{(n \times p + k \times m)}{(n + k)} \tag{10}$$

Where P_s is the predicted value backed upto the node of subtree S, n is the number of training instances taking branch S_i from S and reaching its node, p is the predicted value at the node of S_i and n is the value given by the model at the node of S. k is a smoothing constant with default value 15.

Model trees are more efficient to learn and are more compact and have better prediction accuracy. Model trees can extrapolate and predict values outside the range observed in the training instances. These advantages makes them better suited to the problem at hand than other schemes like regression trees or classifiers like C 5.0 [40],[8].

Common measures of performance include the relative error which is the ratio of the variance of the predicted to the variance of the target values, correlation between the actual and predicted values.

3.1.2.2 M5' Model Tree Learner

The main difference between M5 and M5' is in the smoothing process representation. M5' [8] also improves on the handling of missing values of M5 [40] as well as a previous implementation of M5'. The adjustment to the smoothing process is made by creating a new linear model at each leaf that combines the linear models along the path back to the root. Any need for further smoothing during the prediction process is thus eliminated by the leaf model that automatically create smoothed prediction [40].

If a and b are two attributes at a leaf and have linear coefficients u and v; and b and c

are two attributes in the model at the parent node with linear coefficients w and x, then in the above equation 10 [8]

$$p = ua + vb \qquad m = wb + xc \tag{11}$$

$$P_s = \frac{(nua + (nv + kw)b + kxc)}{(n + k)} \tag{12}$$

This process is continued up to the root giving a single, smoothed linear model which is put at the leaf to be used for subsequent predictions.

3.2 Fractal Analysis : Holder Exponent

A function that displays the characteristics of more than one fractal is referred to as a multifractal function. Three parameters, were identified by Crowell, Shereshevsky and Cukic [31][19], for a Windows 2000 Server System based on a set of three predetermined criteria. These are Memory Available Bytes, Memory Pool Paged Allocations and Memory System Cache Resident Bytes. The extent to which this three dimensional data exhibited multifractality was studied by computing the Hölder Exponent at each point of the data set and then calculating the degree to which the function has a fractal dimension greater than one. Multi-fractal analysis was applied to this three-dimensional data and certain points of fractal breakdown were identified and the second fractal breakdown was chosen to signal a dangerous level of resource exhaustion. That work however did not consider any investigation into analyzing the data using already existing machine learning algorithms for prediction of continuous classes [40][8].

Most of the work done in [6] by Crowell involves parameter measurements on a UNIX system. A strong correlation between the fractality of a memory resource (lg_alloc) and the workload event of system backup. However the same level of correlation between workload and traditional measures of system workload such as percentage CPU usage was not observed. It is seen that the fractality of memory related parameters show more correlation with the workload of the system. This is not seen in sustained behavior of a parameter even it is a rise or fall. It is however seen quite clearly when there is a increase in the fluctuation

of the values of the parameter in question. This leads Crowell to state that the issue of importance might not be workload but the type of work.

Thus large processes consuming many resources and having more fluctuations might have a greater effect on the multifractality (multiple values of the Hölder exponent) of of the memory usage patterns in an operating system. This is a reason for the pattern of stress loading employed in the research work presented in this thesis.

Due to the nature of the data recorded, Crowell [6] found that one parameter dominated the multidimensional holder analysis of the data series. Therefore other parameters under different stress conditions needed to be recorded and analyzed. Crowell also found that the calculation of the fractal dimension D corresponded well to the dominant value of the Hölder exponent of the series. It was found that the series yielded different values of Hölder exponent through the series rather than one single value, however there is a dominant value of the Hölder exponent. The results therefore indicate strong fractality displayed in lg_alloc parameter.

The computation and study of the Holder exponent at a certain point of a time series is very useful for the study of fractal behavior through just the analysis of numbers in a series. This is especially more important because a series may begin to exhibit fractal behavior even without the exhibition of a dramatic rise or fall in the scale of its values. Thus the onset of fractality would be difficult to detect without a plot of an appropriate computed(moving average or holder exponent) value. Crowell further hypothesizes that a computer that exhibits high level of fractality in values of its resources might not be in a stable state. This was supported through examples of observed behavior.

Crowell suggests fractality studies of measures recorded at very small intervals as a next step. Studies suggested include Wavelet based fractal analysis and multifractal brownian motion analysis. Crowell further suggests investigation into techniques used to study network behavior such as Hurst Exponent analysis and long range dependence studies. Simulating operating system crashes to investigate the effectiveness of using fractals to predict operating system crashes was also suggested.

Table 2: An Example of a Random Walk

Random walk Step	Value(Rn = gaussian random number)
0	0
1	0+R0
2	R0+R1
3	R0+R1+R2

3.3 Fractal Analysis : Self Similarity and Hurst Exponent and Long Range Dependence

3.3.1 Random walk

Random walk is also known as brownian motion or Gaussian noise. A simple way to generate a random walk is for every time instant generate a random number and add it to the sum of the previously generated random numbers. An example is shown in TABLE 2. We can differentiate noise from predictable and deterministic components.

3.3.1.1 Autocorrelation Function

The auto-correlation measure is a displays the relationship of a random variable with itself over various lags. The autocorrelation coefficient [-1,+1]for different time lags is given by

$$\rho(k) = E[(X_t - \mu)(X_{t+k})]/\sigma^2 \tag{13}$$

3.3.1.2 Stationarity testing

The estimators for long range dependence and the hurst exponent can fail in the case of non-stationarity of data. It is therefore important to test for stationarity. A run in a time series is a sequence of identical observations. The run test in [24] can be used to detect a monotonic trend in a series with the evaluation of the number of runs. The number of runs should have a mean $\frac{N}{2} + 1$ and variance $\frac{N(N-2)}{4(N-1)}$, where N is the length of the time series.

3.3.1.3 Long memory processes and long range dependence

Long range dependence is a property of a time series which indicates whether the time series preserves some memory of events in the past. That is whether future values of a series are correlated to values in the past. This correlation can be measured through the

autocorrelation function which measures the similarity of a series is with a time shifted version of itself. If a time-series has long range dependence its auto correlation function decays slowly to zero. Whereas if it exhibits more of a short range dependence the decay is much faster. Examples of such processes are ARMA processes or Markov Processes. Hurst exponent H, can be used as a measure of the strength of the long range dependence in a process. A value of H between 0.5 and 1.0 corresponds to the process exhibiting long range dependence. The closer H is to 1.0, the stronger it exhibits long range dependence.

Mathematically a stationary process X_t exhibits long range dependence if

$$\lim_{k\to\infty} \frac{\rho(k)}{[c_p k^{-\alpha}]} = 1 \tag{14}$$

where $\rho(k)$ is the sample autocorrelation function and such that there exists a real number $\alpha \in (0, 1)$ and a constant value $c_p > 0$. Equation 15 states that $\rho(k)$ decays to zero with the approximate rate of $k^{-\alpha}$ where $H = 1 - \frac{1}{\alpha}$.

3.3.2 Hurst Exponent

Hurst invented a new statistical method called the *rescaled range analysis method or R/S analysis method* during his studies of water reservoir construction sizes. Water inflows and outflows are random processes.

The roughness of a surface is given by the the fractal dimension(D). Hurst exponent is related to the fractal dimension as in equation 1 for statistically self similar data.

$$D = 2 - H \tag{15}$$

Statistical self similarity occurs in datasets (for example network traffic) where any section of the data would have the same statistical properties as any other section and as the data set as a whole. Some examples of areas where statistical self similarity has been observed is memory reference traces, congested networks, cartography (coastlines), computer graphics, biology, and medicine. Network traffic has traditionally been assumed to have poisson distribution. However studies have found that network traffic is best modelled with a non-random hurst exponent.

3.3.3 Fractional Brownian Motion (fBM)

Random walks also known as Brownian walks (as described above) can be generated by using a defined Hurst exponent. Such a random walk will exhibit long term memory or can be described as a long term memory process if the Hurst exponent lies in a range $0.5 < H < 1.0$. This type of data sets are referred to as Fractional Brownian Motion (fBM). Fractional Brownian Motion is also referred to as $\frac{1}{f}$ noise or fractional gaussian noise (fGn) or fractal gaussian noise because these data sets are generated from Gaussian random variables or sets of numbers.

fBM can be generated by spectral synthesis using Fourier transform or the wavelet transform. In the case of fourier transform method the spectral density is inversely proportional to frequency raised to the power of β

$$\text{Spectral Density} \propto \frac{1}{f^{\beta}} \tag{16}$$

where

$$\beta \ = 2H + 1 \tag{17}$$

Small Hurst exponent $\Rightarrow$ higher fractal dimension and rougher surface

Large Hurst exponent $\Rightarrow$ lower fractal dimension and smoother surface

3.3.4 Hurst Exponent and Self Similarity

The authors in [9] give a set of very good definitions of self-similar processes. These definitions are quoted here as they were given in that paper.

1. Continuous-time : Let X_t be a continuous-time stochastic process. It is said to be strongly *self-similar* with a parameter H ($0 < H < 1$) called the Hurst exponent, if for any positive stretching factor c, the rescaled process(that is a process that is scaled or shifted to another time scale) with time scale ct, $c^{-H}X_{ct}$, is equal in distribution to the original process $\{X_t\}$. Consequently if $\{X_{t_1}, X_{t_2}, ..., X_{t_n}\}$ are the values of X_t at time instants $t_1, t_2, ..., t_n$ then for all $c > 0$, $\{c^{-H}X_{ct_1}, c^{-H}X_{ct_2}, ..., c^{-H}X_{ct_n}\}$ has the same distribution as $\{X_{t_1}, X_{t_2}, ..., X_{t_n}\}$.

2. Discrete-time process : Let us consider a discrete-time stationary process(what is a stationary process?) $\{X_k\}$ where $k = 0, 1, 2...$, with mean μ, variance σ^2, and an autocorrelation function $\{\rho_k\}$ where $k = 0, 1, 2, ...$ Then the sequence of batch means of the process are given by $\{X_k^{(m)}\}_{k=1}^{\infty} = \{X_1^{(m)}, X_2^{(m)}, ...\}$, where $X_k^{(m)}$ is given by $(X_{km-m+1} + ... + K_{km})/m, \forall k \geq 1$ and $m = 0, 1, 2, ...$

   (a) *Exactly self-similar* : Consider a discrete-time stationary process $\{X_k\}$ with $\rho_k \rightarrow k^{-\beta}$ where $k \rightarrow \infty$ and $0 < \beta < 1$. Such a process is said to be exactly self-similar with a self-similarity parameter $H = 1 - (\frac{\beta}{2})$ if $\rho_k^{(m)} = \rho_k$, $\forall m = 1, 2, 3, ...$ This means that the discrete-time stationary process $\{X_k\}$ and the averaged processes given by $\{X_k^{(m)}\}$ where $m \geq 1$, have identical correlation structure(what does this mean?).

   (b) *Asymptotically self-similar* : The discrete-time stationary process $\{X_k\}$ is said to be *asymptotically self-similar* with the Hurst exponent $H = 1-(\frac{\beta}{2})$, if $\rho_k^{(m)} \rightarrow \rho_k$, as $m \rightarrow \infty$.

The properties of self-similar processes as explained in [9] are described below.

(i)Slowly decaying variance : The variance of the sample mean decreases more slowly than the reciprocal of the sample size. $Var[\{X_k^{(m)}\}] \rightarrow c_1 m^{-\beta_1}$ as $m \rightarrow \infty$.

(ii)Long-range dependence: The autocorrelation function of the stationary process is non-summable ($\sum_{k=0}^{\infty} \rho_k = \infty$). In this case it is observed that autocorrelations have a hyperbolic decay rather than exponential.

(iii)Exhibition of the Hurst effect: Hurst found that many naturally occurring data sets can be represented well by $E[\frac{R(n)}{S(n)}] \sim c_1 n^H$, as $n \rightarrow \infty$, where $0.5 < H < 1.0$ and c_1 is a finite positive constant independent of n[52]. However for a short-range dependent process $E[\frac{R(n)}{S(n)}] \sim c_2 n^{0.5}$, as $n \rightarrow \infty$ where c_2 is a finite positive constant independent of n. This discrepancy between many naturally occurring processes and short-range dependent processes is called the *Hurst effect* or *Hurst Law*

Hurst exponent can be used to measure self similarity exhibited by a process or data set. It gives us an idea of whether a process is a pure random walk or has some underlying trends.

If the random process has an underlying trend it exhibits some degree of autocorrelation. A long term memory process is a gaussian process which has an autocorrelation that has a very long or infinite decay. Long-range dependence is same as long memory process. A long term memory process can be described as a process with a random component and the characteristic of the process is that a past event has a decaying effect on future events. The process retains some memory from past events. However this memory is slowly lost with time.

A long memory process exhibits some autocorrelation. Lets consider a data set where a random variable X has value x_i at time t_i. After a certain time interval d, X has value x_{i+d} at time t_{i+d}. Then the data set is said to exhibit autocorrelation if x_i is correlated with x_{i+d}. However the key feature of a long memory process is that this autocorrelation exhibits a decay. This decay obeys a power law. Simply put a power law is a function such as $x = y^z$. Power laws describe Brownian motion or random walk. In the case of fractals a power law that represents a random walk can also be used to describe fractal characteristics. This is shown in equation 18.

$$p(k) = Ck^{-\alpha} \tag{18}$$

where $H = 1 - \alpha/2$, k is the lag and C is a constant. Here H is the Hurst exponent and $0 < H < 1.0$. If the Hurst exponent of a process is given by $H = 0.5$, the process is a pure random walk or a Brownian time series (if we prefer that terminology) with no autocorrelation. In a Brownian time series there is no correlation between any value of a variable and the future value of that variable.

A process with Hurst exponent as $0.5 < H < 1$ represents positive autocorrelation and persistent behavior. This means that the process is a random process that follows a trend like increases are followed by increases and decreases are followed by decreases ($x_{i+d} > x_i$). Consequently such an increase can be seen even other future values of variable or process X.

A process with Hurst exponent as $0 < H < 0.5$ represents anti-persistent behavior. Such a process exhibits negative autocorrelation. That is the process is a random process

which exhibits mean reversion. That is increases are more likely to be followed by decreases $((X_{i+d} < X_i))$.

3.3.5 Hurst Exponent Estimation Methods

Hurst exponent is not calculated but is estimated. This estimation will have a certain amount of error involved. Given below are some of the techniques that can be used to estimate the Hurst exponent value for a given data set.

3.3.5.1 Rescaled Range Analysis (R/S Analysis)

Hurst exponent can be estimated using Rescaled Range Analysis (R/S Analysis). R/S Analysis was first used in reservoir storage size studies. Let us consider the water flowing into a reservoir during a certain time period(in years) T. Since the water inflow depends on the amount of rainfall and/or the inflow from other streams it is a random process. The mean of the inflow is given by

$$Mean = \frac{1}{T}\sum_{t=1}^{T}\xi(t) \tag{19}$$

If we represent the water inflow for one year as $\xi(u)$ then the deviation from the mean for that year would be $\xi(u) - Mean$.

The accumulated deviation from the mean from years 1 to T is

$X(t,T) = \sum_{u=1}^{t}\{\xi(u) - Mean\}$. The following is a simple pseudo-code to carry out this R/S Analysis.

$sum = 0$

$X(0) = 0$

$for(t = 0; t < T; t{+}{+})$

{

$sum = sum + \xi(t)$

}

$Mean = \frac{sum}{T}$

Where $\xi(t)$ is the inflow in year t.

Accumulated Deviation from mean :

$for(t = 1; t <= T; t{+}{+})$

{

$X(t) = X(t-1) + (\xi(t-1) - mean)$

}

Where $\xi(t-1)$ is the inflow in year t-1. The range, $R(T)$ is the difference between the maximum value of X and the minimum value of X over a time period T

$$R(T) = Max(X(t,T)) - Min(X(t,T)) \tag{20}$$

where $1 < t < T$ The standard deviation of the data set is given by equation 21.

$$S(T) = \sqrt{\frac{1}{T}\sum_{t=1}^{T}\{\xi(t) - Mean\}^2} \tag{21}$$

The rescaled range is given by dividing $R(T)$ by $S(T)$ as in equation 22.

$$\frac{R}{S} = \frac{R(T)}{S(T)} \tag{22}$$

where $S(T)$ = standard deviation over range 1 to T. We now calculate average R/S over multiple regions to get

$$E[R(n)/S(n)] = Cn^H \text{ as } n \to \infty \tag{23}$$

where E[R(n)/C(n)] is the expected value or mean of R/S ($H = 0.5$ for Random walk, $0.5 < H < 1.0$ for persistence and $0 < H < 0.5$ for anti-persistence)

In the case of a random walk we have equation 24.

$$E[\frac{R(n)}{S(n)}] = Cn^{0.5} \text{ as } n \to \infty \tag{24}$$

Such a process is also known as a short-range dependence. To estimate Hurst exponent, plot $log_2(R/S)$ vs $log_2(regionsize)$ and the slope of the line is the estimate of the Hurst exponent.

3.3.5.2 Wavelet Spectral Density Analysis

This technique requires a data set of size that is a power of 2. The Hurst exponent for a data set can be estimated from the wavelet spectral density or the scalogram. Wavelet octave

structure is used for Hurst exponent estimation. A wavelet spectral density plot is generated from the wavelet power spectrum. There are 2^j wavelet coefficients in a wavelet octave. The normalized power is calculated as a summation of the squares of these coefficients, which is the power, and then dividing by 2^j. Normalized power is a requirement for Hurst exponent estimation.

The Hurst exponent is proportional to the slope of a regression line through the set of $\{x_j, y_j\}$ points obtained from the wavelet spectral density. Here x_j is the octave and y_j is the log_2 of the normalized power. The estimation of Hurst exponent by this method is given in equation 25.

$$H = |\frac{(slope - 1)}{2}| \tag{25}$$

Examples of wavelet transforms that can be used are Daubechies D4, Haar and linear interpolation wavelet, which are energy normalized wavelet transforms. It is generally regarded that an energy normalized wavelet transform is required for Hurst exponent estimation. The wavelet used in combination with the type of data can determine the amount of error of the linear regression. [21] shows that for some of the analysis done in that work, the R/S estimation procedure had a lower linear regression error than that obtained from the wavelet transform method.

Other methods for Hurst exponent estimation using wavelet transforms are Haar wavelet [23] and maximum likelihood estimation, Whittle estimator etc.

