Chapter 3

Description and Measurement

In this chapter, a detailed description of the study system, including its hardware, software, and users, is presented. The data collection techniques and tools used to measure the operation of the system are also outlined. Although the information in this chapter does not provide insight into the data, it provides the necessary background to establish the framework of our study. The actual use of the measured data will be discussed in Chapter 4.

The appendix of this chapter, Appendix A3, contains tables of information about the physical configuration of the CDF system and the data collection process.

3.1 Overview of the Study System

The University of Toronto Computing Disciplines Facility (CDF) was chosen as the study computer system. At the time of our study, the CDF computing environment consisted of a file server, a compute server, and 69 diskful workstations that were connected on two Ethernet local area networks that spanned the first two floors of the Engineering Annex building on the University of Toronto campus. Files were shared on a common file server and when needed were relayed over the Ethernet to the client workstations using the Sun Network File System (NFS). The users in the CDF environment were primarily undergraduate students working on assignments for various computer sciences courses. Other users included graduate students, faculty members, and system administrators.

The primary reasons for choosing the CDF system for this study were:

- Limited extensive workload characterization had been previously done on this system.
- If a campus-wide computer system that is similar to CMU's Andrew File System (AFS) were to be designed at the University of Toronto, the CDF system would rep-

resent a typical sub-network. Better understanding of this sub-network may provide insight into the potential design of a campus-wide system.

- The CDF system has periodic fluctuations in the workload that make it interesting from the workload characterization point of view. These fluctuations typically correspond to busy periods at the end of term when assignments were due in several courses. The potential of load sharing during these busy periods is of interest to this study.

- Such an environment may need to be upgraded in size when the number of students or courses offered changes. Studying this system can provide insight into design considerations of this nature.

- The system is quite convenient in a number of respects. It is conveniently located on the University of Toronto campus, and it uses the SunOS version 4 operating system, which provides built-in process accounting and other tools that simplify the measurement and analysis of the workload.

3.2 System Configuration

The CDF system comprised a Sun4 SPARCsystem 600 compute server (eddie.cdf.toronto.edu), a SPARCserver 490 file server (marvin.cdf.toronto.edu), one PC, and 69 Sun IPC SPARCstations with local disks as shown in Figure 3.1.[1] All of these machines were connected to a 10 Mb/sec Ethernet local area network. The 28 machines that were in room 201 and 203 of the Engineering Annex building were attached to the `cdf-ether` Ethernet, while the 42 machines that were located on the first floor, in rooms 107 and 107B, were connected to the `cdf2-ether` Ethernet. Eddie and marvin were located in room 201A of the Engineering Annex building, and were the only machines that were connected to both `cdf-ether` and `cdf2-ether`. In general, groups of 3 to 10 machines shared a bus that led to the repeater for that Ethernet. The repeaters simply propagated electrical signals from one cable to another, without making routing decisions or providing packet filtering. They played no role in the network protocol.

[1] The workstation with the black screen in Figure 3.1 (axiomset) was not working during the data collection period.

