NASA-CR-190975

September 1992

UILU-ENG-92-2232
CRHC-92-17

Center for Reliable and High-Performance Computing

NAG1-613
IN-60-CR
127119
P. 57

A REAL-TIME DIAGNOSTIC AND PERFORMANCE MONITOR FOR UNIX

Hongchao Dong

Coordinated Science Laboratory
College of Engineering
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-92-2232 CRHC-92-17	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION NASA
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) NASA Langley Research Center Hampton, VA 23665
8a. NAME OF FUNDING/SPONSORING ORGANIZATION NASA 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NASA NAG - 1 - 613
8c. ADDRESS (City, State, and ZIP Code) NASA Langley Research Center Hampton, VA 23665	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

A REAL-TIME DIAGNOSTIC AND PERFORMANCE MONITOR FOR UNIX

12. PERSONAL AUTHOR(S) HONGCHAO DONG

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM ____ TO ____	AUGUST 1992	50

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Distributed Unix, network, monitoring, error log, performance log, crash analyzing, multicomputers

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

There are now over one million Unix sites and the pace at which new installations are added is steadily increasing. Along with this increase, comes a need to develop simple, efficient, effective and adaptable ways of simultaneously collecting real-time diagnostic and performance data. This need exists because distributed systems can give rise to complex failure situations that are often un-identifiable with single-machine diagnostic software. The simultaneous collection of error and performance data is also important for research in failure prediction and error/performance studies. This paper introduces a portable method to concurrently collect real-time diagnostic and performance data on a distributed UNIX system. The combined diagnostic/performance data collection is implemented on a distributed multi-computer system using SUN4's as servers. The approach uses existing Unix system facilities to gather system dependability information such as error and crash reports. In addition, performance data such as CPU utilization, disk usage, I/O transfer

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

A REAL-TIME DIAGNOSTIC AND PERFORMANCE MONITOR FOR UNIX

BY

HONGCHAO DONG

B.S., Northwestern Polytechnical University, Xian, China

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1992

Urbana, Illinois

ABSTRACT

There are now over one million Unix sites and the pace at which new installations are added is steadily increasing. Along with this increase, comes a need to develop simple, efficient, effective and adaptable ways of simultaneously collecting real-time diagnostic and performance data. This need exists because distributed systems can give rise to complex failure situations that are often un-identifiable with single-machine diagnostic software. The simultaneous collection of error and performance data is also important for research in failure prediction and error/performance studies. This paper introduces a portable method to concurrently collect real-time diagnostic and performance data on a distributed UNIX system. The combined diagnostic/performance data collection is implemented on a distributed multi-computer system using SUN4's as servers. The approach uses existing Unix system facilities to gather system dependability information such as error and crash reports. In addition, performance data such as CPU utilization, disk usage, I/O transfer rate and network contention is also collected. In the future, the collected data will be used to identify dependability bottlenecks and to analyze the impact of failures on system performance.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my thesis advisor, Professor Ravi Iyer, whose guidance and encouragement made this thesis possible. At a personal level, his guidance and help have been equally invaluable.

I thank Professor Kent Fuchs and all members of the Center for Reliable and High-Performance Computing for a rewarding experience and friendly atmosphere. I have learned much about Unix system and computer networks. It is with pleasure that I acknowledge the opportunities CRHC provided to me. I thank Drs. Dong Tang, Molly Shor and Robert Dimpsey, who were generous to help me with my thesis. I thank my friends here in Champaign-Urbana, specially those in the Coordinated Science Laboratory whose support and friendship made my years at the University a memorable experience.

Final, I thank my dear wife, Qin Wang, whose love and support mean so much in everything I do. For all her love and support, I am grateful.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

The growth in distributed systems spurred by low-cost microprocessor and interconnection technology has made workstation/server systems economically viable in many research and academic environments. Workstation/server systems permit computer users to share CPUs, disks and peripheral devices. They also provide numerous services such as electronic mail, distributed databases, parallel computing, and high speed communication.

1.1 Distributed Computer Environment

The distributed nature of the system is one of its major advantages. However, it is a disadvantage when it comes to diagnosis of errors and general maintenance for the following reasons:

- At present, there are no accurate available network-wide facilities to analyze the multi-computer system after a crash. Each network is a unique collection of products from multiple vendors with no single vendor taking responsibility for the integrity of the entire network. When a fault occurs in a distributed environment, the traditional fault diagnosis process requires system maintenance personnel or diagnostician to log into the suspected workstation or server in order to determine the cause of the fault. The diagnostic procedure might involve the examination of the system logs or the running of diagnostic software. This is not only time consuming but also requires a great deal of experience. The procedure is especially troublesome and the error often goes undiscovered when the problem arises from inter-machine activity.

- When a fault occurs in a distributed environment, most users either report the local system's problem, reboot the local workstation, or simply use another workstation. Contrast this to the response on a dedicated machine where the user will normally contact the system administrator who will log the problem and fix it. By ignoring the problem in the distributed environment, valuable diagnosis data is discarded and the error often remains latent.

- Previous research [2], [4] has shown that system failure rate is dependent on resource usage and that increased resource usage is accompanied by an increased failure rate. Physically, this dependence may be caused by several factors:

- Increased usage exercises more of the system resulting in a higher probability of detecting faults.
- Increased usage results in more stress on the hardware: higher levels of electronic noise, temperature, and mechanical stress.
- Increased usage increases the likelihood of human error which can lead to system failure.

Through load balancing, the distributed system increases the utilization of shared resources. Therefore, these resources become more susceptible to faults. A recent study has shown that nearly half of all failures in a distributed system were due to errors in shared resources [25].

- Most diagnosticians look for unusual patterns in the error log when performing diagnosis. Certain patterns or symptoms become trouble-shooting rules in the diagnostic community. For example, the frequent appearance of the failure of a disk block access indicates that a particular block is corrupted. Frequent parity error messages indicate a memory board problem. These trouble-shooting patterns may become more complex in a distributed system. If the faulty machines are diagnosed individually, the network-wide patterns may not be identified.

A globle diagnosis approach may be addressed with an automatic centralized monitoring tool which periodically gathers performance data in addition to error and crash logs from all units in the network. The collected data can provide a total network-wide view of the system and give the diagnostician the means of isolating and identifying inter-machine problems. Because the monitor could be automated, information would not be lost when a user decides not to report problems. In addition, the orderly logging of all information allows the diagnostician to identify network-wide trouble-shooting patterns. The collection of system performance data will also be extremely useful. Past studies have shown that information on system status at the time of the system failure is important for system diagnosis. Also, by collecting performance data, the insight into the relationship between system activity and errors can be gained.

This thesis proposes an automated distributed performance and error software monitor for Unix-based distributed networks. The monitor has been implemented on a network of workstations and other computing resources at the Center for Reliable and High-Performance Computing (CRHC). The key aspects of this project are:

1. The monitor collects, analyzes, and filters all logs created from system crashes.
2. The monitor collects and filters all error/diagnostic data.
3. The monitor periodically collects performance data that characterizes the workload of the system.
4. The monitor is constructed mainly from the existing system facilities. However, the final monitor is not just a conglomeration of all available system facilities. Extensive study and consultation with performance analysts was

conducted before choosing and modifying the operating system and related facilities in order to collect meaningful data.