3.3.5.3 Wavelet Packet Analysis

[7] presents a new scaling relation to generate the Hurst exponent from the slope of decay for the size distribution of coefficient energies calculated from Wavelet Packet Analysis.

3.3.5.4 Variance Method

In this method a log-log plot of the sample variance and the aggregation level is considered. If the plot is a straight line with slope β being greater than -1 then the series is self similar with long range dependence and the Hurst Exponent of the series is estimated as $H = 1 + \frac{\beta}{2}$[47][24][48].

3.3.5.5 Absolute Moment Method

A log-log plot of the aggregation level versus the absolute first moment of the aggregated series $X^{(m)}$ is taken. If the plot is a straight line the series exhibits long range dependence and its slope is $H - 1$ where H is the Hurst Exponent[47][24][48].

3.3.5.6 Ratio Variance of Residuals

A log-log plot is made of the aggregation level versus the average of the variance of the residuals of the series. If it is a straight line then its slope is $\frac{H}{2}$ where H is the Hurst Exponent[47][24][48].

3.3.5.7 Periodogram Method

The slope of a straight line (if observed) in a plot of log-log plot of spectral density of a time series versus the frequencies gives an estimate of the Hurst Exponent H. The periodogram is given by $I(v) = \frac{1}{2\pi n}|\sum_{j=1}^{N} X(j)e^{ijv}|^2$ where X represents the time series with length N the frequency is v[47][24][48].

3.3.5.8 Whittle

In this method the minimization of a likelihood function is applied to the Periodogram of the time series. This gives an estimation of the Hurst exponent H as well as a confidence interval[47][24][48].

3.3.5.9 Abry-Veitch

The Hurst Exponent is estimated using Wavelets and the energy of the series in various scales.

3.3.6 Long Range Dependence Detection using Bucket Shuffling

Long range dependence in a time series can be detected using a technique called Bucket Shuffling. This technique is helpful in decoupling the short range dependence correlations of a time series from the long range dependence correlations. The time series is partitioned into a series of buckets of length l. An observation X_i in the time series is said to reside in its "Home" bucket $H(i)$ given by $[\frac{i}{l}]$. Each bucket has l items. If two observations X_i, X_j

have the same home they are said to be an in-bucket pair and if not they are said to be an out-bucket pair with an offset given by $|H(i) - H(j)|$. These buckets and/or their contents are now reordered to expose short or long range dependence.

3.3.6.1 Internal Shuffling

To conduct internal shuffling the order of the buckets is kept constant and the contents of each bucket are shuffled. This preserves the long range dependencies and the short range dependencies are subdued. The autocorrelation function will continue to show power-law behavior.[47][24]

3.3.6.2 External Shuffling

The goal of external shuffling is to preserve short range dependencies or correlations of the time-series up to the length of the bucket. This is done by keeping the contents of each of the buckets intact but shuffling the order of the buckets alone. The result is that the long range correlations are subdued and the autocorrelations function does not exhibit significant correlations beyond the bucket size.[47][24]

3.3.6.3 Multiple Level Shuffling

Karagiannis in [48] presents another type of shuffling where each bucket is further subdivided into smaller internal buckets. External shuffling is applied and the order of all the smaller internal buckets that are within a larger bucket is shuffled. This helps to preserve both very short range and long range correlations and while medium range dependencies that comprise multiple smaller internal buckets within a larger bucket are reduced.

3.4 Using Hurst Exponent Estimation to analyze network traffic for self similarity

In [59] the authors give a modified definition of a self similar continuous time stochastic process X and describe the process to be exactly self-similar or asymptotically self-similar by looking at the correlation structure.

The authors of [58] further present techniques to model self-similarity. The authors suggest that using the definition of autocorrelation that they present, a process with such

a structure and which is viewed as a continuous sum of Gauss-Markov processes implies the existence of multiple levels of hierarchies of underlying mechanisms. This accounts for self similarity in the process. While it is difficult to demonstrate the physical existence of a multilevel of hierarchies of underlying behavior and to show why this would cause self similarity, some formal mathematical models have been developed that allow modeling self similar processes. The drawback however is that these techniques do not provide any explanation as to why the self similarity is observed. [58] presents two such models and a third model that attempts to give an explanation for the occurrence of self-similarity. The first two models are 1.The exactly self-similar fractional Gaussian noise 2.The class of asymptotically self-similar fractional autoregressive intergrated moving-average (ARIMA) processes.

3. A model of self-similar processes based on aggregating many simple renewal reward processes exhibiting inter-renewal times with infinite variances.

3.4.1 Impact of self similarity on network performance

3.5 Using Hurst Exponent Estimation to analyze operating system parameters for self similarity

Initially the rescaled range (R_0) is calculated for the entire data set. In this case the average ($RAVG_0$) is same as the rescaled range for the entire data set. Next the range is halved and the rescaled range is calculated for each section giving R_0 and R_1. The average of these two gives $RAVG_1$.This process is continued till the subdivided regions become very small. Each subdivision should have at least 8 data points. A table with the columns $RegionSize$, $RSAverage$, $log_2(regionsize)$, $log_2(RSAverage)$ should be created. A chart with $log_2(regionsize)$ on the x-axis and $log_2(RSAverage)$ on the y-axis should be created. A linear regression line through the plotted points is used to estimate the Hurst exponent. A plot of the autocorrelation for the data set should also be generated along with the approximate power curve.

The classical technique for R/S analysis described in the previous paragraph and in section I.E.2 uses non-overlapping data regions and where the data-size is a power of two.

However there do exist versions of the same technique where overlapping regions are used. There are also versions of this technique that aim to improve the accuracy of the classical R/S calculation as in [52]. These have to still be researched and their suitability to the problem at hand investigated.

An important issue to consider when developing code for the Hurst exponent estimation using either R/S analysis or the Wavelet analysis is that the software needs to be tested with data that has a known Hurst exponent value. Generation of the data is not a trivial issue. [12] discusses and provides a comparison of three methods to generate self similar data. The three methods considered were (i)*fast Fourier transform(FFT) algorithm* based method, (ii)*random midpoint displacement (RMD)* algorithm based method and (iii)the *successive random addition* algorithm based method.

A suggestion made in [22] was to investigate the use of the wavelet packet algorithm for Hurst exponent estimation. This is a possible path of study for estimation of Hurst exponent for operating system parameters.

Another important point to note is that the Hurst exponent by itself may provide sufficient information for forecasting and a relatively large number of data points are needed for the estimation.

CHAPTER IV

EXPERIMENTS CONDUCTED AND APPLICATIONS DEVELOPED

4.1 Design of the Experiment

To collect the data needed for the crash analysis, experiments were performed on personal computers running Windows 2000 operating system. These experiments involved continuous monitoring of operating system resources. In the work presented in this thesis, the data collection interval was set to 1 sec. In the case of a crash or the system being rendered unusable due to a hang-up the recording of the data would immediately stop as the performance tool is one of the tools running on the system.

4.2 Crash Tools

The software used for the experiments included the "Performance Logs and Alerts" Utility available in the Administrative Tools group on Windows 2000 Server and the "System Stress for Windows NT 4.0 and Windows 2000" software available in a subscription to the Microsoft Developer Network (MSDN). Unlike previous work done for this project[19][6] the process of addition of the stress scripts to start different types of stress on the machine was automated and randomized. For this purpose a Win32 Console Application called "StressCtrlRandConsole.exe" was built.

The "System Stress for Windows NT 4.0 and Windows 2000" is a set of tests that are designed to test specific areas of the operating system and to stress the capabilities of the system hardware.

On Server A, a console was started to monitor the progress of any machine that connects to it for the purposes of being stressed.

On Server B a stress command script was invoked which connected to Server A (after authentication) and copied some initial files, after which a stress manager utility was started

by the copied files to enable selection of the "type" of stress and other parameters like time of shutdown, debug options etc. The type of stress chosen for the experiments was "default". After the selection process another dialog window displays the type of stress and the available stress scripts. The "default" stress type consists of 96 stress scripts of different types like access16, word32 and vdm_fltsimulator.

One list box contains all the stress scripts available. Once a stress script is selected and added it is moved to the second window which contains a list of selected scripts. At this point the *StressCtrlRandConsole.exe* Application is invoked to take control of the *Stress Manager Dialog* window and manage the process of selecting and adding the stress scripts. Once a stress script is added and the "Go" button on the *Stress Manager Dialog* pressed, the relevant files are copied from the Server A to Server B into a "stress" folder and execution starts. As the stress scripts are added, the system would be loaded with process and memory requirements to a severe enough level to cause it to either crash or hang-up. While the crash or hang-up process naturally stops the data collection, the process was robust enough to collect data until the penultimate second before the crash. Once the system crashed or hung-up it was forcibly shutdown and restarted. At this point the stress utility's clean up program is invoked which cleans up the remnants of the stress previously performed by deleting the previous scripts. The duration of data collection was found out by checking the created and last modified times of the performance log file.

4.2.1 The StressCtrlRandConsole.exe application

This is a Win32 Console application developed for this work to automate the stress script addition and to randomize the selection and the time of addition of the script. The application was developed using Visual C++ and used the standard C++ rand() function with the current time as the seed to avoid the replication of the same sequence of random numbers each time the application is run. If there are n items left in the list of available scripts the program waits till the random number generator generates a number x between 0 to n-1 (the first item in the list has the index 0). Once the number is generated a message is sent to the Stress Manager dialog window to select the stress script having the index number x.

Later on this was changed such that the random number generated was between 0 and the number of stress scripts left in the list minus 1.

Various aspects of the Stress Manager like window and button handles as well as the messages required to be sent were discovered, by using Microsoft Spy++.

This process is repeated until all the scripts in the available lists window are selected or until the system crashes, whichever occurs first. Other variations of the program also have been developed that add the stress scripts in a linear sequence (i.e 0 to n-1).

4.2.2 The ProcessRemovalConsole.exe application

This is a Win 32 console application that was developed to randomly remove a few of the processes currently running in order to introduce a further randomness in the load increase. This was used in some of the experiments. Initially the startup list of processes on a system were logged and any additional processes added (which were essentially the stress processes) were continuously monitored. The newly added processes became candidates for random selection and removal. A process was randomly selected and removed by forcibly terminating the application. The initial list of processes, number of new processes and the process selected for removal are all displayed in the console window. All this information as well as the delay till the next process removal is sent to Delmonsrvr (discussed in section 4.2.3) on the server Agni over TCP/IP sockets. The next process removal was scheduled to occur after a random delay of $0 \leq t \leq 40$ minutes. This time was selected so that it was approximately twice the similar time period for addition of processes. Thereafter the addition of the first 30 processes every minute the delay between the addition of the remaining processes was randomly selected as $0 \leq t \leq 20$.

It was however tougher to crash or hang-up a machine while using this application in conjunction with the SCtrlRandConsole.exe application. The removal of processes (even though the rate of process removal is slower than the rate of stress addition) tends to decrease the rate of load addition to the system and requires a much longer time for the system to crash.

A Termination of a process terminates all threads of a process. For example if we have

two windows of Internet explorer running. A termination of an internet explorer process will terminate all internet explorer windows.

Hard termination of a process is dirty(there maybe DLLs hanging without their state getting updated).

4.3 Monitoring Tools

4.3.1 Performance Logs

Prior to the start of each stress experiment the a performance log was setup to record data every second for the duration of the experiment. This was setup using the Performance Logs and Alerts Utility. The log was setup to collect data on three parameters identified by Crowell [19][6], as being of interest by showing fractal behavior and not being strongly correlated to each other. This data was stored in a log file which was transferred to Server A and logged for further analysis.

The experiment was run multiple times as described below in the Experiments and Data Collection section, with the time of crash ranging from a couple of hours to 5 days. Most of the times the time to crash was around 24 hours. In some cases the system completely hung up or in some other cases the system displayed a blue screen with a pertinent memory dump message. In one case however (data was not recorded as this happened on a separate machine) all it took to crash a brand new Windows 2000 Server on a dual processor 2.54 GHz machine was to edit a paper in Microsoft Word!

4.3.2 Addmonsrvr and Delmonsrvr

Whenever processes gets added or deleted a message is sent from the machine under stress to the server Agni where two applications coded in C on UNIX, monitor the addition and deletion messages from all machines under stress and logs them in separate log files one for each machine.

4.3.3 EventLogMonitor

This application was developed on the Windows platform to remotely monitor the event logs of the remote machines. In some instances the Event Viewer in the administrative

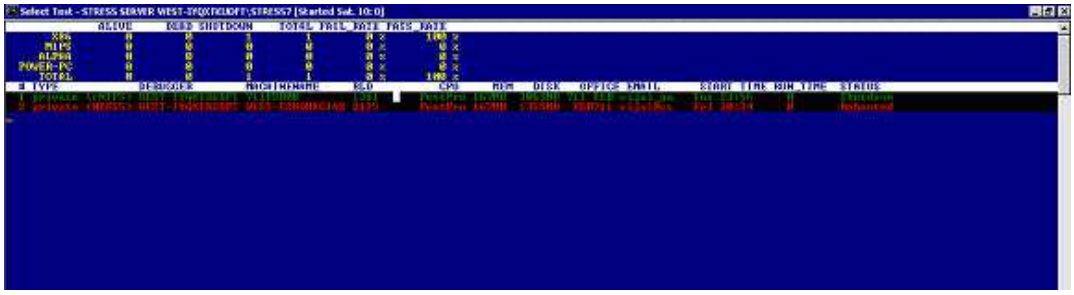

Figure 2: Microsoft Stress Kit's Shareout.exe allows the monitoring of the remote machines being tested.

tools section of Windows 2000 was examined to look for events that might correlate to the behavior of the values of the parameters recorded.

4.3.4 Shareout.exe

Before starting any stress experiment the Microsoft Stress Kit's Shareout.exe application is started. This makes the computer administering the stress scripts to the test machines as a stress server. It continuously displays the state of the the individual machines tested. The status is indicated as Shutdown, Rebooted, Dead, Alive, Aliven etc. This is shown in Figure 2. It displays type of stress, information about machine being tested like name, location, CPU, memory, hard disk capacity contact person's location and email address.

4.3.5 Remote Performance Logs Monitor

This application was developed on the windows platform to remotely monitor various system parameters of remote machines.

4.4 Experiments and Data Collection

4.4.1 Experiment Set 1

These first 10 sets of experiments were conducted by simply adding all the stress processes either manually or with the help of a primitive version of SCtrlRandConsole.exe which implemented minimal fixed delays(few seconds) between stress process addition.

StressControl.exe was developed to add stress tests from the "Default" Stress Test list. It is believed that the time period between the generation of requests to add stress tests is

constant in "normal" conditions. Here normal is defined as conditions that exist immediately after the startup of a Windows 2000 Server when no other applications are running.

Initially all the tests are cleared from the selected tests list. Subsequently tests are added from the available 96 tests in the Default test list.

The Random Stress Controller generates random positive numbers that are greater than 96(the number of stress tests available in the "default" list). The stress loading attempt starts with the addition of the first item in the list and then the subsequent items are randomly chosen based on the numbers generated by the rand() function. Basically the program waits for a number to be generated between 0 and 94(total of 95 numbers. Please remember that the first test from the 96 available ones has already been added). Once that number is generated a stress test with that index is selected from the available tests and moved over to the selected tests. Now there are 94 tests available to be added. Now the program waits for a number to be generated between 0 and 93. This continues until there are no more tests left in the available lists or the computer crashes - whichever may occur first.

1. The time duration between the generation of the requests by itself is not constant.

2. As the stress tests get added and the applications begin to run and spawn new processes the load on the system increases and this causes further delay in the servicing of a message sent to the Microsoft Stress Test Application. So there is introduced a further uncertainty about the exact time when the next random number is generated due to the load on the computer resources. In addition once the number is generated and request is dispatched to the Microsoft Stress Test Application there is more uncertainty about the exact time when it is serviced and the test actually started. Once the test scripts begin to run they generate processes based on the processor scheduling and the individual algorithm.

3. It follows from 2. that we have coarse grained control on the addition of the processes. We can only control the start of the individual test scripts and not how many individual processes or threads. Individual applications and processes are generated by the stress test script.

4. We also do not know the way the tests load the server. That is we cannot predict

the load addition / generation due to the stress tests at this point in order to be able to sequence a specific loading pattern. So it is not clear whether we can assign any specific weights to the various stress tests.

While the above unknowns exist there is a definite advantage in using the current approach. 1. The tests and their load generation algorithms are well proven and manufactured by the OS manufacturer. 2. We do not want to have too much of a control over the generation of the processes as we would like to simulate a real world scenario where process addition is a consequence of the starting of one or more applications "at random".

4.4.1.1 Hardware Setup for crash experiments

For this purpose two machines were set up with Windows 2000 Server operating system. One called Server A to act as the Stress Server and the other called Server B to be the machine that is "stressed". A block diagram representation is given in Figure 2.

Unlike previous experiments where the machines were directly connected to each other (and only each other) using a crossover cable, in these experiments the machines were left connected to the regular LAN in the Department of CSEE at West Virginia University.

The Server A machine had the following configuration. Vendor : Dell Optiplex Gn+ Processor: Intel Pentium RAM: 163, 384 KB RAM Operating System: Windows 2000 Server

Server B machine had the following configuration Vendor: Dell Optiplex GXPro 200 Processor:Intel Pentium 233 MHz RAM Base : 640 K Extended Memory 162816KB Operating System: Windows 2000 Server (Service Pack 2)

4.4.2 Experiment Set 2

In this set of experiments increasing stress was added to a machine without removing any processes. The delays between the addition of processes were random and were between 0 and 20 minutes.

4.4.2.1 Hardware Setup for crash experiments

This experiment setup was further expanded to include six computers Jupiter, Europa, Ganymede, Callisto, Agni and Vijai. Jupiter was alternately used as the server for crash scripts or as a debug machine. Europa, Ganymede and Callisto were used as the machines to be crashed. Agni was a linux server used to record information sent over TCP/IP sockets and Vijai was another server used as a server for crash scripts and remote monitoring (performance logs and event logs). This hardware setup was also used for Experiment Set 3, 4 and 5.

4.4.3 Experiment Set 3:Nominal

Nominal data was collected in three ways.

Nominal Data Locally Recorded Data was collected locally by logging parameters. There was no load except for the regular recording of data every second onto the hard disk.

