August 1986 DAA/LANGLEY UILU-ENG-86-2230
CSG-55

COORDINATED SCIENCE LABORATORY
College of Engineering

FAULT AND ERROR LATENCY UNDER REAL WORKLOAD-- AN EXPERIMENTAL STUDY

Ram Chillarege

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

FAULT AND ERROR LATENCY UNDER REAL WORKLOAD — AN EXPERIMENTAL STUDY

BY

RAM CHILLAREGE

B.Sc., University of Mysore, 1974
B.E., Indian Institute of Science, 1977
M.E., Indian Institute of Science, 1979

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1986

Urbana, Illinois

FAULT AND ERROR LATENCY UNDER REAL WORKLOAD — AN EXPERIMENTAL STUDY

Ram Chillarege, Ph.D.
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, 1986

This thesis demonstrates a practical methodology for the study of fault and error latency under real workload. This is the first study that measures and quantifies the latency under real workload and fills a major gap in the current understanding of workload-failure relationships. The methodology is based on low level data gathered on a VAX 11/780 during the normal workload conditions of the installation. Fault occurrence is simulated on the data, and the error generation and discovery process is reconstructed to determine latency. The analysis proceeds to combine the low level activity data with high level machine performance data to yield a better understanding of the phenomenon. This study finds a strong relationship between latency and workload and quantifies the relationship. The sampling and reconstruction techniques used are also validated.

Error latency in the memory where the operating system resides is studied using data on physical memory access. These data are gathered through hardware probes in the machine that samples the system during the normal workload cycle of the installation. The technique provides a means to study the system under different workloads and for multiple days. These data are used to reconstruct the error discovery process in the system. An approach to

PRECEDING PAGE BLANK NOT FILMED

determine the *fault miss percentage* is developed and a verification of the entire methodology is also performed. This study finds that the mean error latency, in the memory containing the operating system, varies by a factor of 10 to 1 (in hours) between the low and high workloads. It is also found that of all errors occurring within a day, 70% are detected in the same day, 82% within the following day, and 91% within the third day.

Fault latency in the paged sections of memory is determined using data from physical memory scans. Fault latency distributions are generated for s-a-0 and s-a-1 permanent fault models. Results show that the mean fault latency of a s-a-0 fault is nearly 5 times that of the s-a-1 fault. Performance data gathered on the machine are used to study a workload-latency behavior. An analysis of variance model to quantify the relative influence of various workload measures on the evaluated latency is also given.

Error latency in the microcontrol store is studied using data on the microcode access and usage. These data are acquired using probes in the microsequencer of the CPU. It is found that the latency distribution has a large mode between 50 and 100 microcycles and two additional smaller modes. It is interesting to note that the error latency distribution in the microcontrol store is not exponential as suggested by other reported research.

ACKNOWLEDGEMENTS

I would like to express my gratitude and appreciation to my thesis advisors, Professors Ravi Iyer and Edward Davidson. I have learnt from them, more than what a mere academic interaction would have ordinarly provided, at different times and in different ways. Their support, friendship, and guidance have been invaluable. Professor Iyer has spent many long hours in discussing the research and in helping put this thesis together.

It has been a pleasure to be a part of the Computer Systems Group, and I thank all the members, past and present, staff and students, for the stimulating environment and their friendship. Although there are a number of people that I owe thanks to, I would like to mention some in particular. Ron Odom and Joel Emer for the timely loan of measurement equipment. Rick Berry, Subhasis Laha, and Geoff McNiven for their help during the instrumentation of the VAX and Bill Rogers for the slides. Jackie Ziemer, Paula Pachciarz, and Katy Lindquist have been very helpful in our office.

Kelly, my wife, has been a constant source of encouragement and sustenance. I met her during the course of this research and she has been a significant influence towards this end. I thank her for being a friend and a partner.

I am very grateful to my parents for their selfless pursuit to help me get a higher education. Their encouragement and belief have meant a lot to me. My sister, Meera and her husband, Chandra, have always been there whenever I needed them.

You cannot see the seer of sight,
you cannot hear the hearer of hearing,
you cannot think the thinker of thought,
you cannot know the knower of knowledge.
This is your self that is within all.

— *Brhadāranyaka Upanishad*

TABLE OF CONTENTS

CHAPTER PAGE

CHAPTER PAGE

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The widespread dependence on computing resources in our society has made reliability and integrity central issues in computer system design. An important prerequisite in designing for reliability is an understanding of the effect of faults in the system and their manifestation. Due to the complex nature of a system, its behavior under fault is not easy to comprehend.

This thesis concerns itself with the discovery process of faults in the system and how it is affected by the workload. Since a system has components that are not always utilized there is usually a time delay between the occurrence of a fault and its manifestation as a malfunction. This time delay is defined as the fault or error *latency.* An analytical study of latency is unfeasible at this stage. due to a lack of understanding of the complex interactions involved with fault occurrence and manifestation. Hence, a systematic experimental methodology is adopted in this thesis for studying latency and the associated issues.

The understanding of fault and error latency has many ramifications in reliable system design. Primarily, the knowledge of latency is essential for accurate reliability prediction. Large latencies may result in multiple errors thereby rendering many detection and recovery mechanisms ineffective. Furthermore, knowledge of latency is essential to design effective rollback recovery mechanisms. This is necessary since rolling back less than the latency time may result in repeated failures due to corrupt information.

Another motivation for determining latency arises due to the dependency of failures on system activity as reported in [1, 2]. These studies reported a sharp decline in the reliability of computing systems at high utilization. One possible cause of this phenomenon is the latent discovery of faults and errors. Latent discovery suggests that faults occur randomly and an increase in workload reveals the faults thus resulting in a noticeably higher failure rate at higher loads. Hence, it becomes imperative to study fault and error latency under real workload conditions.

1.1. Goal of the Thesis

The primary focus of this research is to develop and demonstrate an experimental approach to study fault and error latency in a machine under real workload.

Fault and error latency are fundamental issues in determining the reliability of large systems and have not been well understood because of the complex nature of the system. Measurements to determine latency become even more complicated when they need to be performed on a production installation.

This research is timely since the present growth of computers is headed toward multiple machines or co-operating clusters of machines where problems due to latency are crucial. The strong emphasis on high availability computing further necessitates a good understanding of the fundamentals of the fault and error discovery process in machines.

1.2. Thesis Outline

Chapter 2 defines fault and error latency and their ramifications. A survey of earlier latency studies and their limitations is presented. The methodology adopted in this thesis for the measurement of latency under real workload follows. The subsequent three chapters are each self-contained and present research on latency in specific subsystems of the machine under study.

Chapter 3 presents a study in error latency in the operating system region of the memory. The instrumentation used, and the measurement and computation of error latency is discussed followed by the error latency distributions that are generated. The validation of the technique is given followed by an estimation of the fault miss percentage, i.e., the chance that a fault is not discovered, an important parameter associated with error latency that can only be estimated in an experimental setup.

Chapter 4 illustrates fault latency in the paged regions of memory. The computation of fault latency and the experimental setup are discussed followed by the latency distributions. The latency information taken together with workload information is further used to develop a workload-latency model.

Chapter 5 examines error latency in the microcontrol store of the processor. The experimental setup is described followed by the analysis and latency distributions. This experimental setup is versatile and can be used for a variety of performance studies as well. The applicability of the data and instrumentation are also discussed.

Finally, Chapter 6 presents a summary of this research and describes the future scope of this work.

CHAPTER 2

LATENCY AND ITS MEASUREMENT

2.1. Fault, Error, and Failure

There has been considerable confusion in terminology regarding *fault*, *error*, and *failure*. In an attempt to provide a conceptual framework for expressing the attributes that constitute dependable and reliable computing, these terms have been informally but precisely defined in [3]. The basic definitions of the terminology are best quoted from the text:

> The **service** delivered by a system is the system behavior *as it is perceived* by another special system(s) interacting with the considered system: its user(s).
>
> A system **failure** occurs when the delivered service deviates from the specified service, where the service **specification** is an agreed description of the expected service. The failure occurred because the system was erroneous: an *error* is that part of the system state which is liable to lead to failure, i.e., to the delivery of a service not complying with the specified service. The cause -- in its phenomenological sense -- of an error is a **fault** [3].

This definition, although precise, is general enough to be applied to a wide variety of systems or applications. In addition, the definitions can also be suitably interpreted in different planes of applicability, such as the physical, electrical or logical planes. This thesis deals with stuck-at fault models that refer to the logical plane. This fault model is widely used and is representative of a number of physical faults. Since it is a commonly used fault model, the results from this study provide an insight which is useful to a large body of applications.

2.2. Fault and Error Latency

The time between the occurrence of the fault and the time of its discovery, i.e., the failure, is defined to be its total latency. The fault model chosen for this study is the single *stuck-at* fault model. This fault model refers to the logical plane and can be caused due to a variety of malfunctions at the physical or electrical plane.

In order to express the effects seen at the logical plane two new terms are introduced, namely, *active* and *inactive* faults. Using Figure 2.1. for illustration, consider a bit in a word containing data with a value of 1. If a s-a-1 fault occurs on that bit, then the fault cannot cause a failure. This *fault* is called *inactive* and is *latent*. If during a write into the word the new data attempts to change the value of this bit to a 0, then the fault becomes *active*. An *active fault* is defined as an *error* since it is *that part of the system state which is liable to lead to failure*. During a subsequent read operation, either the Error Correcting Code (ECC) will detect and correct the error, or, lacking ECC, a failure will occur. The time taken for the *inactive* fault to become *active* is defined as **fault latency**. The time taken for the *error* (an *active* fault) to cause a failure is defined as **error latency**. The sum of the error and fault latency is the **total latency**. If the bit in Figure 2.1. originally contained a value of 0, then the fault is active at the time of its occurrence and hence has a fault latency of zero. Note that the fault latency can be zero, but the error latency is always non zero.

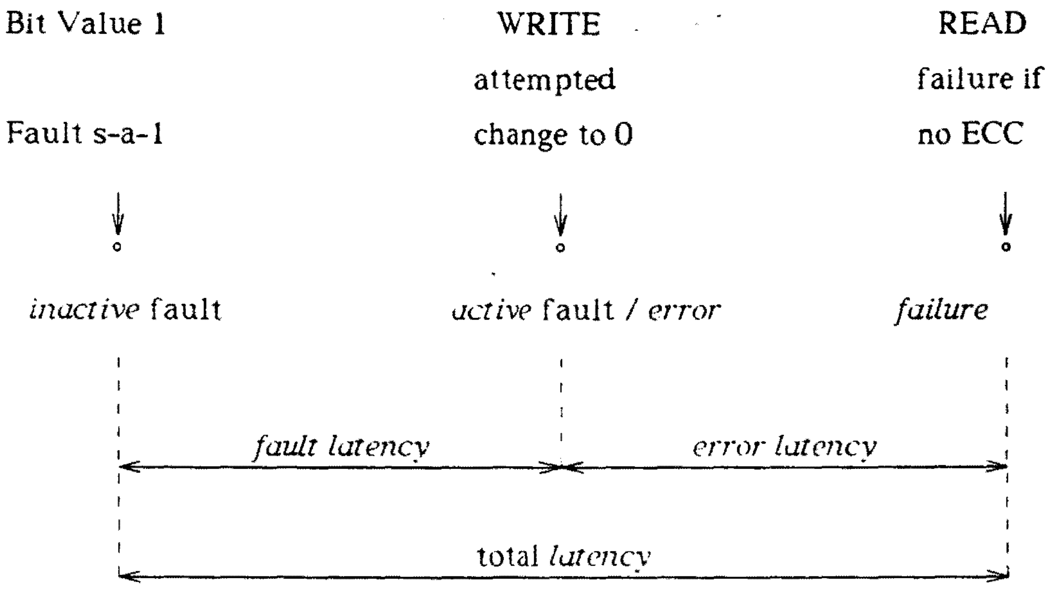

Figure 2.1. Fault and Error Latency.

2.3. Classical Failure Analysis

A number of studies on failure data obtained from computer installations have been performed to study machine reliability, failure trends, or patterns of failure, etc. The data on failures are usually obtained from machine-recorded information in cases when such error logs exist and from operator-recorded information when they do not. Since such data only contain information on the detected failures, nothing is known about failures that were undetected. Additionally, the moment of fault occurrence and error generation is not observable from these data. Thus, the studies are usually limited in their scope to studying the failure event and the associated history of other failures and the environment under which these failures occurred.

A variety of interesting studies has been performed using machine failure data. These studies have analyzed failures that occur in different parts of the system and also separately analyzed hardware and software failures.

2.4. Earlier Latency Studies

There have been a number of studies on latency; however, there have been no studies using real workloads. There is no general technique for the measurement of latency. A gate-level emulation of an avionic miniprocessor is reported in [4] and [5]. A set of specific programs was used to exercise the machine. The programs do not, however, represent a real workload environment. Therefore, the methodology and results are not generally applicable. Another similar experiment is described in [6]. The delay between the occurrence of an error and the moment of its detection is defined as *detection time* in [7, 8] and as *latency difference* in [9]. In [8], Courtois presents a methodology for on-line testing of microprocessors and develops the distribution of detection time of failures affecting the heart of the M6800 CPU die. In [10], Shedletsky shows that error latency is a geometrically distributed random variable for a very general class of faults in combinational digital circuits. Most studies have almost always measured *error latency* or the sum of both *fault* and *error latency*. A significant attempt at determining *fault latency* by Shin is [11]. The authors use an *indirect* technique to estimate fault latency at the pins of the chips in the *CPU* of the Fault Tolerant Multiprocessor (FTMP). Since the exact moment when the fault becomes active is not known, the technique provides only an upper bound for fault latency.

All the studies so far reported have used specific programs or fault injection on special purpose machines. Thus there have been no studies that measure latency in a real workload environment. This study is the first study that measures both fault and error latency under a real workload. These *error latency* measurements, in the unpaged section of the operating system under a real workload, are reported in [12]. Further work on this including the validation of the technique appears in [13]. The measurements on *fault latency* in the paged sections of memory are reported in[14].

2.5. Methodology of Measurement

Hardware faults mostly occur at a device level; hence, measurements to measure fault or error latency have to be performed at a low level. The measurement must acquire data that contain the change of state which activates the fault and propagates the error condition. There are a few alternatives for this measurement, and they predominantly use hardware probes to gather the data. In certain cases a software probe can also be used. Data acquisition with probes is usually complicated by physical problems due to restricted access to circuitry and circuit integration.

The key to making such measurements is to define appropriate probe points that will yield the most relevant data for the specific study. The choice of such probe points is an important issue in the design of the experiment. The probe points are typically defined from a functional stand point, however, in practice it is a trade-off between available accessible points. Since the volume of such data can be very large, the choice should also take into consideration the fact that the high volume of data can be handled by the instrumentation used. The final design of the instrumentation is predominantly influenced by the available instruments. Although it is conceivable that instruments are designed to suit the requirements of the measurement, in reality, the availability of instruments can in fact dominate the project.

There exist two popular schemes for hardware measurements. One of them is counter based and the other trace based. The applicability of each is very dependent on the type of problem. In certain cases either can be used with some degree of adaptability. For this thesis both types of instruments were available, however, only the trace based instrument was used.

2.5.1. Counter-based techniques

Counter-based techniques essentially use counting to make measurements on a certain set of events [15, 16]. Essentially a large number of counters typically ranging anywhere from 128 to as large as 16K are used for this purpose. The probes along with some encoding/decoding logic detect events in the machine that need to be counted. At the end of the measurement period the counters provide a histogram of the various events that were counted. The period of time that the system can be observed continuously is limited only by the length of the counters. Hence, the maximum sampling interval is the time taken by the most frequent event to overflow the counter. The instrument generally provides for backup of the counters and reinitialization.

The advantage that this technique provides is the large sampling interval since providing counters with larger length is relatively simple. Additionally, it is practical to provide a large number of counters since they can be made using memory. The major drawback of the technique is the fact that it is based on counting. Counting necessitates that all the events to be measured are known a priori and are finite. Further, the timing and history information associated with the events are lost.

2.5.2. Trace-based techniques

Trace-based techniques, as the name suggests, provide a trace of events or data [15, 12]. The data from the probes do not necessarily have to be decoded or encoded to be stored in a buffer memory. Thus, the maximum sampling period is determined by the depth of the buffer memory. The start of the sampling may, however, be triggered through a sequence of events. The data may also be qualified with logic so that only a subset of the data seen by the probes is recorded in storage. These instruments also provide for backup of the buffer and reinitialization.

The advantage of this technique is that it provides a very true representation of events or data as they occur in the machine. Since a priori knowledge of specific events is not necessary, such as in the case of the counter-based technique, this is an excellent method for exploratory studies. The data contain history and timing information which are valuable. However, since the buffer memory has only a finite storage, the data acquisition is forced into being a sampling system. This impacts the measurement technique by requiring the sampling method to be validated.

CHAPTER 3

ERROR LATENCY

3.1. Introduction

The study of error latency is an important issue in fault tolerant computing with significant implications in both reliability prediction and testing. The time between the occurrence of a fault and its manifestation as an error has been referred to as *error latency*[10, 3]. Many errors can only be detected when a particular module or subsystem is *exercised*. Thus, although the failures may not be caused by increased utilization, they are *revealed* by this factor, causing a higher *observed* error rate due to increased workload. The difficulties with the measurement of error latency are that the moment of error generation is unknown and failure records only contain information on detected errors.