5. Because the monitor is constructed from existing system facilities, it is completely portable to any Unix machines. All facilities and modifications are written in either Unix commands or C shell.

6. All data collection occurs on-line.

7. All monitoring functions have a minimal impact on system workload.

In the next section related work will be summarized. Following this, in chapter 3, the system monitored will be summarized. Chapter 4 introduces the monitor and data format. The following three chapters 5, 6 and 7, describe three types of information collected in crash, error/diagnostic and performance, respectively.

1.2 Related Research

The major thrust of our work to date has been the creation of a monitor which collects both error and performance data. The data are collected with the future goal of automatic diagnosis and failure prediction in mind. Research in the diagnosis of distributed environments was first done at the Xerox Palo Alto Research Center (PARC) [20]. There METRIC was developed as a kernel software measurement system for a network of NOVAs7. Measurement data was communicated over the Ethernet [20]. Measurement events, which consisted of mini-snapshots of the system state, provided an indication about what was happening on the system. A few years later, an internal project was initiated at Xerox to diagnose computer hardware faults [18]. The diagnostic system consisted of a diagnostic server and over two hundred Dandelion workstations. Each workstation gathered its own events in a log and periodically reported then to the server. However, this work concentrated on collected error events and did not consider performance.

Predicting future errors from the logs of past errors has also been investigated [26]. For instance, a failure prediction method based on intermittent error characteristics of Unix machines was investigated by T. T. Lin [17]. In the study, an on-line diagnostic system on file servers interconnected by an IBM token ring was implemented. Two software systems, the Agent and Diagserver, were developed for the data collection phase.

In measurement-based analyses, an important issue is the interaction between workload and dependability. Past studies have shown that system failures are dependent on the usage of the system resources [2], [4], [3]. For instance, real measurements have shown that CPU-related failures increase exponentially with the resource usage after the system utilization reaches a saturation point [11]. Therefore, measurements of the workload characteristics in terms of resource usage is an important aspect in on-line failure diagnosis and prediction.

Several analytical models that take into account resource-usage effects have been recently proposed in [23], [1]. The issue has also been partially explored through measurement-based analytical models [8]. In the research, analytical modeling and measurements were combined to develop measurement-based reliability/performability models. The results showed that a semi-Markov process, as opposed to a Markov process, was better to model system behavior.

A measurement-based study on the dependability of three different operating systems, the IBM/MVS system, the VAX/VMS distributed system, and the Tandem Guardian fault-tolerant system was investigated in [15]. The study showed that I/O management and program flow control are the major sources of software problems in the IBM/MVS and VAX/VMS operating systems, while memory management is the major source of software problems in the Tandem GUARDIN operating system.

In the above measurement-based studies, real data collected from the computer systems played an important role. It is crucial to have a monitor that can automatically collect performance, error and crash data. Our monitoring facility is developed for this purpose. We hope that the collected information present in the Unix system will help provide to predict the occurrence of errors and diagnose the system when an error occurs.

2. MEASUREMENT ENVIRONMENT AND COLLECTED DATA FORMAT

2.1 SUN Distributed System

The Sun Network File System (NFS) provides a facility for sharing files in a heterogeneous environment of machines, operating systems, application software and networks. Sharing is accomplished through mounting a remote file system. Once a remote file system is mounted, it appears to be a part of the existing file system hierarchy. The network file system avoids the problem of each node in a network maintaining its own file system and thereby duplicating resources.

Figure 1 provides an example of servers and workstations with their file systems mounted on a NFS server. All file transfers take place over the Ethernet. A machine that provides resources to the network is called server, while a machine employing these resources is called a client. A machine may be both a server and a client. For instance, Server Two in Figure 2.1 can access files on the disk of Server One (making it a client) and also serve its clients (making it a server).

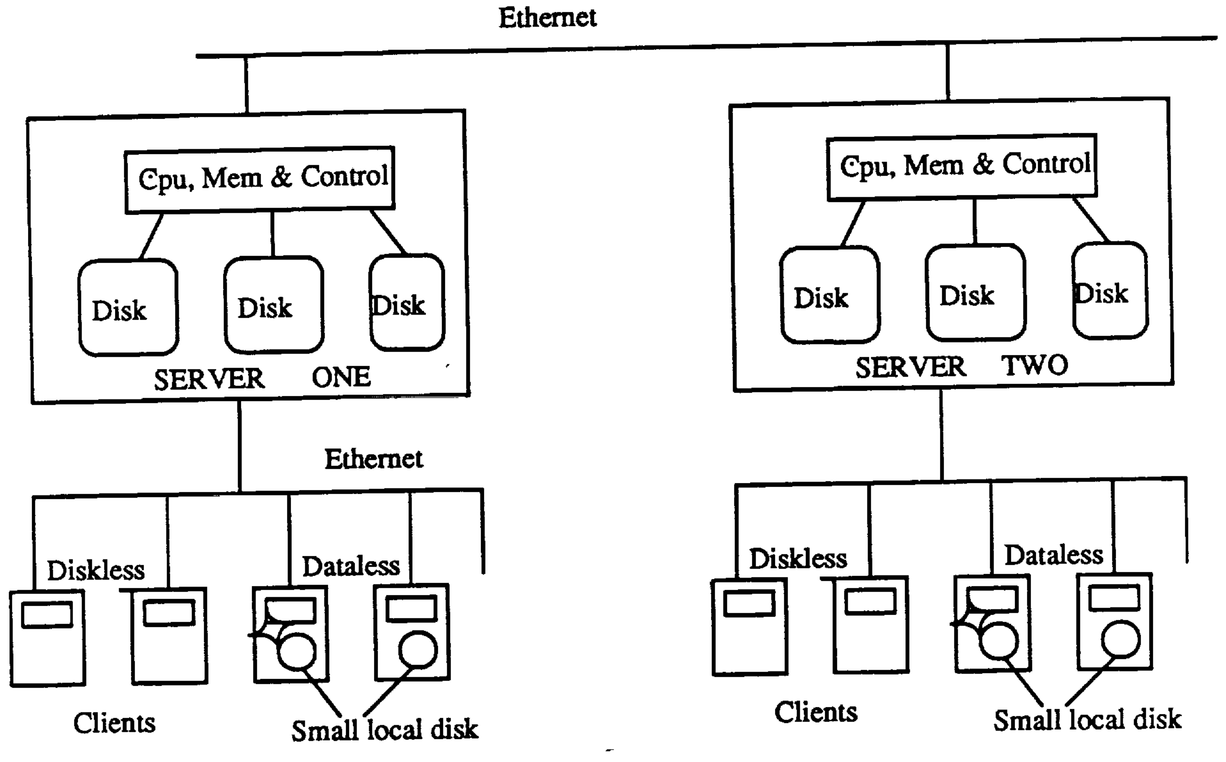

Figure 2.1: Servers and Clients

Both servers and clients run instances of the same copy of the distributed SUN operating system. Each client can have a limited size of local hard disk (normal root and swap), but most large file systems and other shared physical resources reside on the servers. The servers provide large, high-speed disks and network management. Sharing the system resources among many users in a distributed environment dramatically increases the usage of CPUs, disks, and the network. Obviously, the entire SUN system performance is affected if the server cannot provide the required functions.