Nominal Data Remotely Recorded This data was recorded One application (Internet Explorer) was started to add some minimal load just above no load. Data was collected for five consecutive days and later split up in different sizes for analysis.

Artificial Nominal Data This data was artificially generated by generating a day's worth of data with the value of the parameter kept at one constant level. This value was selected from previously recorded data.

4.4.4 Experiment Set 4

In this set of experiments stress or load was added on to a machine in an increasing and decreasing manner. Stress processes were added and with a much lower frequency processes were also removed. It was much tougher to make a computer crash in this way.

4.4.5 Experiment Set 5

Nominal data was also recorded on a Windows XP machine when there was normal daily work taking place.

4.5 Analysis Tools

4.5.1 Weka Machine Learning Tool

Weka is a comprehensive tool suite of Machine Learning algorithms. This is a freely available tool suite downloadable from the internet and is implemented in JAVA. The capabilities of Weka also include data pre-processing, data loading into various learning schemes, analysis of resulting classifiers and their performance. Weka includes classification rule learners, classification tree learners, association rule learners and clustering learners [13] [14].

The M5' implementation in Weka allows the selection of the attribute to be used as the class attribute. It provides 10-way cross-validation for performance assessment. The loaded data is split into ten equal sized blocks and each time a model was constructed using nine blocks and tested on the remaining tenth block. Another option of using a percentage data split is also provided to train on a percentage of the dataset other than 90% as in the case of 10-way cross-validation. Another approach given is to use a designated training set and test the model on the same data set. One can also supply a separate test set for testing purposes [13] [14].

The output of Weka's M5' learner includes a tree representation, its associated linear model equations and evaluation results. All these are shown in Figure 3.

4.5.2 Selfis

SELFIS is a software tool that provides implementation of various algorithms to estimate the Hurst Exponent[47][48][24]. It provides functions to carry out internal, external and multilevel bucket shuffling. Other functions like the estimation of autocorrelation function, calculation of basic statistics and estimation of the power spectrum are also provided. The tool also allows one to zoom in and out of a data plot. The tool was developed by Thomas Karagiannis. Long Range Dependence analysis is provided through Fourier and wavelet transforms as well as data transformation and cleansing algorithms. Results are very conveniently displayed in plots on the screen.

```
=== Run information ===
Scheme:     weka.classifiers.m5.M5Prime -O m -F 2.0 -V 0
Relation:   combined_opc45t.csv
Instances:  18004
Attributes: 7
            Mean1
            Stddev1
            Mean2
            Stddev2
            Mean3
            Stddev3
            Crash_Occurs_In
Test mode:  10-fold cross-validation
=== Classifier model (full training set) ===
Pruned training model tree:
Mean2 <= 84000 :
|  Mean1 <= 1.24e7 :
|  |  Mean2 <= 60300 :
|  |  |  Mean2 <= 40600 :
|  |  |  |  Mean3 <= 1.38e7 :
|  |  |  |  |  Mean3 <= 1.03e7 :
|  |  |  |  |  |  Mean3 <= 8.23e6 :
|  |  |  |  |  |  |  Stddev1 <= 869000 : LM1 (128/10.9%)
|  |  |  |  |  |  |  Stddev1 > 869000 :
|  |  |  |  |  |  |  |  Stddev1 <= 1.78e6 :
..
..
..

   LM1: Crash_Occurs_In = 5640 - 2e-5Mean1 - 6.01e-6Stddev1 - 0.0639Mean2
                  + 0.00524Stddev2 - 1.58e-6Mean3 + 7.89e-4Stddev3
   LM2: Crash_Occurs_In = 4700 - 2.19e-5Mean1 - 9.58e-6Stddev1 - 0.0531Mean2
                  + 0.00524Stddev2 + 1.02e-4Mean3 + 1.49e-5Stddev3
..
..
..
Number of Rules : 452
Time taken to build model: 11 seconds
=== Cross-validation ===
=== Summary ===
Correlation coefficient                  0.8981
Mean absolute error                    331.2964
Root mean squared error                571.6109
Relative absolute error                 29.442  %
Root relative squared error             43.9929 %
Total Number of Instances               18004
```

Figure 3: A section of the model tree generated from the first crash dataset.

4.5.3 Data Analysis 1.0

This is a windows based application developed for this work. It reads processed crash data and generates values for a nominal class. Data from one crash data set was used to obtain a decision tree in Weka. This was then manually coded into if statements which were incorporated into the application. Data from other crash tests were then filtered through this decision tree to obtain predictions made by the M5' Model Tree based nominal classifier. The predicted value was the number of intervals to crash. These predicted values were then compared with actual time to crash and plotted on screen as a graph.

4.5.4 Fractal Analysis 1.0

This is a Java Swing based application that uses the initial classes developed by Crowell [6] with some modifications. The application plots the data values, generates and plots the multidimensional Hölder Exponent and it can compute and plot the Shewhart Series and Crash Prediction plots. It is a multithreaded application and has a processing log printing in a separate threads. User can add comments for the analysis. This application also integrates the Selfis tool. However one needs to have the Selfis tool installed on their system for this feature to function. Other features like reading model trees and processing data with them are planned.

The next chapter discusses how the collected data was analyzed and presents the results.

CHAPTER V

RESULTS AND DISCUSSION

5.1 M5' Data Analysis

The initial 10 crash data sets collected were cleaned and processed before analysis by Weka's M5' learner.

5.1.1 As Is Approach

The data file consisted of data collected at one second intervals. This data was processed by combining data collected in every ten seconds and for each of the three parameters the mean and standard deviation was computed.

Each column(There are three columns in the raw data set one for each of the memory parameters being monitored) was replaced with the mean and standard deviation of each column for 10 time ticks. The date and time column was replaced by a new column which indicated the number of intervals to crash, the crash interval being designated as 0. Each of this processed data sets was called output_classified.csv. This initial processing was performed using AWK. The results were processed by Weka which generated the model tree, the Linear Model Equations and Error Estimates.

Each of the output_classified.csv files were combined into one large file and processed by Weka. Additionally four of the largest data files with greater than 4500 intervals were combined into another file to be processed by Weka. Weka generated a model tree for each data set along with error estimates.

From the results of Weka a model tree generated from the first data set was coded in Visual C++ as the Data Analysis 1.0 application discussed in the previous chapter. Each of the remaining data sets were fed to this program to generate a prediction on the number of intervals to crash. The application also calculated the difference between the predicted interval to crash times of the decision tree and the actual interval to crash time of that

Table 3: Orders of Mean and Standard Deviations

Parameter	**Value**	**Order**
Available Bytes	Mean1	10^8
Available Bytes	Stddev1	$10^7 to 10^8$
Pool Paged Allocs	Mean2	10^5
Pool Paged Allocs	Stddev2	10^3
System Cache Res. Bytes	Mean3	10^8
System Cache Res. Bytes	Stddev3	10^6

Table 4: Results of processing the Computer Crash Data Sets by the M5' Model Tree Learner.

Set	No. of Instances	Correlation Coefficient	Mean Abs Error	Root Mean Sqrd Error	Relative % Abs Error	Root Rel % Sqrd Error
1	459	0.990	13.57	18.61	11.83	14.045
2	6856	0.996	106.18	166.84	6.195	8.429
3	5164	0.927	338.76	558.05	26.24	37.435
4	356	0.96	19.27	28.77	21.65	27.997
5	541	0.925	33.26	59.51	24.59	38.105
6	5124	0.991	110.64	195.69	8.63	13.229
7	1860	0.990	47.84	74.31	10.28	13.840
8	1310	0.977	45.71	79.68	13.95	21.070
9	4957	0.999	45.31	58.57	3.656	4.093
10	3728	0.999	32.32	48.87	3.46	4.541

particular data set. This was output in a file called errorout.txt. The order of each of the mean and standard deviations are listed in Table 3 to compare with the linear regression functions on the leaves of the tree.

Mean1 corresponds to Available Bytes parameter, Mean2 corresponds to Pool Paged Allocs, Mean3 corresponds to System Cache Resident Bytes.

5.1.2 Delayed Data Approach

Another approach taken was to add six more columns to each of the output_classifed.csv files. These six columns were the mean and standard deviations computed for the three parameters in ten second intervals from a certain time period(250 intervals) in the past.

The number of attributes were seven and were *Mean1, Stddev1, Mean2, Stddev2, Mean3, Stddev3, Crash Occurs In.*

The results of M5 prime's cross-validation for all the ten data sets are shown in Table 4. All the results show a very high correlation coefficient. In addition the Relative Absolute Error % and the Root Relative Squared Error are very small. The high correlation and low

Table 5: Mean Absolute Error Comparison

Mean Absolute Error / Number of Intervals (of 10 seconds)	Percentage	DataSet	Correlation Coefficient
13/459	2.8%	1	0.9903
106/6856	1.54%	2	0.9964
338/5164	6.5%	3	0.9274
19/356	5.3%	4	0.96
33/541	6.09%	5	0.925
110/5124	0.2146%	6	0.9912
47/1860	2.5%	7	0.9904
45/1310	3.4%	8	0.9776
45/4957	0.907%	9	0.9992
32/378	0.858%	10	0.999
560/30355	1.844%	All 10 data sets combined	0.8443
331/18004	1.838%	4 data sets with 4500 Intervals each	0.898

errors indicate that this is a promising technique.

A comparison of the Mean Absolute Error obtained in Table 4 for all the sets was made along with the Mean Absolute Error obtained from the M5' processing of all the data sets put together into one set, and four data sets with more that 4500 intervals put together in another data set. These results are shown in Table 5.

From Table 5 it is seen that the mean absolute error of the combined data sets is approximately 560 intervals. This suggests that pending further analysis this technique might be useful to give an advance warning of dangerously high levels of loads (which might lead to a crash) for data sets that are larger than 560 intervals. Of the data sets used for testing this technique this was found applicable to 9 out of 13 datasets (69% success rate).

Figure 4, Figure 5 and Figure 6 show the results of analyzing the output of M5' with the other data sets. For example a decision tree was constructed using the output of M5' from data set 1. The data from the remaining data sets were applied to the tree and the resulting predicted values were plotted on a graph along with the actual values of intervals to go before crash. An error value was also plotted.

In Figure 4. the yellow line indicates the predicted value, which very closely follows the actual value. However, it is seen that in most cases the plots take the form of Figure 5. There is a gradual decline seen in the predicted values (indicating that the interval of crash

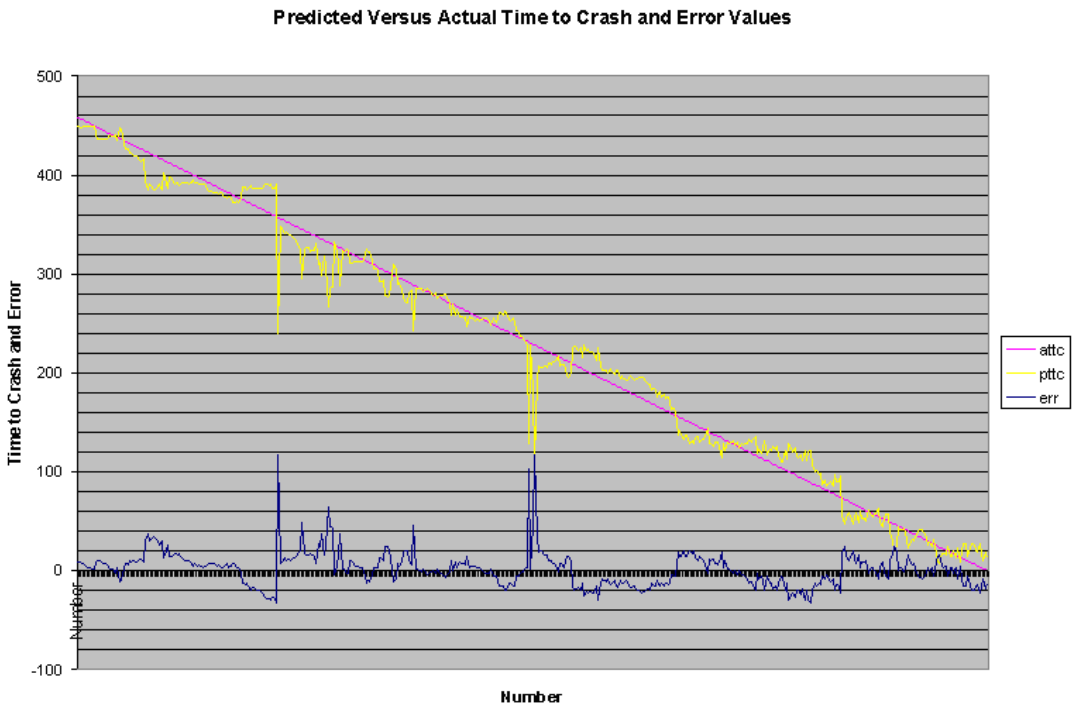

Figure 4: Data Set 1 applied to Model Tree learnt from Data Set 1

Table 6: M5' output for combined data set with data from 250 intervals in the past.

Correlation coefficient	0.9643
Mean absolute error	243.8117
Root mean squared error	456.2966
Relative absolute error	16.8568 %
Root relative squared error	26.8336 %
Total Number of Instances	30355

is approaching), however the scale of these seems much smaller than the actual values.

In addition the graph hugs the origin line. Further processing of this predicted values may be necessary like application of a linear correction to the predicted value so that it closely matches the actual value and is more useful in actual prediction. Further analysis in terms of applying the data to other more general and suitable decision trees is also warranted to further investigate this technique. The high correlations seen in the outputs of M5' certainly are very promising as this indicates a linear relationship between the actual and predicted values. Another advantage of this technique is that it is a very simple way to combine the effects of all the three parameters.

The result of processing the output_classified.csv along with six attributes which are

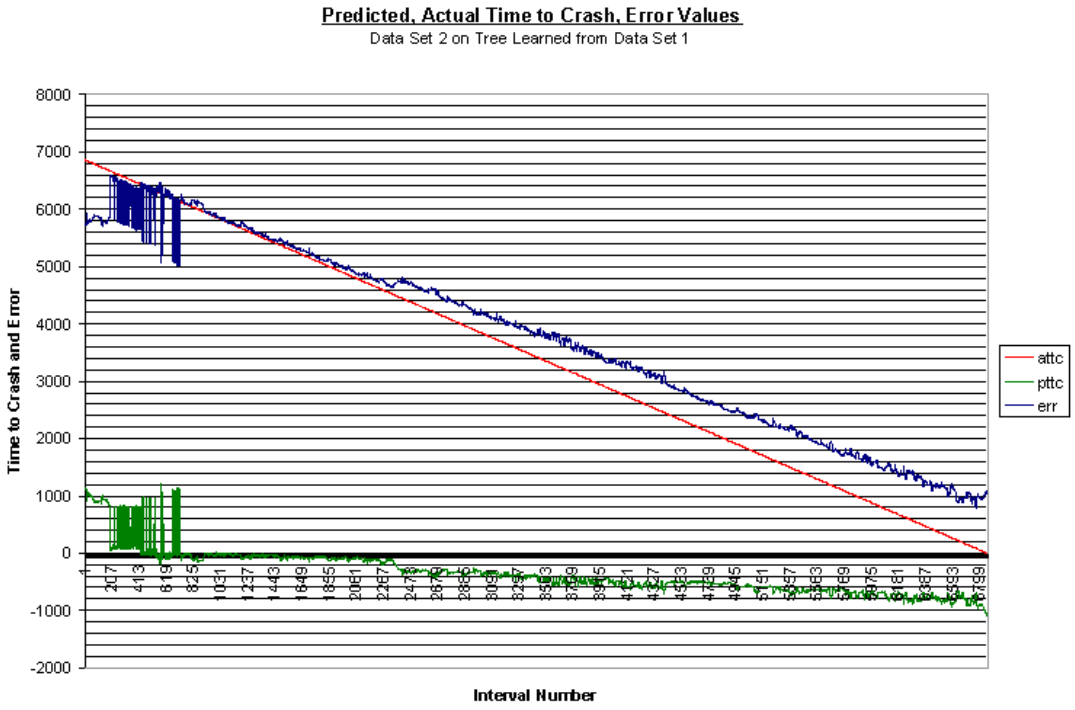

Figure 5: Data Set 2 applied to Model Tree learnt from Data Set 1

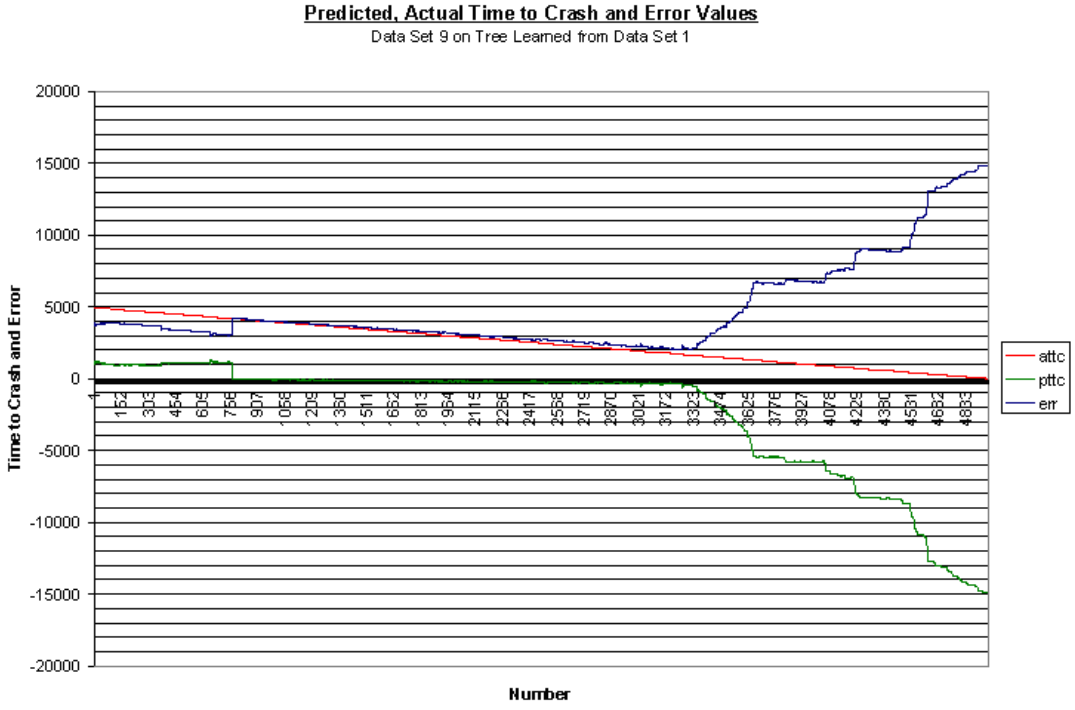

Figure 6: Data Set 9 applied to Model Tree learnt from Data Set 1

Table 7: M5' output with 2-fold cross-validation for combined data set with data from 250 intervals in the past.