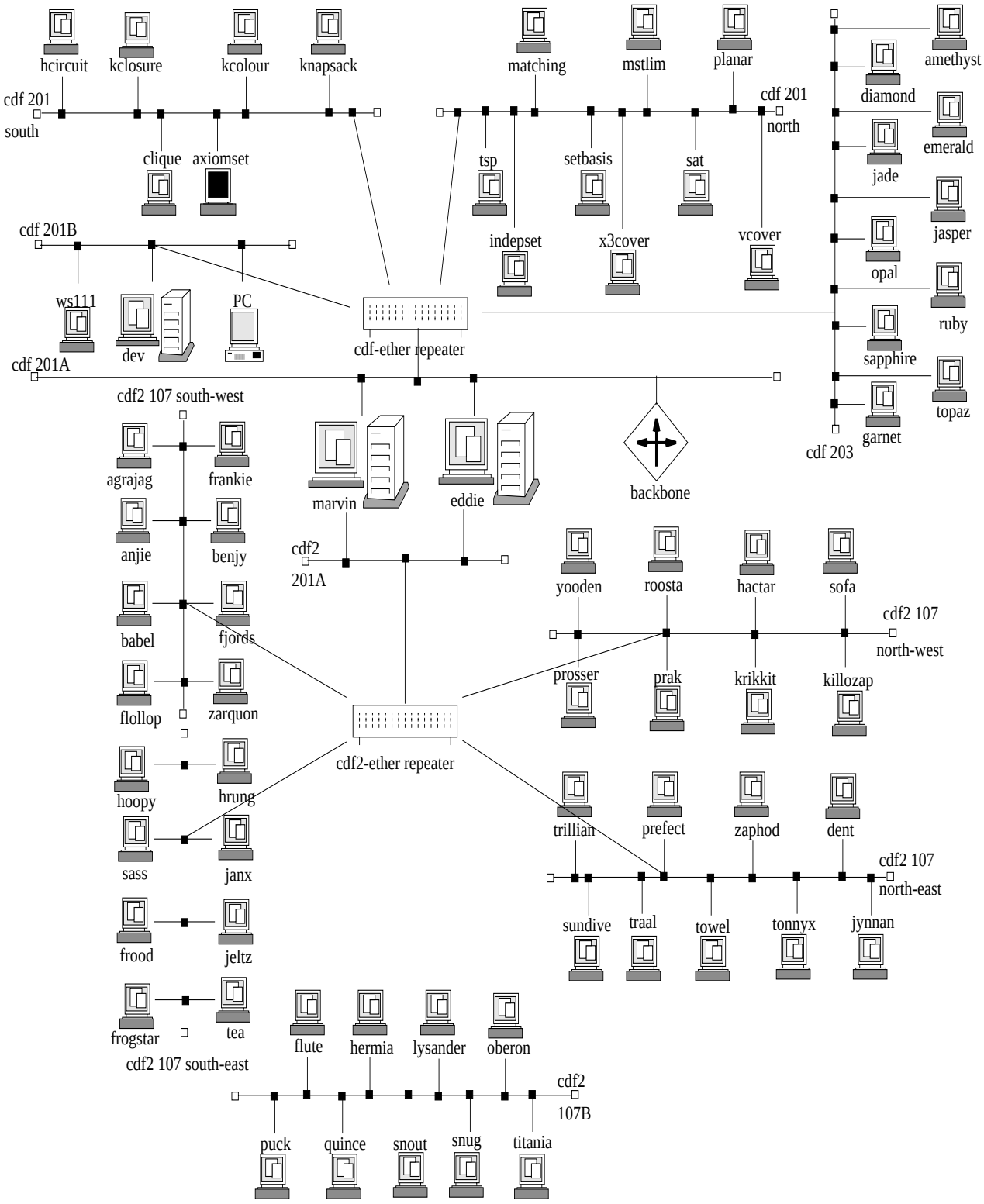

Figure 3.1: The CDF Computing Environment.

3.2.1 Processors and Operating Systems

The CDF system was primarily homogeneous with respect to its processors, with the exception of the file server, marvin, and the compute server, eddie. These hosts needed more computing power due to their specialized functions. Eddie had four CPUs that ran at 40 MHz; marvin had a single processor that ran at 33 MHz. The workstations were each configured with a 25 MHz single processor. All machines used reduced instruction set computing (RISC) technology. A summary of the hardware and software configuration is provided in Table A3.1.

Each host was running SunOS version 4.1.1, with the exception of eddie who was running SunOS 4.1.2 with symmetrical multiprocessor (SMP) support. In this type of architecture, multiple processors share a single common memory and run under the control of a single copy of the operating system. The SMP support in SunOS 4.1.2 is very rudimentary; it uses a simple mutual exclusion (mutex) lock on the kernel. Only one process can run kernel code (e.g., handle an interrupt or process a system call) at a time. Other processes must wait by spinning on the mutex lock.

The data collection in this thesis does not involve any modification of the kernel. All data collection was done using existing UNIX programs and the process accounting facilities provided by the SunOS operating system.

3.2.2 Memory and Displays

Eddie was configured with 128 MB of main memory and marvin was configured with 98 MB of main memory. Dev and ws111 were each configured with 24 MB of main memory. Dev was different than the other IPC SPARCstations, in that its purpose was to store a master copy of the file system that was on each workstation's local disk. The ws111 workstation had more memory than the other IPC SPARCstations because it was used by a system administrator.

The remaining 67 SPARCstations had either 12 MB or 16 MB of main memory. The workstations with 16 MB of main memory had colour monitors, while those with 12 MB of main memory had monochrome display monitors. The workstations each had a local disk with a tmpfs [2] `swap` partition allocated on `/tmp`. Depending on the usage of `/tmp`

[2] Tmpfs is a RAM disk file system type. In this type of file system, modified files never get written out to disk as long as there is some RAM available in which to store them.

on a particular workstation, the total `swap` size varied from 15 to 23 MB on the client workstations. This reduces the load on the network and on the file server, as recently used pages could be fetched into memory from the local disk, instead of having to retrieve them from the file server.

3.2.3 Prestoserve

Marvin was the only host that had a 1 MB Prestoserve synchronous write accelerator installed. A Prestoserve is a pseudo disk driver layer that caches synchronous writes in non-volatile memory. I/O requests for marvin's disk device drivers that had Prestoserve installed were intercepted by the Prestoserve and cached in its non-volatile memory. Whenever the Prestoserve needed to perform actual I/O (e.g., as when the cache needed draining), it called the original driver's entry points to perform the actual I/O on the appropriate device. The Prestoserve did not affect reads, except for data that had recently been written to the Prestoserve and not yet transferred to disk. The contents of the Prestoserve was completely flushed back to the disks on reboot.