There is, in addition, considerable experimental evidence to show that computer reliability is a dynamic function of system activity (as measured by the workload). Workload-failure studies [17, 1] (on IBM machines) and [18, 2] (on DEC machines) provide evidence that CPU and memory failure rates increase rapidly as the system workload approaches saturation. The *cause-effect* relationship in this dependency is unknown, however, it is speculated that one component in this relationship is due to latency [1]. An explicit model for this is given in[19]. Another possible reason for the *observed* workload-failure dependence is the stresses imposed by high currents and voltages. A model based on this is given in [20].

There is no general technique for determining error latency under various workload conditions. Studies on CPU fault latency for an avionic miniprocessor, determined through a gate-level simulation, are described in [4, 21]. A set of a specific programs was used to exercise the machine to reveal faults that were injected into the simulation. Another similar experimental study is found in [6]. Although the approach and results are valid for the case studied, they are not applicable to multi-user systems. Furthermore, it is not practical to measure workload effects through such simulations. Similar studies that do not use a real workload can be found in [22, 23, 24].

In this chapter, a methodology to study the latency characteristics of medium-to-large computer systems is developed. The technique is applied to the memory subsystem, however, the methodology, in principle, is also applicable to the microcontrol store of the CPU. The scope and implication of failure in memory go far beyond the memory subsystem. In addition to the largest number of failures occurring in the memory [25], it has been shown that, a large number of the *CPU* errors are traced to originate from the memory [26].

This is the first attempt at jointly studying error latency and workload variations in a full production environment. The method is based on sampled data of physical memory activity gathered, through hardware instrumentation, during the normal workload of the installation. The data are then used to reconstruct the error discovery process in the system. The measured system is a VAX 11/780 that runs the Unix Berkeley Version 4.2 operating system. The system has 20 to 25 interactive users during the peak hours. The workload comprises a variety of scientific and miscellaneous word and data processing applications. The hardware instrumentation has the advantage of not biasing the workload of the machine during measurement, and sampling provides an effective means to work with the large volumes of data generated by observing the system for multiple days. A detailed validation of the sampling technique is performed, and it is shown that the approach can successfully predict the percentage of undetected errors.

Section 3.2 discusses the instrumentation used, Section 3.3 the measurement, and Section 3.4 the computation of error latency. Section 3.5 shows and discusses the error latency distributions that are generated, and Sections 3.6 the validation of the technique. Section 3.6 also discusses the estimation of *miss percentage*, an interesting attribute of error latency, that can only be estimated in an experimental setup.

3.2. Instrumentation

3.2.1. The system

The instrumentation is an intersting project in itself. As explained above, for the purpose of this study, the emphasis is on memory activity. For the purposes of studying error latency, physical memory activity needs to be measured. This is only possible through hardware instrumentation and direct access to the memory. The backplane of the VAX CPU was probed and the data sampled by the instrumentation. The hardware instrumentation has the additional advantage of not interfering with the regular workload of the system.

The VAX central processor and the memory subsystem are linked through a data path called the Synchronous Backplane Interconnect (SBI). Figure 3.1. shows the organization of the machine. which are given in[27]and [28]. The SBI is a parallel datapath that is multiplexed for address and data, and uses a 200 nsec clock to achieve a maximum information transfer rate of 13.3 million bytes per sec.

The best approach for obtaining memory activity information on the VAX is to monitor the SBI through which all transactions occur. Requests to memory can arise from either the CPU or from the I/O devices. and all of them are transacted through the SBI. Therefore. monitoring the SBI captures all requests to the memory subsystem. The address space on the SBI is partitioned so that addresses to the main memory subsystem. Unibus subsystem. or other adapters are unique thus partitioning access to the subsystems to be individually extracted.

The SBI consists of 84 signal lines that belong to five different groups. namely. arbitration. information transfer. response. interrupt. and control. The information transfer group with 46 signal lines contains the memory activity information. It is used to transfer addresses. data. and interrupt summary information. This group is subdivided into five fields that represent parity check (P). information tag (TAG). source or destination identification (ID). masks (MASKS). and 32 bits of information lines (B). as in Figure 3.2. (page 21).

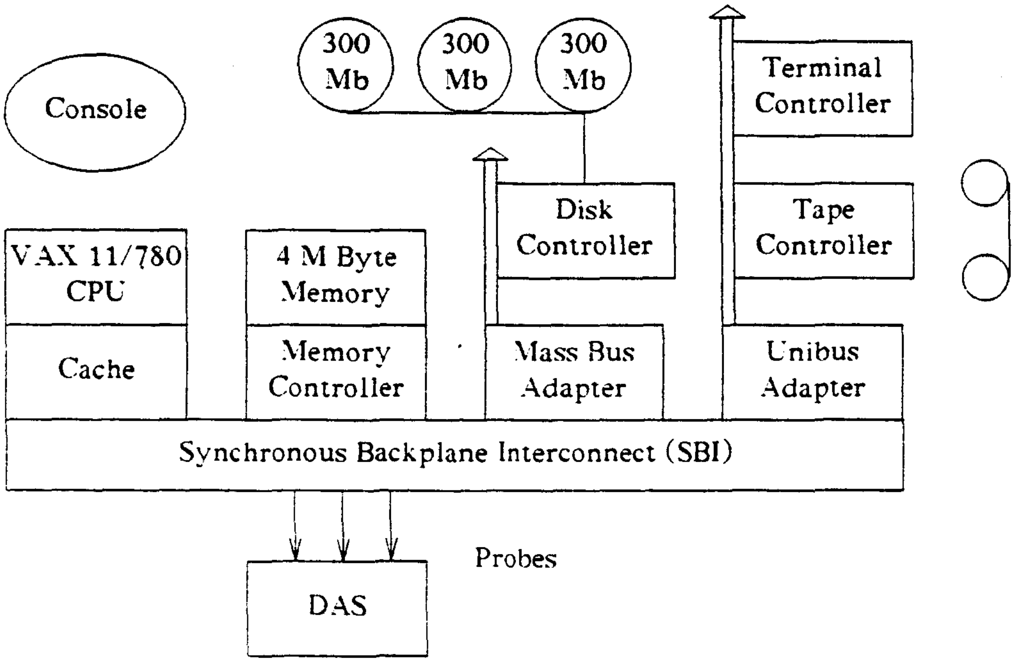

Figure 3.1. VAX 11/780 System Organization.

(1) P field: The parity field of 2 bits provides even parity for detecting single bit errors in the information transfer group. One of the bits provides parity over the TAG, ID and MASK fields and the other over the B field.

(2) TAG field: The TAG field, is 3 bits wide and indicates the information type (being transmitted) on the information lines (B field). This field also determines the interpretation of the ID and the B fields. For example, when the tag code represents COMMAND ADDRESS, the B field contains the address.

(3) ID field: The ID field of 5 bits is used to identify the logical source of the data in a write command and the logical destination of the data in a read command. The address of the location is contained in the B field.

(4) MASK field: The mask field is 4 bits wide and is used to specify operations on any or all bytes of the data in the B field. Each bit in the mask field corresponds to a particular byte in B.

(5) B field: The B field is 32 bits wide (4 bytes) and is used to carry information/data. Depending on the TAG field the 32 bits are interpreted either as one data field of 32 bits or as containing two subfields: a FUNC field of 4 bits which identifies read or write mode and an ADDRESS field of 28 bits containing the physical address which can be either main memory or I/O.

3.2.2. Experimental setup

A Tektronix Digital Analysis System (DAS) 9100 Series was used to monitor and sample data transfer activity on the SBI. The DAS probes used can acquire data at speeds up to 40 nsec which is faster than the clock speed of the SBI (200 nsec). These probes were placed at the card edge connector of the SBI control cards [29] where the SBI signals were accessible. The data were read into the DAS using the SBI clock for external synchronization.

The experiment was controlled from the VAX with the aid of Tektronix 91DVV1 software and some additional programs. The DAS is programmable via an IEEE-488 interface, or an alternative serial line RS232c link to a host machine. It is connected to the VAX via the serial line interface. The software which controlled the experiment was such that it caused negligible overhead and did not bias the experiment. The acquisition system has been tested for data bias against itself. This is done by externally triggering the DAS, acquiring the data, and comparing memory usage distributions generated by this data with the distributions generated from automatically acquired data. It is found that the instrumentation is sound and does not indicate any significant influences of self-bias. The DAS was periodically triggered to acquire data from the SBI, download the acquisition memory, and time-stamp the data. This data was then preprocessed to make it compatible with subsequent input into statistical analysis programs and archived on tapes.

The instrumentation was tested for correct operation and acquisition. This was performed by taking the system down into single user operation, turning the cache memory off and running a test program that accesses specific locations of memory in sequence. Data collected on the DAS was then examined for correct acquisition against the known test program.

The acquisition memory of the DAS for the probes used for this instrumentation has a depth of 511 words. Two types of data were collected. The first is referred to as *regular mode*. This involves logging transactions of every SBI cycle. A sample of data acquired in *regular mode* is shown in Figure 3.3. The first line contains a time stamp for the sample. Each line of data represents one SBI cycle. The data is in one's complement form. Note that there are a number of idle cycles (all 1's). Figure 3.4. shows the decoded version of a single observation. The second is referred to as *compressed mode* and consists of a dense trace of addresses that are acquired by storing only those cycles that contain addresses.

3.3. Measurement

The experiment collects data on memory activity, i.e., physical memory address, access rate and read/write mode. For the purposes of this project, the region in memory where the OS resides is studied. This has the advantage of being the unpaged portion of memory. The data captures the memory activity of the whole physical memory. However, for the purpose of this project the concentration is upon a region of memory which contains the operating system which is largely unpaged. It, therefore, provides an estimate of inherent latency characteristics unaffected by paging. In addition, the errors in the operating system can be fatal. The methodology, however, is equally valid for both the

FIELDS	B					
	P	TAG	ID	MASK	FUNC	ADDRESS
BITS	2	3	5	4	4	28

Figure 3.2. SBI Information Transfer Group Fields.

PROBE:	3D	3C	3B	3A	2C	2B	2A	COMMENTS
	Fri Nov 23 12:15:11 CST 1984							
.15	11111111	11111111	11111111	11111111	11111111	11111111	11111111	idle
.16	10100101	10010101	11111010	01111111	11111111	11110011	10001111	cmd/adr
.17	11111011	00000010	11111111	11111111	11111111	11111011	11101111	data
.18	11111111	11111111	11111111	11111111	11111111	11111111	11111111	idle
.19	00101010	01001100	11110100	01111111	11111111	11111011	10001111	cmd/adr
,20	11111001	11001001	10001010	10000011	11111111	11111011	11101111	data
.21	11111111	11111111	11111111	11111111	11111111	11111111	11111111	idle

Figure 3.3. Acquired Data in Regular Mode.

FIELDS	ADDRESS	FUNC	CNF	unused	MASK	P	unused	TAG	ID
DATA	01011010 00001101 11111100 1101	1111	111	11111	0111	11	11	100	01111
PROBES	3D 3C 3B 3A	3A	2C	2C	2B	2B	2B	2A	2A

Figure 3.4. Decoding the Data.

unpaged and paged portions of the memory. The possible alternatives in collecting representative data were:

(1) Collect data all the time.

(2) Sample the measured system sufficiently, so as to get a representative distribution of memory activity.

The first is not only wasteful but also impractical, given the voluminous nature of the data involved and the large buffer sizes which would be required in the acquisition instrumentation. The second technique is adopted for its versatility and ease of implementation. The data acquisition was performed at intervals of approximately 40 seconds. The sampling is sufficiently frequent to capture the workload behavior. In the preliminary analysis the distribution of memory access stabilizes within 15 to 20 minutes of sampling. Figure 3.5. shows a memory usage histogram of the region studied that is generated from the acquired data. This means that if two 15-minute samples are considered, and the workload changes considerably during this period, it will be reflected in the samples. Thus, the error in the measured latency distribution is limited to less than 15 minutes. Figure 3.6. shows the *User CPU* as a function of the time of day for a typical day. Notice that there is a significant variation in *User CPU* during a day which suggests that the effect of workload on latency should be detectable. In examining the workload profiles, it can be seen that the *average* workload can be considered to be reasonably stable in any 15-minute period.

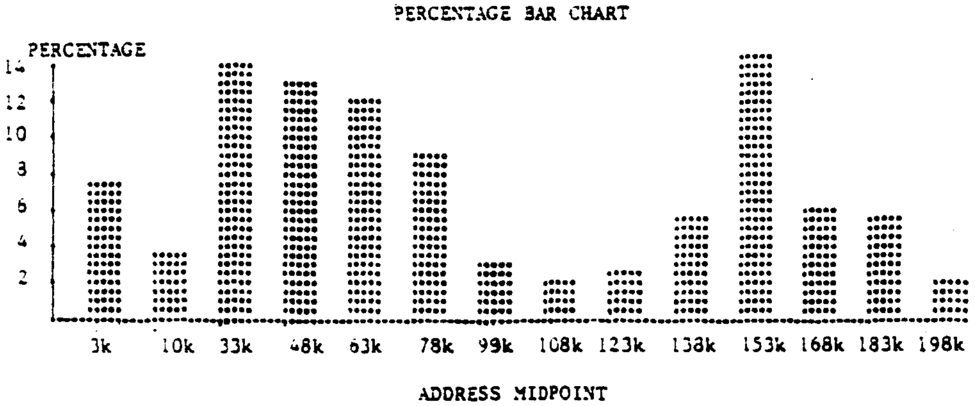

Figure 3.5. Memory Usage Histogram

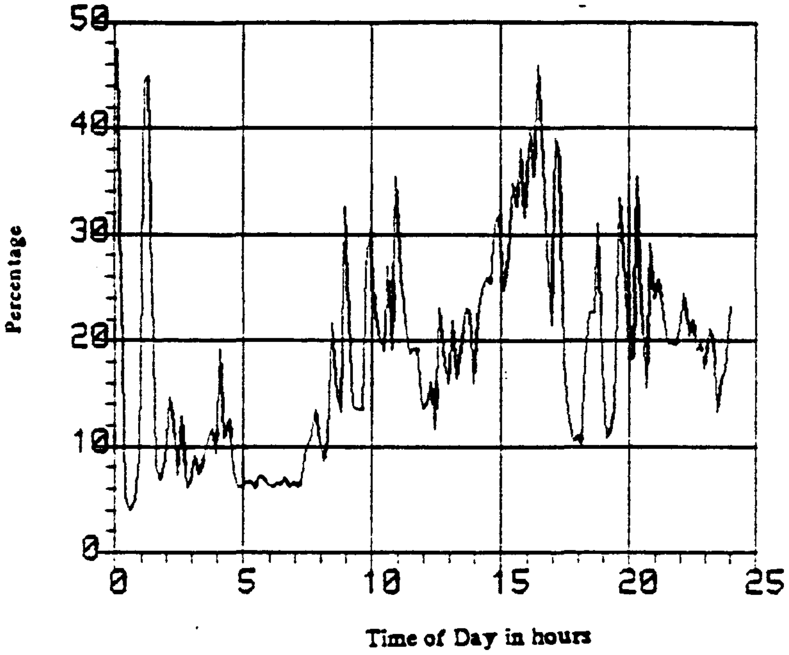

Figure 3.6. User CPU Usage Versus Time of Day.

3.4. Error Latency Determination

For the purpose of this study, it is assumed that any location in the measured region of memory is equally likely to fail. The assumption implies a uniform failure behavior within the region, i.e., the workload variation within the region does not cause any additional failures but merely influences the latency. This allows the determination of the distribution of the discovery process, without being biased by factors which cause faults. The memory addresses chosen to contain a fault are picked from a uniform random distribution. It should be noted that the methodology is equally applicable to other distributions of failure.

3.4.1. Fault model and latency calculation

The fault inserted is a *flipped* or *inverted bit* in a memory location. This is chosen since it results in an active fault which will be detected, during a read on the memory location, by the error detection and correction code (ECC). The time between the occurrence of the active fault, i.e., flipped bit, and its detection is the error latency of the fault (as discussed in Section 2.3). The fault model chosen also conforms the the definitions proposed in the IFIP working group 10.4.