2.2 CRHC Local Network

The CRHC network (Figure 2.2) monitored for this study is a typical local network integrated with various components. Following is a list of the machines connected to the CRHC network:

- Three SUN 4/490 and one SUN 4/280 servers, each with 32M ram and 3 G bytes disk.
- Seventy sun 3, sparc I or sparc II workstations.
- One ENCORE Multimax 510 with 64M ram and 8 processors.
- One INTEL ipsc/2 Hypercube.
- Four VAX and DEC workstations.
- Five HP9000 workstations.
- Four TI Explorer and symbolics workstations.
- A Tandem fault tolerant TMR system (Integrity S2).
- Several IBM-AT, 386, RT, Mac II and four laser printers.

The four SUN 4 servers together provide physical space to store most user file systems and system utilities. Each server has two Ethernet controllers and serves as a gateway to their subnet or cluster of workstations. Each subnet consists of 15-20 workstations

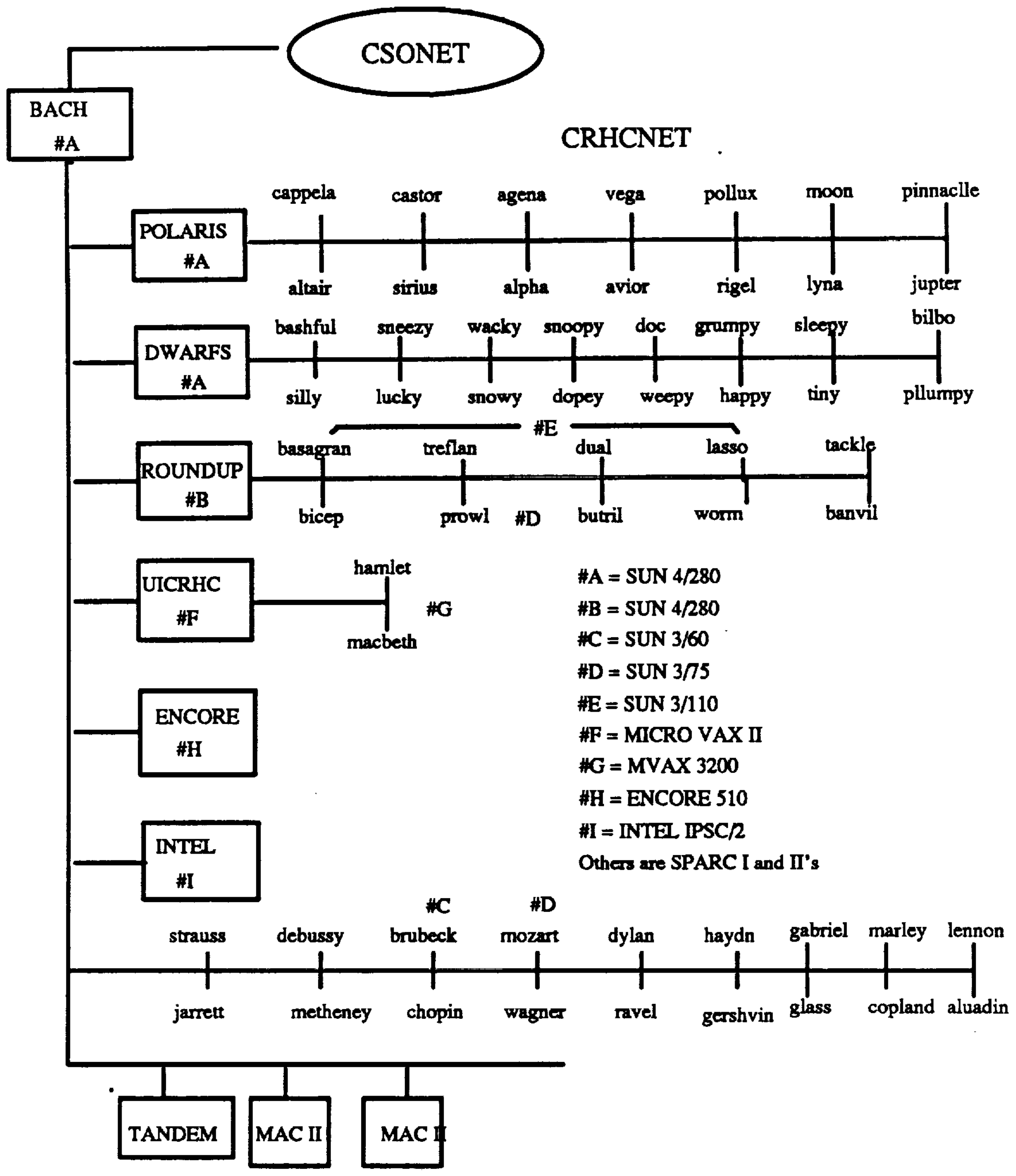

Figure 2.2: CRHC Local Network

or other types of machines. The servers also have CD, 9 track 1/2, 1/4 and 8mm tape drives available to them.

The sharing of certain common software programs is done to save space. For instance, there are only two physical copies of all /usr/local software residing on the network. Uniform access is provided to any user through the NFS sharing. High level communication between machines is based on a client-server model using the remote procedure call (RPC) implemented on top of the internet protocols.

2.3 General Monitoring Environment

In this section, the software monitor is classified and described in a high-level manner. A brief description of the format used by the monitor for the collected data is then given.

All machines in the CRHC network can have a diagnostics and performance monitor. However, SUN 4 servers play the most important role, while most other machines in the network are diskless(without local disk) or dataless (with small local disk mainly serving swap functions) workstations, or other types of machines, on which only one or two users do limited computations. Our implementation makes use of existing operating system facilities, modifies some system configuration files and sets up a software monitor to collect the information on system crash, system error log and system performance on the sun4 servers. At the same time, the system error log information on all workstations are also collected.

2.3.1 Software Monitor

The diagnostic and performance monitors can be classified in terms of software, hardware and hybrid. Hardware monitors typically are the least intrusive but most expensive and inflexible. Hybrid monitors normally detect events with hardware, and record them using software. Although, more flexible than hardware monitors, they too require additional hardware and thus lack portability. Because of the ease of implementation and portability [5], the diagnostic monitors implemented for this study are all software based.

Implementing the proposed monitors consisted of modifying the operating system and then reading contents of certain locations or tables in the kernel memory. When modifying the code, we were careful to maintain the integrity of the operating system and to use event-detection and data-collection techniques that did not appreciably alter the workload characteristics of the measured system.