On marvin the Prestoserve was used with the `/var` directory and with all home directory partitions (`/homes/u1`, `/homes/u2`, `/homes/u3`, `/homes/u4`), except for the "cssu" [3] home directory partition on the `rf2` drive (`/homes/u5`), because the Prestoserve did not support the SMD controller that was used on the Fujitsu Eagle disk drives.

There are a lot of synchronous disk writes in an NFS environment such as CDF, where the NFS server must frequently handle client server requests that modify files. The Prestoserve can produce significant performance improvements in this environment because these writes are essentially performed at memory speeds instead of at disk speeds. In addition, writes that result in a dirty cache hit avoid ever having to perform the previous physical disk writes.

3.2.4 Window environment and login sessions

When users logged into a workstation console, the X Window System started up automatically. A number of `xterm` windows, each corresponding to a user login session, were started on the workstations' display monitor. The interactive `tcsh` UNIX command interpreter was

[3] The cssu account was for the Computer Science Student Union (CSSU). It contained many executable programs, and informational files. This account was used by members of the CSSU council for such things as compiling the computer science yearbook.

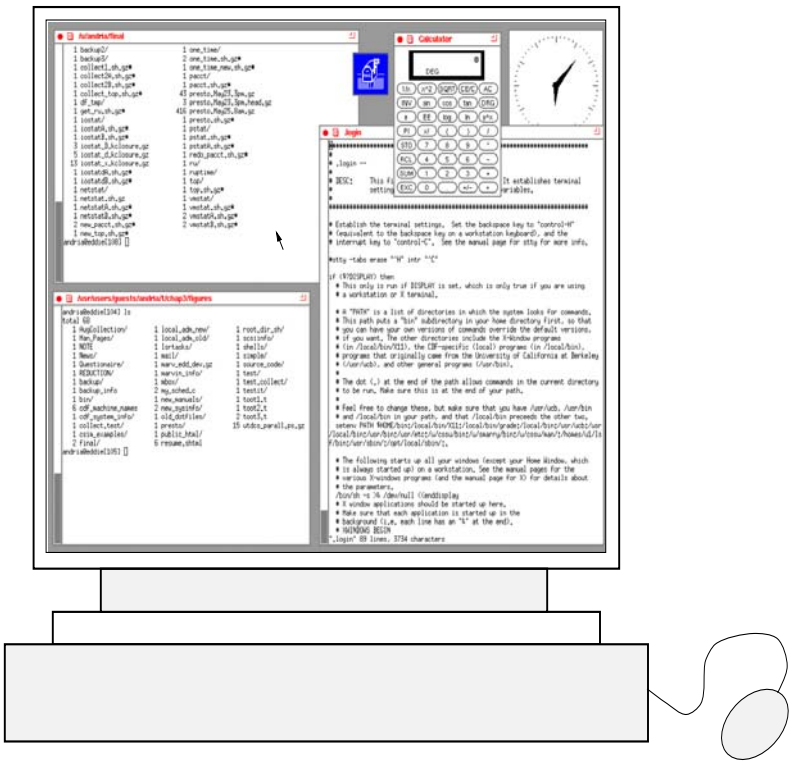

Figure 3.2: Sample Windows on a Workstation in the CDF Environment.

started in each `xterm` window. A window managing program, such as `twm` or `olvwm`, and X Window application programs, such as clocks, calendars, and calculators, were started upon login initiation. A screen on a typical workstation running X Window is shown in Figure 3.2. Users could use the `rlogin` command to start login sessions on other hosts. The X Window environment also allowed a user to start up an `xterm` window on another host.

In most cases a command issued on a local host was executed on that host; however, there were a few exceptions. The `rsh` command could be used to explicitly issue a command on another host in the CDF environment. There were also some software packages with *node-locked* licences that forced execution to be performed on a particular host, regardless of where the application was started. `Maple` and `matlab` were node-locked to eddie. Execution of these programs was supported by an `rsh` to eddie that started the X Window application (either `maple` or `matlab` running in an `xterm` window, or `xmaple` itself) that ran on eddie and displayed on the local workstation.

The mouse could be used to position the cursor (pointer) in one of the windows on the display. Although several application programs and windows could be running simultaneously, it was only physically possible for a user to enter one interactive UNIX command in

one window at a time. Commands could be entered by using the keyboard or by clicking with the mouse.

Besides logging into a workstation console, users could also access the CDF computing environment via a modem line. As these modem lines were connected to eddie, modem login sessions were automatically initiated on eddie. A user could login remotely to other workstations if he or she so desired.