Figure 3.7. shows the algorithm used to generate error latency distributions. A random memory location, say m_1, has a fault f_1. Let the fault be inserted at time t. The data are now scanned to find the first *memory read* to the location m_1. This is when the fault would be detected by the ECC circuitry. In Figure 3.7., location m_1 has three memory reads: one before and two after the

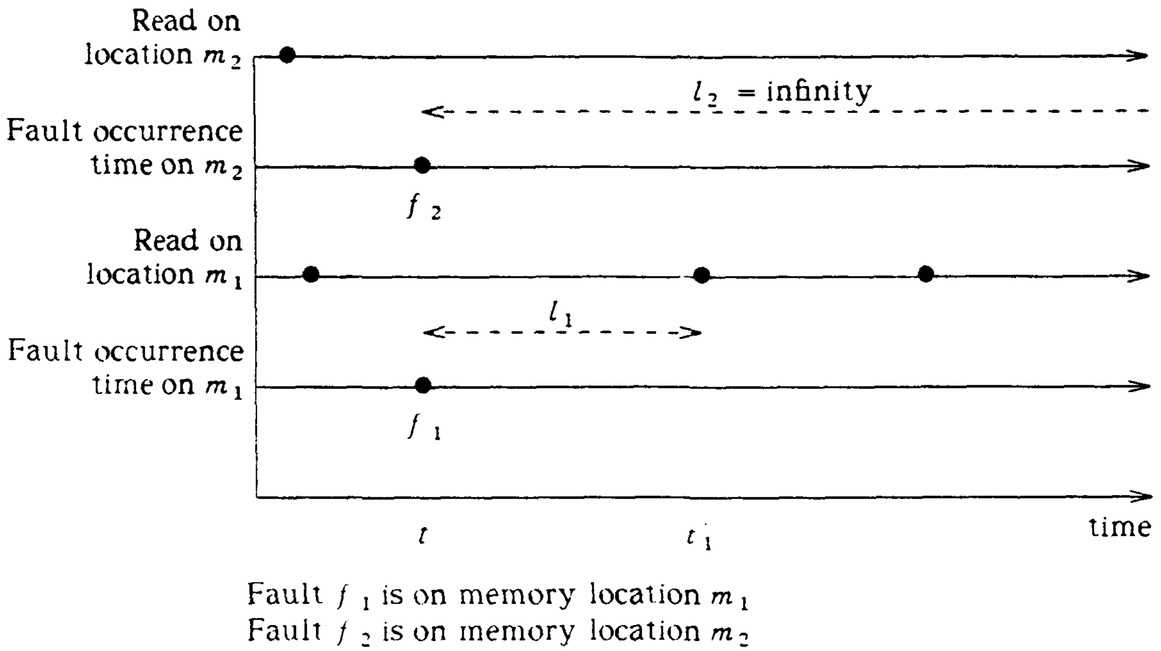
Read on location m_2
l_2 = infinity
Fault occurrence time on m_2
f_2
Read on location m_1
l_1
Fault occurrence time on m_1
f_1
t
t_1
time
Fault f_1 is on memory location m_1
Fault f_2 is on memory location m_2

Figure 3.7. Latency Time Calculation.

fault f_1. Let t'_1 be the time of the first memory read after the fault. Then the latency associated with fault f_1 is $l_1 = t'_1 - t$. The same location may be reaccessed (as in the figure) but that amounts to rediscovery and is not a part of this latency study. If, however, the data set never contains a read to the memory location in fault, then it goes undetected and amounts to a miss. For example, fault f_2 inserted in memory location m_2 is never detected. The misses are used to estimate the percentage of undiscovered faults. This process, of inserting faults and determining the latency, is repeated for a large number of faults, yielding a latency time for each inserted fault. The different latency times, taken together, generate a distribution of the error latency for the *fault occurrence time t*.

3.4.2. Algorithm implementation

In order to work with sampled data a *class* is defined. A *class* is a set of neighboring memory locations which are assumed to have a uniform probability of access. The number of memory locations in a class is termed the *class size*. The class size is chosen to reflect approximately uniform access rates within the class. The class caters to the fact that, although the sampled data are representative of the memory access pattern, it need not contain every distinct address that is generated. Thus, in the computation of error latency, the access to any member of the class can then be considered equivalent to access of the whole class. The algorithm, therefore, uses classes, in place of memory locations, to reconstruct the error discovery process. The class size is chosen small enough so that the memory usage within a class is uniform, i.e., each location in a class has similar access probability. and large enough so that the computation is still tractable. The class sizes that are chosen do not bear any relationship to the physical organization of the memory, although in reality the memory is organized in *classes* to a limited degree. An access to a byte or a word (two bytes) invokes a long word (4 bytes) to be read which corresponds to a class size of 4 bytes. Class sizes, varying from ¼ page to 3 pages, have been experimented with and it was found that the class size did not appreciably change the distribution and that the median varied by less than 5%. Section 3.5. discusses the class size and its ramifications in detail. Figure 3.8. shows the flow of data in the experimental setup and the offline processing.

3.5. Error Latency Distributions

In this section, the latency distributions generated by the technique are described. The workload effect on error latency is determined by placing faults in the data at a specific time of day and computing the error latency in the hours that follow the fault. The fault occurrence time is then moved in time across the entire measurement period, generating a distribution at each step. This generates a family of distributions (one for each fault occurrence time) which, taken together with the workload profile, show how the changes in workload affect error latency. A set of error latency distributions is shown for faults placed under *low* and *high* workload conditions. This demonstrates the variability of the mean latencies showing that it is a strong function of workload. Another set of error latency distributions is shown for measurement

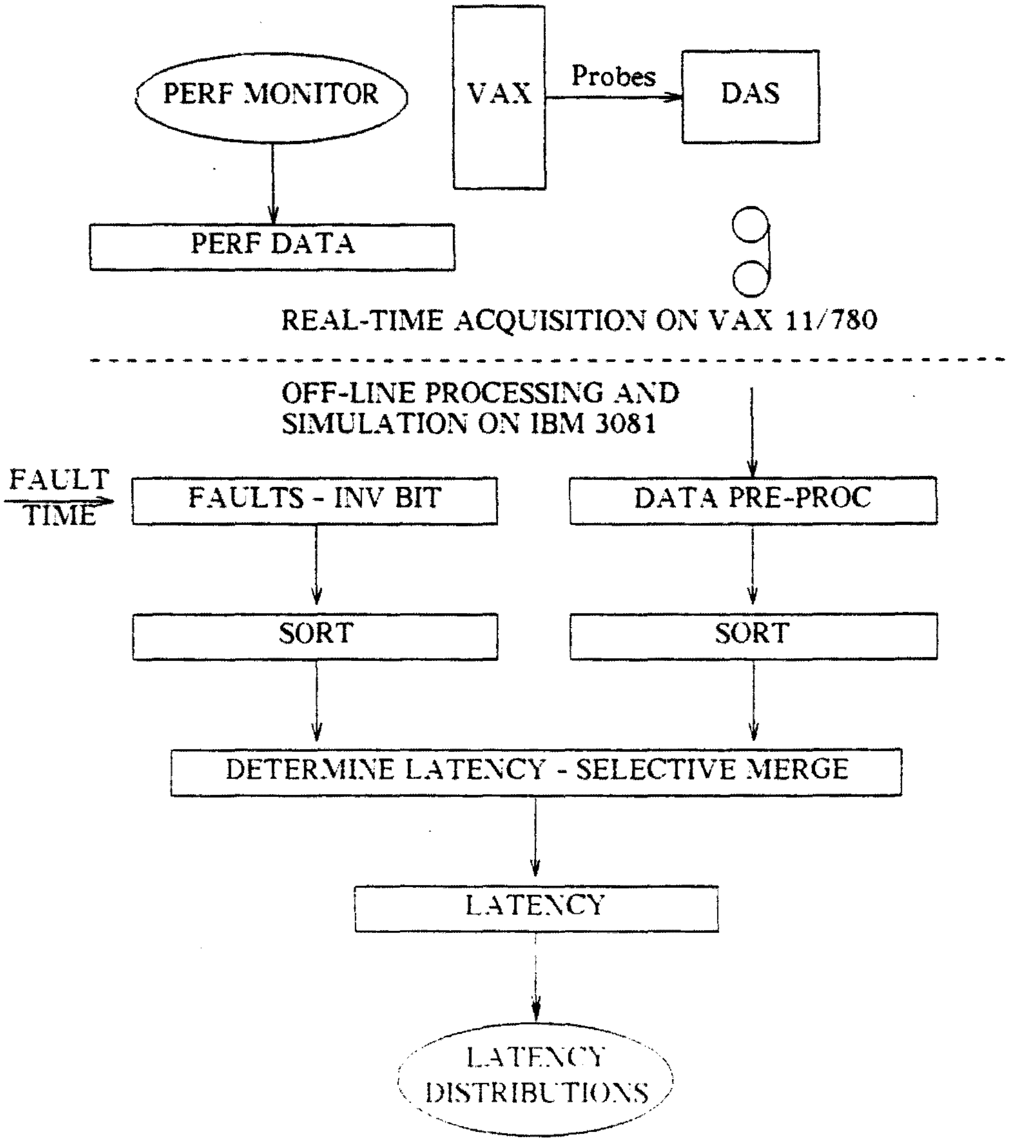
PERF MONITOR
VAX
Probes
DAS
PERF DATA
REAL-TIME ACQUISITION ON VAX 11/780
OFF-LINE PROCESSING AND SIMULATION ON IBM 3081
FAULT TIME
FAULTS - INV BIT
DATA PRE-PROC
SORT
SORT
DETERMINE LATENCY - SELECTIVE MERGE
LATENCY
LATENCY DISTRIBUTIONS

Figure 3.8. Acquisition and Processing of Data.

periods that span *two* days. The cyclic nature of the workload causes the latency distribution on the first day to repeat itself in the second. This demonstrates that the results are representative and not just a freak case. The hazard calculated from the error latency distribution shows that the *observed* error rate, due to the error latency, increases with workload.

3.5.1. Faults at low and high workloads

This subsection shows the error latency distributions for faults placed at two different times of the day. One, when the workload is very low and another, when the workload is high. It is found that there is considerable difference in the two error latency distributions. The mean latency can vary from as large as 8 hours for a fault at low workload, to as short as 40 minutes for a fault at high workload. This clearly demonstrates that error latency is a strong function of the workload that followed the fault.

Recall from Figure 3.6. that the system has a low workload from midnight to 7 a.m. and an increasing workload (intermediate) from 8 to 10 a.m., with a peak around 11 a.m. The intermediate period where workload changes from low to high is of particular interest. Figure 3.9. shows the latency distribution generated with faults inserted at midnight. The distribution is bimodal with the second mode being the larger of the two. The initial peak corresponds to a small period of high activity which usually occurs around midnight. Within the first hour about 10% of the detected errors are found. The bulk of the errors (70%) are found in the second mode. There is a sharp increase in the number of errors being detected about 8 hours after the fault. This corresponds

Parameter	H:MM:SS
Mean	8:03:28
Std Dev	4:01:19
Med	9:18:43

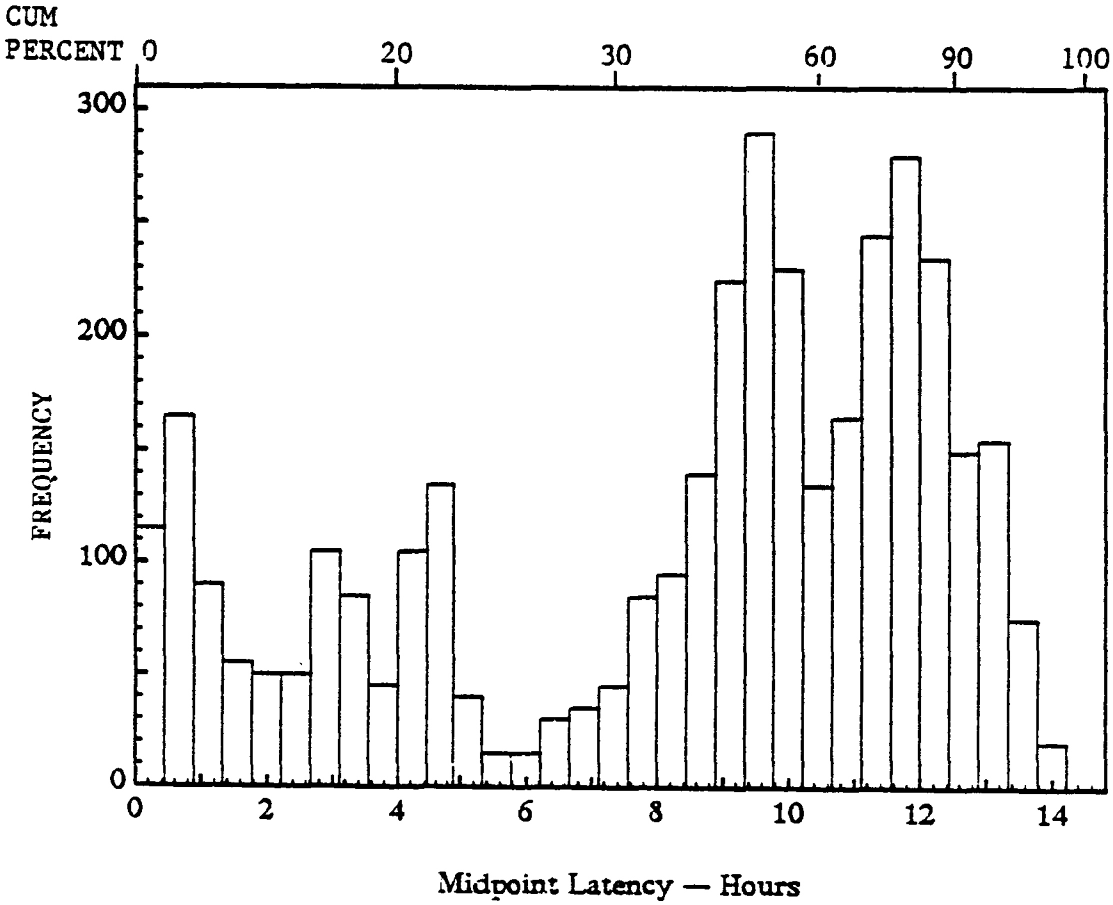

Figure 3.9. Error Latency Distribution - Fault at 00:00 hr.

to 8 a.m. (real time of day) which is the start of the increasing workload period. Also note that there is a dip in the distribution after the mode. This corresponds to a lull in the activity that occurs around 10:30 a.m. or so in this system. This system is largely used by graduate students, whose day starts at around 10:30 a.m. and continues past the lunch hour. The early morning (8 a.m.) rise in activity is due to secretarial and staff users. This clearly shows the influence of workload in determining error latency. The mean latency is 8:03 h:m. Listed in the figure are the percentages which correspond to detected errors only. Nearly 25% of the faults inserted were undetected. Missed faults and the associated miss percentage are discussed in Section 3.6.

Although these distributions presented here are of a specific day, data from a number of different days have been similarly analyzed. No matter how low the workload when the fault occurs, there is always an *initial discovery* of faults that contributes to a mode (though small) in the latency distribution. In Figure 3.9. the initial peak in this distribution is due to a combined effect of the *initial discovery* and also to the fact that there is a peak in the early hours of the morning caused by some system routines. The second mode (larger) is due to the workload that discovers the faults. If, however, the fault occurred at a time during the high workload, say 12 p.m., then the *initial discovery* mode would be dominated by the large discovery due to the high workload. Figure 3.10. has faults inserted at 12 p.m. (well into the high workload period). In contrast to Figure 3.9., the mean error latency is now down to 44 minutes with 70% of the detected errors discovered in the 1st hour. Thus faults occurring at

Parameter	H:MM:SS
Mean	0:44:26
Std Dev	0:29:19
Med	0:43:57

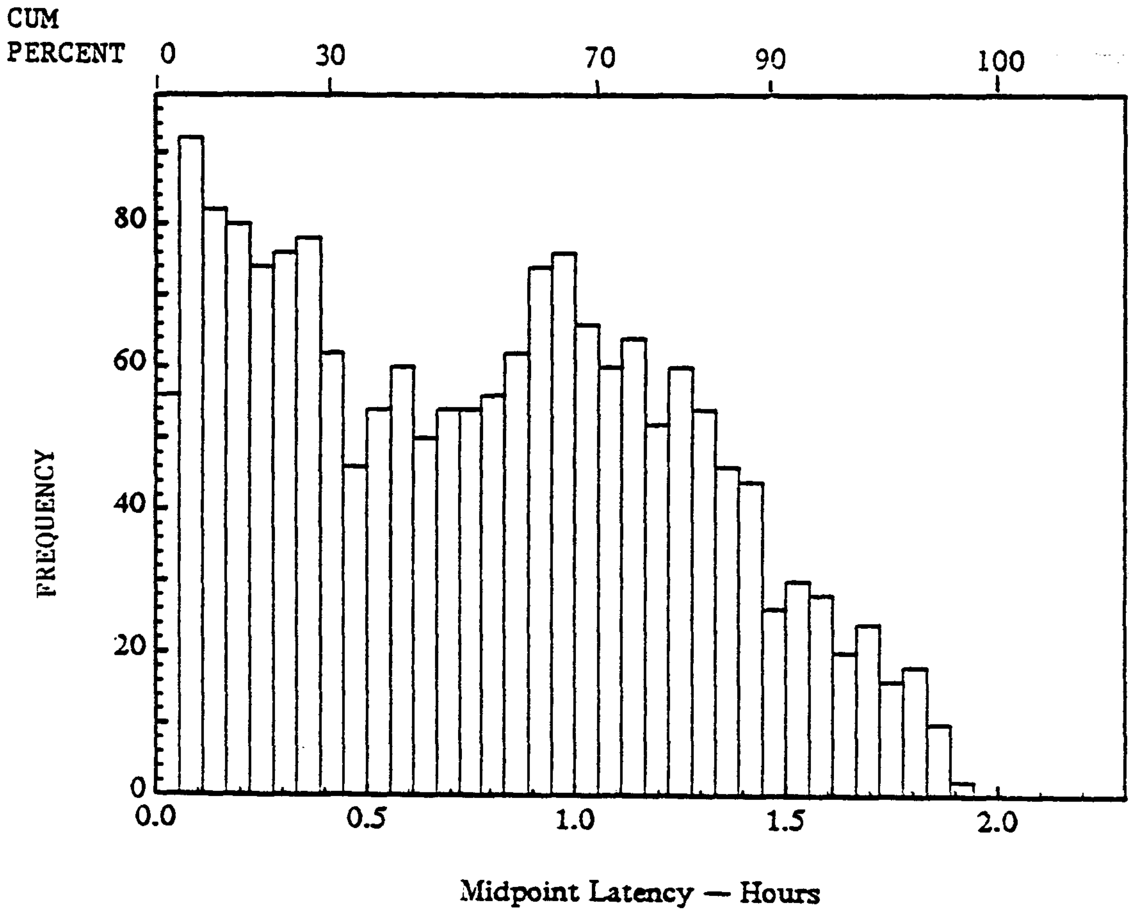

Figure 3.10. Error Latency Distribution - Fault at 12:00 hr.

low workload can be discovered with latencies as large as 10 hours (on the average) versus only 1 hour for errors occurring at high workload.