Correlation coefficient	0.9184
Mean absolute error	369.9304
Root mean squared error	674.1185
Relative absolute error	25.5764
Root relative squared error	39.6432
Total Number of Instances	30355

basically the original attributes from a certain time period in the past (250 intervals) yielded the results shown in Table 6. The same data set used for training was also used for testing. 10-way cross-validation and 5-fold cross-validation of M5' in Weka did not complete for this data set.

This shows a high correlation coefficient of 0.96 again indicating a very strong linear relationship between the actual and predicted values.

An interesting fact from Table 6 is that the Mean absolute error for the combined data set that also considers the values from 250 intervals prior to the values of the current instance have a much lower Mean Absolute Error that is very close to the interval difference between the current instance values and prior instance values.

Table 7 shows the cross-validation summary values for 2 fold cross-validation which did complete and still indicates a much lower mean absolute error than in the case of all the data sets just combined together.

5.2 Holder Exponent Analysis

5.2.1 Crash Data Analysis

Of the 29 crash or hang up data sets collected through experiments, 26 were used for this analysis. For each of the data sets Multi-dimensional Hölder and Shewhart Series were computed. Crash Point Predictions were plotted and their number and location in the data set were recorded. The crash point predictions ranged between 0 and 2. A 0 indicated no crash/hang up. This is a false positive since all the data sets involved a crash or hang up. Data sets having only one crash point predictor was regarded as having a warning. In those data sets having two crash point predictors the second predictor was regarded as a definite

Table 8: Shewhart Series Parameter Values

No.	Local Avg. Time	Dev. Threshold	Change Conf. Time
1	25	5	15
2	50	5	15
3	25	5	25
4	50	5	25
5	25	5	40
6	50	5	40

indicator for a crash.

The analysis involved data sets that had increasing as well as increasing and with some decreasing levels of loads. That is there were data sets with constantly increasing load levels and there were also data sets where as the load increased some processes were randomly removed.

This analysis was repeated for six different combinations of the Shewhart parameters. These parameters are Local Averaging Time, Deviation Threshold and Change Confirmation Time. The Shewhart Series parameter combinations considered are listed in Table 8.

Of The resulting 156 sets of analysis experiments, 88 showed 2 spikes (56.41%), 41 showed 1 spike (26.28%) and the rest showed no spikes. That is in 82.69% of the cases there was at least 1 spike. In all the six sets the data sets were actually the same. The Shewhart parameters that most accurately predicted crashes by demonstrating the most number of dual spikes were 50, 5, 15 (92.3%)and 50, 5, 25 (88.5%).

Figure 7 displays the distribution of the Crash Times. Of the 26 data sets four crashed in less than 14,000 intervals. Six data sets crashed in between 30,000 and 40,000 intervals.

Figure 8 displays the distribution of the time of occurrence of the second spike as a percentage of the whole data set size.

Figure 9 displays the distribution of time to crash from the second spike.

Figure 10 displays three plots. The top one is the plot for each data set indicating the number of intervals to the occurrence of the second spike in each of those data sets. The x-axis is the data set and the y axis is the number of intervals. The middle plot in figure 10 displays the location of the second spike in the data set as a percentage of the whole data

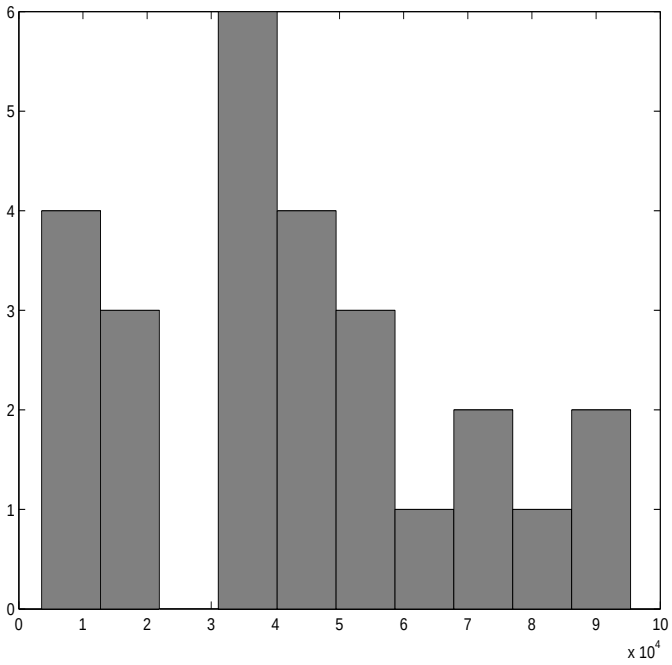

Figure 7: Crash time distributions

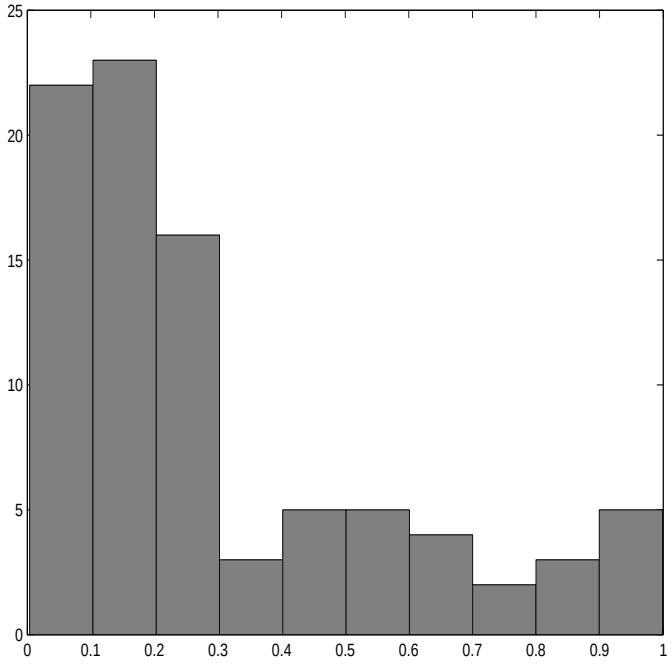

Figure 8: Distribution of percent time to second spike from start of experiment

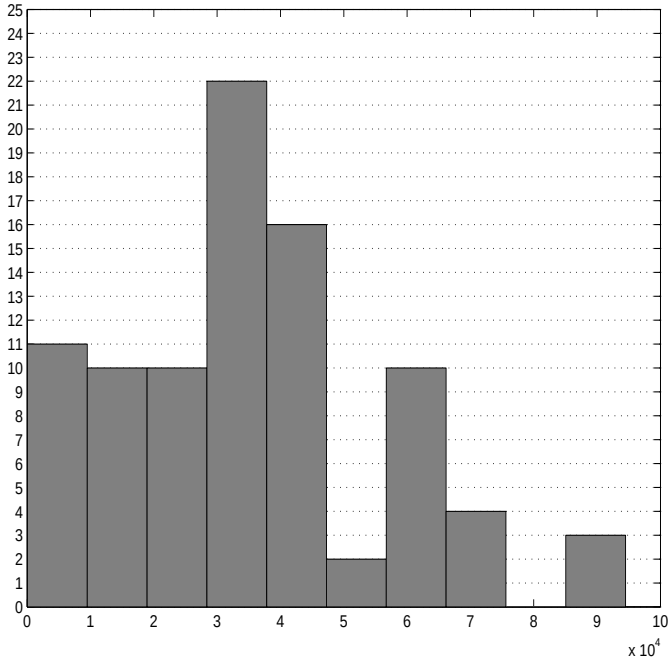

Figure 9: Distribution of second spike to the time to crash

set size. The last (lowest) plot in figure 10 shows the time to crash from the second spike.

In figure 11, the data sets are sorted in the order of the time to crash for all data sets. As can be seen the time to crash is not constant and this is in part due to the randomness in the addition and removal of load. Each experiment is different from another one in the order of the load addition as well as the time of addition of loads. In experiments where processes were removed there is a randomness in the processes eligible for removal(this depends on the processes that were added) as well as the exact process removed and the time it was removed.

5.2.2 Nominal Data Analysis

In [] Crowell presented a novel idea to predict operating system crashes. However no analysis was performed on data from computers that did not crash. Analysis on non-crash data is essential to eliminate false positives and validate that the technique truly predicts crashes or hang-ups in data sets obtained from crashed machines. Data sets were collected as explained in the previous chapter to study nominal(non-crash) data sets. These data sets were subjected to the Holder analysis to study the behavior of the technique. Figure 12

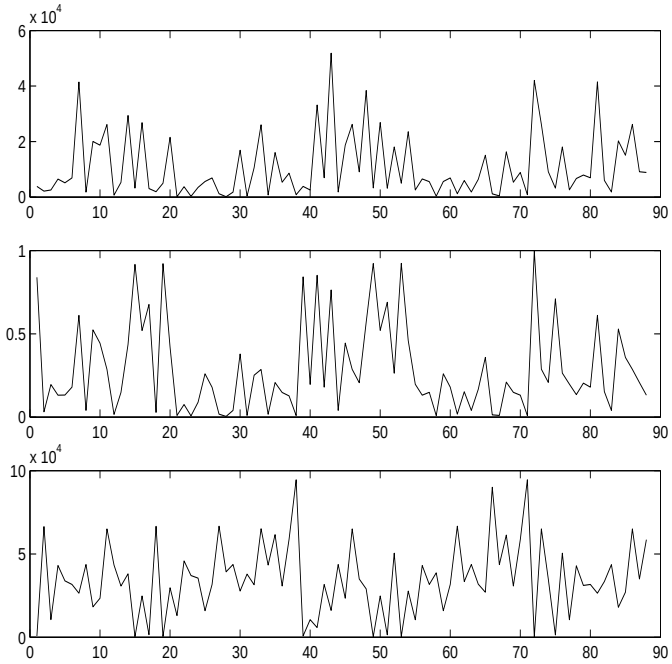

Figure 10: Second spike occurrence location, Second Spike location (Percent), Spike to Crash Time

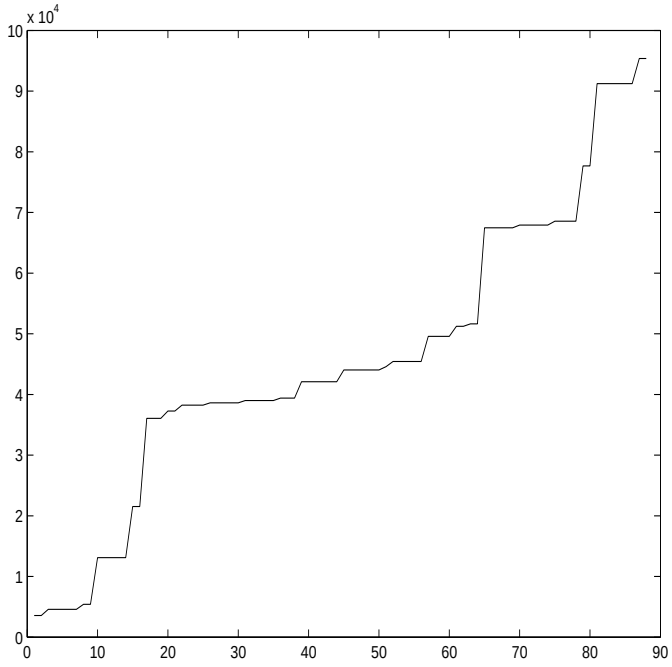

Figure 11: Sorted Time To Crash of all experiments. Time Range from close to an hour to 95000 seconds

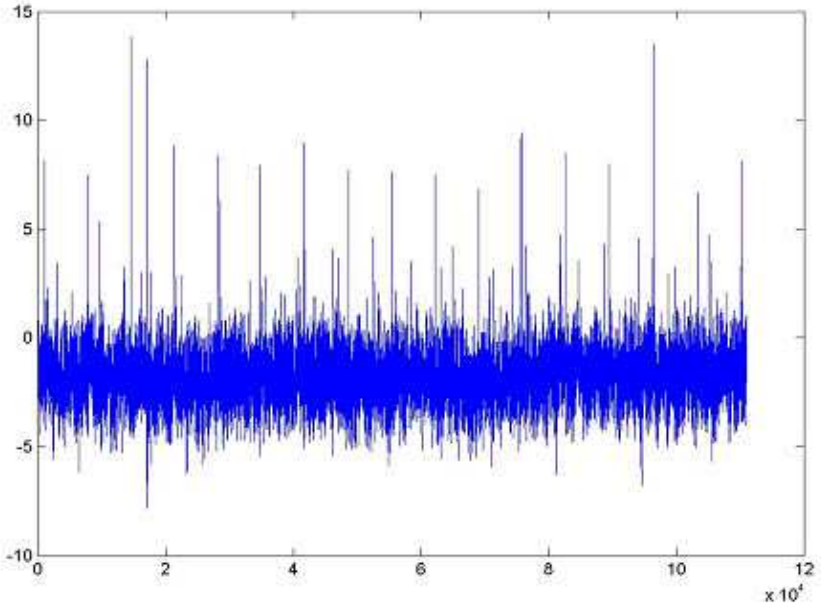

Figure 12: Shewhart series for nominal data collected remotely for one day

shows the Shewharts series and figure 8 shows the Crash Point Predictor lines. Figure 13 shows that in the case of data collected remotely on server *Vijai*when no other processes were running on the tested machine (thus minimizing the load on the system to a very low level) the technique still displayed two spikes.

Figure 14 shows the multidimensional holder exponent plot for the same data set.It was noticed that the values are mostly above 1. This deviation is due to the property of the data set and explains the occurrence of false positives in Figure 12.

Figure 15 displays the nominal data collected on Europa for five days. This data was collected locally on the tested machine Europa. Figure 16 show the multidimensional holder exponent value, the Shewhart's Series and the Crash Point Plots. Notice the abnormally high values of MDHolder (above 1), which indicates the reason for the false positive. This again shows a false positive occurring around one tenth of the whole data set size. This data set showed some initial changes despite the fact that there was no load on the system. These changes can be observed in Figure 15 during the first 10% period of the data set. These changes could have been due to various factors like the system running some initial

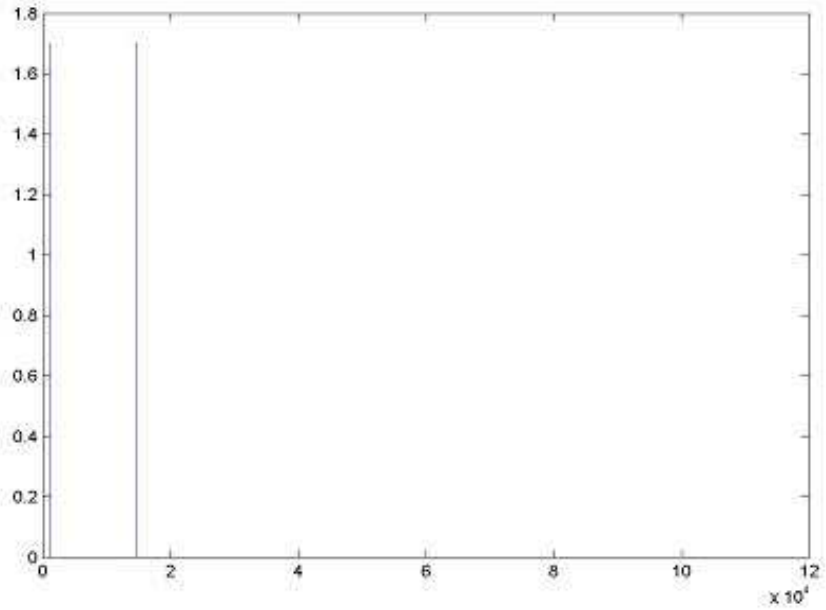

Figure 13: Crashpoint indicators on nominal data collected remotely for a period of one day

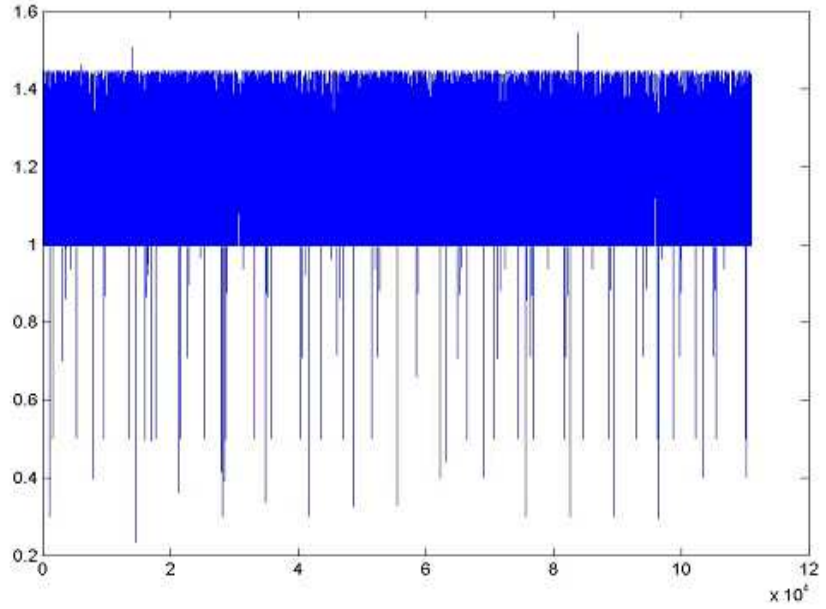

Figure 14: Multidimensional Holder exponent plot for the three parameters on nominal data collected remotely for a period of one day.

processes on startup.

To further eliminate changes like this that might trigger a crash prediction, only the last 4 days of the nominal data set was taken (which did not exhibit any of the changes seen in figure 15 from the start till the 50,000 interval mark). This is shown in Figure 17 and the corresponding Shewhart Series and Crash Point Predictor plots are shown in Figure 18. However even in this case a false positive is observed.