Diagnostic monitors for distributed systems can also be divided according to topology and functionality into three classes: centralized, distributed, and hybrid. In the centralized diagnostic system, error logs generated from each node are collected and saved at a centralized site. This topology has been widely used in server/client distributed systems. In the distributed diagnostic system, error logs are collected and processed immediately by each sub-system. This arrangement does not impact the network as severely as a centralized diagnostic system. A hybrid monitor is a mixture of the above two approaches. Part of diagnostic system resides on individual nodes and part on a centralized diagnostic server.

The monitor presented in this study is a distributed diagnostic system. Based on the physical structure imposed by the distributed environment, the diagnostic system resides on individual servers and one server serves as a centralized diagnostic database center. Most events are collected, processed and diagnosed immediately by the servers. The error log collected among workstations are also transferred to its server. This arrangement was chosen for a number of reasons. First, because there is a large amount of data being collected (crash, error and performance), we thought it wise to avoid excessive network traffic caused by the monitors. Second, the distributed arrangement allows each server to run diagnostic independently and keep the data on the local disk. Third, with distributing the monitors, a single failure will not cause system monitoring to stop completely. Finally, a distributed monitor can be easily expanded as the local area net grows.

2.4 Data Collected and Format

The monitors implemented on the system collect three types of data: system crash data, system diagnostic data, and system performance data. System crash data is collected each time the system fails. Phenomenon or error data is collected for each error. Examples of these are parity errors or abnormal behavior. The implemented monitors also collect performance data (e.g., CPU utilizations, disk usage) continuously. This section provides an overview of the format used for each of these types of data. Detailed information will be given in the following sections.

Table 2.1: Crash Data 1

Summary of crash	hostid time system-age panic-info regs
system stack	current-stack-fp end-of-stack
Processes table	pid ppid pri cpu event p-name p-flags
users structures	p-ptr p-signal ofile-addr start-time
open file table	f-type f-ops date-offset fop-flags
m-buffer usages	streams queues mblocks dblocks buffers

Table 2.2: Crash Data 2

date-time-crash	hostid	p-size	f-size	pc-number	sp-number	panic-mess
ymddhhmmss	xx	xxxx	xxxx	xxxxxxxxx	xxxxxxxxx	aaaaaaaaaa
12	2	4	4	10	10	30

1. System crash data

There are two files created when the system crashes: A large detailed file and a small summary file. The large file is produced by crash script commands and contains the information in Table 2.1.

The small file is a crash log file, which is created from the large crash data file. The crash log data format is shown in Table 2.2, where:

date-time-crash	:The date and time of the crash
hostid	:The id of the hosts
	(1=bach, 2=dwarfs, 3=polaris, 4=roundup)
p-size	:Process table size (number of process is in the system)

Table 2.3: Diagnostic Data 1

date	hostname	program-name	error-information

f-size	:Open file table size (number of files)
pc-number	:Program Counter
sp-number	:Stack pointer
panic-messages	:The panic-messages of the system when the crash occurs

2. System diagnostic data

Every error or warning message generated (diagnostic messages) by the system may be important for diagnostic purposes. Each message is collected by the syslogd and logged in the format shown in Table 2.3, where:

date	: The time of the error occurrence
hostname	: Machine name where syslogd was running
program-name	: The program or daemon name that detected the error
error-infor	: A short message about the error

A filter program was written to eliminate all useless information in the error log file. A lex program also was worked out to decode the log file into numeric format which can be used more efficient by SAS or other statistical programming languages. The

Table 2.4: Diagnostic Data 2

time	hostid	prog-num	log-num	error-messages
yymmddhhmmss	xx	xxxxx	xxxx	aaaaaaaaaaaaaa

decoded log messages contain two parts: a string of decimal numbers and a short error messages. The format is shown in Table 2.4, where

time	:The time of the error
hostid	:hostname (1=bach, 2=dwarfs, 3=polaris, 4=roundup)
prog-num	:the program or process where error is detected
log-num	:the catalog number of the error
messages	:The brief-messages of the error in ASCII

3. System performance data

   The key components in the performance data of SUN 4 servers are collected as follows (Table 2.5).

Table 2.5: Performance Data

Utilization	Memory Usage	System Tables	Network Stat

Utilization:	: CPU disk and tty utilizations
Memory Usage	: Allocated virtual-memory vs. available virtual-memory
System Tables	: Number of files, number of inode, number of processes and swap space usage
Network Stat	: network traffic info: inpkts, outpkts, errors and collisions

3. SYSTEM CRASH ANALYSIS

The goal of the crash analysis is to record meaningful information when a crash occurs. The information will be be used to determine the cause of a system crash. The information is also useful in later studies of failure prediction and fault-performance correlations. The SUN OS reacts to a system crash in a number of predefined and user specified ways. This chapter will first outline the immediate system response to a crash in software and hardware. Following this, the various facilities that SUN provides to analyze a crash will be discussed. Finally, details of our monitoring protocol and what is finally logged in the event of a system crash will be presented.

3.1 Hardware and Software Crash Response

A system crash may have a variety of causes. For instance, faulty hardware, system software error, power supply error, device driver error or mechanical error. In this section, the detailed response of the system to hardware errors and system software errors will be given.

The system must first detect the error before it can make a response. Error detection is usually assisted by built in hardware and software detection mechanisms. Hardware detection mechanisms are based on redundancy techniques such as error detection and correction codes. Software detection mechanisms include consistency checks and the repetitive execution of code. The Unix operating system executes a large number of internal consistency checks on the system image. Device drivers will retry before reporting a fatal error. These internal check failures and retry messages are all recorded in a system error log file.

In general, at the detection of a system fault, the internal processor registers will be saved and an error message will be printed on the console and in the system error log.

The Sun 4 RISC architecture handles most hardware faults through traps and interrupts. When a hardware error occurs during an instruction execution, the hardware writes a value into the trap type field of the Trap Base Register (TBR). This then generates the address of a trap handling routine. The error handling program then checks the error type and prints out an error message.

An example of a hardware error will make the above clear. Suppose there is a memory error during an instruction or data fetch. This will cause a detection of the memory address-not-aligned trap. The hardware will then enter an error state and a value will be written into the trap type field of the TBR. This uniquely identifies the address-not-aligned trap. This number also serves as an offset into the table whose starting address is given in the TBR.

The control is transferred to the trap through the following steps:

1. Disable the trap.
2. Change the Processor State Register (PSR).
3. Decrement the Current Window Pointer (CWP) which points to the current active register window.
4. Save the Program Counter (PC) and next Program Counter(nPC) which holds the next instruction to be executed assuming a trap does not occur.
5. Write into the PC the contents of TBR, and into the nPC the contents of TBR+4.

The trap routine itself will first determine if the error is uncorrectable. Error information will then be printed to the system log. This information is retained for off-line analysis. If the error type is unrecoverable, the system will enter a panic state. The panic state causes the system crash.

If a software error is detected, through an internal consistency check, the system will also panic. A short message indicating which consistency check failed and a two-word description of the inconsistency is printed out. All these information are retained by our monitor. For example, a common consistency check is the program which checks the boundary of a virtual memory address (Figure 3). If an error is detected, the panic routine is called.

The panic routine will print out the error message and then reboot the machine.

