STORAGE SUBSYSTEM WORKLOAD CHARACTERIZATION AND SYNTHESIS

AHMAD A. J. ALI

STORAGE SUBSYSTEM WORKLOAD CHARACTERIZATION AND SYNTHESIS

by

Ahmad A. J. Ali

A Thesis Submitted to the Faculty of

The College of Engineering

in Partial Fulfillment of the Requirements for the Degree of

Master Science in Computer Engineering

Florida Atlantic University

Boca Raton, Florida

December 1995

STORAGE SUBSYSTEM WORKLOAD
CHARACTERIZATION AND SYNTHESIS

by

Ahmad A. J. Ali

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Imad Mahgoub, Department of Computer Science and Engineering and has been approved by the members of his supervisory committee. It was submitted to the faculty of The College of Engineering and was accepted in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering.

SUPERVISORY COMMITTEE:

Imad Mahgoub
Thesis Advisor

S. M. Rezaul Islam

Chairperson, Department of
Computer Science and Engineering

Dean, College of Engineering

Dean of Graduate Studies and Research

11/27/95
Date

ACKNOWLEDGEMENTS

I thank Allah that I am able to finish this work. I wish to express gratitude to my advisor, Dr. Imad Mahgoub, for his guidance, patience and encouragement throughout this work. My thanks to Dr. Rezaul Islam and Andy McNeill for their technical help that made this work possible. I also want to thank IBM Boca Raton for financial assistance as this work was completed under contract number N-UN-270-00 task #3. Many thanks are due to Dr. Gerry Merkel, Gerry Dixon, Ken Terlip and Dr. Mohammad Ilyas.

ABSTRACT

Author:	Ahmad Akhtar Jaweed Ali
Title:	Storage Subsystem Workload Characterization and Synthesis
Institution:	Florida Atlantic University
Thesis Advisor:	Dr. Imad Mahgoub
Degree:	Master of Science in Computer Engineering
Year:	1995

In this study, a tool has been developed which traces the storage subsystem workload at the subsystem level. Due to hardware assistance, this tool has very little overhead. This is achieved by incorporating the tracing routine in the microcode of the IBM SCSI adapter. The I/O traces are collected, in realtime, on a PS/2 connected to the modified SCSI adapter via an asynchronous link. These traces are then processed and studied.

We have developed a scheme to characterize the workload by studying the parameters of the traced workload. These parameters include LBA distributions, interarrival time distribution, size distributions, ratios between read requests and write requests, adapter's cache performance, etc. In this study, workload characterization of storage subsystems in OS/2, AIX and NetWare environments was performed. An algorithm for synthesizing of the workload was also developed and implemented as part of this study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Personal computers are becoming significantly powerful. The environment in which they are used has been changing dramatically in the last few years. With the availability of multitasking operating systems, current generation PCs are now used to process multiuser operations. To meet the demand for high performance, the processing speed of these PCs has been substantially improved, which also increases the demand on the storage subsystems. In an attempt to improve the performance of storage subsystem on one end, RAID technology is becoming more popular in PC systems [1]. On the other end, designers are developing intelligent storage subsystem controllers [2] and hardware cached storage devices [3] to further help the storage subsystem performance keep up with the ever increasing speed of CPUs. Clearly, the storage subsystem performance is a critical factor in the overall system performance.

In order to evaluate and improve the performance of existing and future storage subsystems for an environment, one needs to understand and characterize the workload as seen by the storage subsystem in that environment. The performance evaluation procedures [4, 5, 6] may use simulation, analytical modeling, or measurements of existing

storage subsystems. For all these evaluation methods there is a need for a drive workload which reasonably represents the actual workload, but in reduced form, e.g. less execution time or fewer I/O requests.

Storage subsystem workload is defined as collection of I/O requests that are serviced by the storage subsystem in a specified period of time. The term *workload characteristics* refers to demand placed on the various components of the subsystems. Example of these components are subsystem data transfer channel, drive head assembly, lookahead buffer in drive, etc.

1.1: RESEARCH MOTIVATION

For valid conclusions to be drawn from performance evaluations the drive workload must be representative of the actual workload. Many of subsystem modelling and analysis techniques use estimated workload or measure the workload at the file system level [4, 7, 8]. Most of the above mentioned workload measurement is done at filesystem of MVS operating system. Tools have been developed to measure workload at the file system of Unix [9] and OS/2 [10]. The workload measured at file system is independent of the device driver and the underlying physical storage subsystem. This workload is very helpful for performance evaluation at the operating system level. But of lesser help for physical (hardware) storage subsystem performance evaluation. Development of such tools requires detailed knowledge and source code of the operating system as software hooks have to be inserted in the operating system. The added hooks introduce significant

overhead and a separate tool has to be developed for each operating system of interest available for a particular platform. Figure 1.1 shows a simplistic view of access path to the storage subsystem from an application. It can be noted that a different implementation of a device driver can affect the workload as seen by the storage subsystem. Since, device drivers are considered to be part of the personal computer operating system, therefore workload measured at the storage subsystem would be more operating system independent. So, the workload measured at the storage subsystem level would be more helpful for storage subsystem hardware designers.

There is very little work done to characterize the workload at the physical storage subsystem in Personal Computer environment. In [8], Houtekamer has traced the workload at the MVS operating system in System/370 and then mapped it to physical storage subsystem. This indirect technique can be used to approximately characterize the workload at storage subsystem, but being operating system dependent, it would require knowledge of all operating systems of interest for a storage subsystem. An operating system independent tool that can characterize the storage subsystem workload is desired by storage subsystem designers. Such a tool can be used under different operating systems for a specific PC platform, instead of developing separate tools for each operating system that is available for a particular platform.

Lack of an existing operating independent tool for characterization of workload at the (hardware) storage subsystem level has motivated this research. The target of proposed

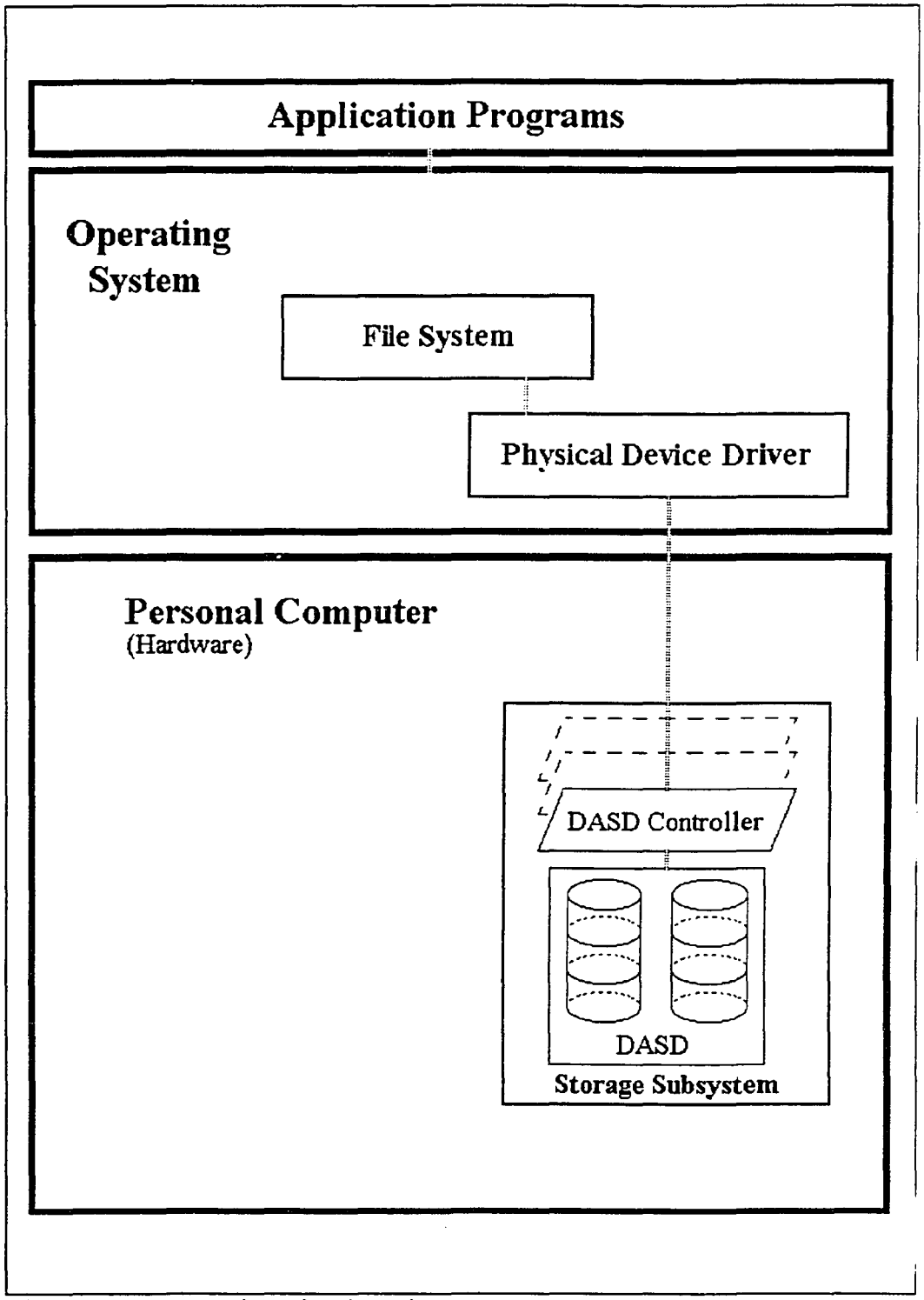

Figure 1.1: A simplistic view of access path to the storage subsystem in Personal Computer.

research is to develop tools and techniques to measure and characterize the storage subsystem workload independent of the operating system in PC systems.

1.2: CONTRIBUTION OF THIS RESEARCH

In this research, we have

- Developed operating system independent tools to measure, process and analyze the workload of the data storage subsystem in personal computers at the storage subsystem level.
- Proposed a scheme to characterize the workload.
- Developed an algorithm for the synthesis of the storage subsystem workload.