5.2.3 Removing False Positives and False Negatives

Due to the false positives observed in the nominal data analysis further analysis was mandated to see if the occurrence of the false positives depended on the parameters selected for the Shewhart Series computation. A detailed analysis was performed to try to tune the parameters to eliminate false positives and avoid false negatives. This is shown in Table 25. It was observed that as the Shewhart parameter values were tuned by raising the Delay or the Alarm it did not have a major effect on the occurrence of spikes. Increasing the threshold value however did. While processing nominal data it was observed that increasing the Threshold from 5 to 10 removed the false positives. This however was too much of an increase in the case of crash data sets. Therefore the increase was done in steps as seen in rows 24 to 27 and 28 to 31 and the holder analysis was done on both crashed and nominal data sets. This was done to see if the false positives that occured in the non crash data sets could be eliminated as the parameter value increased and if false negatives were avoided in crashed data sets as the threshold parameter progressed from 5 to 10.It was however observed that the parameter was either too high or too low. When false positives were removed in non crash data set analysis by increasing the parameter false negatives occurred in crashed data set analysis. When the parameter was decreased to remove the false negatives from crash data sets, false positives occurred in the case of non crash data sets. The aim was to find one or more sets of three parameters which would predict a crash in the case of crash data sets and not predict a crash in the case of nominal data sets.

Four Crash/Hang data sets and four Non Crash Data Sets were to do further analysis and nine sets of Delay, Threshold and Alarm values of most interest were selected from

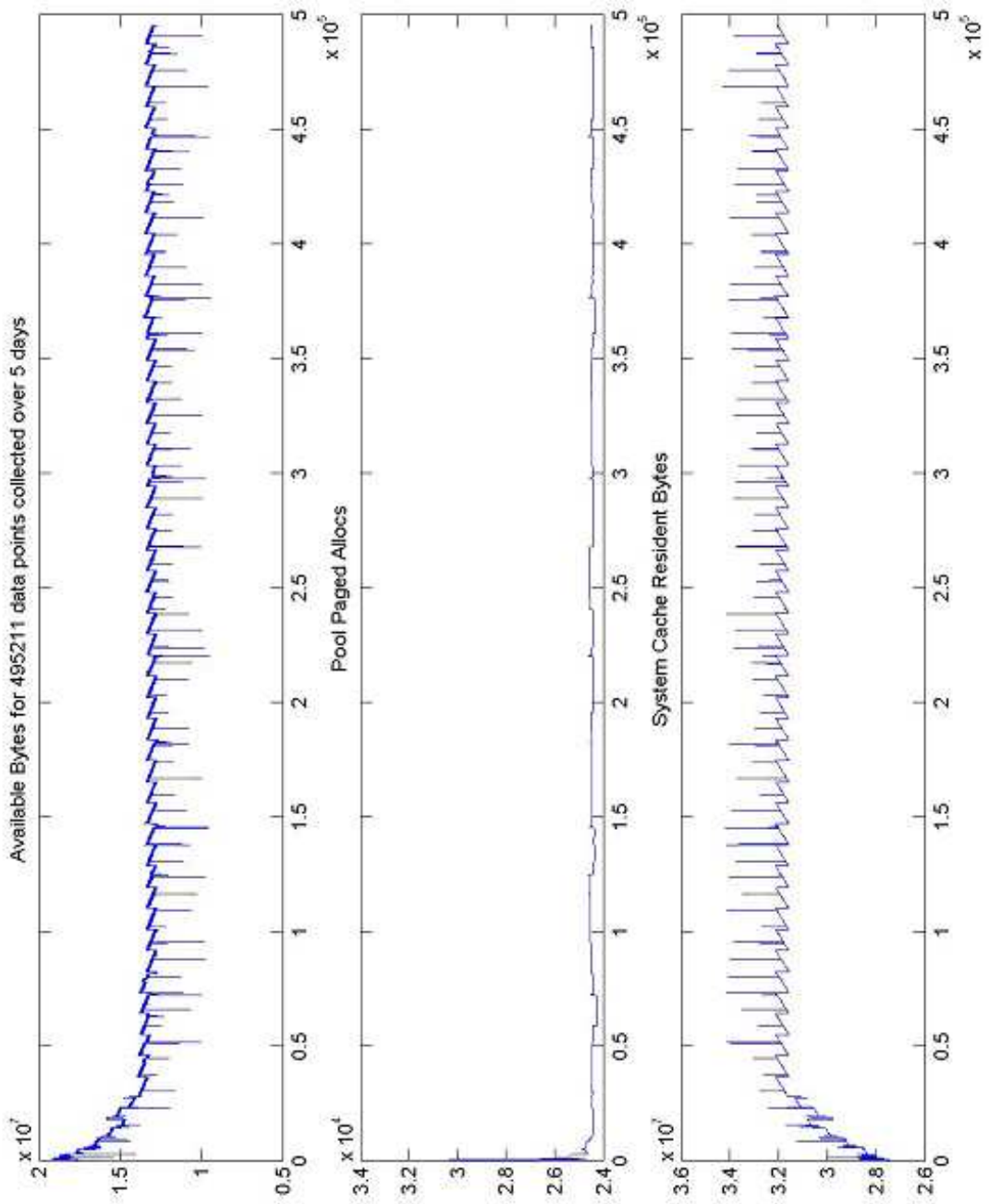

Figure 15: Nominal data (No crash or hangup) collected locally on Europa for 5 days

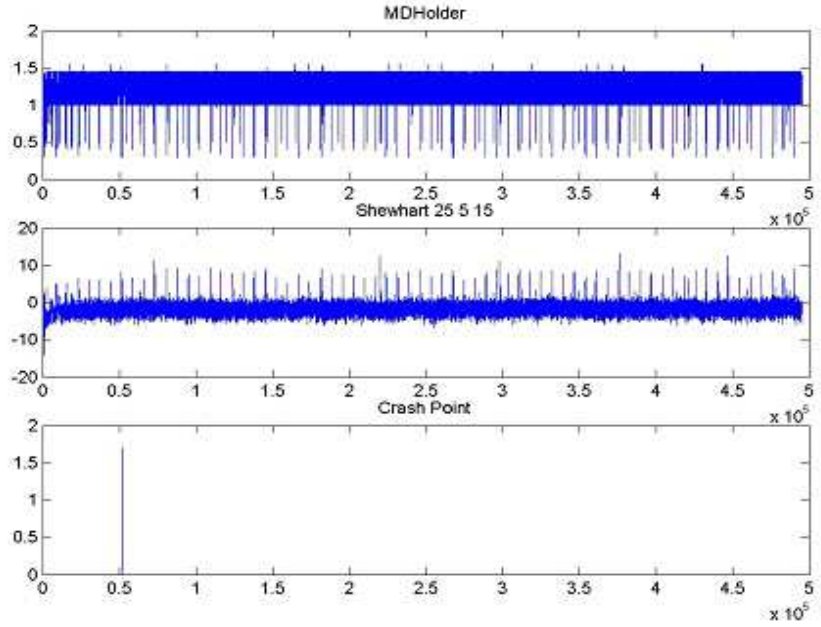

Figure 16: Multidimensional Holder exponent values, Shewharts series with 25, 5, 25 values and crash point indicator.

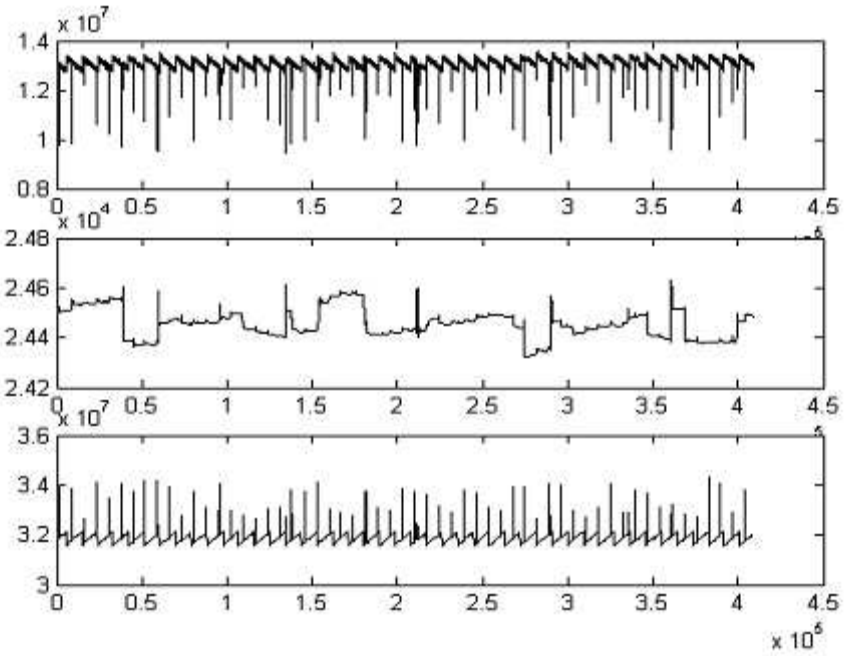

Figure 17: Last 4 days worth of nominal data (from that collected in Figure 12) collected locally on Europa

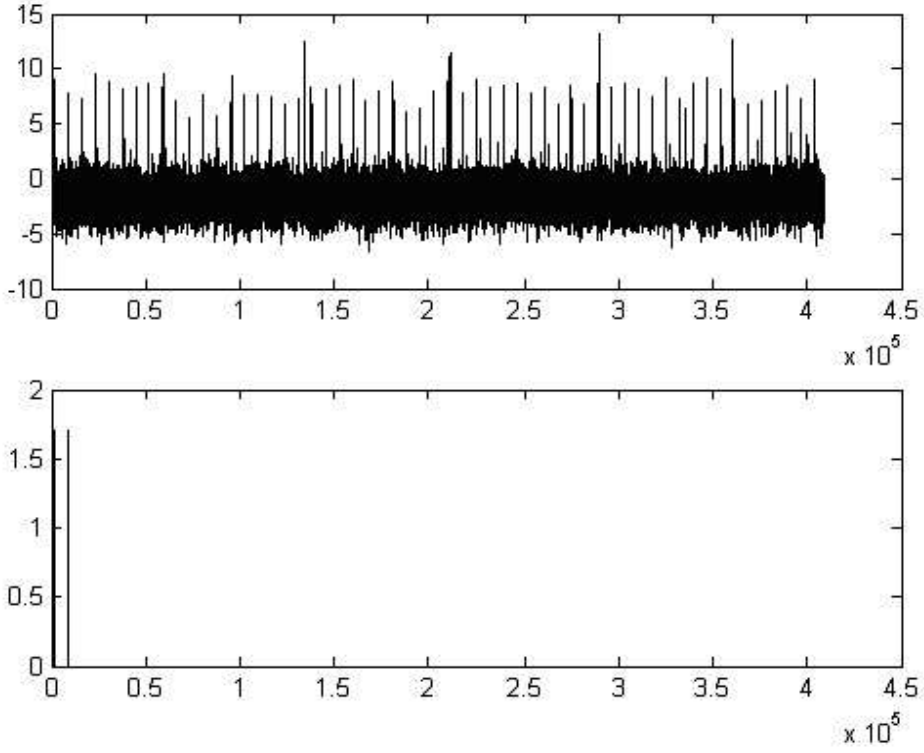

Figure 18: Shewhart Series and Crash Point Indicators for nominal data in Figure 15 collected locally on Europa

Table 9: Various parameters tried for the removal of false positives in the nominal data. Some of these (the most appropriate candidates) were tried on crashed data to see how well they responded. Set Delay Threshold Alarm No. of spikes Position

#	Delay	Threshold	Alarm	Spikes	Location	FP	FN	TP	TN
1	25	15	5	2		1			
2	25	25	5	1					1
3	25	5	15	2	$< 10\%$	1			
4	25	10	15	2	$< 20\%$	1			
5	25	20	15	1	Between 10 and 20%				1
6	25	5	25	2	<10%	1			
7	25	10	25	2	Between 10 and 20 %	1			
8	25	20	25	0					1
9	25	5	40	2	Between 10 and 20 %	1			
10	25	10	40	0			8		1
11	25	20	40	0					1
12	50	15	5	2		1			
13	50	5	15	2	$< 10\%$	1			
14	50	10	15	2	Between 10 and 20%	1			
15	50	20	15	2	Between 10 and 20%	1			
16	50	5	25	2	<10%	1			
17	50	10	25	2	Between 10 and 20%	1			
18	50	20	25	1	Between 10 and 20%				1
19	50	5	40	2	$< 10\%$	1			
20	50	10	40	2	Between 10 and 20%	1			
21	50	20	40	0					1
22	100	20	40	1	Between 80% and 90%				1
23	100	40	50	0					1
24	20	6	40	1	Between 10 and 20%		1		1
25	20	7	40	1	Between 10 and 20%		1		1
26	20	8	40	0			1		1
27	20	9	40	0		1		1	
28	25	6	40	2	Between 10 and 20%	1			
29	25	7	40	2	Between 10 and 20%	1			
30	25	8	40	2	Between 10 and 20%	1			
31	25	9	40	1	Between 10 and 20%		1		1

Table 10: False Positives(FP), False Negatives(FN), True Positives(TP) and True Negatives(TN)

Set	Delay	Threshold	Alarm	FP	FN	TP	TN	Interpretation
1	25	5	15	4	1	3	0	In almost all data sets the technique predicted a crash.
2	25	5	25	4	1	3	0	In almost all data sets the technique predicted a crash.
3	25	5	40	3	2	2	1	Predicted opposite results
4	25	10	40	0	4	0	4	In all datasets these parameters predicted no crash
5	50	5	15	4	0	4	0	In all data sets parameters predicted crashes
6	50	5	25	4	0	4	0	In all data sets parameters predicted crashes
7	50	5	40	4	0	4	0	In all data sets parameters predicted crashes
8	50	10	0	3	3	1	1	Predicted opposite results
9	100	40	50	0	4	0	4	In all datasets these parameters predicted no crash

Table 11: Table for ROC curve with input and output comparison for deciding true positives, true negatives, false positives and false negatives

	Crashed	Did not crash
Predicted Crash	TP	FP
Predicted No Crash	FN	TN

Table 25. Each of the eight data sets was subjected to holder analysis using Fractal 1.0 software with each of the nine parameter sets. The Crash Point Plot was observed for crash point predictor spikes.One Spike or two spikes were considered as *crash predictions* and therefore positive. No spike was considered as *no crash prediction* and therefore negative. Table 10 shows the total number of false positives (FP), false negatives (FN), true positives (TP) and true negatives (TN) for each of the nine parameter sets.

Table 10 shows which combinations of predictions and observations are categorized as TP, FP, TN and FN. The actual observed values after the analysis are shown in table 11. Table 13 displays the values in Table 25 as a percentage of the total number experiments for each set of parameters. The values of this table were used to plot the ROC curve with True Positives vs False Positives in figure 19 and the ROC curve with True Negatives vs False Negatives in figure 20.

Table 12: Table for ROC curve with observed false positives false negatives etc. input and output comparison for deciding true positives, true negatives, false positives and false negatives