Figure 3.1: Virtual Memory Check Program

```
configure();                /* set up devices */

maxmem = freemem;

v = econtig;

i = nproc * ptob(SEGU_PAGES);

if (v + i > (caddr_t) MDEVBASE)

        panic("insufficient virtual space for segu: nproc too big?");

segu = seg_alloc(&kas, v, i);

if (segu == NULL)

        panic("cannot allocate segu\n");

if (segu_create(segu, (caddr_t)NULL) != 0)

        panic("segu_create segu");
```

It is critical for the monitor to distinguish between a reboot caused by a panic and a reboot initiated by a user. The next section details how we have modified the system commands to make this distinction clear.

3.2 Design and Implementation of Crash Analysis

Once a crash has occurred, the panic routine is run and a system reboot is enacted. The system boot up procedure loads the kernel into memory by the PROM monitor. Control is then transferred to the kernel and the machine resumes normal execution.

However, in this way, there is no record (except the small error message) of the state of the machine before the crash. To remedy this, the boot procedure was changed to forces a crash core dump.

The core dump is a copy of the memory image saved in secondary storage. To execute a core dump the SUN OS system first disables virtual-address translation. Then the processor priority is raised to its highest level to block out any device interrupts. Finally, the contents of physical memory are placed in secondary storage.

Because the core dump image is large, the boot up procedure is appended with the *savecore* command which moves the core dump image to a larger disk partition and logs the reboot event was performed in the syslog file.

Commands have also been added which automatically sends mail to the system administrator after each core dump and reboot. It is then the administrator's duty to analyze the core dump and produce the files which summarize the crash in an orderly manner. A script which was created to analyze the core dump and produce the summary of the crash will be described shortly. First, it should be noted that a system reboot is often executed with the system shutdown command even though a system crash has not occurred. To maintain the integrity of our system logs, the shutdown command has been rewritten to require the reason for the shutdown to be specified. In this way, reboots caused by system crashes and those caused by other reasons will not be confused.

The analysis of the core dump by the system administrator was automated with a script command which can be executed by the administrator. Two Unix facilities were considered as tools for the core dump analysis: adb and crash. Adb is a system utility for interactive, general-purpose program debugging. The utility can treat the OS kernel as an executable program and provide a controlled environment for the execution of the kernel. It can also be used to examine the image of the system memory map. However, adb requires extensive familiarity with the operating system source code. We decided not to use it as a vehicle in analyzing the crash core dump.

The crash program is a tool which examines the memory image of a live or crashed system kernel. It displays the values of the system control structures, tables, and other pertinent information. We decided to use the crash program as a vehicle for preliminary crash analysis because it is based on a set of fixed commands and gives detailed information about the usage of system resources and control tables.

Each core dump is analyzed by a script of chosen crash commands. This uniform shell script provides a consistent analysis tool to provide an organized summary of each crash. The shell script is run by the administrator after receiving the mail about a system crash. The shell script provides two outputs for each crash, a long and short

form, in a file called crash.(date).

The long form of the crash summary contains the following:

Crash summary information: Operation system name and version, hostname, hardware architecture, time of crash, age of system, panic messages, program counter address and system stack address.

Information on system resources usage: An image of the dynamic memory buffer reallocation, an image of context register allocation information, an image of allocated streams data block headers, the size of page, proc, datab, dblk, dblock, linkblk, mblk, mblock, msgb queue, stdata, and streams.

Information of every process behavior: Table information for each process including process status, tsize, dsize, ssize, maxrssize, user time, system time, start time, signal disposition, open files and profile flags

Information of the file system usage: Information on the file table including the file no-delay flag and the asynchronous flag. Also a reference and message count for each file to indicate how many processes are referencing the file and how many interprocess communication transact with the file. Also recorded are all mounted file systems, as well as operations, addresses and flags of the mounted file systems.

Physical and virtual address information: Stack information of the users, the kernel and the interrupts. And all physical and vitual address information.

The long form holds too much information for some purposes, so a short form was also created (as shown in Section 2.4). This log contains a one line entry which gives brief information on each system crash.

For instance, *910520232101010101380582f80a674cf832e9f8ialloc: dup alloc*

where:

910520232101	:Crash happened at 1991 May 20 23 hour 21 minute 1 second

01	:Hostid 1=bach
0138	:There are 138 process in system
0582	:There are 582 files in system.
f80a674c	:Program Counter is f80a674c.
f832e9f8	:Stack pointer is f832e9f8.
ialloc: dup alloc	:Panic-messages.

A more detailed example of crash analysis is given in the Appendix of the thesis.

4. SYSTEM DIAGNOSTIC DATA COLLECTION

4.1 System Error Log

Often times an error or abnormal behavior occurs but does not result in a system crash. For instance, a retry on a device driver may be successful, or a parity error may be solved in hardware. While not causing a system crash, these errors may be indicative of failures to come and are helpful in future crash diagnosis. These errors may also result in performance degradation. Therefore, it is important to systematically collect these information. This section will describe how our monitor collects this error/diagnostic information.

When an error occurs, a one line error message is logged in a system error log called the message buffer (*msgbuf*). The message buffer is allocated early in the bootstrapping of the system, and is placed in high memory so that it can be located after a reboot. This allows messages generated just before a crash to be saved. In prior releases of the SUN OS the *msgbuf* could only be read through /dev/kmem with the *dmesg* command.

The command was troublesome because it could not be synchronized properly with the generation of new diagnostic messages and did not provide the time and date of the message. This was a major obstacle in comprehensive diagnostic monitoring facilities.

This problem was solved with the addition of a special device, /dev/log, providing a read only interface to the *msgbuf*. With the /dev/log in place, the *msgbuf* can be read by a user program. The reading of the *msgbuf* is done with a daemon called syslogd. Syslogd collects error messages from /dev/log, from an Internet address family socket specified in /etc/services, and from the special device /dev/klog for kernel messages.

A large number of the error messages handled by the syslogd are superfluous. To filter out the useless messages, the system provides the syslog configure file: syslog.conf. This file allows the system to choose to forward error messages to appropriate log files or users. The file also assigns one of the following priorities to each error message.

1. **emerg:** Panic condition. Normally broadcast to all users.

2. **alert:** Conditions corrected immediately.

3. **crit:** Warnings about critical conditions, such as hard device errors.

4. **err:** Other errors.

5. **warning:** Warning messages (uncritical).

6. **notice:** Not an error conditions, but may require special handling.

7. **info:** Informational messages.

Table 4.1: Error Log Configuration

diagniotic messages	forward to
*.err;kern.debug; auth.notice;user.none	/dev/console
*.err;kern.debug; user.none	/var/adm/messages
auth.warning; daemon; user.none	/var/adm/auth.log
mail.crit; user.none	/var/spool/mqueue/log
lpr.debug	/var/adm/lpd-errs
*.alert; kern.err;daemon.err;user.none	operator
*.alert; user.none	root
*.emerg; user.none	*

8. **debug:** Messages for program debugging.

9. **none:** Do not send message.