The Storage Subsystem Tracing Tool (SSTT) traces the storage subsystem workload at the storage subsystem level. Due to hardware assistance, this tool has very little overhead. These traces are then processed and converted into formats usable by other software tools. The workload characterization scheme is based on study of statistical parameters of the traced workload. These parameters include Logical Block Address (LBA) distributions, interarrival time distribution, size distributions, ratios between read requests and write requests, adapter's cache performance, etc. In this study, workload characterization of storage subsystems in OS/2*, AIX* (IBM's version of unfix) and NetWare** environments was performed.

---

* Registered trademarks of IBM Corporation.

** Trademark of Novell Corporation.

The algorithm developed for workload synthesis is based on a data point clustering method. The synthetic workload retains important characteristics of the original workload while its size is reduced significantly. This saves CPU time when running storage subsystem simulations during the design validation cycle. It is also possible to map the workload for newer subsystems or environments during the synthesis of workload and this greatly helps understanding the expected workload which may not be available during development. Finally, the synthetic workload (or even the originally traced workload) can be used to drive prototypes of new storage subsystem during the engineering validation cycle. This would again save a significant amount of time and resources.

1.3: THESIS ORGANIZATION

SSTT for collecting the workload traces is described in chapter 2. Software tools to process the collected traces and analyze the workload are also described in the same chapter. In chapter 3, we have presented the workload characterization of storage subsystem in various environments. In chapter 4, the representative synthetic workload generation and validation is presented. Conclusion and future directions are included in chapter 5.

CHAPTER 2

DESCRIPTION OF TOOLS DEVELOPED

In this chapter, we discuss the design and implementation of a real time, operating system (OS) independent, tracing tool for storage subsystems in personal computers (PCs). We then discuss tools developed to post process and analyze the traced workload. Tracing storage subsystems involves measuring the workload as seen by the storage subsystem. One way to achieve this is to incorporate software hooks in the OS (e.g. DEKKO* for OS/2 [10]). This enables the monitoring of I/O requests submitted to the storage subsystem as well as collection and storage of information about these requests for later analysis. This requires detailed knowledge of the operating system, and the added hooks increase the OS kernel overhead. Moreover, different sets of code have to be developed to support these hooks in different operating systems. Undoubtedly, this can be tedious, expensive, and time-consuming process. Another way of collecting same information is to trace the workload at the storage subsystem level. The advantages of this approach over the previous one are that the tool becomes operating system independent and due to hardware assistance it introduces minimal overhead.

---

* Trademark of IBM Corporation.

The proposed tracing tools design is based on the later approach. In this design, we modify the storage subsystem controller such that for each I/O request it receives from the host system, a packet of a few bytes of information (trace) about that request is sent to a Data Collection Station (DCS).

In our implementation of this design, we have modified an existing SCSI Host Adapter. The design and implementation of this tool is discussed in section 2.1. Description of tools developed for post processing is included in section 2.2. Section 2.3 presents workload analysis tools.

2.1: STORAGE SUBSYSTEM TRACING TOOL (SSTT)

The SSTT consists of a modified storage subsystem controller and a data collection station (DCS). The DCS is essentially a PS/2 running a data collection and storage program.

2.1.1: MODIFIED STORAGE SUBSYSTEM CONTROLLER

At this time, almost all storage subsystem controllers have a microprocessor on board. Tracing of workload at the storage subsystem level can be achieved by modifying the firmware (micro-code) of a controller and incorporating a port, or a separate I/O channel, on the controller. For each I/O request submitted by the host PC to the storage subsystem, the following information can be collected:

- Type of request (i.e., read, write)

- Logical Block Address (LBA)
- Request size (in blocks)
- ID of the target Direct Access Storage Device (DASD)
- Time when this request arrived at the controller and
- Status of controller cache (hit/miss).

Firmware modification should be such that, while processing a request, the controller's processor would store the desired information bytes at an address that is not being used during its regular activities. This address could be mapped as an input port of a peripheral I/O (PIO) chip. The stored bytes could then be sent by the PIO chip to the DCS. Time stamping of the traces could be done, if possible, by the PIO circuitry, otherwise it could be done at the DCS.

In our implementation of this design, we have modified an existing 32-bit IBM PS/2 MicroChannel SCSI Host Adapter with 512KB cache [11]. Figure 2.1 shows a simplified block diagram of such a card.

This card has a built-in serial port, normally used for diagnostic purposes. In the presence of this serial port we did not have to make any hardware modifications on the card; rather firmware of the card was modified. The modified code stores 5 bytes of information, in controller's RAM, about the request while processing it. One byte sync mark is appended to these 5 bytes and this 6 byte packet of information is transmitted through the port.

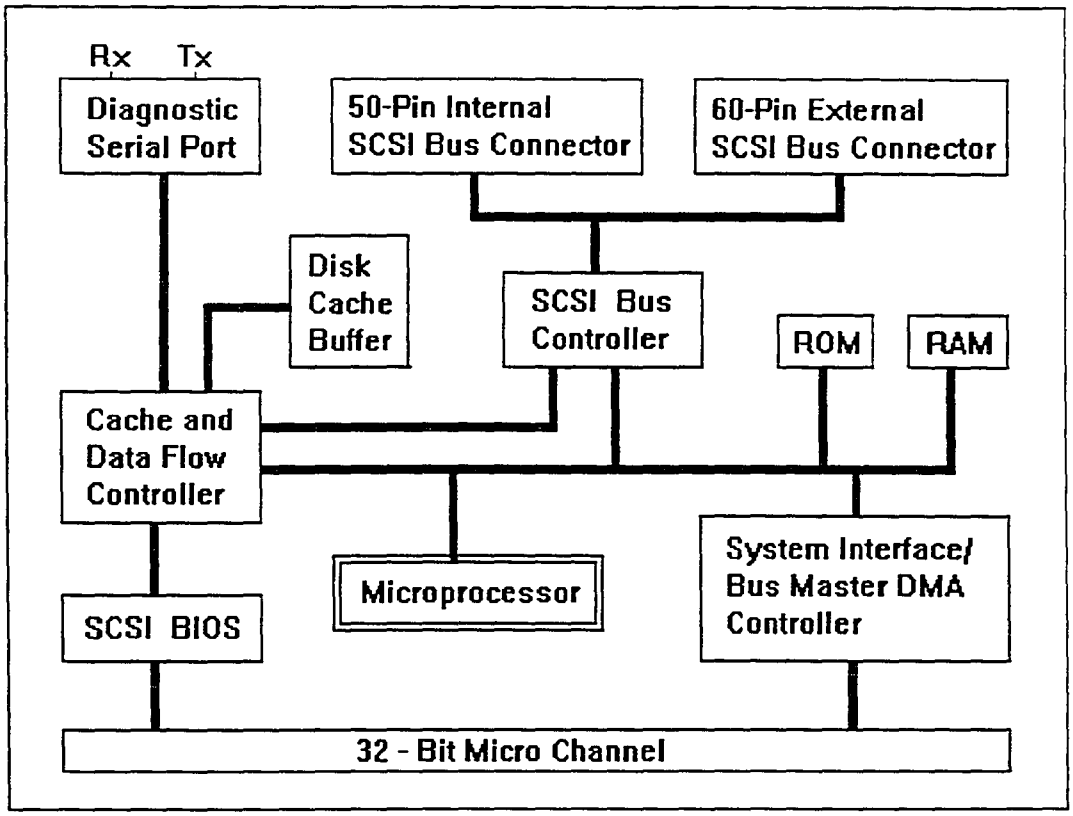

Figure 2.1: Block Diagram of the Micro Channel SCSI Adapter with cache.

Figure 2.2 shows a Generalized Stochastic Petri Net (GSPN) Model [12, 13] of the card's firmware, and Table 2.1 has a brief description of the GSPN. All the points marked by an asterisk (*) show places where code is modified/added. This modified/added code sends a 6-byte packet of information through the diagnostic serial port at 38400 baud, 8 data bits, 1 start bit and no parity bit. Figure 2.3 shows the structure of this information packet. The packet consists of following fields:

- Sync Mark (1 Byte): Declaring beginning of a new packet.
- Logical Block Address (3 byte): Showing beginning LBA of the request. The