3.5.2. Multiple day measurement

In this subsection the effect of a cyclic workload on the error latency distribution is studied by considering a measurement period that spans two consecutive days. The error latency distribution of the first day reappears in the second. This further demonstrates that the distributions generated by this technique are stable and hence representative of the workload in general. This is also explicitly shown by generating three distributions for three different fault occurrence times during these 2 days. A fault is detected with 70% confidence within the first day and incrementally in the following days.

Figure 3.11. shows three latency distributions with the fault occurrence times advanced relative to each other. To make the latency distributions easier to compare, the latency times have been shifted to match up with the real time of day. In Figure 3.11a. the faults occur at 00:00 hours on the first day and the latency times (abscissa) are the same as the time of day. In Figure 3.11b. the faults are inserted at 8:00 a.m.. and the latency times shifted by 8 hours to match up with the time of day. Figure 3.11c. has faults at 4:00 a.m. on the second day with the latency times shifted by 28 hours. Notice that when the faults are inserted in the first day, the pattern of latency distribution of the first day reappears on the second day.

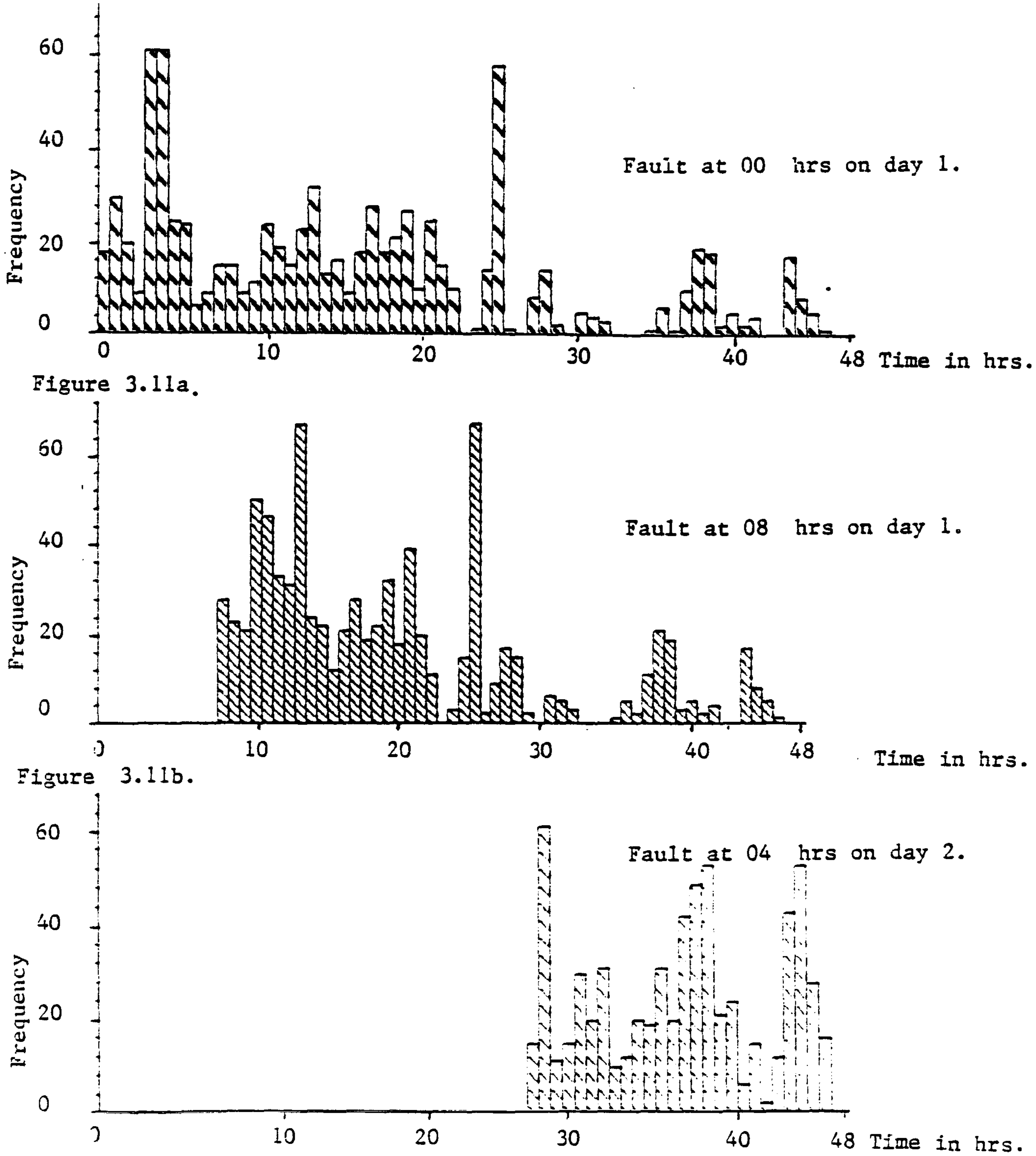

Figure 3.11a.

Figure 3.11b.

Figure 3.11c.

Figure 3.11. Error Latency Distributions for 2 Consecutive Days.

From the latency distributions it is clear that there is a finite number of inserted faults discovered in the second day. When three days of data were studied, there was incremental discovery on the third day. It was found that, typically, the first day reveals a fault with 70% confidence, the second 82%, and the third 91%. Thus, when considering the unit of time as a *day*, the confidence level of detecting a fault is not very dependent on the specific variability in workload. This is due to the fact that the workload cycle over a day reveals faults with a large degree of confidence (70%), and the subsequent fault discovery is incremental. However, the median or 50% confidence level is reached within a day, and this is highly dependent on the workload that follows the fault. The issue of the fault-miss percentage is discussed in detail in the following section.

3.5.3. Latency and hazard

These results suggest that a steady rise in workload sweeps the errors out (higher error discovery) after which few, if any, remain to be discovered (low error discovery). An increase in workload causes a temporary increase in the observed error rate. The error rate drops again after the errors have been discovered. In Figure 3.9., this phenomenon is observable with the steady decline in the number of errors discovered after a large initial discovery. It is of value, therefore, to explicitly determine the change in failure rate that results from the discovery of latent errors by workload changes (in this case the memory access).

The *hazard* or *failure rate*, *h(t)*, is a measure of the *instantaneous speed* of failure. If *F(t)* is the failure distribution function, *f(t)* the failure density function, and *R(t)* = *1* - *F(t)*, the reliability. Then, the hazard *h(t)* is defined to be:

$$h(t) = \lim_{x \to 0} \frac{1}{x} \frac{F(t+x) - F(t)}{R(t)}$$

$$h(t) = \lim_{x \to 0} \frac{R(t) - R(t+x)}{xR(t)}$$

$$h(t) = \frac{f(t)}{R(t)}$$

Thus, $h(t)\Delta t$ represents the conditional probability that a component surviving to age t will fail in the interval $(t, t+\Delta t]$ [30]. For computation of failure rate from data on failure, a discrete definition is used. The discrete functions approach the continuous functions in the limit when the data become large. Thus, hazard over the interval $(t, t+\Delta t]$ is defined as the ratio of the number of failures occurring in the time interval to the *number of survivors at the begining of the time interval*, divided by the length of the time interval [31]. Thus,

$$h(t) = \frac{[n(t) - n(t+\Delta t)]/n(t)}{\Delta t}$$

Figure 3.12. shows three hazard rate plots computed from the three error latency distributions in Figure 3.11. These hazard rate plots reveal some interesting and important characteristics of error latency.

Note that the hazard rate is not constant. This clearly establishes that classical models, assuming exponential distributions to model failure rate due to error latency, are not valid in a varying workload environment. Furthermore, simplifying assumptions such as *linearly increasing* or *linearly decreasing*

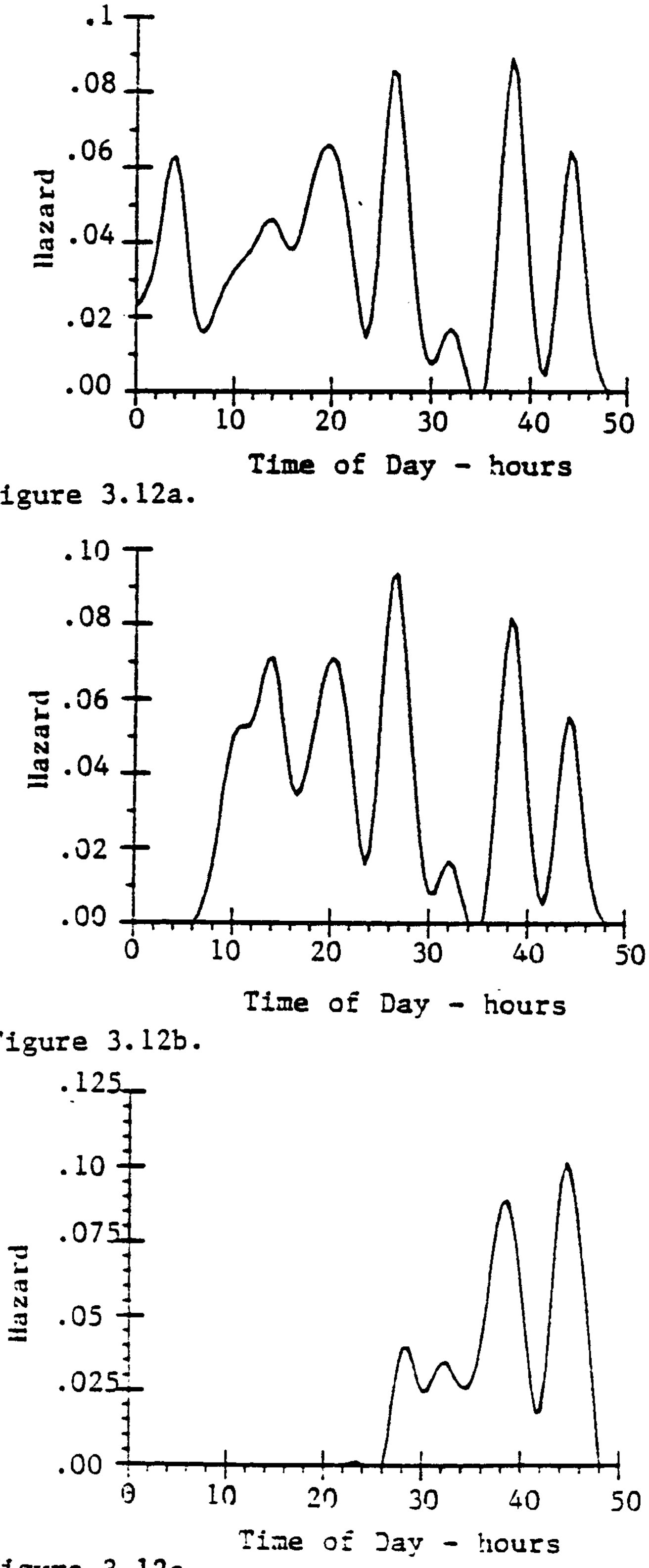

Figure 3.12a.

Figure 3.12b.

Figure 3.12c.

Figure 3.12. Hazard Rates for 2 Consecutive Days.

failure rates that are used to model failure rate are also not representative.

Notice that the change in the hazard in Figure 3.12a. on the second day is comparable to that in the first day even though around 30% of the faults remain. This is seen even when the fault occurrence time is moved across two days. The increase in failure rate due to latency is not so much a function of remaining faults but is dependent on whether or not there is a latent fault.

3.6. Validation and an Analysis of Fault-miss Percentage

The use of sampling to study error latency raises some important questions: does data not recorded between samples significantly affect the computation of error latency and its distribution ? In particular:

- Is the error latency distribution that is computed from the *sampled* data similar to the *real* error latency distribution ?

- Do the memory references not recorded between samples result in a larger *computed* percentage of undetected faults as against continuous measurement ? If yes, what is the *real* miss percentage ?

- What is the effect of the *class size* parameter (see Section 3.1) and the *sampling frequency* on the results obtained ?

This section answers these questions. The distributions are not sensitive to the sampling, however, the computed fault-miss percentage is a function of the sampling frequency and class size. This is best illustrated with a simple example. Consider the implementation of numerical integration. It has one degree of freedom, namely, the step size. In this technique, there are two degrees of freedom, namely, *class size* and *sampling frequency*. The step size does affect the accuracy of the result. However, with the step size within the right range, the computation can be both fast and accurate.

The problem of validating the method is one of estimating the *true* fault-miss percentage, given the *computed* values of fault-miss percentage. It must be noted that the percentage of the missed faults provides an estimate of the probability that a fault goes undetected. A technique for this purpose is discussed below and the technique is verified using data from a region of memory in which the fault-miss percentage is known.

3.6.1. The effect of class size and sampling factor

The first step in this analysis was to look for the possible errors in the latency distribution due to sampling. The effect of sampling on the latency distribution was studied by further sampling the data. For this purpose a sampling factor, s, which measures the decrease in sampling rate over the original sampling, was defined, i.e.,

$$s = \frac{Sampling\ frequency\ of\ collected\ data}{Sampling\ frequency\ of\ new\ sampled\ data}$$

Thus $s = 1$ for the collected data, and $s = 0$ for continuous measurement.

Error latency distributions were then generated for a range of sampling Rates. The distributions did not differ significantly as s increased. This shows the insensitivity of the error latency distribution to the sampling factor used. However, the *computed* fault-miss percentage of undetected faults varied with the sampling factor (s). Recall that the *computed* fault-miss percentage is defined as the percentage of inserted faults that remain undetected during an observation period. A similar dependence was found between the *computed* fault-miss percentage and class size.

3.6.2. Fault-miss percentage

The *computed* miss percentage, during the generation of an error latency distribution, is a function of the sampling factor (s) and the class size (c), i.e., together they form a 3-dimensional surface. Figure 3.13. and Figure 3.14. show the variations in 2 dimensions for a 30 K byte region. Figure 3.13. shows miss percentage (m) versus c for three different values of s. The computed m increases with decreasing c. The curves plotted for different values of s show that m decreases with decreasing values of s. Thus the curve for continuous measurement (corresponding to to $s = 0$) will be below the lowest curve. This curve is estimated in order to determine the real m.

Figure 3.14. shows m versus s for three different values of c. The computed m decreases with decreasing s. The curves plotted for different values of c, however, show an increase in m with decreasing c. Recall that for the measured system, the real class size is 4 bytes. The curve for this real value of c is above the highest curve and is also estimated in order to determine the real m.

The real miss percentage was determined by fitting a multiple regression model to these data and substituting for $c = 4$ and $s = 0$ in the regression model. This, of course, requires backward extrapolation of the regression plane. Following this technique, the real miss percentage for the 30 K byte region was determined to be 38.97% with an r-square of 0.87 and an F value of 194.

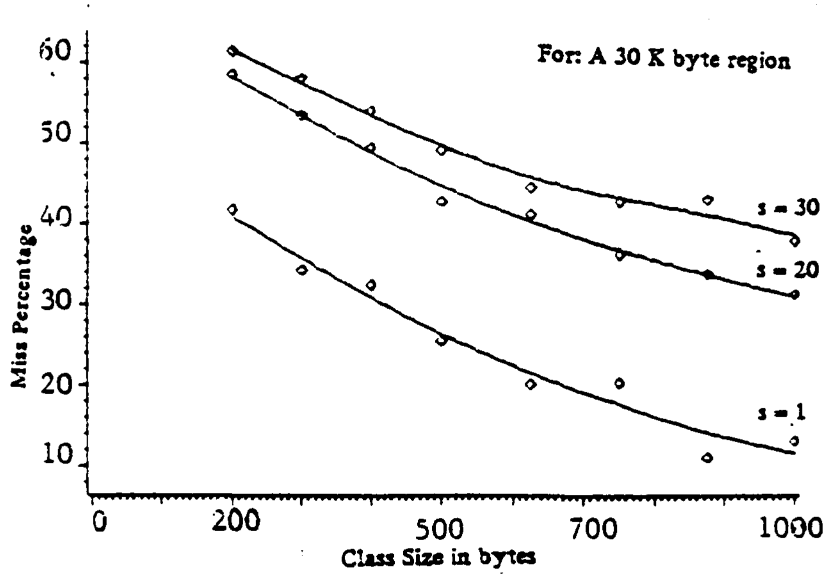

Figure 3.13. Miss Percentage Versus Class Size.

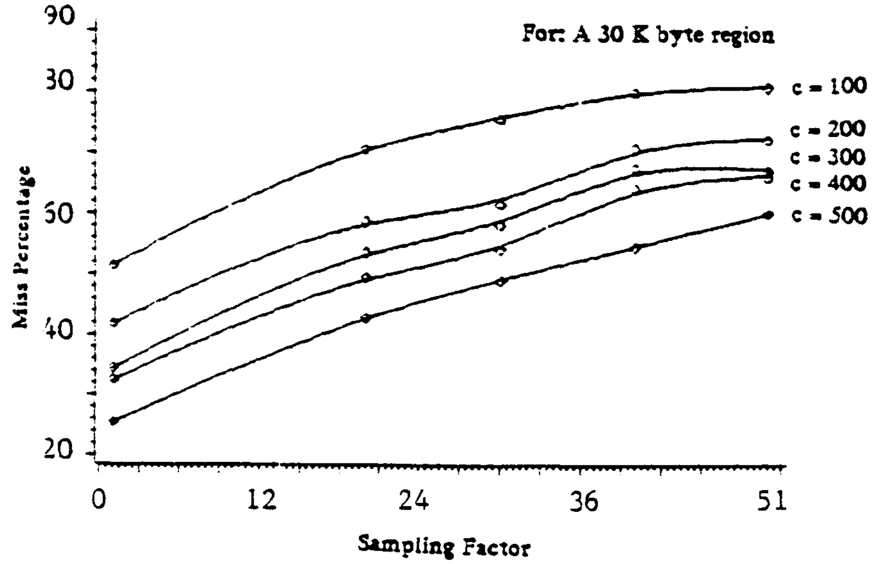

Figure 3.14. Miss Percentage Versus Sampling Factor.