	Crashed	Did not crash
Predicted Crash	21	26
Predicted No Crash	15	10

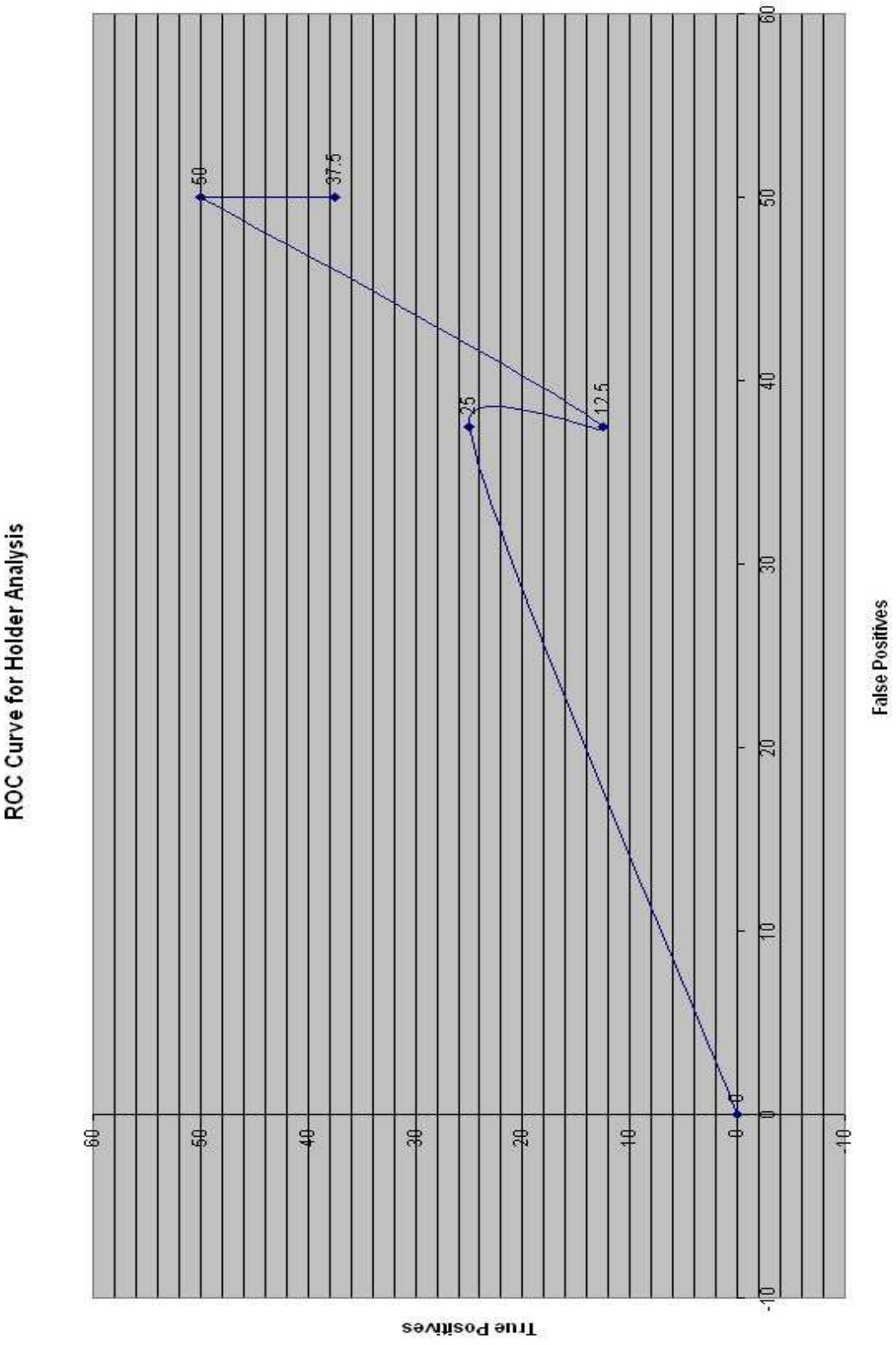

Figure 19: ROC Curve with true positives vs false positives

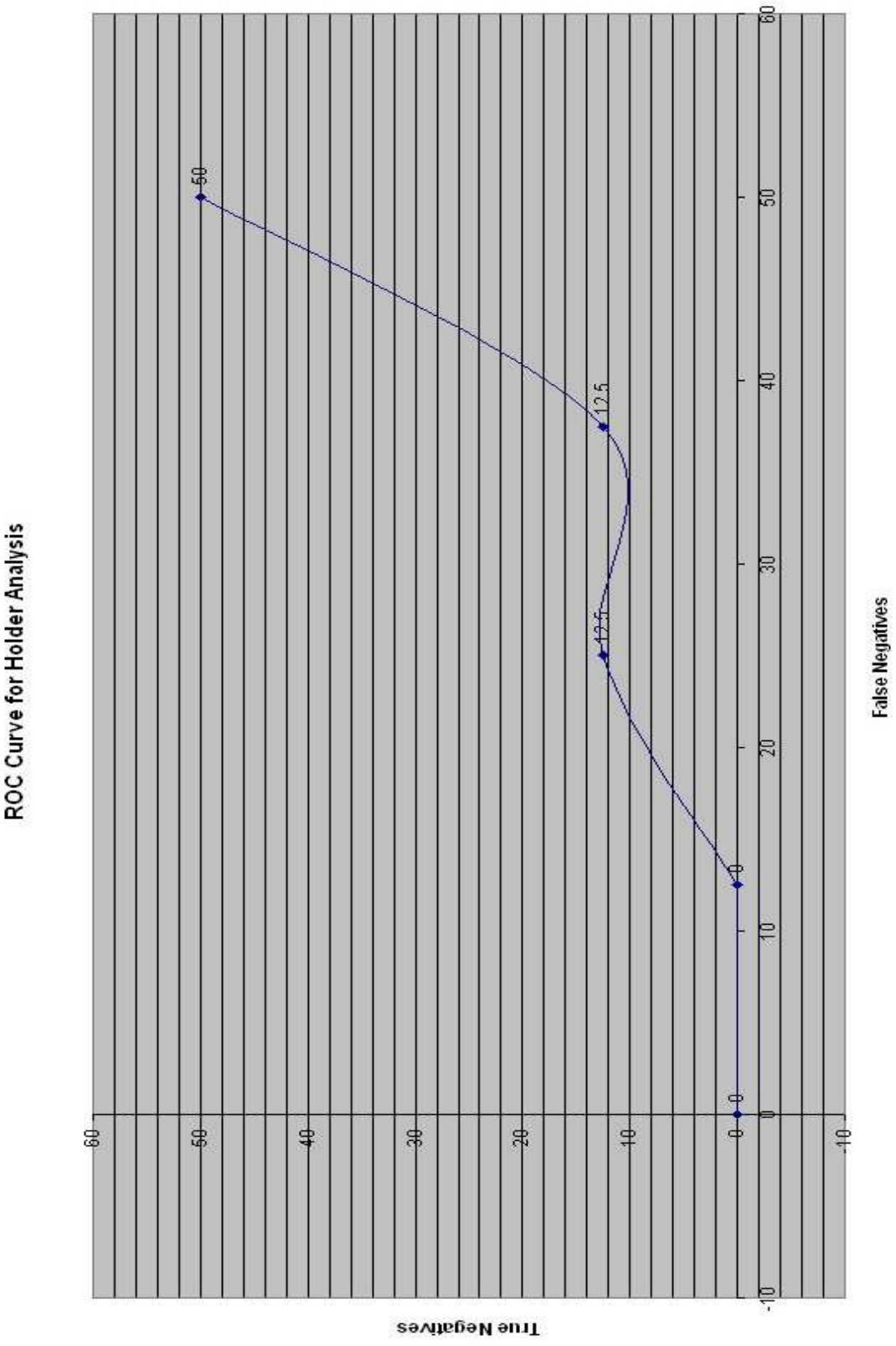

Figure 20: ROC Curve with true negatives vs false negatives

Table 13: Table for ROC curve with observed % false positives, false negatives, true positives and true negatives.

Set	Delay	Threshold	Alarm	FP	FN	TP	TN
10.	25	5	15	50	12.5	37.5	
11.	25	5	25	50	12.5	37.5	
12.	25	5	40	37.5	25	25	12.5
13.	25	10	40	0	50	0	50
14.	50	5	15	50	0	50	0
15.	50	5	25	50	0	50	0
16.	50	5	40	50	0	50	0
17.	50	10	40	37.5	37.5	12.5	12.5
18.	100	40	0	50	0	50	50

5.3 Hurst Analysis

5.3.1 Introduction

This section describes the experiments and analysis of data to analyze operating system behavior for modelling and prediction purposes. The analysis presented here uses the studies fractal properties of the data. The goal of this work is to measure the self similarity and long range dependence or memory behavior of the operating system data. Self similarity and the long range dependence are measured by estimating the Hurst Exponent. The data for the analysis was collected through experiments that involved local and remote data collection. Analysis was performed using SELFIS, a Self Similarity Analysis Tool [47]. SELFIS helps in the estimation of the Hurst Exponent, which can be used as an indicator of self similarity.

5.3.2 Experiments

The following types of data were analyzed.

1. Nominal case data collection on machine (Europa) locally
2. Nominal case data collection on a remote machine (Vijai) to collect data from two different machines (Europa and Ganymede)
3. Artificial data set creation with the last record's values from data collected in case 1 above
4. Crash Data on the machine Vijai

The parameters monitored were

1. Available Bytes
2. Pool Paged Allocs
3. System Cache Resident Bytes

5.3.3 Crash Data Analysis

The question of whether current total value of data versus a rate (per time) value was considered for suitability to the Hurst Analysis. After literature survey and consultation with other experts it was concluded that the only requirement placed on the data was that it should be stationary.

Table 14, 15 and 16 display the results from Hurst Analysis conducted on 29 Crash/Hangup data sets. Table 14 displays Hurst Exponent Values for Available Bytes parameter, Table 15 for Pool Paged Allocs Parameter and Table 16 for System Cache Resident Bytes Parameter. Along with this were plotted the Hurst Exponent Values estimated using Variance Method and the Rescaled Range (RS) method. Plots were also made of the Sizes of data sets ordered by increasing Hurst Exponent to see if there was any relationship between the data set size and the Hurst Exponent.

It is observed from tables 14, 15 and 16 that in all the data sets while the Variance method displayed high Hurst Exponent Values, the R/S method displays very low values. Further the correlation coefficients are very low. Correlations coefficients are expected to be greater than 97% to be assured of the existence of self similarity. This effect could be due to the non-stationarity of the raw data. This therefore necessitates further processing of data to make it stationary.

For this purpose, the data set was also processed by calculating the absolute value of the difference between the value recorded at one time interval to the next. This processed data was then subject to Hurst Analysis to study the Long Range Dependence behavior.

Each data set was processed by taking the absolute differences from one time interval to the next. Each of these data sets were then subjected to Hurst Analysis. The results

Table 14: Hurst Analysis on Available Bytes Parameter

No.	Size	Variance	Correlation(%)	RS Method	RS Correlation
1	67926	0.958	88.34	0.622	92.05
2	39395	0.910	96.53	0.635	94.65
3	34691	0.957	95.42	0.643	94.28
4	37824	0.955	96.09	0.65	94.25
5	42087	0.851	91.71	0.633	95.36
6	91221	0.930	85.9	0.553	90.86
7	44028	0.953	86.33	0.477	85.28
8	77673	0.943	87.15	0.682	93.61
9	36072	0.968	96.31	0.648	93.75
10	67461	0.961	90.5	0.556	89.68
11	44579	0.886	87.9	0.659	94.33
12	112169	0.933	76.29	0.661	92.35
13	95374	0.906	84.28	0.646	93.2
14	21529	0.925	91.19	0.63	94.23
15	45437	0.914	92.64	0.629	94.41
16	38228	0.920	98.44	0.669	96.19
17	38986	0.949	94.72	0.67	95.48
18	38618	0.924	95.09	0.656	94.93
19	4945	0.684	79.34	0.558	90.76
20	3556	0.780	98.04	0.57	94.97
21	51642	0.983	74.32	0.503	85.19
22	4585	0.933	75.63	0.403	71.21
23	68559	0.857	75.1	0.645	94.03
24	5406	0.958	78.93	0.322	63.26
25	51241	0.892	75.11	0.581	89.79
26	18602	0.898	90.7	0.566	90.53
27	13094	0.854	81.88	0.498	82.46
28	49564	0.968	77.96	0.644	93.71
29	37274	0.963	94.19	0.488	85.2
Average	44199	0.914	87.44931034	0.58955172	90.34482759

Table 15: Hurst Analysis on 29 data sets on Pool Paged Alloc parameter

No.	Size	Variance Method	Correlation(%)	RS Method	Correlation
1	67926	1.003	63.68	0.195	34.14
2	39395	1.002	54.22	0.198	46.54
3	34691	0.987	44.37	0.034	7.261
4	37824	1.007	74.88	0.171	40.84
5	42087	1.005	75.25	0.209	45.6
6	91221	0.989	71.89	0.184	39.47
7	44028	0.933	70.77	0.02	4.348
8	77673	0.986	42.13	0.007	1.299
9	36072	1.012	71.56	0.202	40.46
10	67461	0.949	72.97	0.152	39.47
11	44579	1.013	70.17	0.159	34.38
12	112169	1.008	69.37	0.26	48.53
13	95374	1.000	11.92	0.209	45.56
14	21529	1.011	71.06	0.136	35.41
15	45437	1.008	70.4	0.222	48.7
16	38228	1.013	66	0.16	36.58
17	38986	1.010	70.39	0.224	52.71
18	38618	1.011	69.82	0.153	36.75
19	4945	0.946	95.3	0.192	44.7
20	3556	0.977	57.23	0.089	27.52
21	51642	0.951	75.98	0.137	31.58
22	4585	0.983	77.98	0.259	53.91
23	68559	1.011	66.3	0.25	50.55
24	5406	0.832	55.7	0.218	45.56
25	51241	0.981	84.41	0.244	49.7
26	18602	1.009	69.8	0.106	25.41
27	13094	1.013	61.03	0.154	36.14
28	49564	1.011	72.02	0.176	37.75
29	37274	0.999	0.781	0.157	29.88
Average	44198.82759	0.988	64.04762069	0.168172414	36.92234483

Table 16: Hurst Analysis on 29 data sets on the System Cache Resident Bytes Parameter

No.	Size	Variance Method	Correlation(%)	RS Method	Correlation
1	67926	0.986	60.68	0.272	68.96
2	39395	0.903	95.16	0.383	77.56
3	34691	0.968	77.69	0.332	66.42
4	37824	0.856	62.36	0.288	62.34
5	42087	0.795	59.82	0.436	76.4
6	91221	0.926	83.45	0.257	59.98
7	44028	0.952	76.95	0.247	51.67
8	77673	0.952	80.24	0.114	32.12
9	36072	0.973	86.31	0.337	67.5
10	67461	0.95	83.59	0.296	60.67
11	44579	0.896	50.38	0.227	51.09
12	112169	0.992	50.71	0.387	69.62
13	95374	0.978	93.54	0.323	66.16
14	21529	0.88	88.73	0.382	75.89
15	45437	0.805	69.83	0.382	74.95
16	38228	0.917	94.45	0.424	77.68
17	38986	0.791	76.31	0.299	69.23
18	38618	0.768	74.27	0.24	58.66
19	4945	0.985	78.91	0.357	68.82
20	3556	0.704	81.29	0.316	72.24
21	51642	0.992	54.23	0.313	59.37
22	4585	0.951	70.9	0.383	76.6
23	68559	0.946	85.84	0.332	62.93
24	5406	0.991	66.33	0.196	41.98
25	51241	1.001	14.35	0.311	62.87
26	18602	0.97	55.36	0.263	57.59
27	13094	0.986	74.55	0.068	16.98
28	49564	0.822	65.36	0.232	56.56
29	37274	0.947	63.1	0.093	27.02
Average	44198.82759	0.917	71.54103448	0.292758621	61.02965517

Table 17: Hurst Analysis conducted on the absolute difference values of recorded parameters of 29 data sets

No	H_AB	H_PPA	HSCRB
26	0.616	0.651	0.683
20	0.637	0.688	0.698
19	0.64	0.556	0.673
5	0.641	0.736	0.711
21	0.654	0.638	0.702
24	0.692	0.567	0.65
12	0.703	0.637	0.738
22	0.707	0.664	0.668
23	0.718	0.73	0.642
17	0.733	0.628	0.655
15	0.735	0.758	0.727
4	0.745	0.592	0.646
14	0.756	0.544	0.588
13	0.759	0.649	0.665
2	0.76	0.603	0.6
18	0.768	0.578	0.643
25	0.771	0.648	0.676
16	0.773	0.705	0.678
11	0.778	0.642	0.628
28	0.783	0.677	0.718
9	0.784	0.683	0.673
10	0.799	0.708	0.696
27	0.806	0.637	0.696
6	0.806	0.699	0.698
1	0.81	0.673	0.729
8	0.812	0.683	0.719
3	0.818	0.623	0.699
29	0.839	0.667	0.75
7	0.844	0.763	0.698

are shown in table HurstAbsDiff29Datasets. The correlation values were also observed to be greater than 97%.

The results in Table 17 show that all the data sets collected in the case of crashes or hang-ups exhibit self similarity and long range dependence. That is the data set retains memory of events occurring in the past. This makes the data set very suitable for modelling and predictions. Analysis is therefore mandated on nominal data sets. The Hurst Exponent Analysis conducted on nominal data sets is presented in the next section.

The available bytes parameter values in 16 of the above 29 data sets were then broken

down into separate data sets of one hour each and subjected to Hurst Analysis. Hurst Exponent and the Correlation Coefficients for each hour in each data set were recorded for the Available Bytes Parameter. The hourly behavior of the hurst exponent was plotted and a line drawn through the points using linear approximation. The slopes of all these lines were then recorded and plotted as an XY Scatter Plot. This was also done for the nominal data sets. The results are shown in figures 21 and 22.

Figure 16 shows that a large majority of the crash data sets exhibit slopes of between -0.005 and -0.02. In the case of nominal data sets the slope differ from the crash data sets by at least an an order of magnitude and lie between 0.00001 and 0.0018. This clearly shows that the nature of the change in the values of the parameters changes in a certain way as we approach a system overload or crash. The fact that the slope is negative indicates that the self similarity of the absolute differences reduces from normal circumstances to hang-up or crash circumstances. It is also observed that while the processed nominal data sets show a negative slope too, the self similarity levels are much lower than in the case of crash or hang-up data sets.In addition the hurst exponent levels are almost steady and the slopes are very small indicating that the data of nominal data sets indicates very low levels of long range dependence.

A third set of analysis was started on the Multidimensional Hölder Exponent computed on all data sets.

5.3.4 Nominal Data Analysis

In the case of crash data as the bucket size is increased (100, 250, 500, 1000, 2000) for internal shuffling the hurst parameter as well as the correlation coefficient increase. As the bucket size is increased (100, 250, 500, 1000, 2000) for external shuffling the hurst parameter is below 0.5 and increases approaching 0.5(0.467-0.47). The Correlation coefficient also marginally increases. It is very close to but just below 90%. (89.86). Employing multilevel shuffling increases the H value for RS method (0.705) with 95.70 % correlation coefficient.