Only system administrators and users who were given special permission were allowed to login to marvin and dev. This restriction was made to reduce the load on these machines, and to maximize security in the system. We had special permission to use marvin during the collection of our data in the CDF environment.

3.2.5 Disks

SPARCstation Disks

Each of the 68 Sun IPC SPARCstations in the CDF computing environment was configured with a 200 MB disk drive on a single SCSI disk controller. The disks were either Maxtors or Quantums as outlined in Table A3.2.

There was no automatic caching done on the local disks and no local state information was backed up. Instead, a specific subset of the complete `/usr` and `/local` partitions on marvin was made available on each workstation's local disk. The parts that were missing on the local disks were replaced by symbolic links to an NFS-mounted complete version of `/usr` and `/local` on marvin. Eddie stored a complete version of these directories on its local disk.

Each night a command script was run on each workstation to compute a pseudo directory of the local disk. This directory was copied to dev, and compared against a master copy of the directory. Any differences were then corrected on each workstation, one at a time. The complete versions on the servers and the partial versions on the workstations were updated automatically in a single sweep from the master copy on dev.

Each workstation was also configured with a 3.5 inch floppy disk drive. Users could use the UNIX `mtools` collection of programs to manipulate MSDOS files on floppy disks placed in these disk drives.

Disks on Dev

Unlike the other 68 Sun IPC SPARCstations, in addition to having a 200 MB `sd0` disk drive and a floppy disk drive, dev was configured with an Exabyte 8 mm tape drive and a 1.2 GB Fujitsu M2266S disk drive. The tape drive was used to make backups on each weekday night. The 200 MB `sd0` disk contained the master copy of `/usr` and `/local` partitions for the workstations' local disks. The 1.2 GB `sd1` disk contained a complete copy of `/` (root), `/usr`, `/var`, and `/local`. It also contained the `/admin` directory.

Disks on Marvin

Marvin had a total of eight disk drives: one Fujitsu M2654SA, four Seagate Sabres, and three Fujitsu Eagles. The Fujitsu M2654SA was on an SCSI controller; the Fujitsu Eagles were on an SMD controller; and an IPI controller was used for each pair of Seagate Sabre disks. The Fujitsu M2654SA SCSI contained a home directory (`/homes/u4`) and some repositories. One IPI drive held the `/` (root), `/user`, `/local` (primarily read-only executable files and data files), and `/var` directories (log files and accounting records). The remaining three IPI drives were configured for home directories and `/tmp` (swap). The `/homes/u1` directory was entirely devoted to staff users and instructors, while all other home directories were for students. Two of the Fujitsu Eagles held backup partitions, while the third held the home directory for the "cssu" account.

The `/backup` partitions were only used by "root" unless a user issued the `getback` command. The `getback` command could be used by users to retrieve a backup copy of a file from these `/backup` partitions. A "root" process called `backup` was used to continuously backup five different directories on CDF. This process swept over the first four home directories (the `/homes/u5` directory was not backed-up) and the user mail spool directory on disk `id0`. It copied over recently changed files to the Fujitsu Eagle backup directories. As soon as one disk had been scanned, it proceeded to the next in sequence.

Disks on Eddie

The compute server, eddie, had two disk drives on two separate SCSI controllers. The Maxtor (`sd0`) was entirely devoted to `/news`, whereas everything else was on `sd1`, the Fujitsu M2652SA disk drive. A complete copy of `/usr` and `/local`, as well as `/var`, `/src`

and `/tmp`, were stored on this disk.

3.2.6 Network

In the CDF environment, the Sun Network File System (NFS) protocol was used to transfer file requests across the Ethernet to the file server. NFS provides a convenient way to transparently access files on remote file systems ([San85], [SGK+85]). NFS is independent of the machine architecture and the operating system on which it runs. Heterogeneous hosts in a distributed system transparently send requests across the network using remote procedure call (RPC) messages. The server and the client hosts each have a virtual file system (VFS) request handler that can convert between RPC messages and local file system requests.

NFS uses a synchronous stateless protocol. The file server (marvin) does not maintain state information about each transaction. Each procedure call comes complete with all of the information that is necessary for its completion. The clients can not send subsequent requests until previous ones have been acknowledged.

When large files were transferred to and from the file server in the CDF environment, they were packaged into as many 8 KB blocks as possible, and then the remainder of the file was put into 512 byte block(s). When an 8 KB block had to be transferred over the Ethernet, it was fragmented into several 1.5 KB (including preamble, data, and frame check sequence) Ethernet packets. Since NFS does not maintain state information, if any of these packets was lost, a complete retransmission had to be performed.