Although such extrapolation is a commonly used technique, questions may be raised about its validity. For this purpose, further analysis was performed, which is discussed in the following section.

3.6.3. Verifying the miss percentage estimation

The key to verifying the extrapplation is to show that the estimated miss percentage is indeed correct. This is possible since in the data there exists a region of memory where the real miss percentage is known. Data from this region (which contains the kernel of the operating system), are used in verification. The region has very high usage and complete representation in the sampled data. Access to this region is exhaustive; hence, it has a zero *m* during a 24 hour period.

The data from this region were truncated in time, to decrease the period of observation, thereby increasing the *m*. It was then further sampled to emulate higher sampling factors. Analysis, similar to that in Section 5.1, was then performed to study the variation in *m* as a function of *c* and *s*. This analysis showed relationships among *m*, *c* and *s* similar to that observed in other regions, i.e., a plane. Figure 3.15. and Figure 3.16. show these variations in 2-dimensions. A regression model was then fitted to this and the *m* determined. The miss percentage obtained by extrapolating the regression plane was compared with the known miss percentage for this region. The real miss percentage is 0.06 and the predicted miss percentage is 0.09, which is close (with a regression coefficient of 0.91). This proves the validity of the technique used to predict miss percentage. In addition, it also shows that the fault-miss

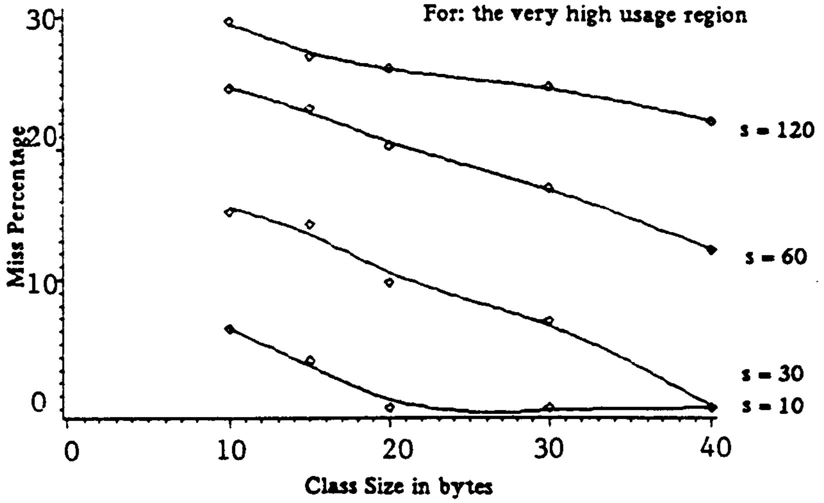

Figure 3.15. Verification: Miss Percentage Versus Class Size.

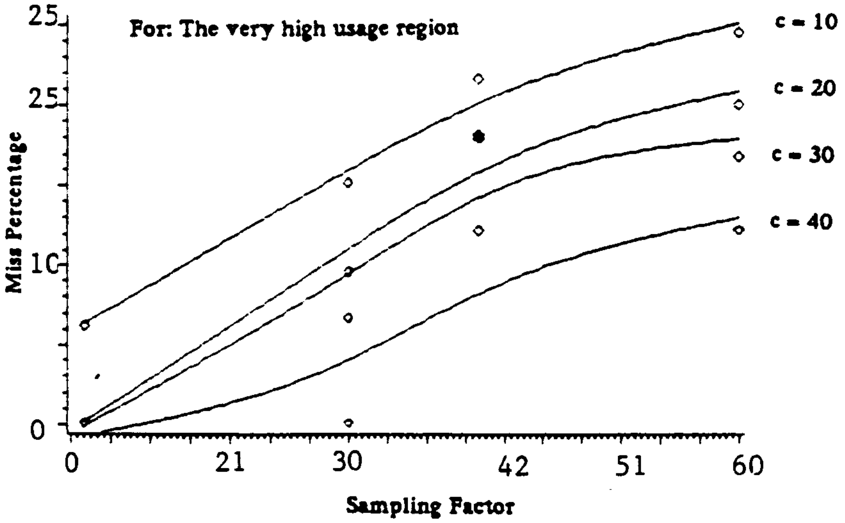

Figure 3.16. Verification: Miss Percentage Versus Sampling Factor.

percentage can vary significantly from one region of memory to another. depending on the level of usage.

It is important to note that the real miss percentage varies significantly depending on the activity in the region. Thus, different regions of memory can have widely varying miss percentages. Figure 3.17. illustrates this by comparing the miss percentage versus class size curves from two different regions of memory. Region A is the same as used in the earlier figures and is 30 K bytes in size, and Region B is 200 K bytes. From Figure 3.17. it is clear that region B has a higher miss percentage than region A. These figures are generated from data when the system was observed for 24 hours. It is to be noted that the miss percentage will also vary depending on the period of observation of the system. This is evident from the latency distributions of multiple days, where there is a small but significant discovery during the second day (Figure 3.11.). These issues illustrate that the miss percentage in a system has a large variability between regions: from 9% to 80% during a 24 hour period. Hence, using an average value does not well reflect its variability. It can only be expressed with reference to a specific region of memory and a period of time that the system is observed.

In summary it has been shown that:

(1) the error latency distribution is insensitive to the sampling technique used for measurement.

(2) the *computed* fault-miss percentage, during the generation of an error latency distribution, is a function of the sampling factor and class size.

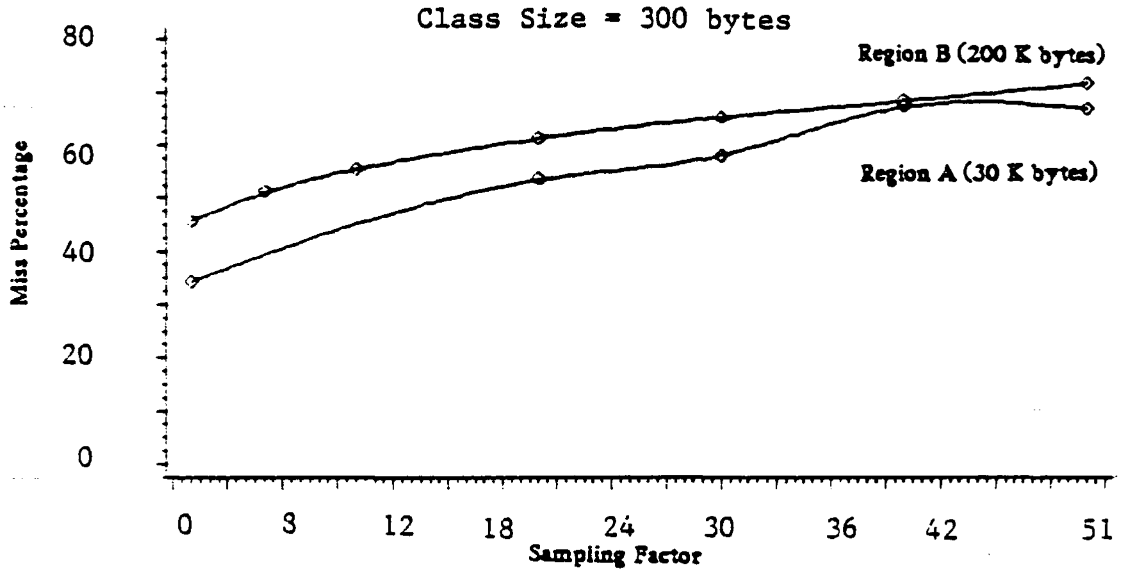

Figure 3.17. Variation in Miss Percentage Between Regions.

(3) the *real* fault-miss percentage is estimated using a multiple regression model. The technique is shown to be valid using data for which the miss percentage is known.

(4) the fault-miss percentage can only be expressed with reference to a specific region of memory and for a finite period of time.

3.7. Summary

This chapter illustrates a practical approach to the study of error latency under real workload. The determination of error latency is an important and unsolved issue in fault tolerant computing with significant implications in both reliability prediction and testing. The method is based on sampled data of the physical memory activity gathered by hardware instrumentation on a VAX 11/780 during the normal workload cycle of the installation. The data collected are then used to reconstruct the error discovery process in memory under different workload conditions. The use of sampling is validated by an analysis of the sampling factor, the class size, and the *computed* fault miss percentage. A regression based projection is used to determine the *real* fault miss percentage. This is verified using data for which the miss percentage is known. The analysis and its verification substantiate the overall approach of using sampling to reconstruct the error discovery process.

The results provide general guidelines for understanding latency behavior. The study finds that the mean error latency, in the unpaged memory containing the operating system, varies by a factor of 10 to 1 (in hours) between the low and high workloads within a day. The *hazard rate*, computed from the error latency distribution, clearly shows that the *observed* failure rate increases during higher workloads. Analysis using consecutive days of data shows that a fault is typically discovered the same day with 70% confidence, 82% confidence within the next day and 91% confidence within the third, i.e., there is a small but significant fault discovery in the second day and third day. This method in addition to determining error latency also provides a means to study the fault-miss probability. The fault miss percentage is seen to vary widely between regions of memory depending on the activity, i.e., workload, and can only be expressed with reference to a specific region of memory and a finite observation period. As with any statistical analysis, caution should be exercised in extrapolating the absolute numbers obtained in this study to other non-similar systems. However, the development of workload based reliability models, based on the general characteristics of the latency distribution found here, is an area of future study.

CHAPTER 4

FAULT LATENCY

4.1. Introduction

. The study of latency is an important issue in fault tolerant computing with far-reaching implications to both reliability measurement and evaluation. *Fault latency* is the time between the physical occurrence of a fault and its corruption of data, causing an error [3]. The difficulty with fault latency measurement is that the time of fault occurrence and the exact moment of error generation are unknown. The detection of a failure is the only record of the errors caused by a fault. Thus, faults that do not cause a failure are completely missed. The only feasible way to quantify latency is through an experimental setup, wherein the time of fault is controlled, and the error generation time is observable.

This chapter describes an experiment to accurately study the fault latency in the memory subsystem. This is the first attempt to measure *fault latency* in the *memory* with a real workload on the machine. The experiment employs real memory data from a VAX 11/780 at the University of Illinois. Fault latency distributions are generated for *stuck-at-0* (s-a-0) and *stuck-at-1* (s-a-1) permanent fault models. Results show that the mean fault latency of a s-a-0 fault is nearly 5 times that of the s-a-1 fault. Large variations in fault latency are found for different regions in memory. An analysis of variance model to quantify the effect of various workload measures on the evaluated latency is also given.

There have been a number of studies on latency; however, they have almost always measured *error latency* or the sum of both *fault* and *error latency* [6, 7, 5]. An evaluation of these techniques is given in [11]. Considerable confusion in terminology regarding *fault*, *error* and *failure* has occurred in the literature. In this thesis the definitions for *fault*, *error* and *failure* are stated as proposed by the IFIP WG 10.4 [3]. Almost all the studies so far have used specific programs or fault injection on special purpose machines. In [12] the measurement of *error latency* under a real workload in the unpaged section of the operating system is described. The first significant attempt at determining *fault latency* is found in [11]. The authors use an *indirect* technique to estimate fault latency at the pins of the chips in the *CPU* of the Fault Tolerant Multiprocessor (FTMP). More discussion on this is presented in Section 4.6.

4.2. Computation of Fault Latency

This section describes the algorithm used to calculate fault latency. Subsequent estimation of error latency based on the calculated fault latency is also described. The computation is best described by following the calculations with respect to a single bit position in a word that is chosen to contain a fault. Consider a bit position *b* of a word *w*. The value of *b* changes between 0 and 1 as a function of time. Figure 4.1. shows the contents of *b* as it changes in time. The times of change are indicated as t_1, t_2, etc. However, although the bit *b* might not change, the word *w* could have changed without affecting bit *b*. The times of change in the words are T_1, T_2, etc. A change in the bit b implies a change in the word w but the converse need not be true.

4.2.1. Fault latency calculation

With reference to Figure 4.1., consider two fault times $t_{f\,1}$ and $t_{f\,2}$. Let F_1 be a s-a-0 and F_3 be a s-a-1 fault at time $t_{f\,1}$. Let F_2 be a s-a-0 and F_4 be a s-a-1 fault at time $t_{f\,2}$. Note that in the description, a s-a-0 and a s-a-1 fault are inserted at the same time to illustrate the computation. In practice, however, only one of the faults can occur on a memory location at any given instant. Fault F_1 occurs at $t_{f\,1}$ during which time bit b is 0 and hence the fault is latent. At t_3 the bit b is written with a 1. The fault F_1 causes bit b to be stuck at 0 and the fault becomes active. Therefore, the fault latency associated with the fault F_1, namely, L_{s-a-0}^{f} is

$$L_{s-a-0}^{f} = t_3 - t_{f\,1}$$

A read performed on the word w any time after t_3 will be detected by the ECC as an error. In the figure the fault F_3 is a s-a-1 fault occurring at the same time as F_1, i.e., $t_{f\,1}$. In this case, however, the value of the bit b is 0, and the fault is active as soon as it occurs. Hence, the fault latency is

$$L_{s-a-1}^{f} = 0$$

Similarly, the fault latences for faults F_2 and F_4 are clearly,

$$L_{s-a-0}^{f} = 0$$

and

$$L_{s-a-1}^{f} = t_2 - t_{f\,2}$$

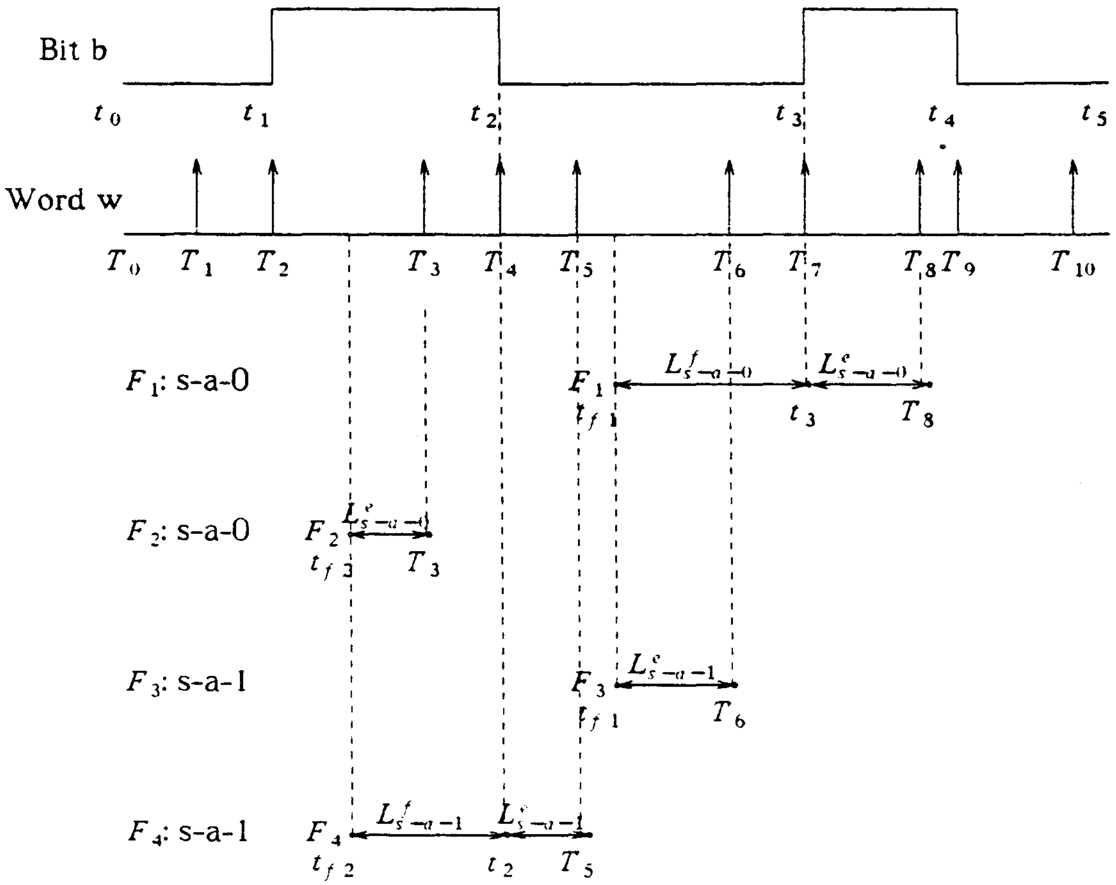

Figure 4.1. Latency Calculation.

The above discussion referred to a single fault (either s-a-0 or s-a-1). A distribution of fault latency for a given fault occurrence time is generated by inserting a large number of faults (approximately 1000) into randomly chosen bit positions of randomly chosen words. The resulting latences, taken together, yield a fault latency distribution for the given fault occurrence time.

4.2.2. Estimating error latency

An error occurs when a fault becomes active. The error latency is the time from when the error occurs until a failure results or a discovery of the error is made. For the memory subsystem, a single active fault is discovered on the read following the fault to the word in memory. In the VAX 11/780, the memory write operation, called *write masked*, checks the ECC before it updates the location. Thus, the write operation will also detect the error. The data (as will be described in Section 4.3.) used for this experiment is generated by periodic memory scans, which detect changes that take place in the contents of the scanned memory locations. Although these data only contain information on when the write operations take place and not the read operations, it can still be used to *estimate* error latency.