The syslog.config file was configured in our implementation to send messages as indicated in Table 6. The table shows that all diagnostic error messages are recorded in a file entitled /var/adm/messages. All errors at the err level severity or higher print out the error message on the console and also log the message into the /var/adm/messages file. A user login authorization system warning level or higher will be forwarded to /var/adm/auth.log. Mail and printer errors will be forwarded to /var/spool/mqueue/log and /var/adm/lpd-err, respectively. The users *root* and *operator* are informed of any alert level messages. All users will be informed by any emergency level messages excluding user messages.

The amount of data collected is still huge with this facility. To prevent the collected data from filling up the root partition, the data must be archived periodically. This is done by the /usr/lib/newsyslog routine which is run automatically by the cron process.

The configuration file allows a certain amount of filtering, but a significant portion of the messages collected are still unrelated to the hardware and software errors. These have been eliminated with a post-filtering program.

For analysis purposes, We catagorized the remaining messages into seven groups: cpu, disk, memory, peripheral device, network, software and miscellaneous. Each message is represented by the format shown in Section 4.2. In order to facilitate data processing and analysis, a decoding program was written which converts each error message into a string of decimal numbers. The decimal data can then be used as input data for SAS or other statistical languages to perform systematic analysis.

4.2 The Implementation of Distributed Diagnostic System

The diagnostic information collecting is based on long term. The basic thrust of our efforts is to get as much system diagnostics information as possible. Because in most cases, once data are collected, it is not possible to request more information.

Our implementation of the diagnostic data collection is based on the distributed diagnostic system (Fig 5.1). The diagnostic information generated from each host will be collected in its server. The distributed diagnostic system includes four steps.

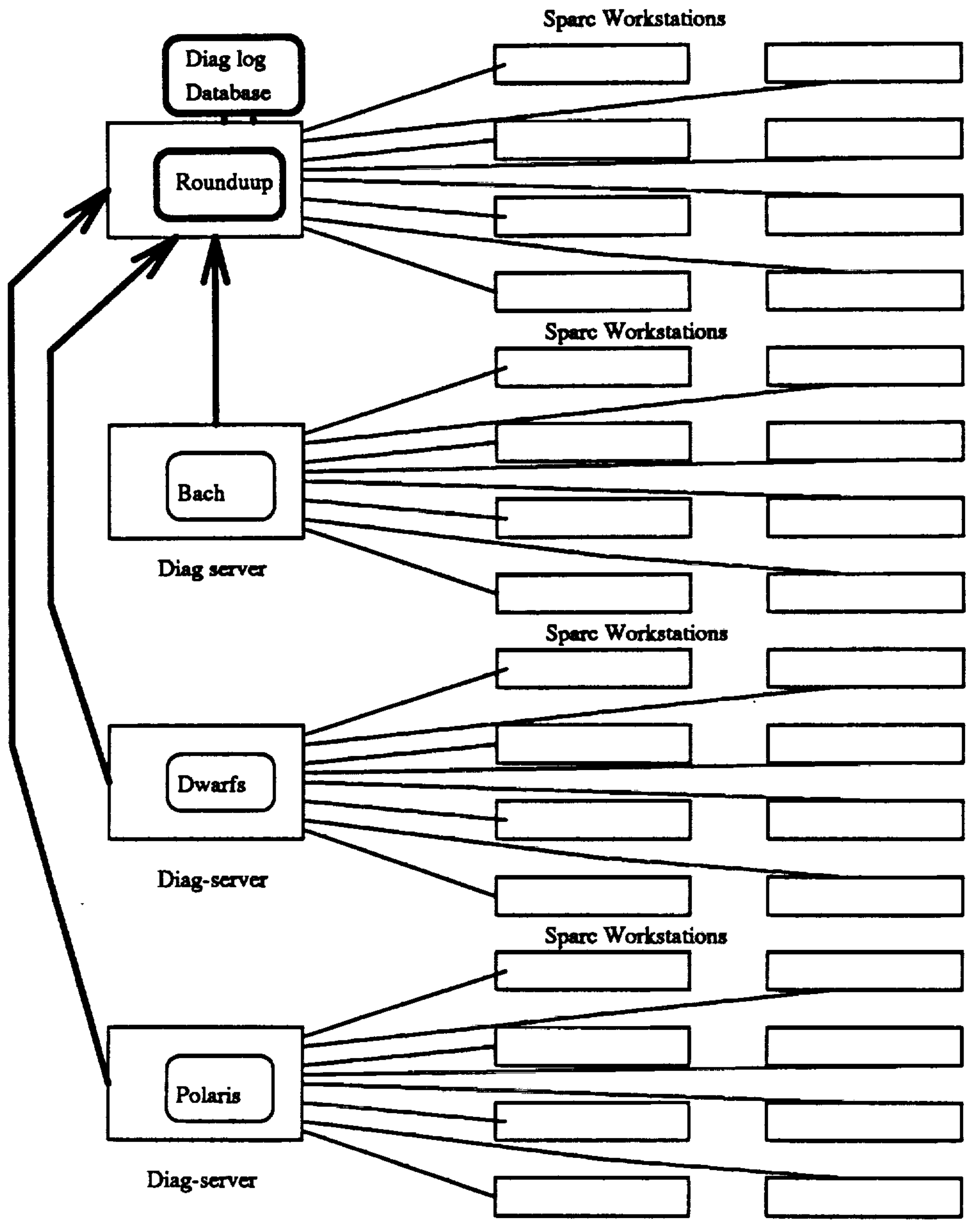

Figure 4.1: Distributed Diagnostic System

- First, each machine (server and workstations) logs diagnostic information in a local file.

- Second, The server collects diagnostic information from machines on its subnet which consists of 15-20 workstations. The server will independently and regularly reformat and reallocate the collected data on its disk.

- Third, after a fixed interval (two weeks in our case), the servers transfer all diagnostic information collected to one specific machine–a centralized diagnostic database site.

- The centralized diagnostic database site processes all information collected from the CRHC network (about 70 of servers or workstations). It creates an integrated diagnostic database file which records all diagnostic information collected in the CRHC network.

All the four steps are done automatically by the computer system. Several system utilities have been modified to help to automate the diagnostic process.

First, on each diagnostic server, /usr/lib/newsyslog run automatically by cron process which moves old log file messages to a specific disk partition. The current diagnostic file always contains the most recently error-log information.

Significant portion of the diagnostic messages are not related to hardware and software error. Thus a filter program is needed to eliminate all useless messages. The function of the filter is to examine the error patterns and delete the entries that match the strings

Table 4.2: Final Diagnostic Data Format

date	hostname	servername	program-name	error-information

we want to delete. The filter program runs on each diagnostic server to produce the minimized final diagnostic data.

In order to have the uniform data formate, a C shell program has been worked out to convert diagnostic data into the format shown in Table 4.2, where:

date	: The time at which error occurred.
hostname	: Machine name where syslogd was running.
servername	: Diagnostic server name where machine reported error.
program-name	: The program or daemon name that detected the error.
error-infor	: A short message about the error.

At the centralized diagnostic database site, all collected diagnostic data from machines in the CRHC network are organized in chronicle. The data will permanently reside on the site for further analysis.