The Ethernet used a carrier sense multiple access with collision detection (CSMA/CD) system; so before an Ethernet device could put a packet onto the Ethernet, it first had to listen to determine if another device was already transmitting. Once the device found the Ethernet to be clear, it could start sending a packet while also listening to hear if another device started sending at the same time. If another device started to send a packet while the first device was in progress, a *collision* would occur. Since only one device can transmit on the Ethernet at a time, both devices would have to "back-off" and attempt to retransmit.

3.3 Workload Overview

The week of Monday December 6^{th} through Friday December 10^{th}, 1993 was chosen for the collection of the data in this thesis because it was a busy week near the end of the

semester when final assignments were due in several courses. The `deadlines` command could be used by students on the CDF computers to determine assignment due dates for CDF courses, and to plan their work so that it would not coincide with expected busy periods. Table 3.1 shows which assignments were due in the CDF environment during the week of data collection.

CS Course [4]	Asst	Weight	Due Date	Instructor
csc378	5	10%	Mon Dec 6/93 2:10 pm	Tony W H Lai
csc485/2501	3	25%	Tue Dec 7/93 9:00 am	Graeme Hirst
csc2321	3	35%	Tue Dec 7/93 4:10 pm	Christina Christara
csc350	4	10%	Wed Dec 8/93 7:10 pm	Christina Christara
csc468/2204	3	19%	Thu Dec 9/93 2:00 pm	Scott Graham
csc418/2504	3	16.7%	Fri Dec 10/93 11:00 am	Michiel van de Panne
csc324	3	20%	Fri Dec 10/93 12:10 pm	K. Lawrence Chung
csc228	3	19%	Fri Dec 10/93 1:00 pm	Jim Clarke

Table 3.1: Assignments due during the Collection Interval

[4]Courses with two number listed were offered as both graduate and undergraduate courses.

The data in Table 3.1 indicate that the end of the week of data collection could be quite busy, as assignments were due in the large csc468 and csc418 classes. The csc468 course was an upper level operating systems class in which the students had a large Object Oriented Turing assignment due. The csc418 course was a computer graphics course that was working on a C based 3-dimensional graphics package for the final course assignment. The workload characterization analysis in the next chapter will show which periods were busiest throughout the week of data collection.

3.3.1 User Workload

A questionnaire was distributed to a professor or a teaching assistant in each computer science course that had an account in the CDF computing environment. The questionnaire explained the purpose of the study, and then asked the following three questions:

1. Do students in your course have assignments that are required to be done on the CDF computers? If not, for what purpose are they likely to be using their CDF computer accounts?

2. Are there any assignments on CDF that are due during or just after December 6-10, 1993? What is the due date of any such assignment? What type of assignment is it, and what types of software packages or UNIX commands are the students likely to use to complete the assignment?

3. In general (for the other assignments in your course), are there any other commands that students are likely to use?

The purpose of the questionnaire was to provide an overview of the types of commands that students were likely to use in each course. The second question was asked in case the instructor had forgotten to submit course due date information to the `deadlines` program.

The questionnaire revealed that most students in all courses used their computer account for communication. This included reading network news groups (`nn`, `xrn`, `trn`) and electronic mail (`mail`, `from`) for course and non-course purposes. Most students also used standard UNIX file commands such as `ls`, `cp`, `mv`, `more`, and `grep`, and editing commands such as `vi`, `emacs`, or `xedit`. Word processing in the CDF environment could be done using `nroff`, `troff`, or `latex`. The most common printing commands were `lpr`, `lpq`, and `enscript`. For some courses, assignments were submitted for marking using the `submit` command. This command copied the submitted files to a directory that could be accessed by the instructors.

There were also a number of X Window application programs, such as `xbell`, `xbiff`, `xcal`, `xcalc`, `xclipboard`, `xclock`, `xcpustate`, and `xdvi`. Most of these commands were placed in the background in a separate window, and ran continuously throughout the course of a user's login session at a workstation.

Table 3.2 summarises the course-specific UNIX commands that were used by students in each course. In many of the programming courses that used the C programming language, students used some combination of an editor, a preprocessor (`cpp`), a make file (`make`), a compiler (`cc` or `gcc`), a linker (`ld`), a debugger (`dbx`, `gdb`), and an executable program. Students who used Turing (`turing`) or Object Oriented Turing (`oot`) started up an environment that provided built-in preprocessing, compiling, debugging, and running facilities. The `oot` program started up an X Window application, whereas the `turing` program entered a text-only turing programming environment. There was also a terminal object oriented turing (`toot`) compiler available that, unlike `oot`, did not have to be used on a workstation.