In [12], extensive instrumentation was performed using hardware probes to observe low-level operations on the memory and I/O. From this study it was found that about 73% of the memory operations were *write masked* and the remaining *read extended*. This high percentage of writes is most likely explained by the fact that the machine has a write-through cache. Since the majority of the memory operations are *write masked*, using only the write times for error detection provides a good estimate of error latency. However, as a read operation can occur before a write operation, this estimate provides an upper bound for error latency.

Using this method to estimate the error latency it can be seen from Figure 4.1. that T_8 is the first write that takes place after fault F_1 becomes active. Hence the error latency,

$$L^e_{s-a-0} = T_8 - t_3$$

The total latency for fault F_1 is

$$L_{s-a-0} = L^f_{s-a-0} + L^e_{s-a-0}$$
$$= T_8 - t_{f\,1}$$

Similarly, the error latency for fault F_3 is

$$L^e_{s-a-1} = T_6 - t_{f\,1} .$$

and as it had a fault latency $L^f_{s-a-1} = 0$, the total latency for the fault is the same as its error latency. Again, for fault F_2 the error latency is

$$L^e_{s-a-0} = T_3 - t_{f\,2}$$

and for F_4,

$$L^e_{s-a-1} = T_5 - t_2$$

From these estimates the total latency, which is the sum of the fault and error latency, can be calculated.

Thus, to determine the fault latency one needs data on the contents of physical memory and its time of change. The collection and implementation to evaluate fault and error latency distributions are discussed in the next section.

4.3. The Experiment

The purpose of the experiment is to get data from which the fault latency can be determined as described in the earlier section. The measured system is a VAX 11/780 that runs the Unix operating system Berkeley Ver 4.2 which is used mostly for scientific computing and a variety of miscellaneous data processing activities. The VAX 11/780 system studied has 4 M byte of main memory, three 300 M byte disk drives, a tape drive, and many miscellaneous terminals and printers. During the peak hours it has about 20 to 25 interactive users who work on a wide range of applications. The primary data used in the experiment are derived from actual scans of physical memory.

4.3.1. Memory data scanner

The physical memory of the VAX at this installation is 4 M bytes. Since the size of the memory is very large, it is impractical to scan the whole memory periodically. Representative samples from different regions of memory were chosen to capture the variation in memory usage. The choice of sample size was based on engineering judgment so as to keep the data manageable, yet ensure that it well reflected the system behavior. Concepts used to determine appropriate sample sizes were similar to those discussed in [12]. Four regions of size 10 K bytes evenly spaced in the 4 M bytes of physical memory were sampled. The memory data scanner copies the contents of randomly chosen locations from the selected regions at periodic intervals.

The scanning rate was chosen from knowledge of the distributions of the lifetimes of data in the memory locations. Data were initially acquired at a high rate (< 5 sec). the rate being qualified by the number of *identical* memory content values that were generated in the consecutive scans. The distribution of the lifetimes of the data showed a mode around 30 seconds, and more than 75% of the changes in the contents of memory locations occurred after 1 minute. Hence, a 15-20 second scan interval was considered reasonable.

4.3.2. The experimental setup

Figure 4.2. shows the experimental setup and a flow of data. Concurrent simulation is performed for all the inserted faults to determine latency times and, hence, generate latency distributions. A computationally efficient scheme is used, whereby a distribution for a chosen fault occurrence time is generated in one pass over the data. This is accomplished by preprocessing the raw data, as shown in Figure 4.2., to generate two different data sets. The *bit change dataset* contains only the times of transition of the randomly chosen bit positions that contain faults. The other, *word change dataset*, contains the times of changes in the words that contain faults. The primary reason for this separation is that a word in memory can change in value without affecting the value of a particular bit in it. Concurrent simulation of all faults can now be performed for a given fault occurrence time by one selective merge of the two data sets. Distributions for different fault occurrence times use the same data sets but use another pass through the merge.

Workload and performance data are also gathered on the machine during the memory scans. These data are used to merge with the estimated total latency to generate a workload-latency model.

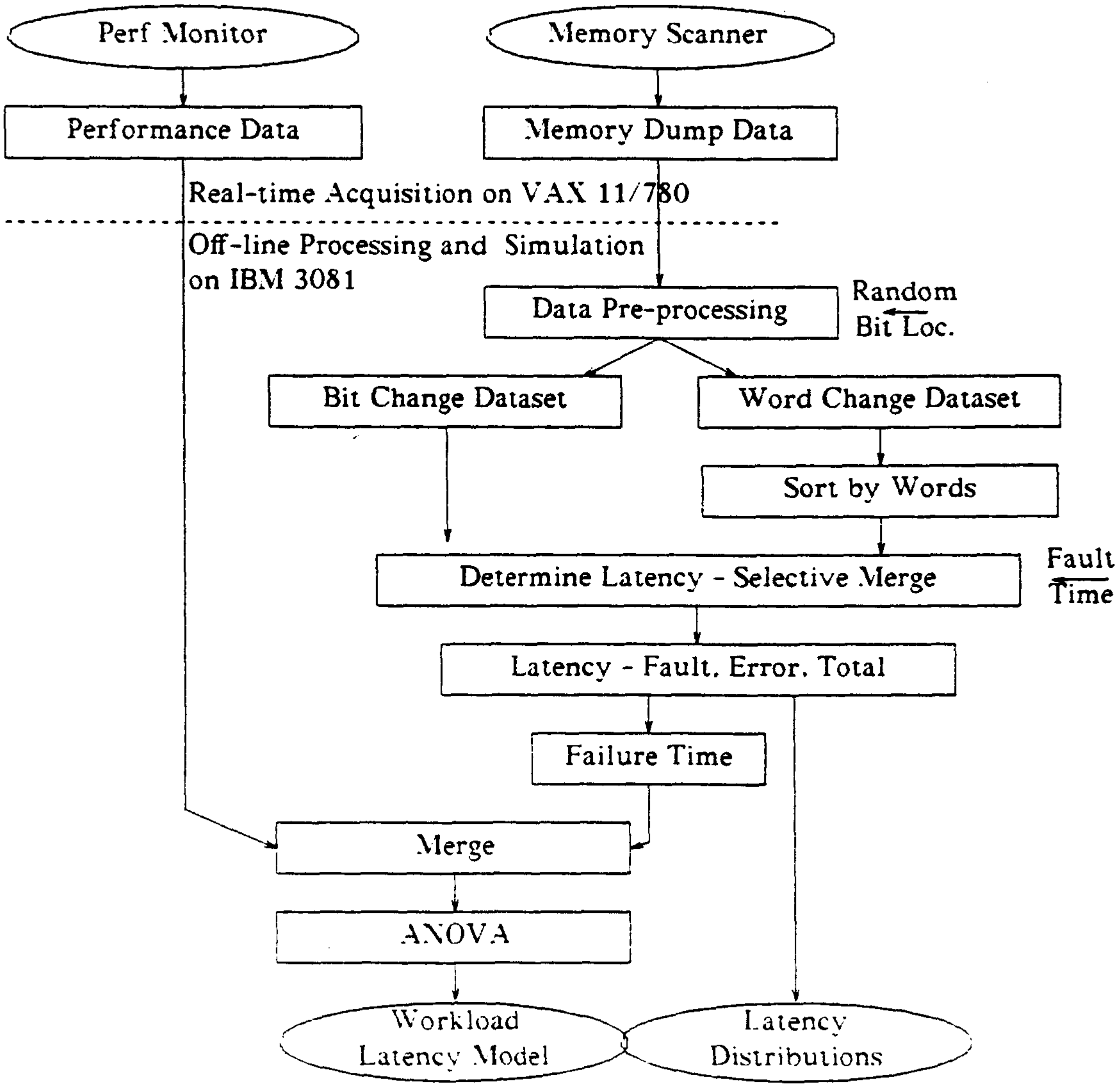

Figure 4.2. Acquisition and Processing of Data.

4.4. Latency Distributions

4.4.1. S-A-0 and S-A-1 distributions

Latency distributions for both s-a-0 and s-a-1 faults are computed (as explained in Section 3) by inserting a large number of faults at a specified fault occurrence time. A family of such distributions is generated for different fault occurrence times for different regions of memory. Since the salient characteristics of the distributions repeat themselves among the different regions of memory and repetitions of the experiment, only the distribution for a representative fault occurrence time is discussed here.

Figure 4.3. and Figure 4.4. show the latency distributions for a s-a-0 and a s-a-1 fault, respectively. Figure 4.3a. shows the **fault latency distribution** for a s-a-0 fault at 9:30 a.m. when there is a medium-to-high workload at this installation. The vertical axis is the latency midpoint of the histogram and the horizontal axis the frequency. Beside each horizontal bar the frequency and its percent contribution are shown. A total of 960 faults was inserted to generate the distribution. Figure 4.3b. shows the estimated **error latency distribution** and Figure 4.3c. the estimated **total latency distribution**. Figure 4.4. similarly shows the corresponding latency distribution for a s-a-1 fault, at the same time.

Notice that *fault latency* for the s-a-0 fault is nearly 5 times that for the *fault latency* of the s-a-1. However, the error latency estimate of the s-a-1 fault is more than twice that of the s-a-0. The total latences of the two are comparable. An explanation for the observed results follows.

```
FREQUENCY BAR CHART           (Mean: 1:10:24 Std. 1:20:14)
MIDPOINT
Fault Latency                                         FREQ   CUM.
h:mm                                                        PERCENT
           !
0:00       !************************************** 376      39.17
0:30       !*******                                 73      46.77
1:00       !**************                         139      61.25
1:30       !***********                            105      72.19
2:00       !*****                                   52      77.60
2:30       !***********                            114      89.48
3:00       !*****                                   49      94.58
3:30       !**                                      15      96.15
4:00       !*                                       11      97.29
4:30       !                                         4      97.71
5:00       !**                                      22     100.00
              -----+----+----+----+----+----+----+---
                   50  100  150  200  250  300  350
```

Figure 4.3a. DISTRIBUTION OF FAULT LATENCY - FOR S-A-0 FAULT.

```
FREQUENCY BAR CHART  (Mean: 0:20:40 Std. 0:31:14)
MIDPOINT
Error Latency                         FREQ     CUM.
h:mm                                          PERCENT
          !
0:00      !************************** 513     53.44
0:15      !***********                220     76.35
0:30      !*                           28     79.27
0:45      !****                        82     87.81
1:00      !                             2     88.02
1:15      !***                         65     94.79
1:30      !                             0     94.79
1:45      !**                          41     99.06
2:00      !                             9    100.00
          -----+----+----+----+----+-
              100  200  300  400  500
```

Figure 4.3b. DISTRIBUTION OF ERROR LATENCY ESTIMATE - FOR S-A-0 FAULT.

```
            FREQUENCY BAR CHART  (Mean: 1:31:04  Std. 1:16:55)
MIDPOINT
Latency                                             FREQ     CUM.
h:mm                                                        PERCENT
0:00       !*******************                     142      14.79
0:30       !****************                        120      27.29
1:00       !************************************    268      55.21
1:30       !************                             87      64.27
2:00       !*********                                66      71.15
2:30       !*******************                     144      86.15
3:00       !*********                                68      93.23
3:30       !***                                      19      95.21
4:00       !***                                      20      97.29
4:30       !                                          2      97.50
5:00       !***                                      24     100.00
           ----+---+---+---+---+---+---+---+---+
              30  60  90 120 150 180 210 240 270
```

Figure 4.3c. DISTRIBUTION OF TOTAL LATENCY ESTIMATE - FOR S-A-0 FAULT.

Figure 4.3. S-A-0 Latency Distributions

```
         FREQUENCY BAR CHART   (Mean: 0:14:39 Std. 0:31:55)
MIDPOINT
Fault Latency                                      FREQ    CUM.
h:mm                                                      PERCENT
          !
0:00      !************************************    725     75.52
0:15      !****                                     72     83.02
0:30      !                                          0     83.02
0:45      !**                                       34     86.56
1:00      !                                          4     86.98
1:15      !****                                     73     94.58
1:30      !***                                      52    100.00
          -----+----+----+----+----+----+----+-
              100  200  300  400  500  600  700
```

Figure 4.4a. DISTRIBUTION OF FAULT LATENCY - FOR S-A-1 FAULT.

```
         FREQUENCY BAR CHART  ( Mean: 0:45:25  Std. 0:47:53)
MIDPOINT
Error Latency                                            FREQ     CUM.
h:mm                                                             PERCENT
          !
0:00      !*****************************************     407      42.40
0:30      !************                                  116      54.48
1:00      !*********************                         213      76.67
1:30      !*********                                      87      85.73
2:00      !******                                         55      91.46
2:30      !********                                       82     100.00
3:00      !                                                0     100.00
          -----+----+----+----+----+----+----+----+-
              50  100  150  200  250  300  350  400
```

Figure 4.4b. DISTRIBUTION OF ERROR LATENCY ESTIMATE - FOR S-A-1 FAULT.

```
FREQUENCY BAR CHART                    (Mean: 1:00:22  Std. 0:47:35)
MIDPOINT
Latency                                                  FREQ     CUM.
h:mm                                                             PERCENT
          !
0:00      !*************************************         276      28.75
0:30      !***********                                    84      37.50
1:00      !*****************************************     306      69.38
1:30      !**************                                106      80.42
2:00      !*************                                  96      90.42
2:30      !************                                   92     100.00
3:00      !                                                0     100.00
          ----+---+---+---+---+---+---+---+---+---+-
             30  60  90 120 150 180 210 240 270 300
```

Figure 4.4c. DISTRIBUTION OF TOTAL LATENCY ESTIMATE - FOR S-A-1 FAULT.

Figure 4.4. S-A-1 Latency Distributions

These investigations suggest that the difference between s-a-0 and s-a-1 fault latences is due to the unequal lifetimes of 1's and 0's in the system. This difference in the lifetimes is most likely due to the way memory is allocated and released by programs. It is conjectured that often block storage allocations may result in clearing large sections of memory to zero, e.g., array initialization. The program, however, may use only a fraction of the allocated storage resulting in a large number of zeros in the memory. Discussions with system programmers have suggested that this phenomenon can occur in both user programs and system utilities. This may be true in non-Unix operating systems as well.

The results obtained for fault and error latency are now compared. Intuition leads one to believe that in general fault latency should be larger than the error latency, because updates to a word need not necessarily change the faulty bit. However, a (single) active fault is always discovered by the next access or update to the word. It can be shown that this intuition would be true, provided that the probability of a fault being inactive, i.e., the fault is latent were the same for both s-a-0 and s-a-1 faults. It is found that the above intuition is true for the s-a-0 fault but not for the s-a-1 fault. This difference is attributed to the fact that the average lifetime of a 0 is much longer than that of a 1. As a consequence of the difference in lifetimes, the s-a-0 fault remains latent with a probability of approximately 0.7 (the probability for a s-a-1 is about 0.3).

So far the analysis pertains to the distributions in a region of memory. It is found that the mean fault latency and error latency estimates vary considerably from one region of memory to another. Typical variations in the mean fault latency for a s-a-0 fault can range from 9 minutes to 50 minutes and for a s-a-1 fault from 8 seconds to 6 minutes. Although the mean latences have large variations, the distributions from different regions are similar. This variation in the means is attributed to the existence of different activity spots in the memory. Thus it is clear that a single estimate of latency is not adequate. The variation of latency with activity, caused by the workload, is analyzed and quantified in the next section.

4.4.2. Workload-latency model

Workload-failure models generated in [32, 2] relate failure rates to workload. It is believed that an important component of the workload-failure relationship is due to error latency. Since the time of error occurrence was not known in the above studies, an explicit workload-latency model could only be surmised.

To investigate the workload-latency dependency, latency due to faults injected under various workload conditions is determined using the method described earlier. The mean latency under different workloads is analyzed using an analysis of variance (ANOVA) [33]. The ANOVA analysis can be used to estimate the relative influence of different sources of variation on the values of a performance index. Thus, in this case the relative influence of various workload measures on latency is estimated. Workload data are gathered by running a performance monitor on a machine. This performance monitor records the average value of a number of high level performance parameters every 30 seconds. Table 4.1 shows the various performance measures that were recorded and used in the ANOVA analysis.

The ANOVA analysis reveals that the workload measures, namely, *active virtual memory* (AVM), *user CPU* (CPUUS), and *page reclaims* (PRE) had a large main effect. More than half of the contribution due to the interaction terms were due to the interaction between AVM and PRE. Figure 4.5. is a pie chart that shows the different workload parameters and their contributions to the model. Figure 4.6. contains a time-of-day plot for the three workload measures that had a large main effect. The other workload terms that also influenced the variation are *system CPU* (CPUSY) and the *context switch rate* (INTCS). The resulting linear model had a R-Square of 0.84 for latency. This analysis was done for a workload range that can be termed as medium-low to high and which corresponds to a CPU utilization of above 25 percent. The very low workload range has been specifically excluded since activity in that workload range tends to be very low and needs to be independently studied.

4.5. Discussion and Significance of Results

The scope and implications of fault latency in the memory go far beyond the memory subsystem. In [26] it is shown that the largest number of faults occur in the memory, and, in addition, it has been shown that a large number of the *CPU* errors originate in the memory. Thus, the importance of fault latency in the memory cannot be over emphasized.

In [11] the fault latency of CPU pin level faults is studied through fault injection and error detection. It is not possible to compare the two results since

TABLE 4.1. WORKLOAD PARAMETERS RECORDED BY THE PERFORMANCE MONITOR.