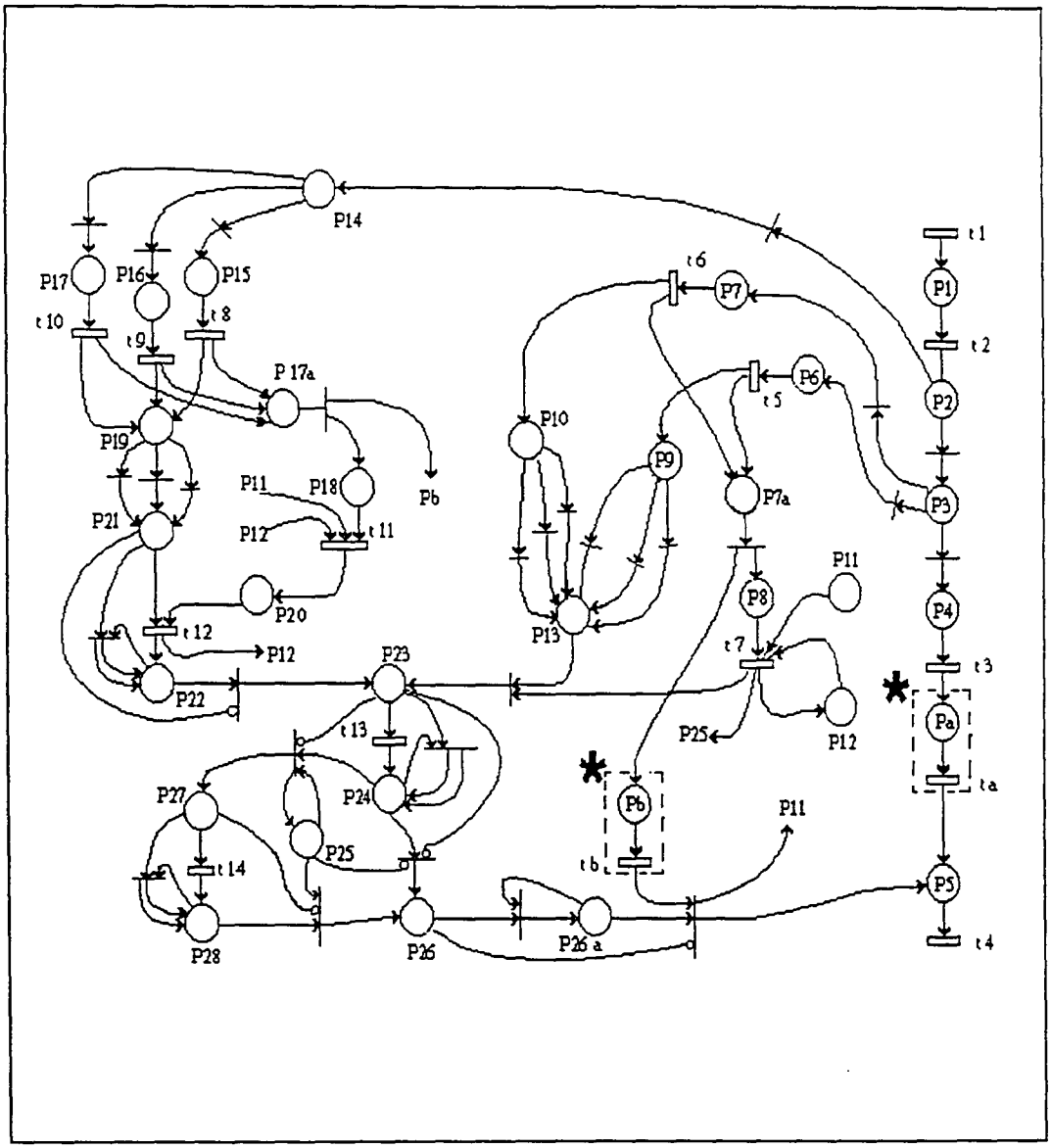

Figure 2.2: GSPN model of IBM SCSI attachment controller.

p1	arrival of a job
p2	select read or write
p3	select read hit, read miss or bypass
p4	read hit
p5	job done
p6	read cache bypass
p7	read cache miss
p7a	no op
p8	get drive, SCSI bus and send msg & cmd
p9	select data xfer length
p10	select prefetch length
p11	SCSI bus available
p12	drive available
p13	no op
p14	select write cache bypass delete update
p15	write cache bypass
p16	write cache delete
p17	write cache update
p17a	no op
p18	get drive, SCSI bus and send write cmd
p19	select xfer length
p20	write data xfer ready
p21	write data xfer waiting for cmd complete
p22	no op
p23	start drive (seek, latency and xfer)
p24	drive complete
p25	house-keeping (no op)
p26	house-keeping (no op)
p26a	no op
p26b	no op
p27	start read data xfer
p28	read data xfer complete
pa	send trace packet
pb	send trace packet
t1	inter-arrival time
t2	preprocessing
t3	overhead read hit
t4	post processing
t5	overhead read cache bypass
t6	overhead read miss
t7	read cmd and msg time
t8	write cache bypass overhead
t9	write cache delete overhead
t10	write cache update overhead
t11	write cmd time
t12	write data xfer time
t13	drive time
t14	read data transfer time
ta	trace packet xfer time
tb	trace packet xfer time

Table 2.1: Description of the GSPN model in Figure 2.2.

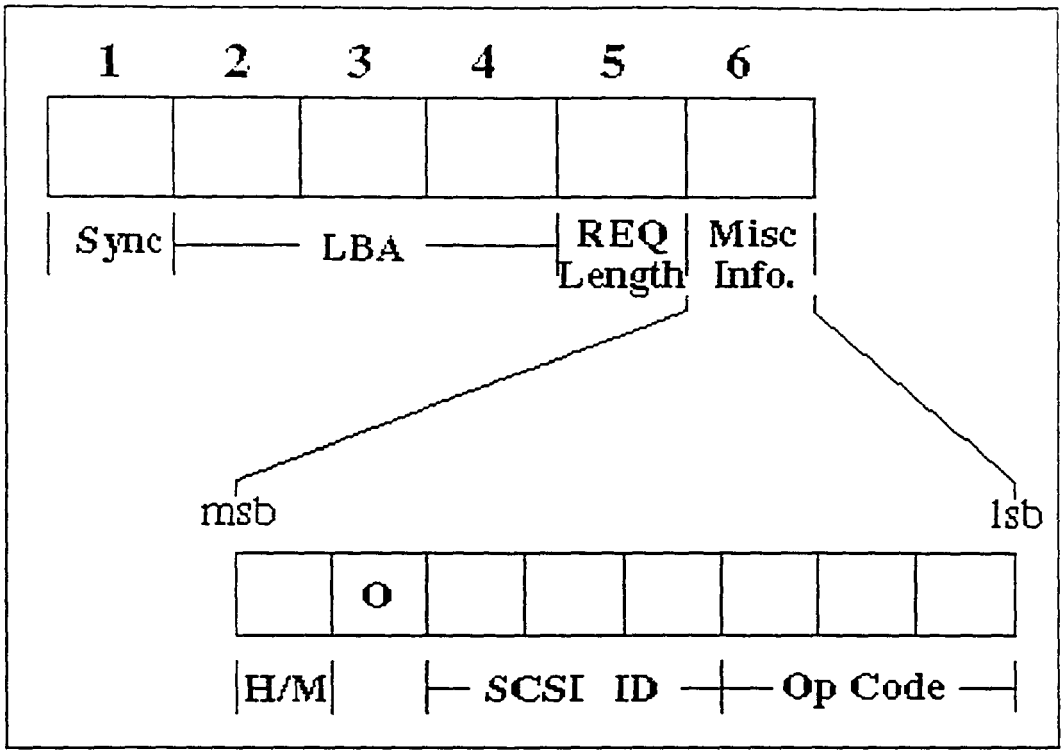

Figure 2.3: Structure of a Workload Trace Packet.

LBA field is big enough to represent an address space of 16 Mega Blocks. For a 512 byte block this translates to 8GB of storage capacity.

- Request Length (1 Byte): Showing length or size of the request in blocks.
- Miscellaneous Information (1 Byte): This byte has 1 bit (bit7) indicating adapter cache hit/miss (H/M), next bit (bit6) is always zero. 3 bits (bit5-bit3) represent the SCSI device ID and the last 3 bits (bit2-bit0) show the request opcode.

For each byte of data 10 bits (1 start bit, 8 data bits and 1 stop bit) are transmitted. Therefore, the transmission time of a packet (containing N bytes) at the rate of R baud

(bits/second) can be given as following,

$$Transmission\ time\ per\ packet = 10N/R \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{Equation 2.1}$$

From above, the transmission time for a 6 byte packet at 38000 baud is 1.56 msec. In all cases, except read hit in the controller cache, transmission time of the packet is overlapped by the SCSI device request service time (see Figure 2.2). Therefore, in a worst-case situation (i.e., read hit in controller cache) the modified/added code introduces a maximum 1.56 msec overhead. Depending upon SCSI bus contention and DASD response, other scenarios introduce overhead values of much less then 1.56 msec.

The extra code introduced above causes some tracing overhead, which can increase the service time of the storage subsystem. We estimated the increase in the service time using PC Magazine Laboratory Benchmark Series (Release 5.6). We used an IBM PS/2 Model 95 (80486 25 Mhz) with two different types of hard drives (C and D) to run the Disk Performance Tests of the benchmark with and without the modified microcode. The benchmark basically,

- Computes an average time for a DOS access. This average is computed for accesses to different sectors of the drive.
- Then it performs DOS file access operations involving reading/writing records in sequential and random patterns. It reports results for both sequential and random access operations. The DOS file access test is performed for different sizes of

records. The results shown are average time in seconds to perform each part of the test (the average is taken for 5 iterations).

Table 2.2 shows computed results for drive 'C' and table 2.3 shows results for drive 'D' as reported by the benchmark. The following are few observations based on the PC Magazine Benchmark results.

- The maximum overhead of the additional code for DOS disk access is less than 3%.
- For DOS files access operations involving 512 byte records, the overhead of the added code is very significant (37.86% to 44.63%) for both sequential and random read operations. For random write operations the overhead is less than 9% and around only 2% for sequential write operations. Apparently, for 512 byte read requests we have a higher rate of cache hit and as explained earlier, for a read cache hit, the added code introduces a maximum overhead.
- For DOS file access operations involving 4KB records, the overhead for sequential read and random read operations is less than or equal to 9% and 20%, respectively. For other operations the overhead is very minimal and strangely on drive D the overhead for random write operations is a little negative (-0.83%). This may be due to randomness or due to rounding real numbers in computations of delay and resolution or clock used by the benchmark. The benchmark reports results only upto 2 digits after the decimal and the difference in this case is 0.01.