- Artificial Nominal Data

  For a flat line (artificial data)the R/S method gives a value close to 0.5 (0.519) and

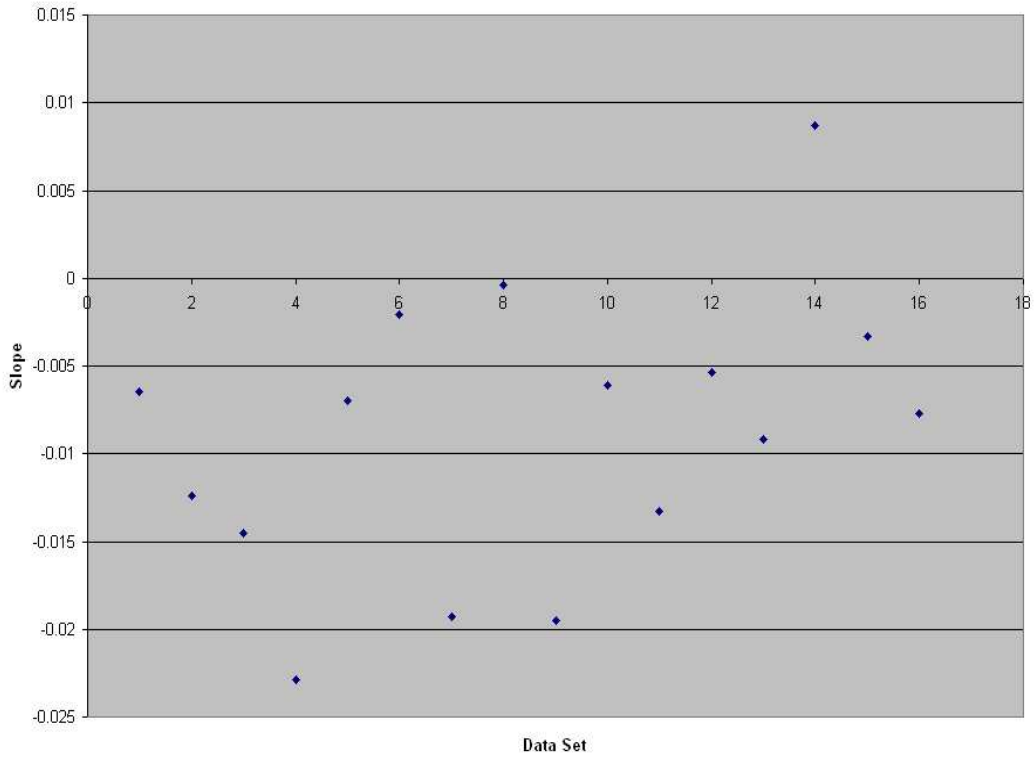

Figure 21: Slopes of linear approximation of Hurst Exponent computed for each hour of a crash data set

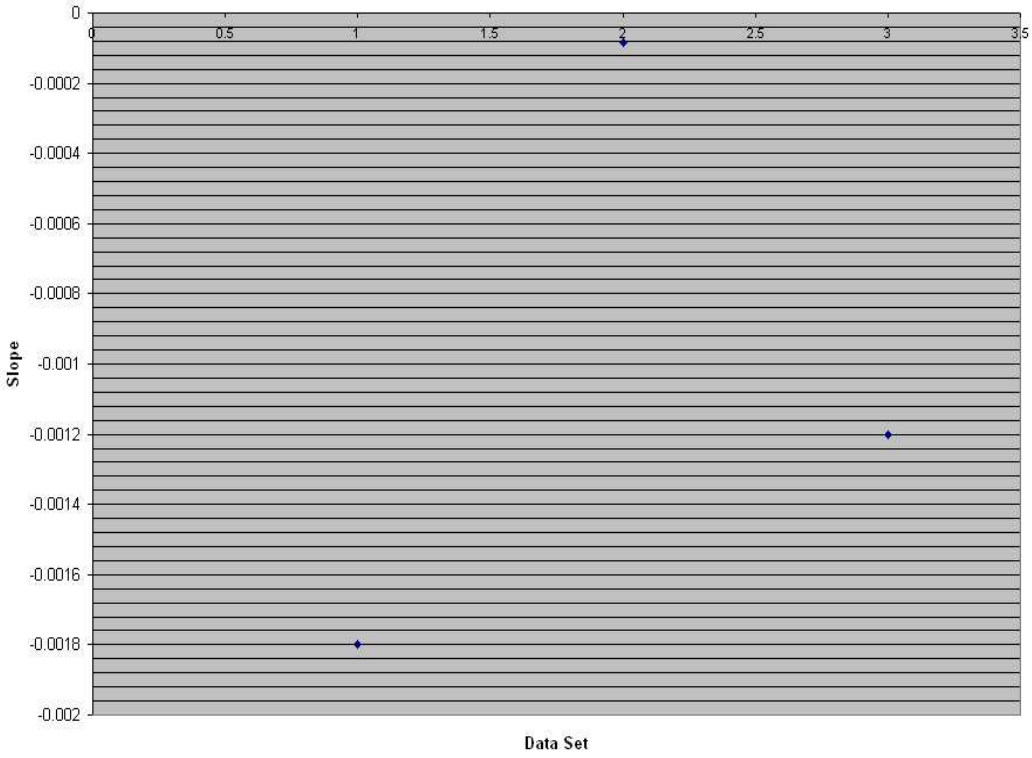

Figure 22: Slopes of linear approximation of Hurst Exponent computed for each hour of nominal data sets set

Table 18: Hurst Estimation with internal bucket shuffling with Artificial Data

Bucket Size	Hurst Exponent H	Correlation
100	0.522	99.91
250	0.522	99.91
500	0.522	99.91
1000	0.522	99.91
2000	0.522	99.91

Table 19: Hurst Estimation with external bucket shuffling with Artificial Data

Bucket Size	Hurst Exponent H	Correlation
100	0.522	99.91
250	0.522	99.91
500	0.522	99.91
1000	0.522	99.91
2000	0.522	99.91

the correlation coefficient is 99.92%. However the tool was not able to compute an estimate with variance method.

Table 18 and table 25 show the hurst exponent estimation values with internal and external bucket shuffling respectively. Even with multi level shuffling of level 1 size = 100 and level 2 size = 50 the values are the same.

- Nominal Data Locally Recorded

  In the case of nominal data that was locally recorded on one of the test machines, The hurst analysis results are shown in Table 20. By using a linear approximation this displays a fairly steady slope for the hurst exponent through out the data set. Table 21 and table 22 display the results of the Hurst Analysis using internal and external bucket shuffling. The low correlation coefficient numbers point to the data exhibiting very poor self similarity.

- Nominal Data Remotely Recorded

  In Table 23, AB = Available Bytes PPA=Pool Paged Allocs SCRB = System Cache Resident Bytes AB Section = A section of the data set was taken which only consisted of (seemingly) randomly changing data. This pattern was found to repeatedly occur

Table 20: Hurst exponent value with Variance method on data for each day in the data set of nominal data collected on Europa.

Day	Hurst Exponent Variance Method
1.	0.805
2.	0.775
3.	0.804
4.	0.849
5.	0.799

Table 21: Hurst Estimation with for locally recorded nominal data with internal bucket shuffling

Bucket Size	Hurst Exponent H	Correlation Correlation
100	0.840	74.80
250	0.841	74.94
500	0.843	74.81
1000	0.847	75.57
2000	0.832	78.31

Table 22: Hurst Estimation with external bucket shuffling for locally recorded nominal data

Bucket Size	Hurst Exponent H	Correlation Correlation
100	0.752	93.9
250	0.743	86.43
500	0.788	90.27
1000	0.862	85.74
2000	0.866	81.62

Table 23: Hurst exponent estimation values and correlation values using Variance and R/S analysis methods with the data sets of the three parameters over nominal data collected remotely

Data	Timeline	Variance Method Hurst Estimation	Correlation % Coefficient	R/S Method Hurst Estimation	Correlation % Coefficient
AB	1 day	0.995	20.36	0.038	19.01
PPA	1 day	1.010	66.66	0.110	27.57
SCRB	1 day	0.998	7.717	0.316	96.51
AB Section 4375 Records (180-4554)		1.014	63.83	0.182	91.13

over a saw tooth like waveform of the data. Here again the low correlation coefficient indicates a data set exhibiting very poor self similarity. Table 24 and 25 show the results of hurst analysis done on remotely recorded nominal data while employing internal and external bucket shuffling which again exhibits poor self similarity.

1. (a) Parameter 1 seems to exhibit high Hurst Exponent values using Variance Method (but not exceeding 1)
   (b) Parameter 2 exhibits values about or slightly greater than 1 using Variance method
   (c) Parameter 3 exhibits Hurst Exponent values close to 1 (but not exceeding)
   (d) A section of the data set was taken which only consisted of (seemingly) randomly changing data. This pattern was found to repeatedly occur over a saw tooth like waveform of the data. This exhibits values greater than 1 from the Hurst Exponent.
2. In all the cases above the Hurst Exponent computed using the R/S Method gave results where the Hurst Exponent was $0 < H < 0.5$ where as in the case of Variance method it was mostly $0.5 < H < 1.02$. In other data sets (> 30) of similar source the behavior is similar.
3. Using the other methods like Whittle or Periodogram methods gives H values even greater than 1 (more than what was observed for Variance Method). As the bucket size (100, 250, 500, 1000, 2000) was increased for internal shuffling the hurst parameter as well as the correlation coefficient increase.

As the bucket size is increased(100, 250, 500, 1000, 2000) for external shuffling the hurst parameter is below 0.5 and increases approaching 0.5(0.467-0.47). The Correlation coefficient also marginally increases. It is very close to but just below 90%. (89.86)

When employing multilevel shuffling an increased H value is observed for RS method (0.705) with 95.70 % correlation coefficient).

Table 24: Hurst Estimation with internal bucket shuffling for remotely recorded nominal data

Bucket Size	Hurst Exponent H	Correlation Correlation
100	1.006	53.93
250	1.006	54.71
500	1.006	54.47
1000	1.006	52.34
2000	1.006	43.21

Table 25: Hurst Estimation with external bucket shuffling for remotely recorded nominal data

Bucket Size	Hurst Exponent H	Correlation Correlation
100	0.691	88.93
250	0.772	83.06
500	0.868	79.32
1000	0.910	81.30
2000	0.931	70.56

5.3.5 Combining all three data sets for Hurst Analysis

In the case of Holder Exponent a multi-dimensional holder exponent value was calculated thus combining the effects of all three parameters that were being monitored. This was used to generate the Shewharts Series. Similarly it would be beneficial to see if the combination of the three signals can lend us any idea of the self similarity of the chaotic nature of this combination.

For each data set the multi-dimentional Holder exponent value series was generated and the self similarity of the signal was calculated by generating the Hurst exponent value. This value was generated using the Variance Method.

The table 26 summarizes the results.

Figure 23 shows the XY Scatter plot the Hurst exponent values of the multi-dimensional holder data. This clearly shows a separation between the self similarity of crash data sets and that of the nominal (non-crash data sets).

The values close to the 0.629 line chosen to demarcate the crash and non crash data sets can in various situations be interpreted as near false positives or near false negatives. However simple internal shuffling shows the presence of high self similarity in crash data

Table 26: Hurst Exponents of Multi-Dimensional Holder Series of Crash and Nominal Data

No.	Data Set	Hurst Value	Crash or Nominal	Method
1.	02083019	0.896	Crash/Hangup	Variance
2.	02101120	0.790	Crash/Hangup	Variance + Int. Shuffling
3.	02102215	0.797	Crash/Hangup	Variance
4.	02102419	0.867	Crash/Hangup	Variance
5.	02102909	0.825	Crash/Hangup	Variance
6.	02110221	0.855	Crash/Hangup	Variance
7.	02110417	0.717	Crash/Hangup	Variance
8.	02111121	0.816	Crash/Hangup	Variance
9.	02111214	0.893	Crash/Hangup	Variance
10.	02111313	0.835	Crash/Hangup	Variance
11.	02111418	0.960	Crash/Hangup	Variance
12.	03020523	0.844	Crash/Hangup	Variance
13.	03020600	0.874	Crash/Hangup	Variance
14.	03020710	0.842	Crash/Hangup	Variance
15.	03021312	0.841	Crash/Hangup	Variance
16.	03021313	0.779	Crash/Hangup	Variance
17.	03021412	0.846	Crash/Hangup	Variance
18.	03021815	0.727	Crash/Hangup	Variance
19.	03040720	0.974	Crash/Hangup	Variance
20.	03041015	0.781	Crash/Hangup	Variance + Int.Shuffling
21.	03042219	0.892	Crash/Hangup	Variance
22.	03051215	0.834	Crash/Hangup	Variance
23.	03052310	0.956	Crash/Hangup	Variance
24.	03052616	0.771	Crash/Hangup	Variance
25.	03052714	0.799	Crash/Hangup	Variance
26.	03110521One	0.563	Nominal 1	Variance
27.	03110521Two	0.585	Nominal 2	Variance
28.	03110521Three	0.558	Nominal 3	Variance
29.	03110521Four	0.580	Nominal 4	Variance
30.	00112233One	na	Nominal 5	Variance
31.	03110521Day1	0.593	Nominal 6	Variance + Ext. Shuffling

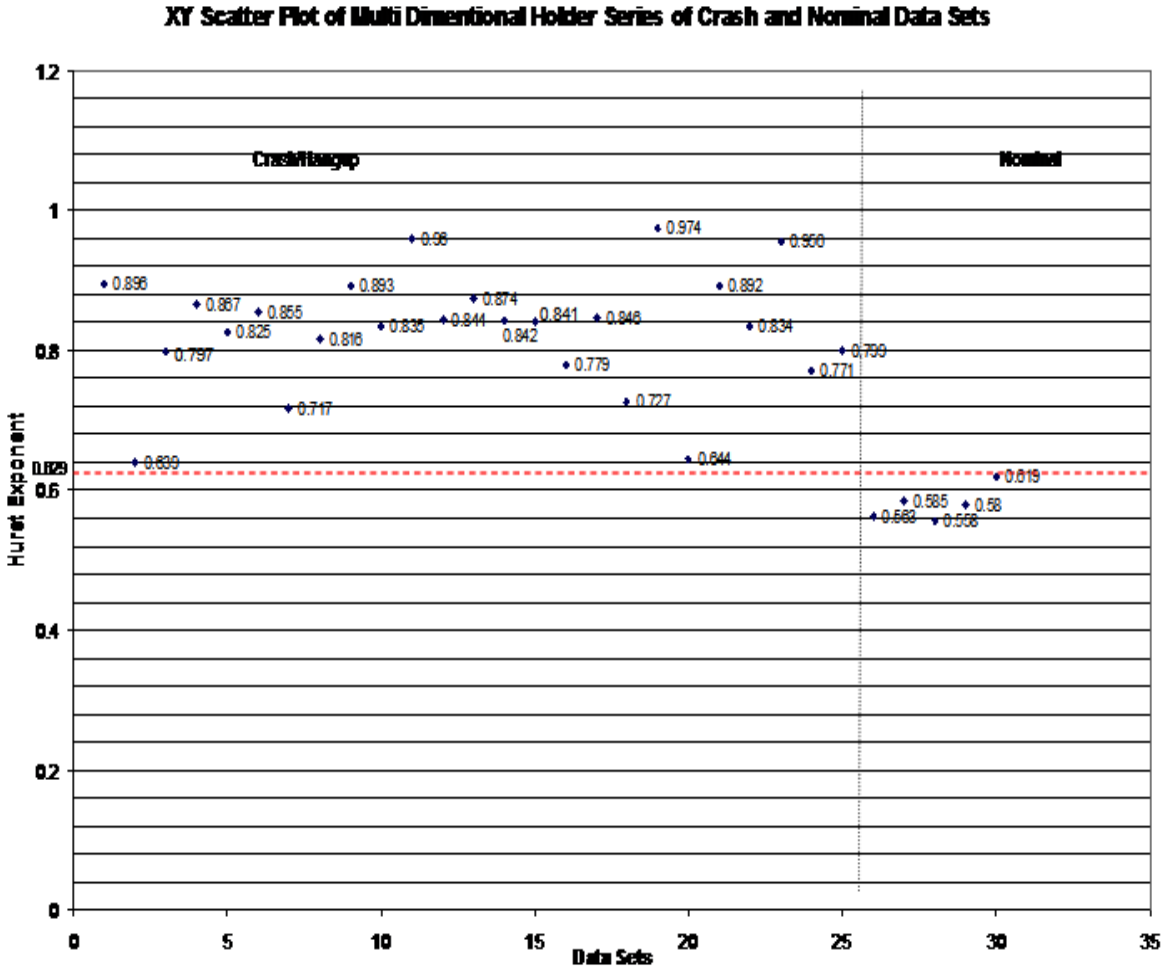

Figure 23: XY Scatter Plot of Multidimensional Holder exponent values for Crash and Nominal Data.

Table 27: Correcting Near False Postives and Near False Negatives

No.	Data Set	FP or FN	Processing
2101120	0.639	Near False Negative	
2101120	0.79	True Positive	Int Shuffle
3041015	0.644	Near False Negative	
3041015	0.781	True Positive	Int Shuffle
03110521Day1	0.845	False Positive	
03110521Day1	0.619	True Negative	Ext Shuffle

sets and simple external shuffling shows the lack of such self similarity in the non crash data sets. This is shown in table 27.

Further it is seen that the hurst exponent of nominal data sets tends towards 0.5 indicating little or no self similarity, where as the values of the crash data sets are very high and point to high self similarity.

CHAPTER VI

CONCLUSIONS

M5 Prime Method M5' shows promise in its application to crash data sets for giving advance warning using the model trees generated from it.

The high correlation coefficients obtained after cross-validation was performed by Weka indicate a strong linear relationship between the actual and predicted values. However the nature of the plots of the predicted values indicate that the technique has to be further developed and maybe used in conjunction with other techniques to develop a reliable predictor of operating system crashes.

This study also indicates that for data sets larger than 560 intervals (based on the ten data sets studied here) an advance warning can be given.

Holder Exponent and Shewhart Series Analysis Crash predictions using multidimensional holder exponent and shewhart series plots successfully detects changes and gives crash prediction warnings. At this point one or more sets of Shewharts parameters (Delay, Threshold and Alarm) have not been found that consistently remove false positives from non crash data sets and false negatives from crash data sets.

Hurst Exponent Analysis and Long Range Dependence It has been clearly shown here that the crash and hang up data sets exhibit self similarity making them candidates for the use of prediction techniques used in many well known problems like Nile river flows. In addition it has also been shown here that the nature of the self similarity changes as one approaches a crash or hang up. This is consistently seen and is not observed in the case of nominal data. The multi-dimension holder exponent of the three parameters monitored also clearly shows a self similarity in the crash data sets and no self similarity in the non-crash data sets.

6.1 Future Work

1. Further research has to be conducted into the user of M5 Prime Model Tree Learners as they show promise in being able to predict operating system crashes. Investigation into making changes to the model tree and developing a better predictor as well as investigation into the use of the model tree learner in conjunction with another technique is recommended.

2. Further investigation is required in the use of Holder Exponent Analysis and Shewhart Series to tuning the three parameters in order to develop a reliable predictor.

3. Further analysis by studying the delta of change of the hurst exponent from one hour to the next can be performed to give a much better idea of the behavior of the time series. Predictive methods mentioned in [38] and [4] can used to predict the behavior of the parameter in question. Further, the lessons learnt from the study of the self similar nature of Ethernet Traffic and the studies conducted on the impact of self similarity on network performance can be applied to operating system data for development of predictive models.

APPENDIX A

RUNNING THE CRASH TESTS

A.1 List of stress tests in the Microsoft Stress Kit

1. csd
2. default
3. directx
4. german
5. germancsd
6. hct
7. ie4
8. lastrun
9. netmeet
10. nt
11. nt4onnt5
12. ntam
13. ntds
14. ntjpn
15. ntnet
16. ntpnp
17. nttrmsrv

18. ntwin32k
19. ntwx86
20. ole
21. pkclient
22. pksrvtype
23. private
24. shellnt
25. tester

A.2 The default *set of stress scripts*

The Microsoft Stress Test Kit has the following stress groups in the *default* set.

1. required_group -1
2. internet_stuff 0
3. openglconflict 0 1
4. gdi 2
5. dsgrp 2
6. olebasic 1
7. mem 0
8. vdm 0 2
9. confoc 0 1
10. app 0 1
11. filesystem 1 2

12. net 0 1

13. direct3d 1 3

14. directdraw 1

15. directsound 1

16. directinput 0

17. lsasam 1 2

18. misc 1

19. shell 1 3

20. rpc_group -1

21. rpc_group2 -1

After each of the groups the numbers indicate the following-

-1 = Run all tests in this group

0 = Don't include in the initial 16 tests selected

1 5 = run atleast one test but no more than 5 from this group

Each group consists of a set of tests. For more information please contact Microsoft Inc.