Mnemonic	Function	Description	Units
CPUUS	CPU	User time	percent
CPUSY	CPU	System time	percent
CPUID	CPU	Idle time	percent
MAVM	MEM	Active virtual pages	number
MFRE	MEM	Size of free list	number
PGRE	PAGE	Page reclaims	per sec
PGPI	PAGE	Pages paged in	per sec
PGPO	PAGE	Pages paged out	per sec
PGFR	PAGE	Pages freed	per sec
ININ	FAULTS	Device interupts	per sec
INSY	FAULTS	System calls	per sec
INCS	FAULTS	Context switch	per sec
PRR	PROCS	Processes in run queue	number
PRB	PROCS	Processes blocked (I/O, etc.)	number
PRW	PROCS	Processes runnable but swapped	number

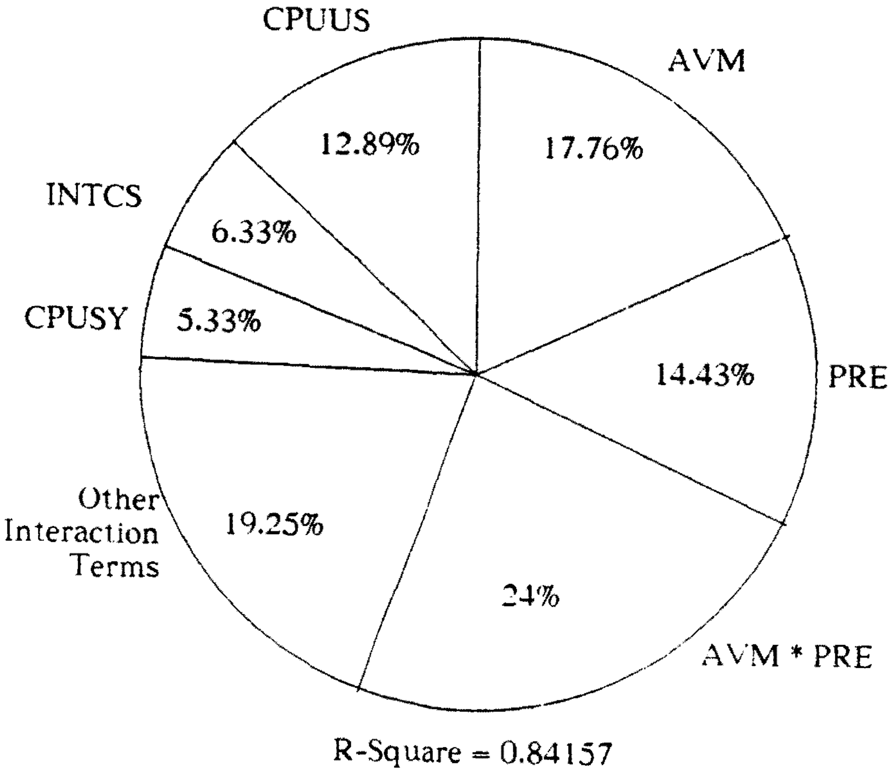

Figure 4.5. Pie Chart Showing Workload-latency Relationship.

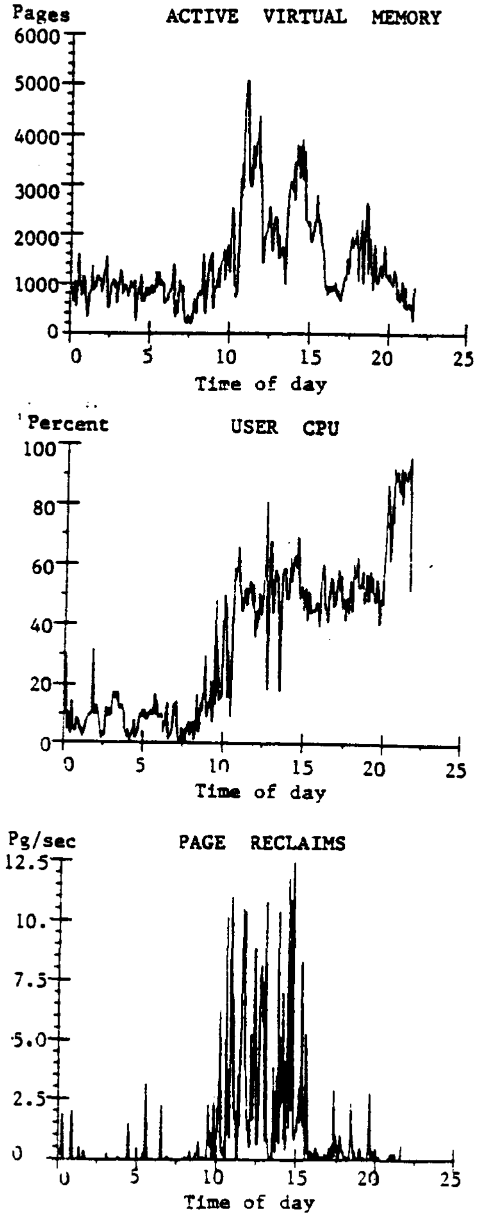

Figure 4.6. Plot of Three Workload Measures.

they relate to two entirely different system components. However, a comparison of the methodology is instructive. The method in [11] is somewhat indirect since the exact moment when the fault becomes active is not known. Thus, this provides only an upper bound for fault latency. Since, in this technique, the exact moment when the fault becomes active is known the fault latency computation is accurate. This is primarily due to the nature of the memory data that were collected.

The measurements on fault and error latency distributions of s-a-0 and s-a-1 faults show:

(1) The fault latency of a s-a-0 is much larger than that for s-a-1. The ratio of the two is approximately 5:1

(2) The estimated error latency for a s-a-0 is smaller than that for s-a-1.

(3) The differences in (1) and (2) are attributed to the difference in lifetimes of zeros and ones in the memory.

4.6. Summary

This chapter has demonstrated a technique to accurately determine fault latency under real workload conditions in the memory subsystem. This technique used real memory scan data from a VAX 11/780 running Unix. Fault latency distributions were generated for s-a-0 and s-a-1 permanent fault models. The mean fault latency of a s-a-0 fault is nearly 5 times that of s-a-1 fault. It is likely that the above phenomenon is characteristic of other systems as well. Large fault latences are a reason for concern since they can result in multiple errors. From the data, the s-a-0 fault is clearly a cause for greater concern. An estimate of the error latency was also provided and a workload-latency model developed using ANOVA. Workload and latency have a linear relationship for a medium-to-high workload range.

CHAPTER 5

ERROR LATENCY IN THE MICROCONTROL STORE

5.1. Introduction

In many Central Processing Unit (CPU) designs, the area taken up by the microcontrol store is significant; therefore, its reliability is important. The VAX 11/780 is a microprogrammed machine; hence, the microcontrol store is a key element in the operation of the CPU. Errors in the microcontrol store can cause catastrophic failures.

The execution of an instruction in the 11/780 requires a sequence of microoperations. The sequence of microoperations is determined by the microprogram contained in the microcontrol store. The microcontrol store in the 11/780 consists of programmable read only storage (PCS) and a writable diagnostic control store (WDCS). The microword in the 11/780 is 96 bits wide with additional 3 parity bits. Each microword is comprised of several fields which control specific functions in the processor. A detailed description of the fields and the format can be found in [34]. The PCS provides storage for 4K microwords and the WDCS has a writable storage for 1K words. Thus the micro address is 13 bits wide. The WDCS is mainly used for modifications to the original microprogram and for user-written microcode. The microcode is loaded into the WDCS during system startup and, for the most part during the running of the machine, the microcode is read-only. Thus only an *active fault*, i.e. an *error*, in the microcode can cause failure. Although the parity will detect any single errors in each 32 bit field of the 96 bit word, there is no recovery from these errors. In this chapter the error latency associated with such faults in the microcontrol store will be measured and analyzed.

5.2. Instrumentation

The type of data that are needed for measuring error latency in the microcontrol store is similar to the data used for the error latency measurements in the memory (Chapter 3). Essentially, data on the access and use of the different microwords are required to measure error latency. These data are generally available from the microsequencer in the machine or the instruction decode logic. The primary function of the microsequencer is to provide the address of a word in the control store. Description of the 11/780 microsequencer that is necessary for the instrumentation is presented in the following section. Full details appear in [34].

5.2.1. The microsequencer

The microsequencer controls the entry to the microprogram during the normal program flow and also during special conditions such as powerup, microtraps, stalls, console operations, and microword patches. The address of the next microword to be executed is broadcast on the microprogram counter bus (UPC bus) to the PCS and the WDCS. In the case of a decision point branch, the lower-order bits of the microaddress are generated by the instruction decode logic. The most significant bit (bit 12) of the microaddress determines which control store is addressed. When bit 12 of the microaddress is 0, the PCS is addressed, and when it is 1, the WDCS is addressed.

The source of the microword address is dependent on the mode of the microsequencer operation. These modes are dependent on conditions that are generated in other parts of the CPU. The typical conditions are power up/down initialization, maintenance, cache stall, microtrap, microword patch, and normal operation. During the normal mode of operation, the next microaddress is selected from the Jump and Branch Enable field of the current microword. Under a microword patch, the microsequencer generates an address for the WDCS. This is done by a lookup table through a Field Programmable Logic Array (FPLA) which contains PCS addresses that require changes to the corresponding WDCS addresses containing the new microcode. This causes a *no-op* cycle to fetch the new microaddress. When a microtrap occurs, the microsequencer generates specific vector addresses which contain trap handling conditions in the CPU. This also causes a *no-op* cycle in which the new microaddress can be formed. The utrap causes microword registers to be cleared and an abort cycle to be generated. A cache stall mode is initiated when a cache miss occurs. In this mode the execution of the next microinstruction is temporarily prevented. Under a cache stall mode the microprogram is in a *no-op* state, and this can continue for several cycles until the stall condition is negated. In the maintenance mode the console can control various operations of the microsequencer. During power up/down initialization the microsequencer is forced to a constant microtrap vector.

5.2.2. Data acquisition

The addresses that are generated by the microsequencer are visible on the UPC bus. These addresses are accessed by probes placed on the backplane of the microsequencer card. From the discussion above, it can be seen that the microaddress is not valid on every processor cycle due to the different modes of operation. Hence, the *stall* and *abort cycle* signals are used to qualify the cycles when the data are valid.

The probes on the DAS that are used to acquire the data have an acquisition memory that is 1024 words deep. Thus, each sample will contain 1024 microaddresses. Similar to the data acquisition in Chapter 3, the acquisition can be in either a regular or a compressed mode. In the regular mode, each CPU cycle is stored; thus, the data contains cycles which include cache stalls and abort cycles. In the compressed mode, the stall and abort cycles are ignored. The regular mode acquisition is particularly useful for performance measurements since cache stall can be studied. For analysis that only needs the microaddress trace, the compressed mode is preferred.

The DAS is connected to the host machine (a Gould 9050) via an RS232-C port. This facilitates programming the DAS through a GPIB protocol and provides up-loading of the acquisition memory to store on tape. In this setup, the DAS can be periodically triggered and the system repeatedly sampled.

The VAX11/780 provides a facility called the Performance Monitor Enable (PME). The PME is a signal in hardware that can be seen on the backplane and is also a bit in one of the registers in the process control block. If the PME bit is set. then the hardware PME signal is set whenever the particular process executes. This enables monitoring of a specific process in a mix. This instrumentation facilitates qualifying the data acquisition as the PME bit provides a means to make microaddress or cache stall measurements on a specific process. By looking at the transition times of the PME signal, one can study the context switch times which are particularly hard to study in a simulation environment.

5.3. Measurement and Analysis

5.3.1. Microcode usage distribution

The total address space of the microcontrol store is 5K words. This is comprised of 4K of PCS and 1K of WDCS. Measurements were made during the medium workload and a mix of interactive and batch programs. In this workload the usage distribution of the microcode stabilized with around 32 to 48 acquisitions, each containing about 1000 microaddresses. This microcode usage distribution is shown in Figure 5.1. By studying the microcode usage distribution, it is clear that a small portion of the microcode accounts for a large part number of the access. This type of usage is typical for machines with large instruction sets.

5.3.2. Interaccess time

The time between access to the same microword in the control store is called the interaccess time. This interaccess time can be measured from the data provided that it is less than 1000 cycles since the acquisition buffer is 1000 words deep. Computing the interaccess time provides two useful measures.

```
                              FREQUENCY BAR CHART
MIDPOINT
MICROADDRESS                                                   PERCENT    CUM.
                                                                        PERCENT
             !
      0      !***************************************           28.27     28.27
    200      !*********************                             15.31     43.57
    400      !****                                               3.18     46.75
    600      !**************                                    10.46     57.21
    800      !********                                           5.85     63.06
   1000      !*                                                  0.71     63.77
   1200      !**                                                 1.81     65.58
   1400      !                                                   0.32     65.90
   1600      !                                                   0.34     66.24
   1800      !                                                   0.28     66.52
   2000      !                                                   0.07     66.59
   2200      !****                                               2.74     69.33
   2400      !                                                   0.19     69.52
   2600      !*                                                  0.47     69.99
   2800      !*                                                  0.77     70.76
   3000      !                                                   0.22     70.98
   3200      !*********                                          6.76     77.74
   3400      !**                                                 1.36     79.10
   3600      !**                                                 1.20     80.30
   3800      !********                                           5.91     86.21
   4000      !*                                                  0.38     86.59
   4200      !***                                                2.04     88.63
   4400      !**                                                 1.45     90.08
   4600      !                                                   0.35     90.42
   4800      !***                                                2.07     92.49
   5000      !**                                                 1.24     93.73
   5200      !*****                                              3.62     97.35
   5400      !***                                                2.33     99.68
   5600      !                                                   0.04     99.73
   5800      !                                                   0.27    100.00
   6000      !                                                   0.00    100.00
             -----+----+----+----+----+----+----+----
               1000 2000 3000 4000 5000 6000 7000

                             FREQUENCY
```

Figure 5.1. Usage Distribution for the Microcontrol Store.

First, one sees the distribution of the interaccess time. Second, the percentage of the used microaddresses that have an interaccess time that is less than 1000 cycles can be determined. This is necessary to develop a measure of confidence for the error latency distribution that is later computed.

Figure 5.2 shows the interaccess time distribution computed over a large number of acquisition samples. The number of samples required to stabilize the interaccess time distribution is comparable with the numbers needed to stabilize the usage distribution. It is found that the microwords that have on the average interaccess time less than 1000 cycles constitute over 80% of the microwords that are commonly used. This percentage is determined by comparing the microwords in the interaccess time distribution with those in the microcode usage distribution. This essentially quantifies the coverage of the error latency distribution that can be computed from these data. In summary, the error latency distribution that is generated will be limited to latencies that are a maximum of 1000 cycles which is the case 80% of the time.

5.4. Error Latency Calculation

Recall from Chapter 2 that the error latency is the time between the occurrence of an error and the consequent failure. In the case of the microcontrol store, there is only an error latency issue since the measured microcode is read only. Error latency is computed by simulating the occurrence of active faults (inverted bit) in the data and determining the time taken to cause failure. The failure will be caused on the following use of the microword.

Microword interacces time distribution :
- For 80 % of the Addresses

```
                           FREQUENCY BAR CHART

MIDPOINT
DELTIM                                                PERCENT     CUM.
                                                                  PERCENT
          !
     0    !************                                 5.31        5.31
    25    !*************                                6.00       11.32
    50    !**********                                   4.62       15.94
    75    !********                                     3.46       19.40
   100    !*************                                5.77       25.17
   125    !**********                                   4.39       29.56
   150    !************************                    10.85       40.42
   175    !**********************                      10.16       50.58
   200    !***************************                 12.24       62.82
   225    !****************                             7.39       70.21
   250    !*****************                            7.85       78.06
   275    !*****                                        2.31       80.37
   300    !***                                          1.39       81.76
   325    !**                                           0.92       82.68
   350    !***************                              6.70       89.38
   375    !**                                           0.69       90.07
   400    !***                                          1.39       91.45
   425    !***                                          1.15       92.61
   450    !*                                            0.23       92.84
   475    !                                             0.00       92.84
   500    !*                                            0.23       93.07
   525    !                                             0.00       93.07
   550    !*                                            0.23       93.30
   575    !                                             0.00       93.30
   600    !                                             0.00       93.30
   625    !                                             0.00       93.30
   650    !                                             0.00       93.30
   675    !                                             0.00       93.30
   700    !                                             0.00       93.30
   725    !***************                              6.70      100.00
          -----+----+----+----+----+--
              10   20   30   40   50
```

Figure 5.2. Microword Interaccess Time Distribution.

From the discussion in the previous section on interaccess time, it is clear that 80% of the errors have a latency which is less than 1000 cycles. Figure 5.3. illustrates the error latency calculation. Errors are inserted on randomly chosen microwords at a given instant in time. With reference to Figure 5.3., at time t errors e_1 occur on microword w_1 and e_2 on w_2. The first access and use of microword w_1 will cause a failure at t^*; hence, the latency for e_1 is l_1. If however, there is no access to a microword in the sample, as is the case of w_2, that error may be larger than 1000 cycles. This process of determining latency is repeated over a large number of samples which then provide an average behavior. The intersample time is randomly distributed, and the microwords chosen to contain a fault are drawn from a uniform distribution. Error latency is computed for the same set of randomly chosen erroneous locations over many samples. This results in generating a stable error latency distribution. The number or samples required to stabilize the distribution is comparable to that needed to stablize the usage distribution.