DISK PERFORMANCE TESTS FOR DRIVE 'C':	with modified code (sec)	without modified code (sec)	% of over-head
DOS DISK ACCESS	27.29	26.85	1.64
DOS FILE ACCESS (SMALL RECORDS)			
512 RECORDS/512 BYTES EACH			
File Create	8.97	8.97	0.00
Sequential Write	9.98	9.82	1.63
Sequential Read	2.57	1.80	42.78
Random Write	13.26	12.49	6.16
Random Read	2.65	1.86	42.47
*TOTAL (512*512)*	*37.43*	*34.94*	*7.13*
64 RECORDS/4096 BYTES EACH			
File Create	1.67	1.65	1.21
Sequential Write	1.49	1.49	0.00
Sequential Read	0.50	0.46	8.7
Random Write	1.85	1.81	2.21
Random Read	0.38	0.33	15.15
*TOTAL (64*4096)*	*5.89*	*5.74*	*2.61*
TOTAL (SMALL RECORDS)	*43.32*	*40.68*	*6.49*
DOS FILE ACCESS (LARGE RECORDS)			
16 RECORDS/16384 BYTES EACH			
File Create	0.84	0.79	6.33
Sequential Write	0.70	0.70	0.00
Sequential Read	0.43	0.43	0.00
Random Write	0.80	0.79	1.27
Random Read	0.76	0.77	-1.30
TOTAL (16/16384)	*3.53*	*3.48*	*1.44*
8 RECORDS/32768 BYTES EACH			
File Create	0.77	0.75	2.67
Sequential Write	0.56	0.56	0.00
Sequential Read	0.44	0.43	2.33
Random Write	0.63	0.62	1.61
Random Read	0.58	0.59	-1.69
TOTAL (8/32768)	*2.98*	*2.95*	*1.02*
TOTAL (LARGE RECORDS)	*6.51*	*6.43*	*1.24*
TOTAL (SMALL & LARGE RECORDS)	*49.83*	*47.11*	*5.77*

Table 2.2: Disk 'C' performance as reported by the benchmark

DISK PERFORMANCE TESTS FOR DRIVE 'D':	with modified code (sec)	without modified code (sec)	% of over-head
DOS DISK ACCESS	22.96	22.31	2.91
DOS FILE ACCESS (SMALL RECORDS)			
512 RECORDS/512 BYTES EACH			
File Create	7.35	7.33	0.41
Sequential Write	7.77	7.60	2.19
Sequential Read	2.56	1.77	44.63
Random Write	10.66	9.79	8.89
Random Read	2.84	2.06	37.86
*TOTAL (512*512)*	*31.18*	*28.54*	*8.47*
64 RECORDS/4096 BYTES EACH			
File Create	1.13	1.13	0.00
Sequential Write	1.04	1.04	0.00
Sequential Read	0.54	0.50	8.00
Random Write	1.19	1.20	-0.83
Random Read	0.37	0.31	19.35
*TOTAL (64*4096)*	*4.27*	*4.18*	*2.15*
TOTAL (SMALL RECORDS)	*35.45*	*32.72*	*7.70*
DOS FILE ACCESS (LARGE RECORDS)			
16 RECORDS/16384 BYTES EACH			
File Create	0.46	0.46	0.00
Sequential Write	0.38	0.38	0.00
Sequential Read	0.34	0.33	3.03
Random Write	0.44	0.44	0.00
Random Read	0.45	0.45	0.00
TOTAL (16/16384)	*2.07*	*2.06*	*0.48*
8 RECORDS/32768 BYTES EACH			
File Create	0.39	0.38	2.63
Sequential Write	0.28	0.28	0.00
Sequential Read	0.25	0.24	4.17
Random Write	0.32	0.33	-3.03
Random Read	0.30	0.31	-3.23
TOTAL (8/32768)	*1.54*	*1.54*	*0.00*
TOTAL (LARGE RECORDS)	3.61	3.60	0.28
TOTAL (SMALL & LARGE RECORDS)	*39.06*	*36.32*	*7.54*

Table 2.3: Disk 'D' performance as reported by the benchmark

- For DOS file access operations, involving 16KB and 32KB, the overhead of added code is very minimal (3%) except for sequential read operations where it is upto 4.17%. Here, again, we see some negative overheads which may be explained as discussed previously.

The PC Magazine Laboratory Benchmark used in performing the tests is primarily for measuring the performance index in a DOS environment and can be influenced by the OS characteristics. Therefore, we wanted to use some tool that would provide a similar performance index without being influenced by the OS. The SCSI Exerciser program, developed by IBM engineers for IBM internal use, works at the BIOS level and gives full control of the SCSI device under test. Results obtained using the SCSI Exerciser on a third hard disk drive (drive E) in the same IBM PS/2 model 95, are shown in Table 2.4.

From the numbers and figures obtained using PC Magazine Benchmark and the SCSI Exerciser, it can be concluded that:

- Any time there is a SCSI device access the impact of the additional code is almost invisible (< 0.33 ms or 1.7 %).
- For small read requests that are served from the controller's cache one sees a maximum impact. This is in agreement with the intuitive estimates obtained by inspecting the GSPN model of the controller card.

It should be noted that in most of the systems the demand on the controller is much

below its throughput capacity, and a realistic workload does not consist of a high percentage of read hits at the controller cache level. Therefore, most of the time the overhead due to the modified/added code would be less than 0.33 msec.

	with modified code (msec)	without modified code (msec)	% of over-head
NO CACHE BYPASS (SMALL REQUESTS)			
READ SAME LOCATION			
512 Byte requests	51.04	36.23	40.88
4096 Byte requests	56.8	41.13	38.10
8192 Byte requests	63.54	47.3	34.33
WRITE SAME LOCATION			
512 Byte requests	139.33	139.33	0.00
4096 Byte requests	139.50	139.35	0.11
8192 Byte requests	141.1	139.51	1.14
SEQUENTIAL (LBA=LBA+18) READ			
512 Byte requests	82.2	82.2	0.00
4096 Byte requests	93.6	93.6	0.00
8192 Byte requests	107.1	107.1	0.00
SEQUENTIAL (LBA=LBA+18)WRITE			
512 Byte requests	194	193.53	0.24
4096 Byte requests	195.09	193.6	0.77
8192 Byte requests	196.88	193.6	1.69
CACHE BYPASS (16384 BYTE REQUESTS)			
SEQUENTIAL REQUESTS (LBA=LBA+32)			
Reads	12.19	12.19	0.00
Writes	23.83	23.82	0.04
RANDOM REQUESTS			
Reads	38.06	37.8	0.69
Writes	37.95	37.8	0.40

Table 2.4: Disk performance as reported by the SCSI Exerciser

2.1.2: DATA COLLECTION STATION

The Data Collection Station, as described earlier, receives traces sent by the modified storage subsystem controller (tracing card) and stores them (in a hard disk drive) for later analysis. A DCS should be fast enough to handle the trace collection activity and it must have:

- An I/O port that can be connected to the storage subsystem controller's trace output port
- Storage capacity to store traces for a reasonable length of time, and
- If time-stamping of traces is not done at the controller level then a high resolution timer (at least 1 msec) is also required for time-stamping.

In our implementation of a DCS we found that a 386-20MHz-based PS/2 with a hard disk drive of 80MB can be programmed to meet the requirements of a DCS. It has a built-in serial port (COM1) that can easily be programmed and connected to the tracing controller's serial port using a special cable. Its system clock can be programmed to give a resolution of 1 msec. It has enough processing power to receive the traces at a high rate, time-stamp them and store them on the hard disk. The description of the data collection program (called COLLECT) running on the DCS follows.

The serial port (COM1) is programmed to give an interrupt (0x0C) after receiving each byte of the information from the card. The COM1 is programmed at 38400 baud, 8 data bit, 1 start bit and no parity bit. It is achieved by loading different registers of the serial

port controller [14] as listed in table 2.5.

```
PORT Hex 03F9  := Hex 00   ;Disable data recv'd interrupt.
PORT Hex 03FB  := Hex 80   ;Enable divisor latch.
PORT Hex 03F8  := Hex 03   ;Divisor Low byte.
PORT Hex 03F9  := Hex 00   ;Divisor high byte.
PORT Hex 03FB  := Hex 03   ;Disable divisor latch.
                           ;No parity, 1 stop bit and
                           ;8 data bits.
PORT Hex 03FC  := Hex 0B   ;Enable COM1 interrupts.
PORT Hex 03F9  := Hex 05   ;Enable data receive interrupt
                           ;and line error interrupt.
PORT Hex 03FA  := Hex 00   ;Disable FIFO.
PORT Hex 0021  := Hex 00   ;Interrupt mask register reset.
```

Table 2.5: Setting registers of serial port controller.

The existing clock in a PC gives a resolution of about 54.9 msec (= 3,600,000 ÷ 65,536). In an IBM PC (or compatible) a timer/counter chip issues an interrupt (timer tick) every 54.9 msec. The interrupt controller forwards it as interrupt number 8 to the system processor, and the BIOS handler [15] services it as follows:

- Keeps a 4-byte count of timer ticks at memory location 0004:006C
- Wraps it to zero after 24 hours (1,573,024 ticks)
- Controls diskette drive motor
- Issues a software interrupt hex 1C.

Originally the timer/counter chip generates a timer tick by dividing 1.193 MHz input frequency by 65536 [16] to give an output frequency of about 18.2 Hz (1/18.2 = 0.0549 sec). For a 1 msec (1 KHz) resolution we need a frequency divider value of 1193. This

value can be loaded in the latch register of the timer/counter by accessing port hex 40 (low byte first).

We also have replaced the BIOS interrupt handler for interrupt 8 with our code, which simply keeps a 4-byte count of timer ticks at memory location hex (B800:1F50). The following are the reasons for replacing the BIOS routine:

- The CPU has to execute it 1000 times a second instead of about 18.2 times a second, so we want its execution time to be smaller.
- We do not need the interrupt hex 1C.
- We do not want the count of timer ticks to wrap after 1,573,024 (hex 1800B0) ticks (26.22 min at 1 msec ticks).
- We do not use the diskette drive during the trace collection.
- DOS reads the Time-of-Day (timer tick count) when closing files and, if the tick count has exceeded 1,573,024, then it gives a divide-by-zero error. Therefore, Time-of-Day location (0004:006C) is not used; rather a different memory location is used for the count of ticks.

Trace reception is interrupt-driven. The serial port controller gives interrupt 0x0C after receiving each byte from the tracing card. The interrupt service module performs the following tasks:

- Checks if circular buffer is full; if so then it givers an error message and drops the

data byte and is ready to service the next interrupt.

- Moves the data byte to a circular buffer (32KB)
- Checks if the data byte was a sync mark (see Figure 2.3)
- If it was a sync mark then the interrupt handler writes a 4-byte time stamp in the circular buffer just after the sync mark. It uses the modified system clock (1 msec resolution) for time stamping.

Data storage is done by a module of COLLECT running in the foreground during the trace collection. This module is always checking whether there is any data in the circular buffer. If so, then it removes the data from the circular buffer and writes it in a series of files on the hard disk. As a precaution against data loss due to power or any other failure, a data file is closed if no data is received for 10 minutes or the size of the data file has reached 256KB.

2.2: POST PROCESSING TOOLS

The data collected by the program COLLECT is in binary form and called "raw data". The raw data may have some inconsistencies due to,

i) Time stamps caused by data bytes that are same as sync mark,

ii) Serial communication line errors (voltages driven by the card are between 0v and 5v), or

iii) COM1 or circular buffer overflow.