APPENDIX B

RUNNING THE FRACTAL APPLICATION

The Fractal Application was developed using Oracle's JDeveloper. Include the files into a project with JDeveloper and run the application.

Open the data file to be processed and select from the menu to either run Multi-Holder Exponent computation or Shewarts series computation.

APPENDIX C

RUNNING THE JAVA HOLDER COMPUTATION PROGRAMS OF JOHN CROWELL

Please note that the source code may have to be modified if you wish to process a data file of size larger than 499999 records.

1. Create folders: These instructions are for a Windows 2000 or XP operating system.

   (a) Create a folder "C:\softwareaging".Place the data files here. The data files should have a naming format as Flog_xxxxxxxx.csv. Where the eight x characters represent a number which may be used to indicate when the data file was generated. For example to indicate that the data was collected on November 5, 2003 with the hour part of the time being 9 pm the data file name should be Flog_03110521.csv.

   (b) Under that folder create a folder called "java" to hold the .java files

   (c) Also create a folder called "classes" to hold the compiled java byte-code files

   (d) Create a folder "C:\Perflogs" and in that create folders "C", "H" and "K" for the three parameters we are dealing with here.

2. Compile : You must have the Java SDK installed and have the path variable set to the bin directory of the SDK so that you can execute javac and the java commands.

   (a) Change directory to "C:\softwareaging\java". This is where you have the files compute.java, holder.java, MA.java, MDholder.java and Shewhart.java

   (b) Run the command "javac -d ..\classes *.java"

   (c) Change directory to "C:\softwareaging\classes"

3. Clean data files: Data files can have different names and different formats of data. They have to be "cleaned" and put in the right format. If there are quotes around

the data values(the windows performance monitor places quotes) be sure to remove them using a unix command or with an awk script. The data format of the data file should be as follows.

(a) Column 1 is the "Date" column, Column 2 is the "Available Bytes" column Column 3 is the "Pool Paged Allocs" column and Column 4 is the "System Cache Resident Bytes" column

(b) The first row should have the headings. The program automatically excludes the heading row from processing.

4. Locate data files : Place the data files (with the file name and data formatted as described in 3. above)into the folder "C:\Perflogs".

5. Run "compute.class": Change directory to "C:\softwareaging\classes" and execute the command "java compute". The files have the necessary output statements to keep you aware of the processing thats going on. Please note that the set of files provided here are the modified versions of John's files. The modifications performed were to use only 3 parameters and have status messages printed.

6. Computing the Shewhart's Series : The Shewhart algorithm is used for detecting the moment of downward shift in a time-series. For every data point in a time series the mean and standard deveiation of the current fragment of the time series is computed using all the previous points. If a certain inequality holds true for a number of concurrent points it indicates the detection of a sustained drop. In this thesis a second drop was attempted to be discovered to give a clear alarm of a crash.

(a) usage:"$javaShewhart < datafilename >< delay >< threshold >< alarm >$ ". Set delay to 25 or 50, threshold to 5 and alarm to 15, 25, 40. Try different combinations of the tuning parameters delay, threshold and alarm.

(b) Copy the $.dat_MDholder$ file created in the classes directory into a separate folder, say "C:\Shewharts". Use that as the input to shewhart program along with the

tuning parameters as the other input arguments. Please note that the program takes a long time to run.

REFERENCES

[1] A. Pfening, S. Garg, A. P. M. T. and Trivedi, K. S., "Optimal software rejuvenation for tolerating software failures," *Performance Evaluation*, vol. 27 and 28, October 1996.

[2] Aadland, D., "Detrending time-aggregated data," October 2002.

[3] Barnsley, M., *Fractals Everywhere*. Academic Press, 1988.

[4] Beran, J., *Statistics for Long-Memory Processes. Monographs on Statistics and Applied Probability*. New York, NY: Chapman and Hall, 1994.

[5] Cameron L. Jones, Greg T. Lonergan, D. E. M., "Wavelet packet computation of the hurst exponent," *Journal of Physics A: Mathematical and General*, vol. 29, pp. 2509–2527.

[6] Crowell, J., *Multifractal Analysis of Memory Usage Patterns, M.S. Thesis*. Morgantown, West Virginia: West Virgnia University, 2001.

[7] Daubechies, I., *Ten lectures on wavelets*. Philadelphia, PA: SIAM, 1992.

[8] Eibe Frank, Yong wang, S. I. G. H. I. H. W., "Using model trees for classification," Tech. Rep. 1, Department of Computer Science, University of Waikato, Hamilton, New Zealand, 1998.

[9] Falconer, K., *Fractal geomety; Mathematical foundations and Applications*. John Wiley and Sons, 1990.

[10] Feder, J., *Fractals*. New York: Plenum Press, 1988.

[11] Gammel, B. M., "Hursts r/s analysis," *http://www1.physik.tu-muenchen.de/ gammel/matpack/html/Mathematics/RSAnalysis.html*.

[12] Gray, J., "Why do computers stop and what can be done about it?," tech. rep., HP Labs, June 1985.

[13] Ian H. Witten, E. F., *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. San Francisco: Morgan Kaufmann, 1999.

[14] Ian H. Witten, Eibe Frank, L. T. M. H. G. H. S. J. C., "Weka: Practical machine learning tools and techniques," *In: Proc ICONIP/ANZIIS/ANNES'99 Int. Workshop: Emerging Knowledge Engineering and Connectionist-Based Info. Systems. (1999) 192-196"*, 1999.

[15] Ian Kaplan, L. K., "Estimating the hurst exponent," *http://www.bearcave.com/misl/misl_tech/wavelets/hurst/*.

[16] J. S. Marron, C. P., *Internet Traffic Data Analysis Study Group*. Department of Statistics, University of North Carolina, http://www-dirt.cs.unc.edu/netdata/networkdata2004.htm, February 2003.

[17] Jeong, H., McNickle, D., and Pawlikowski, K., "A comparative study of generators of synthetic self-similar teletraffic,"

[18] Jiang, M., Nikolic, M., Hardly, S., and Trajkovic, L., "Impact of selfsimilarity on wireless data network performance," *"Proc. IEEE ICC'01", Helsinki, Finland*, pp. 477–481, June 2001.

[19] Jonathan Crowell, Mark Shereshevsky, B. C., "Using fractal analysis to model software aging," tech. rep., Lane Department of CSEE, West Virginia University, Morgantown, WV, May 2002.

[20] K. S. Trivedi, K. V. and Goseva-Popstojanova, K., "Modeling and analysis of software aging and rejuvenation," *IEEE Annual Simulation Symposium*, April 2000.

[21] Kaplan, I., "Estimating hurst exponent," *www.bearcave.com*.

[22] Kaplan, I., "Spectral analysis and filtering with the wavelet transform," *www.bearvave.com*, vol. 10, August 2002.

[23] Kaplan, L. and Kuo, C. J., "Fractal estimaton from noisy data via discrete fractional gaussian noise (dfgn) and the haar basis," *IEEE Trans. on Signal Proc.*, vol. 41, December 1993.

[24] Karagiannis, T., *SELFIS: A Short Tutorial*. University of California, Riverside, Department of Computer Science, November 2002.

[25] Koutsoyiannis, D., "Supplementary material for the paper climate change, the hurst phenomenon, and hydrological statistics,"

[26] Leland, W. E. and Wilson, D. V., "High time-resolution measurement and analysis of lan traffic: Implications for lan interconnection.," *In Proceeedings of IEEE Infocomm '91*, pp. 1360–1366, June 1991.

[27] M. Barnsley, R. Devaney, B. M. H. P. D. S. R. V. Y. F. M. M., *The Science of Fractal Images*. Springer-Verlag, 1988.

[28] Mallat, S., "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Transactions on Pattern Recognition and Machine Intelligence*, vol. 11, pp. 675–693, July 1989.

[29] Mandelbrot, B. B., "Long-run linearity, locally gaussian processes, h-spectra and infinite variances," *Intern. Econom. Rev.*, vol. 10, pp. 82–113, May 1969.

[30] Mandelbrot, B. B., *The Fractal Geometry of Nature*. New York: W. H. Freeman & Co., 1983.

[31] Mark Shereshevsky, Jonathan Crowell, B. C. V. G. Y. L., "Software aging and multifractality of memory resources," *International Conference on Dependable Systems and Networks*, p. 721, 2003.

[32] Marshall Brain, R. R., *Win32 System Services: The Heart of Windows 98 and Windows 2000*. Addison Wesley, December 2000.

[33] Martin J. Fischer, T. B. F., "Fractals, heavy-tails, and the internet," *SIGMA, Mitretek Systems, McLean, Virgnina*, 2001.

[34] Mengzhi Wang, tara Madhyasta, N. H. C. S. P. c. F., "Data mining meets performance evaluation¿ fast algorithms for modeling bursty traffic," *18th International Conference on Data Engineering, 2002*, 2002.

[35] Murad S. Taqqu, Vadim Teerovsky, W. W., "Estimators for long-range dependence: an empirical study," *Fractals*, vol. 3, no. 4, pp. 785–798, 1995.

[36] P. Devynck, G. Wang, G. A. G. B., "The hurst exponent and long-time correlation," *27th EPS Conference on Contr. Fusion and Plasma Physi. Budapest*, vol. 24B(2000), pp. 632–635, June 2000.

[37] Parnas, D. L., "Software aging," *In Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy*, pp. 279–287, May 1994.

[38] Peter J. Brockwell, R. A. D., *Time Series: Theory and Methods*. New York, NY: Springer-Verlag, 1987.

[39] Prendergast, B. T., *Windows Architecture I & II MCSD Study Guide*. Wiley Publishing, 1998.

[40] Quinlan, J. R., "Learning with continuous classes," *Proceedings of AI'92 (Adams and Sterling Eds), World Scientific*, pp. 343–348, 1992.

[41] S. Garg, A. van Moorsel, K. V. and Trivedi, K. S., "A methodology for detection and estimation of software aging," *Int'l. Symp. on Software Reliability Engineering,ISSRE 1998*, November 1998.

[42] S. Garg, A. Puliafito, M. T. and Trivedi, K. S., "On the analysis of software rejuvenation policies," *Annual Conference on Computer Assurance (COMPASS)*, June 1997.

[43] S. Garg, A. Puliafito, M. T. and Trivedi, K. S., "Analysis of preventive maintenance in transactions based software systems," *IEEE Transactions on Computers*, January 1998.

[44] Saupe, D., *Algorithms for random fractals, Chapter 2 of The Science of Fractal Images by Barnsley et al.* Springer-Verlag, 1988.

[45] Schreiber, S. B., *Exploring Windows 2000 Memory*. Addison Wesley, July 2001.

[46] Tabini, M., "The mtlib memory-tracking library, a memory-leak-tracking system for c developers," *Dr. Dobb's Journal*, October 2003.

[47] Thomas Karagiannis, M. F., "Selfis: A tool for self-similarity and long-range dependence analysis. extended abstract,"

[48] Thomas Karagiannis, M. F., "The selfis tool: Long-range dependence and self-similarity analysis," *1st Workshop on Fractals and Self-Similarity in Data Mining: Issues and Approaches (in KDD) Edmonton, Canada*, July 2003.

[49] Trivedi, K. S., "Software bugs," *http : //srejuv.ee.duke.edu/software_bugs.htm*, 1993.

[50] UCI, "Collection of different datasets http://www.ics.uci.edu/ mlearn/mlsummary.html,"

[51] V. Castelli, R. E. Harper, P. H. S. W. H. K. S. T. K. V. and Zeggert, W. P., "Proactive management of software aging," *IBM Journal of Research & Development*, vol. 45, March 2001.

[52] V. Teverovsky, Murad S. Taqqu, W. W., "A critical look at lo's modified r/s statistic," *Journal of Statistical Planning and Inference*, vol. 80, pp. 211–227, May 1998.

[53] Vaidyanathan, K. and Trivedi, K. S., "A measurement-based model for estimation of software aging in operational software systems," *Int'l. Symp. on Software Reliability Engineering ISSRE 1999*, November 1999.

[54] Vaidyanathan, K. and Trivedi, K. S., "A comprehensive model for software rejuvenation," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, April-June 2005.

[55] Vern Paxson, S. F., "Wide-area traffic: The failure of poisson modeling," *IEEE/ACM Transactions on Networking*, vol. 3, pp. 226–244, June 1995.

[56] Walter Willinger, Murad S. Taqqu, R. S. D. V. W., "Self-similarity in high-speed packet traffic: Analysis and modeling of ethernet traffic measurements," *Statistical Science*, vol. 10, no. 1, pp. 67–85, 1995.

[57] Wang, Y. and Witten, I., "Inducing model trees for continuous classes," *Proc of Poster Papers, 9 th European Conference on Machine Learning, Prague*, April 1997.

[58] Will E. Leland, Murad S. Taqqu, W. W. D. V. W., "On the self-similar nature of ethernet traffic," tech. rep., Bellcore Inc., Boston University, Morristown, New Jersey, March 1993.

[59] Will E. Leland, Murad S. Taqqu, W. W. D. V. W., "On the self-similar nature of ethernet traffic," *IEEE/ACM Transactions on Networking*, vol. 2, pp. 1–15, February 1994.

[60] Y. Bao, X. S. and Trivedi, K. S., "A workload-based analysis of software aging and rejuvenation," *to appear in IEEE Transactions on Reliability.*

[61] Y. Bao, X. S. and Trivedi, K. S., "Adaptive software rejuvenation: Degradation model and rejuvenation scheme," *Proceedings of the International Conference on Dependable Systems and Networks, DSN2003,San Francisco*, pp. 241–248, June 2003.

[62] Y. Huang, C. M. R. Kintala, N. K. and Fulton, N. D., "Software rejuvenation: Analysis, module and applications," *In Proc. 25th International Symposium on Fault-Tolerant Computing, Pasadena, CA,*, 1995.

VITA

SUMMARY

- 6 years experience in the design & development life cycle of object oriented applications, distributed systems, database management & customer relationship management systems
- MSEE and pursuing MS in Computer Science with excellent academic record
- Brainbench Certified C++ Programmer

PROFESSIONAL SKILLS

Languages Java, C++, C, TCL/TK, PL/SQL, XML, JavaScript, HTML, AWK, Shell Scripts

IDE Visual Studio 6.0, Visual Studio .NET, Oracle Jdeveloper10g, DELPHI

Technologies J2EE, MFC, MPI, TCP/IP, UDP, RPC, Web Services

Database Administration Sybase SQL Server 12.0, Microsoft SQL Server 2000, Oracle 9i Database, Oracle 9i Developer Suite, Oracle Warehouse Builder 9.0.3.37, MySQL 4.0.13, Postgresql 7.4

Operating Systems UNIX, FreeBSD, Linux, WINDOWS XP, 2000

CASE Tools Rational Rose Enterprise Edition, Teamwork CASE Tool, Scitools Understand Java

Siebel eBusiness 2000 Siebel Tools, EIM, EAI, Siebel VB, Call Center Enterprise, Server and Application Administration, Requirements Analysis, Workflow and Assignment Manager

Other CRM Tools Scopus Foundation, Synchronicity

Application Servers Weblogic Server, Zope, Tomcat, Axis, Websphere Server, MS IIS

Enterprise Systems IBM DB2 Content Manager and IBM WebSphere II OmniFind Edition

WORK EXPERIENCE

- (December 2004 - Present) Software Engineer, IBM Inc.
- (October 2004 - November 2004) Software Engineer, Global Pundits Inc.
- (September 2004 – October 2004) Consultant, Information Solutions Group, The World Bank
- (June 2004 – August 2004) Summer Intern, Information Solutions Group, The World Bank

- (December 2003 – May 2004) Graduate Assistant, Business Intelligence Project, Office of Information Technology, WVU
- (Jan 2002 – December 2003) Research Assistant, NASA IV and V and Limbic Systems Research Projects Department of CSEE, WVU
- (Jan 1999 – Dec 2001) Member of Technical Staff 1, Broadband Access/ FPGA Groups Lucent Technologies Microelectronics /Agere Systems
  - (Oct 2000 Dec 2001) BBACS Organization Siebel Implementation Project
  - (Jan 2000 – Sept 2000) Siebel 99.5 and 2000 Implementation Projects Setup Siebel eBusiness 2000 and 99.5 for FPGA, Sonet and BBACS Groups.
  - (Jan 1999 – Dec 2001) CRM System Development/ Administration
- (Nov 1998 – Jan 1999) Software Developer-Insurance Software Development Project, Norell Information Systems, Cincinnati Financials, Ohio
- (Aug 1996 – Aug 1998) Research Assistant/Teaching Assistant- NSF Funded Project, West Virginia University, Morgantown, WV

EDUCATION

1. Master of Science in Computer Science WEST VIRGINIA UNIVERSITY Morgantown, WV, Expected, Spring 2006, GPA 4.0/4.0, Advisor: Prof. B. Cukic
2. Master of Science in Electrical Engineering WEST VIRGINIA UNIVERSITY Morgantown, WV, August 1998, GPA 4.0/4.0, Advisor: Prof. S. K. Tewksbury
3. Bachelor of Engineering in Electronics and Communications Engineering, ANDHRA UNIVERSITY, Visakhapatnam, India, May 1996, Advisor: Prof. G. S. N. Raju
4. Honors in Systems Management National Institute of Information Technology (NIIT), India, May 1994, Grade Excellent

ACHIEVEMENTS

1. Awarded IBM Bravo Awards April and May 2005
2. Publicly recognized in All Hands Meeting for work performed on TIMES
3. Awarded Lucent Performance Award for implementation of a Customer Relationship Management (CRM) System for the SONET/ONS Customer Applications Group, August 2000

4. Placed in the top of the batch at the end of MSCS and MSEE Programs

5. Brainbench Certification in C++, September 2003

PUBLICATIONS

- Validation and Reliability Estimation of a Fingerprint Image Registration Software, ISSRE 2004, St-Malo, France
- Software Aging and Multifractality of Memory Resources, DSN-2003, San Francisco, CA, 2003
- Int. Memory, Network Architectures for Cluster Organized, Parallel DSP architectures, Santa Cruz, CA. 1998
- A Parallel DSP Testbed with a Heterogeneous and Reconfigurable Network Fabric, IEEE 1997, ISIS, Austin, TX