Figure 5.4. shows the error latency distribution. Note that there is a large mode in the 50 to 100 cycle range. There are also two other modes around 250 and 600 cycles. The mean for this distribution is 310 cycles with a standard deviation of 267. Unlike the error latency distribution in the memory which can be very large and can have a second large mode, this is skewed to have the shorter latences dominate. It is interesting to compare this with the interaccess time distribution. The interaccess time distribution has modes around 200 and 350 cycles. If the access of the microcode was uniform, which it is not, one

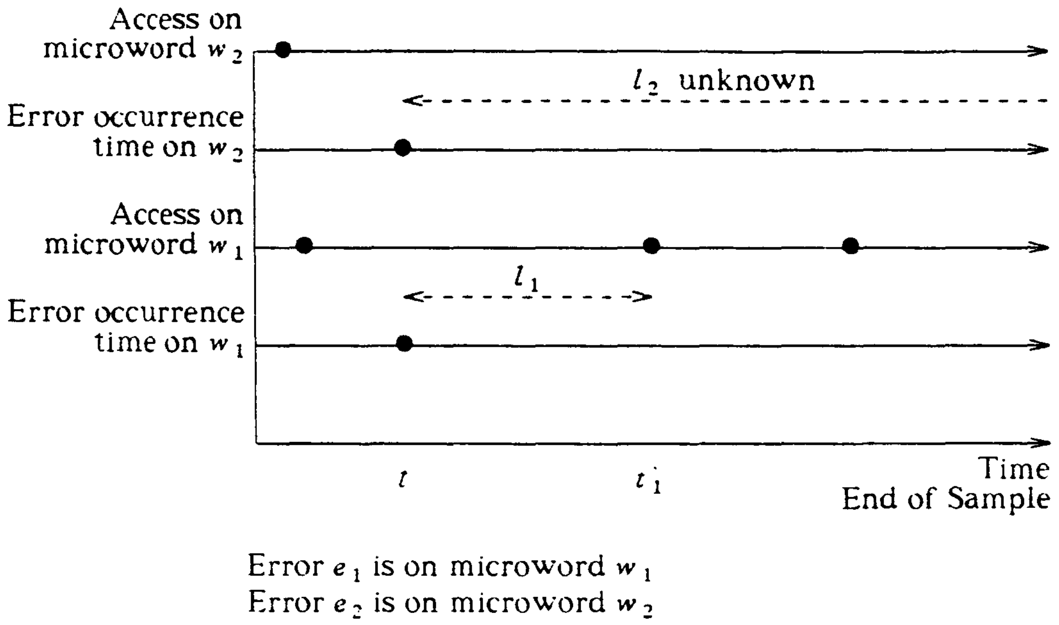
Access on microword w_2
Error occurrence time on w_2
Access on microword w_1
Error occurrence time on w_1
l_2 unknown
l_1
t
t_1
Time
End of Sample
Error e_1 is on microword w_1
Error e_2 is on microword w_2

Figure 5.3. Error Latency Time Calculation.

```
                              FREQUENCY BAR CHART     (Mean 310 Cyc; Std Dev 267)
MIDPOINT
LATENCY                                                             PERCENT    CUM.
CYCLES                                                                       PERCENT
     0    !***************************************                    10.30    10.30
    50    !****************************************************       13.74    24.04
   100    !***************************************                    10.37    34.41
   150    !******************************                              7.91    42.32
   200    !*********************                                       5.61    47.93
   250    !****************************                                7.39    55.31
   300    !*********************                                       5.67    60.99
   350    !****************                                            4.20    65.19
   400    !*****************                                           4.51    69.69
   450    !*************                                               3.58    73.27
   500    !*************                                               3.35    76.62
   550    !**********                                                  2.53    79.15
   600    !***************                                             3.91    83.06
   650    !***************                                             4.10    87.16
   700    !************                                                3.22    90.38
   750    !**********                                                  2.66    93.04
   800    !*****                                                       1.45    94.48
   850    !*******                                                     1.99    96.47
   900    !******                                                      1.55    98.02
   950    !*****                                                       1.36    99.38
  1000    !**                                                          0.62   100.00
          --------+-------+-------+-------+-------+-------+----
                 200     400     600     800    1000    1200

                              FREQUENCY
```

Figure 5.4. Error Latency Distribution for the Microcontrol Store.

would expect a one-one relationship between the distributions. Results show that this is not the case.

The [5] study on error latency in an Avionic miniprocessor using a gate level simulation and test programs had faults injected into the ALU, address processor, and micromemory. Although a direct comparison between the two is not possible, since this study is restricted to the microcontrol store, a discussion on the two results is instructive. The latency distribution reported in the McGough study showed an almost exponential behavior, with the slight non monotonicity attributed to statistical fluctuation. In this study it is found that the distribution is not exponential. The distribution has a large mode with two other smaller modes.

5.5. Summary

This chapter determines error latency in the microcontrol store of a VAX 11/780 processor. The microcontrol store is a significant part of the processor; hence, errors in the control store cause a catastrophic failure of the machine. Microaddress traces occurring during the regular workload of the machine are gathered from probes placed in the microsequencer of the processor. The latency distribution has a large mode between 50 and 100 microcycles and two additional smaller modes. It is interesting to note that the error latency distribution in the microcontrol store is not exponential as noted in a similar study performed using a gate level simulation of an avionic processor.

CHAPTER 6

CONCLUSIONS

6.1. Summary and Discussion of Results

This thesis has developed a systematic experimental approach to study fault and error latency under real workload. The determination of these latences is an important and unsolved issue in fault tolerant computing with significant implications in both reliability prediction and testing. The methodology, based on gathering relevant low level data from the machine during the normal workload cycle of the installation, was demonstrated on a VAX 11/780 system. In particular, error latency in the system region (largely unpaged) of the memory was studied by using hardware probes placed on the synchronous backplane interconnect and gathering data on physical memory access and usage. Fault latency in the user sections of the paged memory was determined using data from memory scans. This latency information, taken together with the workload data on the machine was used to develop a workload-latency relationship. Error latency in the microcontrol store is determined by using probes in the microsequencer and gathering data on the microaddress sequence executed.

Chapter 3 studies the error latency in the system region of the memory. The data collected are used to reconstruct the error discovery process in memory under different workload conditions. The use of sampling is validated by an analysis of the sampling factor, the class size, and the *computed* fault miss percentage. A regression based projection is used to determine the *real* fault miss percentage. This is verified using data for which the miss percentages are known. The analysis and its verification substantiate the overall approach of using sampling to reconstruct the error discovery process.

The results provide general guidelines for understanding latency behavior. The study finds that the mean error latency, in the unpaged memory containing the operating system, varies by a factor of 10 to 1 (in hours) between the low and high workloads within a day. The *hazard rate*, computed from the error latency distribution, clearly shows that the *observed* failure rate increases during higher workloads. Analysis using consecutive days of data shows that a fault is typically discovered the same day with 70% confidence, 82% confidence within the next day and 91% confidence within the third; i.e., there is a small but significant fault discovery in the second and third days. This method, in addition to determining error latency, provides a means to study the fault miss probability. The fault miss percentage varies widely between regions of memory depending on the activity, i.e., workload, and can only be expressed with reference to a specific region of memory and a finite observation period.

Chapter 4 demonstrates a technique to accurately determine fault latency under real workload conditions in the memory subsystem. This technique used real memory scan data from a VAX 11/780 running Unix. Fault latency distributions were generated for s-a-0 and s-a-1 permanent fault models. The mean fault latency of a s-a-0 fault is nearly 5 times that of s-a-1 fault. It is likely that the above phenomenon is characteristic of other systems as well. Large fault latences are a reason for concern since they can result in multiple errors. From the data, the s-a-0 fault is clearly a cause for greater concern. An estimate of the error latency was provided and a workload-latency model developed using ANOVA. Workload and latency have a linear relationship for a medium-to-high workload range. More experimental analysis on different machines is suggested to further understand the latency problem.

Chapter 5 determines error latency in the microcontrol store of a VAX 11/780 processor. The microcontrol store is a significant part of the processor; hence, errors in the control store cause a catastrophic failure of the machine. Microaddress traces occurring during the regular workload of the machine are gathered from probes placed in the microsequencer of the processor. It is found that the latency distribution has a large mode between 50 and 100 microcycles and two additional smaller modes. It is interesting to note that the error latency distribution in the microcontrol store is not exponential as compared with a similar study performed using a gate level simulation of an avionic processor.

As with any statistical analysis, caution should be exercised in extrapolating the absolute numbers obtained in this study to other nonsimilar systems.

6.2. Suggestions for Future Research

This thesis has extensively analyzed the problem of fault and error latency with respect to hardware faults in a single machine. With the growth of the computing environment into distributed machines and clusters of machines, the problems of latency gain even greater importance. Fault and error latency in such environments remain to be studied. Although a similar methodology may be employed, the joint collection of data and simulation of error in multiple environments provides a challenging problem for future research. Some extensions to this research are:

Specific Environment

It will be interesting to see this study performed on different computing environments. Specifically, it is likely that the behavior of batch and interactive workloads may be different. Real-time applications and other specific applications are also likely to have different latency behaviors.

CPU Study Extension

The data acquired on microcode usage have a number of different applications. In particular the microcode usage trace, taken together with the microcode of the machine, can be used for some excellent studies on fault propagation and diagnosis. Since, the fields of the microword are known the severity of a fault in the CPU can be determined by simulating faults in the CPU and exercising them with the microcode trace. This study can also lead to some very useful information on designing diagnostics for the machine.

Multiple Faults

The possibility of multiple faults occurring due to large latences is an important consequence. This issue has to be specifically studied. It will be interesting to see the effect of different fault arrival distributions given that there now exists some idea of the latency distributions.

Software failures

A large number of failures occur due to software faults. Latency of software faults in a production system needs investigation. By definition, software faults are latent; however, the concept of a software fault needs more precise definition. It is likely that the existence of an active software fault is conditional on a variety of workload conditions. This condition *may* in some sense be termed the moment of error generation. The problem of detecting software error generation is quite complex and requires further research.

REFERENCES

[1] R. K. Iyer, S. E. Butner, and E. J. McCluskey, "A Statistical Failure/Load Relationship: Results of a Multi-Computer Study," *IEEE Transactions on Computers*, vol. C-31 no. 7, pp. 697-706, 1982.

[2] X. Castillo and D. P. Siewiorek, "Workload, Performance and Reliability of Digital Computing Systems," in *Proceedings 11th International Symposium on Fault-Tolerant Computing*, Portland, Maine, pp. 84-89, 1981.

[3] J. C. Laprie, "Dependable Computing and Fault Tolerance: Concepts and Terminology," in *Proceedings 15th International Symposium on Fault-Tolerant Computing*, Ann Arbor, Michigan, pp. 2-11, 1985.

[4] J. G. McGough and F. L. Swern, "Measurement of Fault Latency in a Digital Avionic Mini Processor Part II," NASA Contractor Report 3651, 1983.

[5] J. G. McGough, F. L. Swern, and S. Bavuso, "New Results in Fault Latency Modelling," in *Proceedings IEEE EASCON Conference*, Washington, DC, 1983.

[6] J. H. Lala, "Fault Detection, Isolation and Reconfiguration in FTMP: Methods and Experimental Results," *Proceedings 5th Avionics Systems Conference*, 1983.

[7] B. Courtois, "Some Results about the Efficiency of simple Mechanisms for the Detection of Microcomputer Malfunction," in *Proceedings 9th International Symposium on Fault-Tolerant Computing*, Madison, Wisconsin, pp. 71-74, 1979.

[8] B. Courtois, "A Methodology for On-line Testing of Microprocessors," in *Proceedings 11th International Symposium on Fault-Tolerant Computing*, Portland, Maine, pp. 272-274, 1981.

[9] J. J. Shedletsky, "A Rollback Interval for Networks with an Imperfect Self-checking Property," *IEEE Transactions on Computers*, vol. C-27 no. 6, pp. 500-508, 1978.

[10] J. J. Shedletsky and E. J. McCluskey, "The Error Latency of a Fault in a Combinational Digital Circuit," in *Proceedings 5th International Symposium on Fault-Tolerant Computing*, Paris, France, pp. 210-214, 1975.

[11] K. G. Shin and Y. H. Lee, "Measurement and Application of Fault Latency," *IEEE Transactions on Computers*, vol. C-35 no. 4, pp. 370-375, 1986.

[12] R. Chillarege and R. K. Iyer, "The Effect of System Workload on Error Latency: An Experimental Study," in *ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems*, Austin, Texas, pp. 69-77, 1985.

[28] DEC, *VAX Architecture Handbook*. Digital Equipment Corporation, 1980.

[29] DEC, *KA780 Field Maintenance Print Set*. Digital Equipment Corporation, 1980.

[30] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*. Englewood Cliffs, N.J.: Prentice-Hall, 1982.

[31] M. L. Shooman, *Probabilistic Reliability: An Engineering Approach*. New York: McGraw-Hill, 1968.

[32] R. K. Iyer and D. J. Rossetti, "A Statistical Load Dependency of CPU Errors at SLAC," in *Proceedings 12th International Symposium on Fault-Tolerant Computing*, Santa Monica, California, pp. 363-372, 1982.

[33] W. Mendenhall, *Statistics for the Engineering and Computer Sciences*. California: Dellen Publishing Co., 1984.

[34] DEC, *KA780 Central Processor Technical Description*. Digital Equipment Corporation, 1979.

VITA

Ram Chillarege was born in [redacted] on [redacted] [redacted]. He received his B.Sc. degree from University of Mysore, India, in September 1977. He obtained his B.E. degree in Electrical and Electronics Engineering and his M.E. degree in Automation from the Indian Institute of Science, Bangalore, India, in 1977 and 1979, respectively. At the University of Illinois he was employed as a research assistant with the Computer Systems Group at the Coordinated Science Laboratory from 1981 to 1986.

His current research interests include measurement and performance analysis, computer architecture, high-performance systems, reliability, and VLSI systems.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

ORIGINAL PAGE IS
OF POOR QUALITY

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS
None

2a. SECURITY CLASSIFICATION AUTHORITY
NA

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE
NA

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release, distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
UILU-ENG-86-2230 (CSG-55)

5. MONITORING ORGANIZATION REPORT NUMBER(S)
NA

6a. NAME OF PERFORMING ORGANIZATION
Coordinated Science Laboratory
University of Illinois

6b. OFFICE SYMBOL *(If applicable)*
NA

7a. NAME OF MONITORING ORGANIZATION
JSEP (Joint Services Electronics Program)
NASA (National Aeronautics & Space Administratio
GRB (Graduate Research Board [U of Il)

6c. ADDRESS *(City, State and ZIP Code)*
1101 W. Springfield Ave.
Urbana, IL 61801

7b. ADDRESS *(City, State and ZIP Code)*
NASA: NASA Langley Research Center, Hampton, VA
JSEP: Office of Naval Research, 800 N. Quincy, Arlington, VA
GRB: 125 Coble Hall, 801 S. Wright, Champaign, IL

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
JSEP, NASA, GRB (see 7a)

8b. OFFICE SYMBOL *(If applicable)*
NA

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
JSEP: N00014-84-C-0149
NASA: NAG-1-613
GRB: (no i.d. number)

8c. ADDRESS *(City, State and ZIP Code)*
(see 7b)

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
N/A	N/A	N/A	N/A

11. TITLE *(Include Security Classification)* Fault and Error Latency Under Real Workload— An Experimental Study

12. PERSONAL AUTHOR(S)
Ram Chillarege

13a. TYPE OF REPORT
Technical

13b. TIME COVERED
FROM – TO –

14. DATE OF REPORT *(Yr., Mo., Day)*
August, 1986

15. PAGE COUNT

16. SUPPLEMENTARY NOTATION
NA

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Error and Fault Latency, Failure Rate, Experimental Study, Workload Effects, Analysis of Variance

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This thesis demonstrates a practical methodology for the study of fault and error latency under real workload. This is the first study that measures and quantifies the latency under real workload and fills a major gap in the current understanding of workload-failure relationships. The methodology is based on low level data gathered on a VAX 11/780 during the normal workload conditions of the installation. Fault occurrence is simulated on the data, and the error generation and discovery process is reconstructed to determine latency. The analysis proceeds to combine the low level activity data with high level machine performance data to yield a better understanding of the phenomenon. This study finds a strong relationshop between latency and workload and quantifies the relationship. The sampling and reconstruction techniques used are also validated.

Error latency in the memory where the operating system resides is studied using data on physical memory access. These data are gathered through hardware probes in the machine that samples the system during the normal workload cycle of the installation. The technique

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. □ DTIC USERS □

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL

22b. TELEPHONE NUMBER *(Include Area Code)*

22c. OFFICE SYMBOL
none

EDITION OF 1 JAN 73 IS OBSOLETE.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

provides a means to study the system under different workloads and for multiple days. These data are used to reconstruct the error discovery process in the system. An approach to determine the fault miss percentage is developed and a verification of the entire methodology is also performed. This study finds that the mean error latency, in the memory containing the operatng system, varies by a factor of 10 to 1 (in hours) between the low and high workloads. It is also found that of all errors occurring within a day, 70% are detected in the same, 82% within the following day, and 91% within the third day.

Fault latency in the paged sections of memory is determined using data from physical memory scans. Fault latency distributions are generated for s-a-0 and s-a-1 permanent fault models. Results show that the mean fault latency of a s-a-0 fault is nearly 5 times that of the s-a-1 fault. Performance data gathered on the machine are used to study a workload-latency behavior. An analysis of variance model to quantify the relative influence of various workload measures on the evaluated latency is also given.

Error latency in the microcontrol store is studied using data on the microcode access and usage. These data are acquired using probes in the microsequencer of the CPU. It is found that the latency distribution has a large mode between 50 and 100 microcycles and two additional smaller modes. It is interesting to note that the error latency distribution in the microcontrol store is not exponential as suggested by other reported research.