The raw data is processed in two steps to convert the collected traces into a readable ASCII format. In the first step binary trace data is extracted from the raw data and in second step this data is converted to an ASCII format.

XTRACT

This is a smart software tool developed to extract data from the collected raw data. It removes,

- o time stamps that were placed after data bytes that are same as sync mark,
- o erroneous bytes introduced due to serial communication line errors,
- o incomplete packets due to port or buffer overflow.

It looks for the first sync byte. Then it validates this sync looking byte (i.e. data byte that is same as sync mark) by examining the packet following it and the next sync in the following manner.

i) Find next sync looking byte. Assume that the byte found is byte1 (see Figure 2.3) of the packet.

ii) Next four bytes constitute the present time stamp, skip them.

iii) Byte2 through byte6 are picked using following test;

IF present byte is sync looking

THEN Position of next byte = Position of present byte + 5

ELSE Position of next byte = Position of present byte + 1.

iv) IF next byte after byte6 is sync looking then previous packet is valid and go to step2 else go to step1, starting after the previously assumed sync mark.

v) If program finds four consecutive sync looking bytes which fail the test then it gives a menu driven control to the user and user has to point towards the next valid sync mark.

vi) When time stamp of a packet is greater then the time stamps of preceding and proceeding packets then it is considered inconsistent and the packet is dropped.

When XTRACT step is successfully completed the data is in usable binary form called FIX form. The size of a FIX file is always multiple of ten bytes.

BNJ

It converts traces from binary FIX format to BNJ (an ASCII) format. The BNJ format has six fields as following,

1. OpCode
2. LBA accessed
3. Request size in blocks
4. Inter arrival time in milliseconds
5. SCSI id of the target device
6. Controller cache hit/miss

All the fields are separated by single space. The program computes inter arrival time for a request by subtracting the previous time stamp from the present time stamp. The inter arrival time for the first request is assumed to be 10 msec as there is no previous packet and hence no previous time stamp.

For example, if the host makes a 3-block read request going to drive 6 at an LBA of 23642, and there is a controller cache hit (1) and 31 msec has elapsed since the last request was made by the host, then this would be traced as "R 23642 3 31 6 1". Similarly, if the next request comes after 17 msec going to drive 4 at an LBA of 316378 for writing 23 blocks, and this one causes a controller cache miss (0), then this would be traced as "W 316378 23 17 4 0". Or when put together these would look as follows,

```
R 23642 3 31 6 1
W 316378 23 17 4 0
```

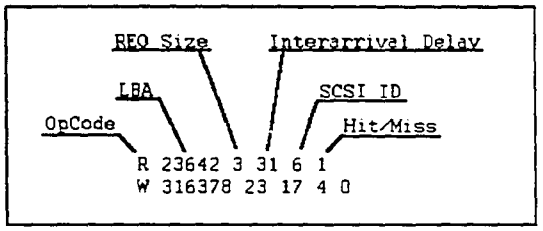

Figure 2.4: BNJ format traces.

These workload traces can be directly fed to different programs for statistical analysis, workload characterization, DASD simulations and cache algorithm performance analysis.

2.3: ANALYSIS TOOLS

STATS:

As the name hints this program performs the statistical analysis of the traces. It takes traces in BNJ format and computes different statistical characteristics of the traces and gives results in ASCII form which are easy to read and can be fed to program like Lotus 123, Quatro and FreeLance for graphical presentations of characteristics of interest. The program STATS basically computes,

- Inter arrival time frequency, density and distribution
- LBA frequency, density and distribution
- Load distribution across drives for each hour. The load distribution shows the number of requests for each hour and read to write ratio and the ratio of controller cache hit and miss.
- It also gives the ratios in terms of overall values.
- The resolution of inter arrival time and LBA is user selectable thus giving option of zooming-in statistically.

The DASD Animator (SHOWFIX):

This is a graphical animation tool giving a visual presentation of a DASD responding to the workload. It shows the pattern of head movement across different sectors in a linear DASD address space. The user has option of selecting and focusing on certain address range or type of transaction request. This program requires that input trace file binary FIX formatted and the display monitor be at least a VGA monitor.

2.4: OTHER RELATED TOOLS

BNJ2FIX:

This program is used to convert the trace file from BNJ (ASCII) format to FIX (binary) format. It is primarily used when trace files in BNJ form are desired to be fed to the DASD animator. It recreates time stamp by simply summing up all the inter arrival times up to that trace. The recreated timestamps may not be same as those of the original FIX file.

CUTCOPY:

This tool copies down a chunk of traces from a binary trace file into a new trace file. It is usually needed when certain portion of traces requires a very close study and the big size of trace file is causing problems. It can also be used to break a trace file into multiple files due to distinct changes in the workload.

FILEMIX:

When the traced system has multiple controllers then there are multiple trace files, each with its own sequence of time stamps. This utility merges multiple trace files into one file reordering traces as per their timestamps. It is important to start tracing all controllers at the same time otherwise we need to specify an offset time for each file. Since this utility works with timestamps it can only process trace files in FIX format. Although FIX formatted files can be recreated using BNJ2FIX but this may falsely imply that each controller handled a transaction at the beginning time.

GAPFIND:

This utility is used to help find interarrival times larger than the given threshold. The threshold is a user specified parameter in milliseconds. It shows the value of time stamps and sequence number of the transaction coming after an inter arrival time greater than the threshold.

RATECALC:

This program computes the maximum instantaneous load on the storage subsystem in terms of transactions per second. It slides a window of user defined number of transactions over the trace file. That is it finds the minimum time taken by a burst of 'x' requests, where 'x' is defined by the user.

LONGCOLL:

This is basically a modified COLLECT tool. The microcode of the controller is further modified to send a two byte packet at transaction arrival time then it sends a six byte information packet and then a two byte packet is sent at transaction completion time. The data transmission rate is 57600 baud. The two byte packets consist of a SYNC mark byte and an information byte. This information byte is 1110XYYY, where X represents the arrival (0) and completion (1) and YYY is the device id of the target DASD. The six byte info packet is same as in figure 3. The LONGCOLL puts a time stamp only when it receives a two byte packet. This helps compute the service time of the DASD. The tracing overhead with this code has not been determined but intuitively it should be a little higher

than the one computed previously. Using equation 2.1, the transmission time for 10 bytes at 57600 baud is 1.7361 millisecond, while for 6 bytes at 38400 baud it was 1.56 ms; i.e. an increase of 11%. This overhead increase has full impact in case of hit in controller cache on a read request.

LONGXTR:

This program is used to post process the binary raw data collected by the LONGCOLL. It combines the three packets of information, collected for each transaction traced by the modified controller, into an eleven byte packet. First ten bytes are same as of FIX format and the eleventh byte gives the service time in millisecond. This program has no capabilities to handle any error in the raw trace files.

PIKOUT:

This is a primitive utility to remove single byte from the binary trace file. It is sometimes needed when LONGXTR can not work due to some erroneous byte.

CHAPTER 3

WORKLOAD CHARACTERIZATION

Workload characterization for any subsystem involves measuring the workload as seen by the subsystem and then searching for some quantifyable patterns in the workload. In case of data storage subsystems of PCs, all requests coming down to the subsystem can be identified by,

- the type of request, i.e. read, write, etc,
- the size of the request, and
- the location/destination of the request on the storage media.

At the storage subsystem level no information about the operating system or the file system or even which request belongs to which task/file is available. The tracing tool, described in chapter 2, provides the information about all the requests coming down to the storage subsystem. We can characterize the storage subsystem workload by computing,

- The ratio of read requests to write requests.
- Distribution of request size frequency.

- Distribution of LBA frequency (or LBA footprint).
- Distribution of inter-request LBA-distances.
- Distribution of inter-arrival time frequency.
- Hourly demand on the system in terms of number of requests per hour.

The distribution of request size frequency gives an idea about demand on the data transfer channel. Distribution of LBA frequency shows how requests are distributed on the disk space. Knowledge of read-to-write ratio, request size frequency distribution and LBA frequency distribution helps us in performance analysis and optimization of storage subsystem level (controller level or device level) cache design. Distribution of inter-request LBA-distance shows demand on the head movement mechanism. A big number of large inter-request LBA-distances indicates that the workload demands heavy head movement. Distribution of inter-arrival time frequency reveals instantaneous demand on the storage subsystem. Demand on the system for each hour shows the trend of resource utilization over longer periods of time. This can help to redistribute the load to hours where the system is relatively less busy, thereby improving subsystem response time in busy hours.

In this chapter, the storage subsystem workload in Netware, OS/2 and AIX operating system environments is traced using the tracing tool (SSTT) described in the chapter 2. The traced workload is then characterized by obtaining the above mentioned distributions.

3.1: CHARACTERISTICS OF WORKLOAD TRACED IN REAL LIFE NETWARE ENVIRONMENT

In this section, we present the characteristics of the storage subsystem workload in a PS/2 file server running Novell Netware version 3.11. The site profile is given in Table 3.1. The storage subsystem traces containing detailed information about the I/O requests were collected at the storage subsystem level using the SSTT as described in chapter 2. These traces were later processed using tools described in chapter 2 to characterize the workload. The characteristics of the workload at the storage subsystem are summarized in Table 3.2. Traces were collected in three periods as follows:

- day-one for 22.56 hours.
- day-two for 23.65 hours.
- day-three for 99.09 hours.

Applications
Business: Database/Transaction
Office Support: E-Mail, Wordprocessor, and Spreadsheet
Representative Cycle Time: Generally majority of activity is during 8am to 5 pm business day.
Server Configuration
System: IBM PS/2 model 8595 (486/33MHz) with 20MB RAM.
Hard Drives: 2 320MB IBM SCSI drives connected to a single SCSI adaptor.
Network Adapter: IBM 16/4 Token Ring/A.

Table 3.1: The Site Profile.