Course Number	UNIX Commands Used
csc104 csc108 csc148	First year courses mainly used the CDF PC labs. The only active accounts on CDF were of users who dialed into eddie to work on Turing assignments.
csc228	basic C commands: make, cpp, cc (or gcc) text formatting: nroff, troff, or latex
csc260	maple, oot, toot, turing
csc300	network news, mail, printing
csc318	network news, mail, printing
csc324	prolog, lisp, oot (or toot)
csc350	matlab (ran on eddie only)
csc378	no assignments required computer use
csc418/2504	cc, make, dbx (or gdb), render
csc428/2514	SUIT
csc434/2509	EMPRESS, cc, make, dbx
csc465	no assignments required computer use
csc468/2204	oot, turing
csc470/2206	cc, make, dbx, maple, csim, map, gnuplot, xgraph
csc485/2501	prolog, lisp
csc2307	f77, cc, make, dbx
csc2321	matlab (ran on eddie only)
csc2535	xerion modules, fep, gnuplot, latex

Table 3.2: Course-specific commands on CDF during the 1993 Fall Semester

The following courses used specialized software: csc324 and csc485 used `lisp` and `prolog`; csc2321 used `matlab`; csc428 used `SUIT`, and csc434 used `EMPRESS`.

3.3.2 System Workload

A significant portion of the workload was generated by system programs and scripts. The system accounts that were active in the CDF system during the data collection period were: "root," "daemon," "sys," "nobody," and "news."

Table 3.3 shows the various daemons that were run by system users during the collection interval. Whenever a user on a particular host invoked the `finger` command, for example, this would start up an `in.fingerd` daemon on that host.

On eddie, a system "news" user continuously executed a number of different command scripts. At 15 minute intervals a script was run to transmit network news articles to a remote network news transfer protocol (NNTP) server. Whenever a remote NNTP host sent new news articles to eddie, the `nntpd` daemon was invoked and a command script was

Daemon Name	System User	Description
in.identd	sys	TCP/IP IDENT protocol server
nntpd	news	Network News Transfer Protocol server
in.fingerd	nobody	remote user information server
lpd	daemon	line printer daemon
in.comsat	root	biff server
in.ftpd	root	file transfer protocol server
in.ntalkd	root	remote user communication server
in.otalkd	root	server for talk program
in.rlockd	root	network lock daemon
in.rlogind	root	remote login server
in.rshd	root	remote shell server
in.telnetd	root	DARPA TELNET protocol server
lpd	root	line printer daemon
xdm	root	X Display Manager with support for XDMCP
Xsun	root	Sun server for X Windows Version 11

Table 3.3: Systems Users running Daemons during the Collection Interval

started to copy the new articles to eddie's news disk. The `nntpd` daemon was also invoked on eddie whenever a user started up a network news reading program from any host.

A short script that consisted of six commands was run by "root" at 5 minute intervals on each workstation throughout the data collection. This script was used to collect and update information for the `xrwho` X Window application program.

At 12:05 am each morning a lengthy shell script, called `earlymorn`, was started on marvin. This script verified that the local disks on the workstations were identical to the master copy that was stored on dev. It also cleaned up files from the `/tmp` directory of the workstations' local disks. On each host it made backup copies of various accounting information, including the last login information and limited fields of the daily process accounting records.

On marvin, the `backup` command was repeatedly run by the "root" user. The backup command was run on each of the 5 directories (one at a time) that were backed up for use with the `getback` command.

As some load sharing work had previously been done in the CDF system ([ZZWD91], [WZAL93]), before collecting our data we verified that no load sharing software was running in the CDF computing environment; such software could significantly skew the results of the study.

Unlike in the user workload, there is sometimes periodicity in the system workload. The "root" and "news" users run certain command scripts at fixed time intervals. If periodic elements in the workload can be identified, this may simplify the modelling of these components. For our model design in Chapter 6, the system workload will be examined to determine if any periodic elements can be identified. As the user workload, on the other hand, is quite unpredictable, stochastic models may prove best for modelling these elements.

3.4 Data Collection Techniques

UNIX end user commands were used to collect the data. Commands were submitted in C-Shell (`csh`) program files. The advantage of using this method of data collection is that the complicated task of modifying the kernel is avoided. This method of data collection, however, is not without fault. The analyst has less control over the type of data that can be collected, and it is more difficult to customize the data collection. In addition, the workload unit may have to be chosen as a compromise of what is available.

As mentioned in Chapter 2, selection of the workload unit should be done at an early stage in the workload model design. Since the model being designed in our study is to examine the potential of load sharing, the physical resources used by commands were chosen as the basic workload unit. This information was collected from the system process accounting statistics that were automatically generated on each host in the CDF system.

In addition to the process accounting records, two other types of data collection were used: *static* and *dynamic* collection. The static collection provided general information about the physical characteristics of the hosts in the system (see Table 3.5). It was collected just once at the beginning of the data collection period. Since a snapshot of the system at any particular point in time is not necessarily representative of how the system was operating, a dynamic collection using the UNIX maintenance statistical commands was also used (see Table 3.7).