5. SYSTEM PERFORMANCE DATA COLLECTION

As mentioned before, empirical studies uncovered a relation between the workload on a system and faults experienced by the system [2][9][11]. The performance or workload data of the system may therefore be useful in predicting faults or in diagnosing the system after a fault. Thus, workload performance data is collected along with the crash and diagnostic data by the monitor. This section summarizes the performance information collected.

The performance of the distributed Unix system is summarized by the usage of the individual resources. The following five components were chosen to summarize the performance of each individual machine.

1. CPU utilization

2. Disk I/O usages

3. Network I/O transfer rate

4. Ram and virtual memory usage

5. File cache efficiency

There are various system utilities in Unix which can be used to collect the performance data of individual components. For our situation, it is important to collect data over a long period of time. A key factor in choosing the monitoring software is to minimize the performance perturbation of the collection facility. The execution of the data collection should have no adverse effect on the regular use of the system.

User level commands were chosen to collect performance data because they allow great flexibility and portability. In addition, the commands could be easily added and dropped on the various machines. Performance data is collected with a shell script that executs the chosen system commands every 10 minutes. The following commands were used.

IOSTAT The iostat command quantifies the I/O of terminals and hard disks. It also quantifies the percentage of time the CPU's spend in various states. For each disk, the number of seeks and data transfers completed and the number of words transferred are counted; for the terminals, the number of input and output characters are counted. Also, at each clock tick, the state of each disk is examined and a tally is made if the disk is active. Transfer rates for the disks are also measured. For example, a typical iostat uotput is:

```
    tty          xd0            xd1            xd4            xd5      cpu
tin tout bps tps msps  bps tps msps  bps tps msps  bps tps msps  us ni sy id
  0    1  12   2 13.2    6   1 15.8    3   0 17.4   30   5 12.1  14 11 13 62
```

Table 5.1: Iostat Data Format

TTY	Disk 1	Disk 2	Disk 3	CPU
Usage	Usage	Usage	Usage	Usage

where:

tin: The number of characters read in from the terminals.

write: The number of characters written out to the terminals.

rps: The number of reads per second for each disk.

wps: The number of writes per second for each disk.

util: The Disk utilization.

us: The percentage of time the system had spent in the user mode.

ni: The percentage of time the system was in user mode running low priority processes.

sys: The percentage of time the system had spent in the system mode.

id: The percentage of time the system was idle.

Table 5.1 summarizes the output of iostat.

PSTAT The pstat command records the contents of various kernel tables such as the system inode table, text table, process table, and open file table. The command also displays status information for all terminals and swap space usage.

Table 5.2: Pstat Data Format

files	inodes	processes	swap-spase

For example, the pstat returns swap space information as follows:

```
5304k allocated + 528k reserved = 5832k used, 132824k available
```

This indicates that the system is using 5832k of virtual memory and there is 132824k of virtual memory available.

An example of a Kernel table is the following:

172/582 files 291/322 inodes 59/138 processes 5864/138656 swap

where:

The first number is the number of table system used and second number is the number of table system can provide.

The table also indicates that the system is currently running 59 processes but could support an additional 69 processes if necessary.

Table 5.2 summarizes all the information collected by pstat for our monitoring implementation.

NETSTAT The netstat command quantifies contention on the network. The command provides a list of active sockets for each protocol and statistics of various other

Table 5.3: Netstat Data Format

ethernet-controller 1	ethernet-controller	loopback
(CRHC-net)	server subnet	localhost
traffic stat	traffic stat	

network data structures. For example, the information of ethernet traffic is the following:

```
Name      Net/Dest      Address   Ipkts    Ierrs  Opkts    Oerrs  Collis  Queue

ie0 1500  uiuc-crhc-net4roundup   2202170  811    2537605  14     784     0

ie1 1500  uiuc-crhc-net roundup   3787908  183    4212515  0      21464   0

lo0 1536  loopback-net  localhost 2253     0      2253     0      0       0
```

The data is the traffic statistics which shows how much input, output, error and collision on each ethernet interface.

The results from this command are summarized in Table 5.3.

The vmstat command was considered but it was found that the data produced was prodigious and the perturbation caused was high. Timing measurements have been conducted on the performance measurement facilities and it has been determined that the perturbation caused by the monitoring is small. An interval of ten minutes was chosen to reduce this perturbation and the amount of data collected. Ten minutes is short enough to collect meaningful information.

6. CONCLUSION

This thesis has introduced a flexible, portable, distributed monitoring facility that collects system crash data, diagnostic/error messages, and performance data. The monitor has been implemented on a local network consisting of a large number of Sun workstations as well as a variety of other computing resources such as the Encore Multimax. To maintain portability, the monitor was constructed from existing system Unix facilities. All collection are automated and done on-line. Care was taken to minimize the perturbation caused by the monitoring software.

The organized collection of data after a system crash required the system bootup procedure to be modified to perform a core dump, move the dump to a large partition, and send mail to administrator. The system shutdown command was also modified to record the exact reason for shutdown (logging purposes). A crash command script which analyzes the core dump was also written. The collection of error/diagnostic data required the use of the syslogd daemon and the creation of filtering and decoding software.

Performance data collection was accomplished with the periodic invocation of iostat, netstat, and pstat.

An important issue which arose from this study and deserves future attention is the collection of error/diagnostic messages through the device driver /dev/log. This facility allows easy access to the message buffer from the user level. With this facility, and the kernel level printf command (print to message buffer) the collection of all kinds of operating system level information becomes trivial. The OS is modified with printf commands and the syslogd daemon then collects these. This allows easy expansion of monitoring facilities and also may help in kernel debugging.

The goal of this research has been the concurrent collection of crash, diagnostic, and performance data. With the monitor in place, data is continuously being collected. This data will be helpful in the diagnosis of system crashes. The data can also be used in future studies of failure prediction and workload/fault correlations.

APPENDIX A THE EXAMPLES OF COLLECTED DATA

```
LONG CRASH FILE
***************
  A) Information about time of crash, time of system since last
  crash and the panic message.

>status
version: SunOS Release 4.1.1 (GENERIC) #1: Fri Oct 12 18:17:55 PDT
1990 Copyright (c) 1983-1990, Sun Microsystems, Inc.
machine name:   bach.crhc.uiuc.edu
machine type:   SUN 4/470
time of crash:  Mon May  20 23:21:01 1991
age of system:  107 days, 21 hr., 11 min.
panic: ialloc: dup alloc
...
```

B) Contents of the key registors when crash occur.

```
>pcb

        registers: pc f80a674c

        sp f832e9f8

        psr:      ae4

        flags:        0

        uwm        0 wbcnt 0 wocnt 0 wucnt 0
```