Period:	day-one
Total Transactions:	117731 (Drive 0: 74594; Drive 1: 43137)
Read Operations:	32.4% (38178)
Adapter Cache hit in Read:	14.4% (45.5% of total read)
Write Operation:	67.6% (79553)
Adapter Cache hit in Write:	0.47% (0.7% of total write)
Mean Interarrival time:	683.52 msec (St. Dev. : 8124.90 msec)
Mean I/O Request size:	4.45 Sectors (St. Dev. : 3.41 Sectors)
Period:	day-two
Total Transactions:	742191 (Drive 0: 74054; Drive 1: 165)
Read Operations:	46.6% (34600)
Adapter Cache hit in Read:	20.6% (44.3% of total read)
Write Operation:	53.4% (39619)
Adapter Cache hit in Write:	7.31% (13.3% of total write)
Mean Interarrival time:	1147.04 msec (St. Dev. : 12228.42 msec)
Mean I/O Request size:	4.74 Sectors (St. Dev. : 3.39 Sectors)
Period:	day-three
Total Transactions:	180321 (Drive 0: 179746; Drive 1: 575)
Read Operations:	43.0% (77596)
Adapter Cache hit in Read:	21.8% (50.7% of total read)
Write Operation:	57.0% (102725)
Adapter Cache hit in Write:	7.86% (13.8% of total write)
Mean Interarrival time:	1978.18 msec (St. Dev. : 10228.42 msec)
Mean I/O Request size:	4.39 Sectors (St. Dev. : 3.38 Sectors)

Table 3.2: Summary of the storage subsystem workload.

3.1.1: I/O REQUEST SIZE DISTRIBUTION

Figures 3.1, 3.2 and 3.3 show read and write I/O request size frequency distribution of day-one, day-two and day-three, respectively. Clearly, all reads are 8-sector requests and most of the writes (more than 60% of all write request) are one sector requests. Relative frequency of write requests exponentially drops with increasing request size up to 7-sector but increases at 8-sector. We observed a considerable number of 8-sector write requests on day-one. 8-sector write requests were more frequent than 7-sector request but much less frequent than 1 and 2 sector requests in day-two and day-three. I/O request size frequency distribution is presented in Table 3.3.

The read request to write request ratio is observed to range from 32.4:67.6 (day-one) to 46.6:53.4 (day-two). This is almost opposite of the 70:30 read to write ratio observed by the file system workload studies. This could be attributed to the huge file system caching. The file system, due to caching, could be filtering a significant number of read requests and doing lazy writes back to the storage subsystem. This notion is further strengthened by the observation (table 3.3) that more that 60% of write requests are small (1 block) in size while 44% - 50% of read requests (8 blocks each) resulted in adapter cache hit. From table 3.3, the total volume (in blocks) of data read and written by the storage subsystem can be computed. It is interesting to note that the ratio of blocks read to blocks written is 57:43, 78.6:21.4 and 78.5:21.5 for day-one, day-two and day-three respectively.

Period: day-one

Sector	1	2	3	4	5	6	7	8
Write	50746	6722	1261	802	415	270	169	19168
Read	0	0	0	0	0	0	0	38178
Total	50746	6722	1261	802	415	270	169	57346

Period: day-two

Sector	1	2	3	4	5	6	7	8
Write	28665	5641	791	445	292	232	160	3393
Read	0	0	0	0	0	0	0	34600
Total	28665	5641	791	445	292	232	160	37993

Period: day-three

Sector	1	2	3	4	5	6	7	8
Write	77792	14604	2105	1052	902	718	705	4847
Read	0	0	0	0	0	0	0	77596
Total	77792	14604	2105	1052	902	718	705	82443

Table 3.3: I/O Request Size Frequency Distribution.

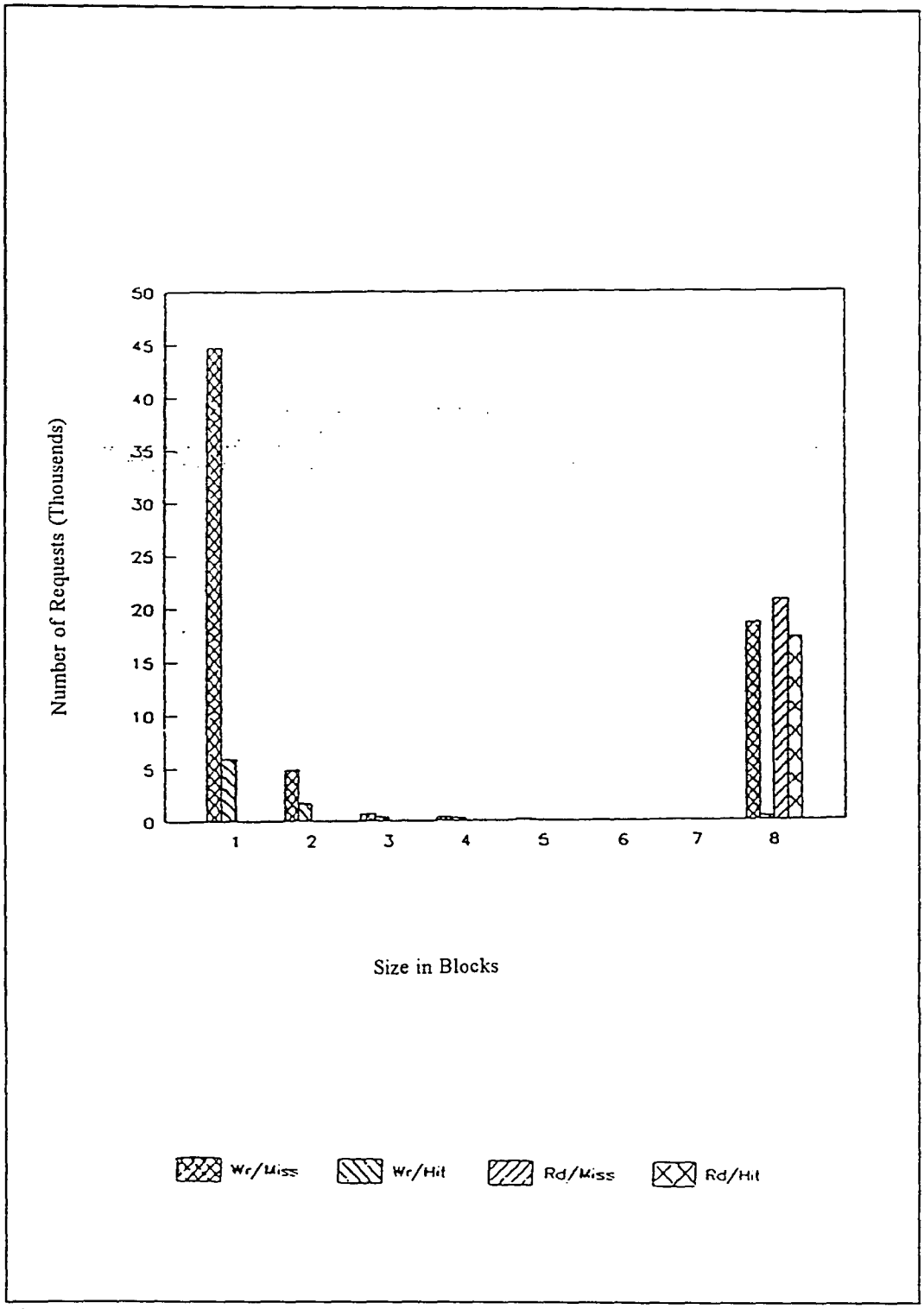

Figure 3.1: I/O request size distribution on day-one.

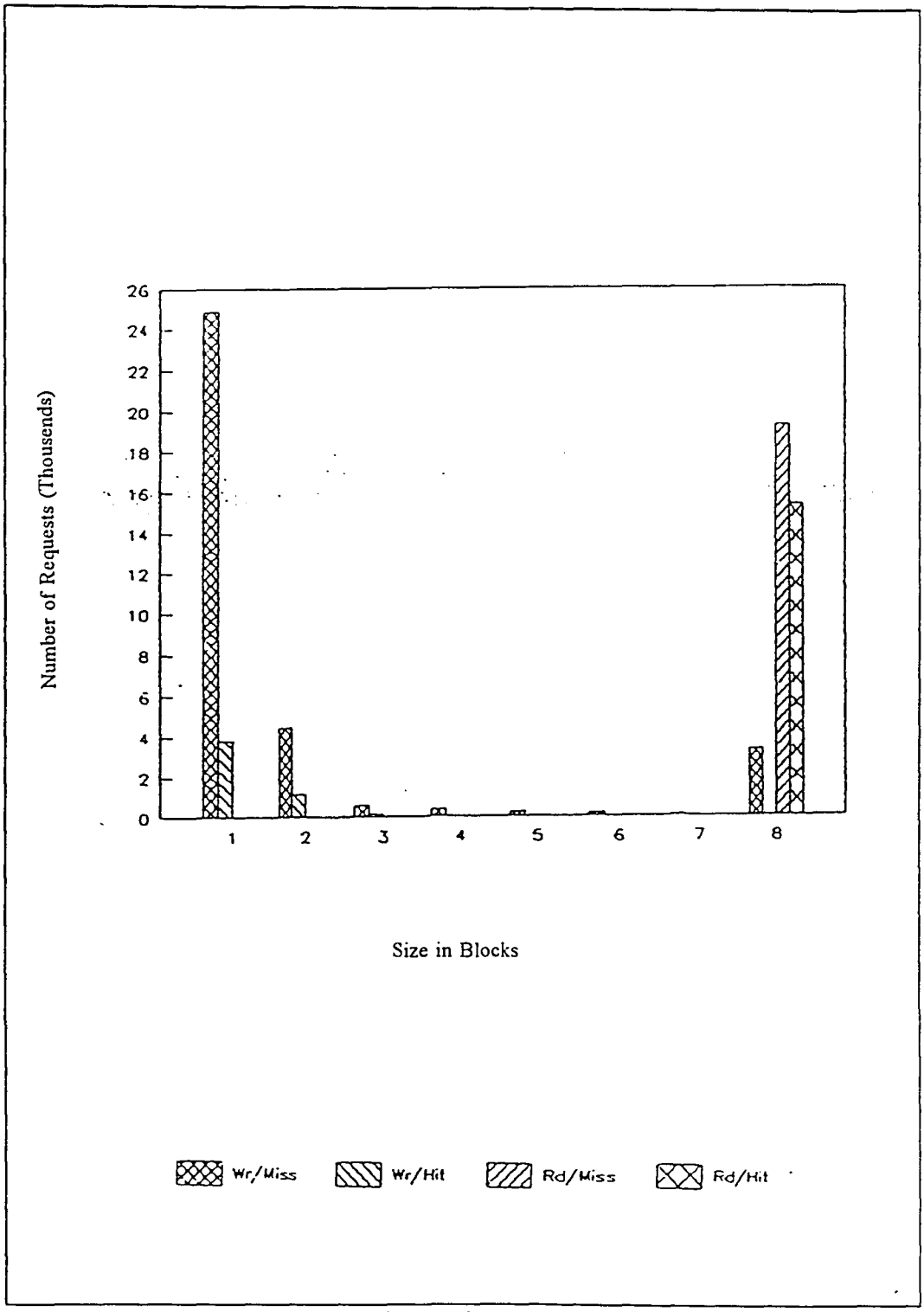

Figure 3.2: I/O request size distribution on day-two.