3.4.1 Process Accounting Data Collection

The process accounting data were collected from December 6^{th} until December 10^{th}, 1993. Each time a process finished running, a record storing all resource consumption and timing statistics for that process was automatically written to the `/var/adm/pacct` file on that host. Each process was identified based on the command name that was used to start its

execution. The `pacct` file stored a record for each completed process in an encoded form. To extract the records into a comprehensible ASCII form, the `acctcom` command had to be used with the `pacct` file.

At about 2:30 am each morning, the `earlymorn` system shell script used the `acctcom` command to process the `pacct` file on each host. The ASCII output of the `acctcom` command was stored in a backup directory on marvin, and the unprocessed `pacct` file in the host's `/tmp` directory was then removed. Since not all fields needed for our workload analysis were stored in the processed process accounting files, the `earlymorn` system shell script was modified so that a complete copy of the unprocessed `pacct` file on each host was copied to a different backup directory on marvin.

At a later date, the `acctcom` command was used to extract data from the unprocessed `pacct` files using the command "`acctcom -ahikrt pacct`" [5] (for each host). The process accounting fields extracted are shown in Table 3.4. It was not expected that all of these fields would actually be required, but as previously mentioned, it is better to collect too much data than not enough.

acctcom Field Name	**Description**
COMMAND NAME	process command name
USER NAME	name of user submitting the command
START TIME	the time that the process started
REAL (SECS)	elapsed run time for the process
SYS CPU TIME (SECS)	system CPU time used by the process
USER CPU TIME (SECS)	user CPU time used by the process
CHARS TRNSFD	total number of characters read from and written to devices
BLOCKS R/W	total number of physical disk block (local or file server) reads and writes
KCORE MIN	cumulative amount of kilobyte segments of memory used per minute of run time

Table 3.4: Process Accounting Fields Selected

The process accounting records that resulted from the actual collection of the data for this study were ignored in the data analysis used to design the workload model. It was quite straightforward to determine which process accounting records these were, as each process accounting record was recorded with the name of the user who started that process.

[5] An explanation of the acctcom flags that were used in the data reduction is provided in Table A3.7.

3.4.2 Static Data Collection

At the beginning of the collection period, at 12:00 am on December 6^{th}, the commands listed in Table 3.5 were issued on each host to collect information about the physical characteristics of each host. These data were collected just once on each host in the system at this time. Some of these data could be used to determine configuration parameters that would be needed if the CDF system were to be simulated.

Command	Description
fpversion	Print information about the system's CPU and FPU types.
host -t hinfo $host	Look up information about an internet host using the domain server. The **-t hinfo** option gives information about the hardware and operating system used by the host.
sysinfo -level all	Display all system information. This included information about devices, OS version, architecture, memory, disks, CPU.
mem -v	Report on the amount of physical memory.
df	Report on the amount of free disk space on each file system.
cp /etc/mtab .	Get a copy of the static file system mounting table file.
netstat -i netstat -r	Show network status. The **-i** option shows the state of the interfaces that have been auto-configured, and the **-r** option shows the routing tables.

Table 3.5: Commands Used to Collect Static System Information

3.4.3 Dynamic Data Collection

Information that could not be obtained from the process accounting records was extracted by the dynamic data collection. Unlike the accounting records that were recorded for each command, the statistical commands, which are listed in Table 3.7, were used to record activity over a certain time interval.

All of the client workstations and servers had synchronised time clocks. On marvin the primary data collection script was remotely sent out to each workstation, prior to 12:00 am on December 6^{th}, using the `rsh` command.[6] An example of a primary data collection script is shown in Figure A3.1. This primary script started a separate script for the collection of each command that is shown in Table 3.7. An example of the C-Shell code that was used for a 24 hour collection of data from the `iostat -D` command is shown in Figure A3.2.

[6] Alternatively, the `at` or `crontab` command may have been used to start the scripts at a specified time.

Due to the massive amount of data that are generated by statistical commands, these collection scripts were only run on Tuesday December 7^{th}, and on Thursday December 9^{th}. As there was not enough storage space to do a full data collection on all hosts, 12 hosts were chosen for a 24 hour collection and the remaining hosts were used for only an 11 hour collection. Information about specific collection times is provided in Table 3.6.

Commands	**Collection Period and Date**
iostat, vmstat, netstat, nfsstat, pstat	Collected on 12 hosts over two 24-hour periods: Dec 7^{th} from 12:00 am to 11:59 pm Dec 9^{th} from 12:00 am to 11:59 pm Collected on 57 hosts over two 11-hour periods: Dec 7^{th} from 9:00 am to 8:00 pm Dec 9^{th} from 9:00 am to 8:00 pm
top	Collected on 69 hosts over a 4-day period: Dec 6^{th} 12:00am - Dec 9^{th} 11:59pm
ru	Collected on one host over a 6-day period: Dec 6^{th} 2:00am - Dec 10^{th} 11:59pm

Table 3.6: Data Collection Periods

The statistical commands that generated a lot of data were sampled less frequently. As shown in Table 3.7, a 20 second interval was used with the `iostat` and `vmstat` commands, whereas other commands used 12 or 15 minute sampling intervals. A 20 second interval is a desirable choice for the shorter intervals because it is a common divisor of the longer collection intervals, and thus simplifies the task of data correlation.