C) List information of all processes at the time crash, such as process id, program name and process status. Following example show ttere are 138 processes when crash happened.

```
>proc

PROC TABLE SIZE = 138

SLOT ST PID   PPID  PGRP   UID PRI CPU  EVENT     NAME         FLAGS

   0 s     0     0     0     0   0   0 f817eac8                load sys

   1 s     1     0     0     0  30   0 f823d374 init         swapped pagi

   2 s     2     0     0     0   1   0 f823d428              load sys

   3 s   127     1   127     0  28   0 ff036568 rarpd        swapped pagi sel

   4 s    65     1    65     0  26   0 f817e61c portmap      swapped pagi

   5 s   202   168   202     0  26   1 f817e61c rpc.rquotad  swapped pagi
```

```
. .   ...   ...   ...     .  ..   . ........ ............   .......
. .   ...   ...   ...     .  ..   . ........ ............   .......
. .   ...   ...   ...     .  ..   . ........ ............   .......
```

D) Usage of system stacks. In this example, the system used about 1 Kbytes memory space (f8478f58 - f8478a70), which indicates the system was lightly used when crash happened.

```
>stack
KERNEL STACK:
FP: f8478a70 END OF STACK: f8478f58

f8478a70:          3  f8478ab0  f8006afc  f823ee2c
f8478a80:         40         7  ff012f68         0
f8478a90:      a8750  114000c5  f80cd1d0  f8475f58
f8478aa0:         80         9         1         7
f8478ab0:   f8478a98         0  f81a4c00  f81a4f44
f8478ac0:   f818b400  f8475f58  ff005b20  f8478b40
f8478bd0:      c6000  f818b400         0  ff194300
f8478be0:   f81a7a70  f8478c10  f80e2640  f8478c18
........    ........  ........  ........  ........
........    ........  ........  ........  ........
........    ........  ........  ........  ........
```

E) Detail info of each process, such as time of process start, physical address of the program and files opened by the process.

```
>u 0-137

PER PROCESS USER AREA FOR PROCESS 0

        command:  proc ptr: f823d2c0  sess ptr: f8185150  tty ptr: 0

        no cntrl tty

        current directory structure (at ff013fe8):

        ref 1 len e38c


        vnodes: current directory: ff039cd8 root directory: 0 tty: 0

        ofile: f81534e8 pofile: f81535e8 lastfile: -1 cmask: 0000

.....  ........ ......  ........ ......... .. ....... ....

.....  ........ ......  ........ ......... .. ....... ....

.....  ........ ......  ........ ......... .. ....... ....


PER PROCESS USER AREA FOR PROCESS 1

        command: init proc ptr: f823d374 sess ptr: f8185150 tty ptr: 0

        no cntrl tty

        start: Fri Feb 15 17:09:42 1991

        current directory structure (at ff013fe8):

        ref 1 len e38c
        vnodes: current directory: ff039cd8 root directory: 0 tty: 0

        ofile: f827b4e8 pofile: f827b5e8 lastfile: -1 cmask: 0000

..... ........ ......  ........ ......... .. ...... ....

..... ........ ......  ........ ......... .. ...... ....

..... ........ ......  ........ ......... .. ...... ....
```

F) Information about files opened in system at the time of crash.

such as size, status and operation of the file,

```
>file

FILE TABLE SIZE = 582

SLOT  TYPE RCNT   MSG          OPS      DATA   OFFSET     CRED FLAGS

   0    S     1     0    _socketops ff652c8c        0 ff0246b8 read write

   3    S     1     0    _socketops ff654a8c        0 ff0246b8 read write

   4    S     1     0    _socketops ff65638c        0 ff0246b8 read write

   5    V     1     0    _vnodefops ff0ee69c f8170784 ff024d84 read

   7    S     1     0    _socketops ff65528c        0 ff0246b8 read write

   8    V     3     0    _vnodefops ff053f3c      23f ff0246b8 read write

   .    .     .     .    .......... ........      ... ......... ....

   .    .     .     .    .......... ........      ... ......... ....

  50    V     1     0    _vnodefops ff0ee69c ff649e50 ff0246b8 read

  51    S     1     0    _socketops ff65668c        0 ff0246b8 read write
  .    .    .    .    .......... ........     ... ......... ....
  .    .    .    .    .......... ........     ... ......... ....
```

```
SHORT CRASH LOG
**************

   910520232101010138O582f80a674cf832e9f8ialloc: dup alloc

where:
   910520232101 ;Crash happened at 1991 May 20 23 hour 21 minute 1 second
   01           ;Hostid 1=bach,
   0138         ;There are 138 process in system.
   0582         ;There are 582 files in system.
   f80a674c     ;Program counter.
   f832e9f8     ;Stack pointer.
   ialloc: dup alloc ; panic-messages.

...

DIAGNOSTIC MESSAGES
*******************
...
```

```
Jun 30 17:53:47 polaris vmunix: id000h: block 669508 (1741852 abs):

write: Conditional Success. Data Retry Performed

Jun 30 17:53:47 polaris vmunix: id000h: block 669508 (1741852 abs):

write: Conditional Success. Data Retry Performed

Jul  1 18:33:18 polaris vmunix: /home/polaris3: file system full

Jul  1 18:36:21 polaris last message repeated 6 times

Jul  2 13:30:16 polaris vmunix: ie2: Ethernet jammed

Jul  4 13:30:18 polaris vmunix: ie2: Ethernet jammed

Jul  5 14:39:50 polaris vmunix: st0:  I/O request timeout}

...
```

REFERENCES

[1] B. E. Aupperle, J. F. Meyer, and L. Wei, "Evaluation of fault-tolerant systems with nonhomogeneous workloads," in *Proceedings of 19th International Symposium on Fault-Tolerant Computing*, pp. 159–166, June 1989.

[2] S. E. Butner and R. K. Iyer, "A statistical study of reliability and system load at slac," in *Proceedings of 10th International Symposium on Fault-Tolerant Computing*, 1980.

[3] X. Castillo and D. P. Siewiorek, "A performance-reliability model for computing systems," in *Proceedings of 10th International Symposium on Fault-Tolerant Computing*, 1980.

[4] X. Castillo and D. P. Siewoerk, "A workload dependent software reliability prediction model," in *Proceedings of 12th International Symposium on Fault-Tolerant Computing*, pp. 279–285, June 1982.

[5] D. Ferrari, G. Serazii, and A. Zeigner, "Measurement and Turning of computer systems ," N. Y.: Prentice-Hall Inc. August 1983.

[6] A. Goyal et al., "The system availability estimator," in *Proceedings of 16th International Symposium on Fault-Tolerant Computing*, 1986.

[7] R. A. Howard, *Dynamic Probability Systems.* New York: John Wiley & Sons, Inc., 1971.

[8] M. C. Hsueh, R. K. Iyer, and K. S. Trivedi, "Performability modeling based on real data: A case study," *IEEE Transaction on Computers*, April 1988.

[9] R. K. Iyer, S. E. Butner, and E. J. McCluskey, "A statistical failure/load relationship: Results of a multicomputer study," *IEEE Transaction on Computers*, vol. C-31, no. 7, pp. 697–706, July 1982.

[10] R. K. Iyer and D. J. Rossetti, "A statistical load dependency model for CPU errors at SLAC," in *Proceedings of the 12th International Symposium Fault-Tolerant Computing*, pp. 363–372, June 1982.