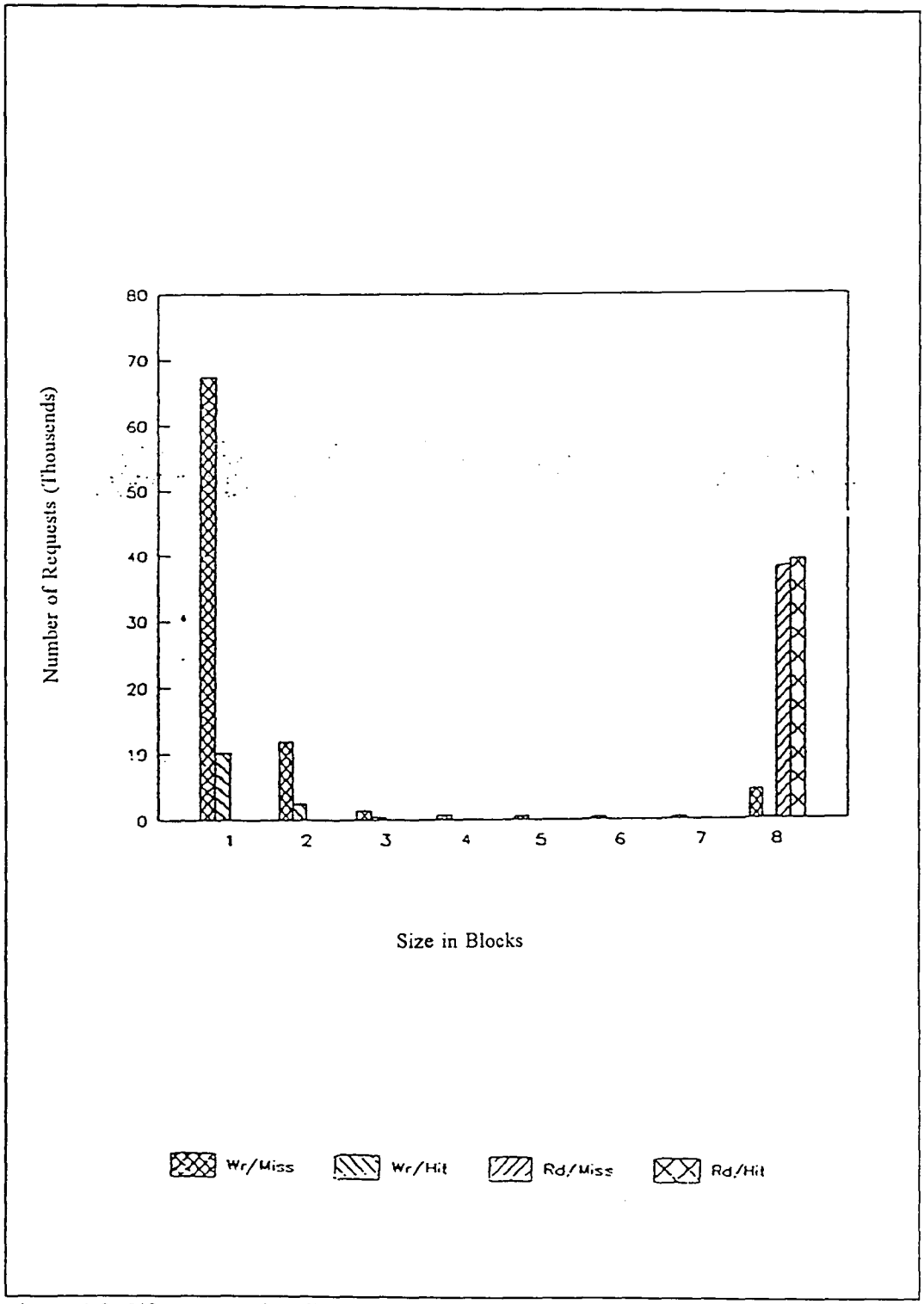

Figure 3.3: I/O request size distribution on day-three.

3.1.2: INTERARRIVAL TIME DISTRIBUTION

Interarrival time between successive requests in day-one, day-two and day-three are shown in figures 3.4, 3.5 and 3.6, respectively. Each of these figures shows two peaks, one around 20 msec and another around 70 msec.

It was also observed that in day-one both drives were active with drive 0 and drive 1 receiving 63% and 37% of the total requests, respectively. In day-two and day-three drive 1 was practically inactive and received less than 0.4% of the subsystem load (see Table 3.2). Note that since the storage subsystem does not support command queuing at the drive level, interarrival time is in fact approximately equal to the service time of the storage subsystem when there are request queued at the host device driver. This is because of the fact that the device driver running at the host does not send I/O request to the storage subsystem unless the drive is free and ready to serve them. Therefore, if the subsystem has only one active drive and queued requests pending the interarrival time exactly represents the service time of the subsystem when the host device driver overhead (which is normally negligible compared to storage subsystem service time) is subtracted.

It is interesting to note the mean and standard deviation of the interarrival time around peaks as presented in the Table 3.4. Mean and standard deviation of interarrival time on day-one is less than day-two and day-three due to two active drives (see Table 3.2). This fact is also evident from analyzing mean and standard deviation around peaks at 20 msec and 60 msec. As explained in previous paragraphs, the first peak around 20 msec

represents the subsystem response time which is summation of delays in the SCSI adapter and drive(s).

Period	Only First Peak (≤40 msec) Mean/St Dev.	Only Second Peak (≥55 msec and ≤70 msec) Mean/St Dev.	Both First and Second Peaks (≤70 msec) Mean/St Dev.
day-one	19.24/6.70	60.79/3.22	24.86/17.52
day-two	20.48/6.80	59.65/2.94	32.98/18.98
day-three	20.37/7.35	60.20/3.09	33.74/19.51

Table 3.4: Mean and Standard Deviation of Interarrival Time

However, the second peak at the 60 msec is due to sequential (mostly) read requests at routine backup operation. Since backup operations are normally done to a relatively slow device such as tape or optical drive the interarrival times are relatively large. In addition, the st. de. of interarrival time around 60 msec (≥ 55 msec and ≤ 70 msec) is also small due to sequential read request which get a large number of hits in the SCSI adapter cache and almost deterministic interval between successive requests from the backup device.

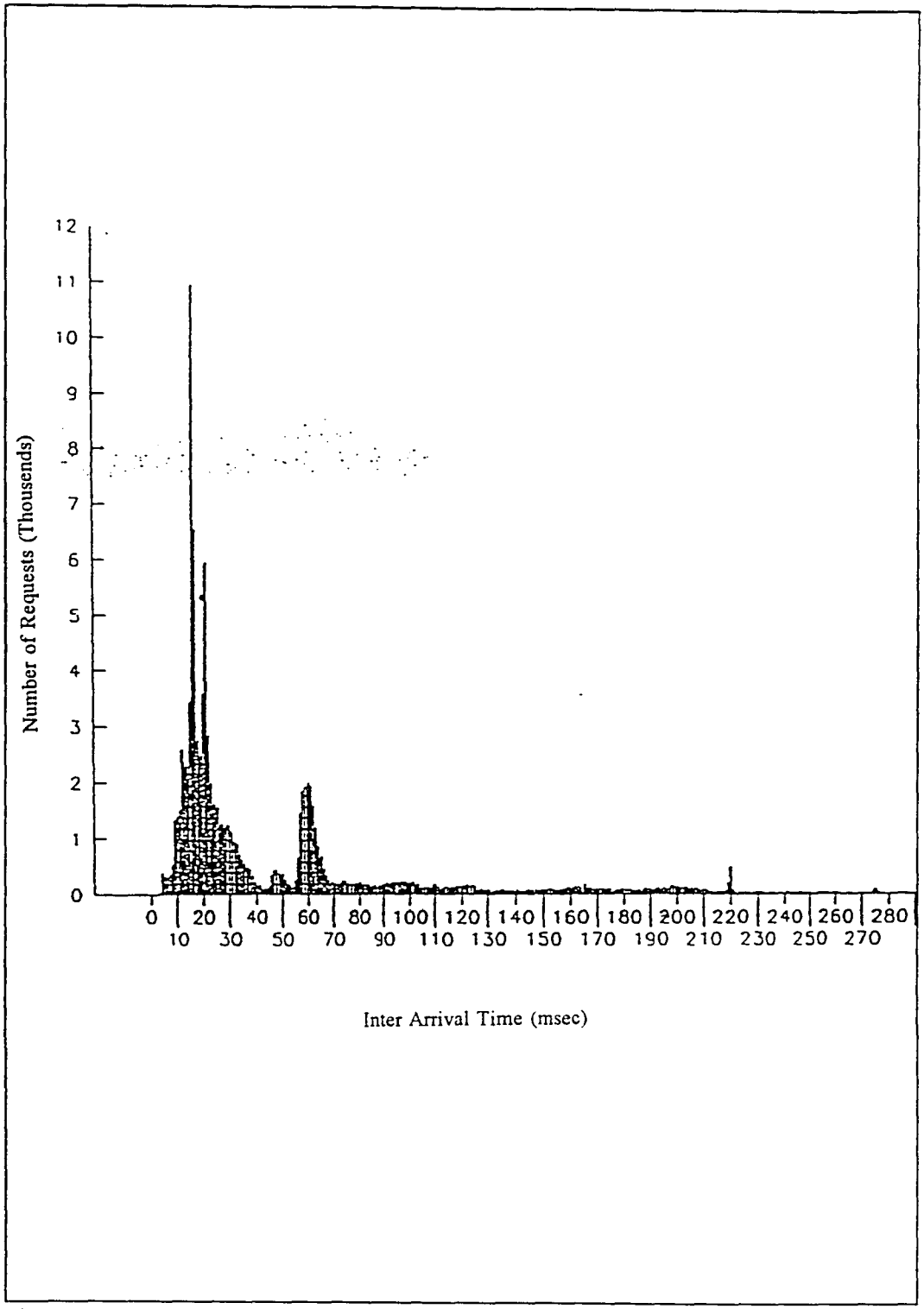

Figure 3.4: I/O request Interarrival time distribution on day-one.