The `iostat`, `vmstat`, `netstat`, and `top` commands used an interval command line parameter to determine the frequency of the output. This helped to control the *drift* [7] to the exact frequency specified. The remaining commands that are listed in Table 3.7 were executed in a loop, with a `sleep` statement for the appropriate frequency placed at the end of the loop.

Since the `ru` command worked remotely (i.e., information was displayed for all remote hosts), it was the only command that was executed on just one host in the system. The `ru` command showed which users were logged in at each workstation. The `-c` option provided a count for multiply-logged users, and the `-d` option caused users to be listed for each

[7] Drifting occurs when commands take a long time to complete because the system is extremely busy, or because the command produces an excessive amount of output. The starting time of the next command is delayed because of the time taken for the previous command to complete.

Command	Frequency	Type of Data
iostat -D 20 3780 iostat -d 20 3780	20 seconds	disk read/writes per second disk transfers per second
vmstat -S 20 3780	20 seconds	swapping activity
netstat -i 900	15 minutes	network file system statistics
nfsstat -c	15 minutes	file system activity
pstat -T pstat -s	15 minutes	system table information swap space page information
top -d 482 -s 720	12 minutes	utilization, load, memory
ru -cd	30 minutes	user login sessions on each host

Table 3.7: Data Collection Frequencies for Statistical Commands

host, with the hosts listed alphabetically. The output from the `ru` command was used to determine which users were on a given host at any particular time during the data collection interval.

The `top` command collected information about the UNIX load average, the system utilization, and the amount of memory in the system. The **-d** and **-s** flags were used to specify the number of times that data were collected (482), and the number of seconds (720 seconds or 12 minutes) between displays, respectively.

The `iostat` command printed the I/O statistics. Its **-D** option reported on disk transfers and its **-d** option reported on the reads and writes per second. The `netstat` command showed the network status. The `nfsstat` command with the **-c** option displayed network file system statistics for the client. `pstat -T` showed information about how full the system tables had become. `pstat -s` showed information about the swap space usage in the system. `vmstat` reported on virtual memory statistics. Its **-S** option specified that swapping rather than paging activity should be shown.

3.5 Summary

The data collection techniques outlined in this chapter were used to collect data from the CDF system during the week of December 6^{th} through December 10^{th}, 1993. Successful data collection was carried out on 71 hosts in the CDF system. Only one host, axiomset, was not operating during the time of our data collection. The command scripts had to be restarted on several hosts that went down and then came back up again within the collection period. In total, 184 MB of data were collected for the accounting records, 42 MB of data

for the dynamic collection, and 1 MB of data for the static collection. Drifting was minimal.

We made use of existing UNIX commands to collect the measurement data for this thesis. No modification of the kernel was needed for our data collection. UNIX commands were submitted using a series of C-Shell scripts that were spawned out to each host in the system using a remote shell (`rsh`). Our data collection was used to collect static information that provided a "snapshot" of the system at a particular instant in time, dynamic information that recorded the system state during a specified time interval, and process accounting records that recorded the resource usage of commands.

The techniques used to collect our data were able to successfully retrieve an extensive set of workload data from the CDF system, without any kernel modification. The method of using scripts to start the commands at specified time intervals was impeccable. Other commands, such as the UNIX `crontab` or `at` command, may also have been suitable for initiating our commands and scripts.

In carrying out the measurement collection in this chapter, we increased our understanding of potential problems that might arise in the collection of data from distributed systems. We offer the following guidelines for others who are faced with the task of collecting a large amount of data from a distributed computing system:

- Test your data collection scripts in advance to ensure that all data will be collected properly. Ensure that no data will be missed or overwritten.
- Verify that your data collection does not interfere with network activity by collecting your files to the local disks of each workstation.
- Ensure that sufficient disk space is available for your collected data; limit your measurement collection if the available storage space is not adequate.
- Examine the intended usage of your data to ensure that you are collecting sufficient data from the system.
- Monitor your data collection closely so that problems can be rectified as quickly as possible; in this way, information loss will be reduced to a minimum.

The data that were collected by the measurement techniques that we have outlined in this chapter will be analyzed extensively in the following two chapters. The analysis in Chapter 4 will determine a specific subset of the large data set that has been collected in this chapter that is appropriate for inclusion in the model that we will design for the CDF system (in Chapter 6).