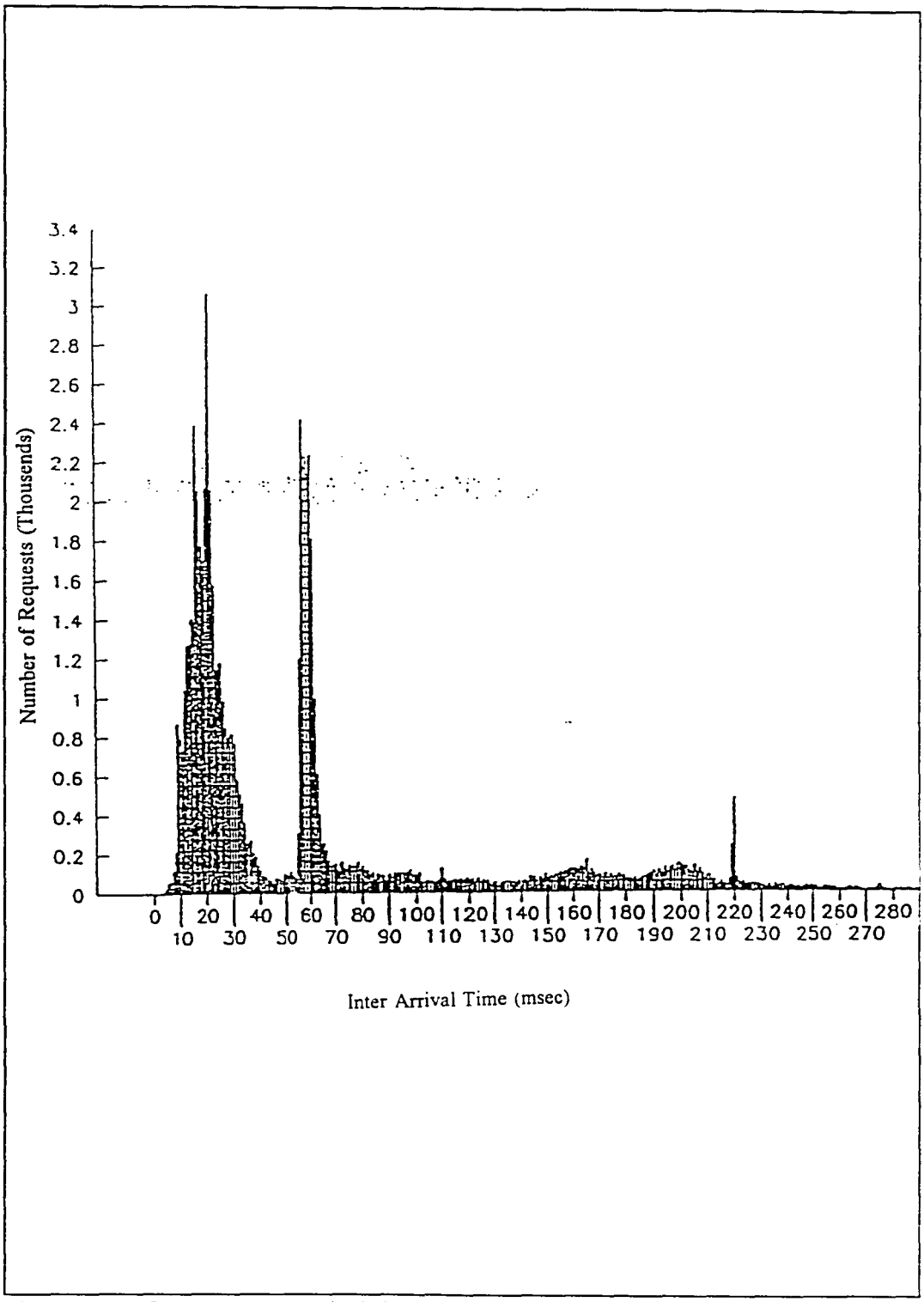

Figure 3.5: I/O request Interarrival time distribution on day-two.

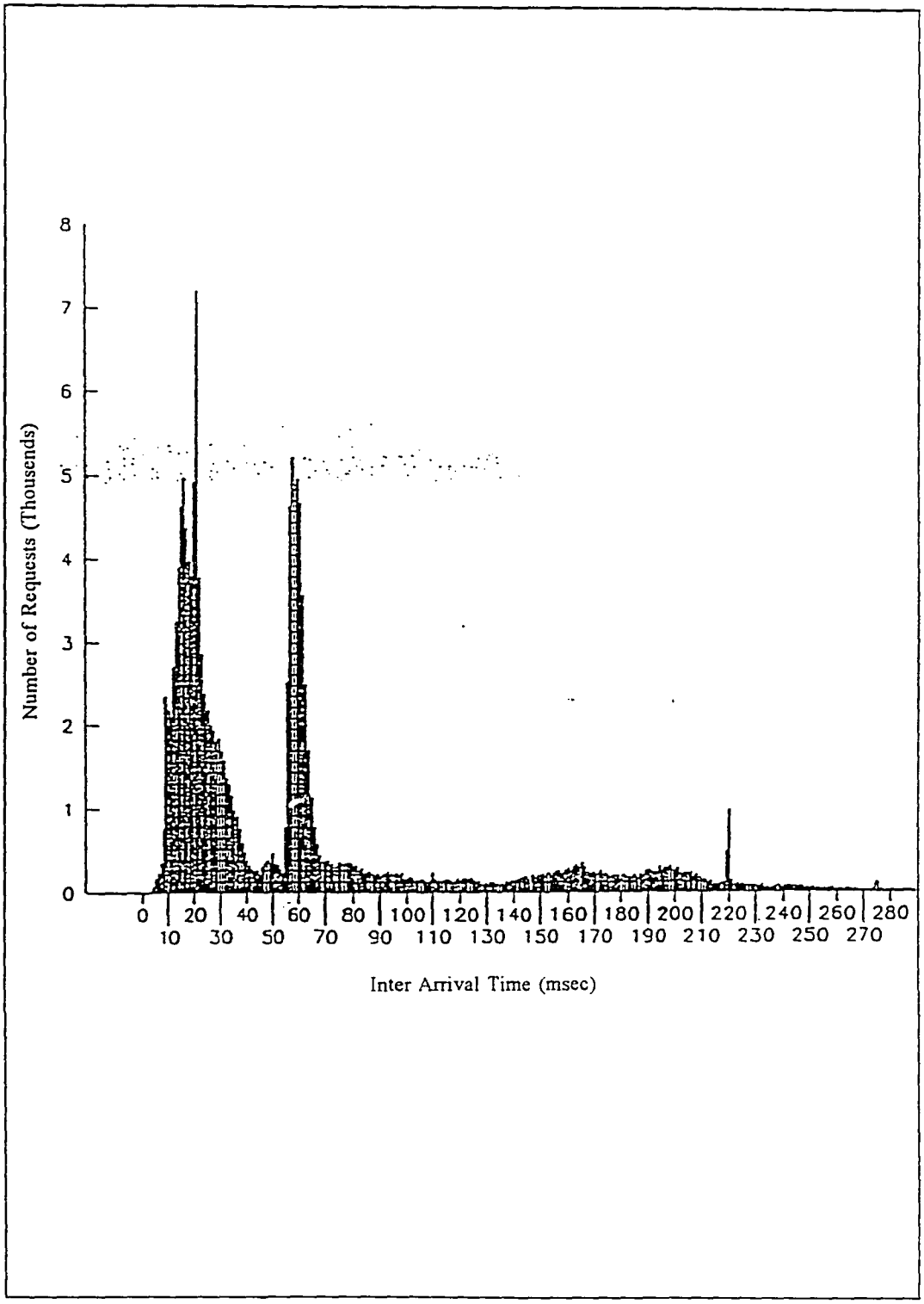

Figure 3.6: I/O request Interarrival time distribution on day-three.

3.1.3: DISTRIBUTION OF THE LOGICAL BLOCK ADDRESS OF I/O REQUESTS

Figures 3.7, 3.8, 3.9 and 3.10 show the distribution of the LBA on day-one for read and write I/O requests for drive 0 and 1. In figure 3.7 (drive 0), read requests are distributed approximately uniformly over the first 70% of the storage address space. However, the write requests (figure 3.8) are mainly concentrated into several LBA regions. Drive 1 shows similar request patterns where write requests (figure 3.10) are mostly concentrated into one LBA region. Similar to drive 0, drive 1 I/O requests are mainly restricted to first 70% of the available storage address space. Figures 3.11 and 3.12 show LBA distributions of I/O requests on day-two and figures 3.13 and 3.14 show LBA distribution of I/O requests on day-three. I/O requests patterns on day-two and day-three are very much similar to day-one except that there was almost no activity in drive 1 as shown in Table 3.2.

3.1.4: SEQUENTIAL I/O REQUESTS

The Novell Netware file system block size was set to eight sectors during volume definition. Therefore, all read I/O request size to storage subsystems are 8-sectors and write requests size vary from one to eight sectors. When an application program running in a client system requests more than eight sectors and they are not found in the host cache memory, the host must issue more than one I/O request to the storage subsystem. Clearly, file system block size has serious storage subsystem performance implications.

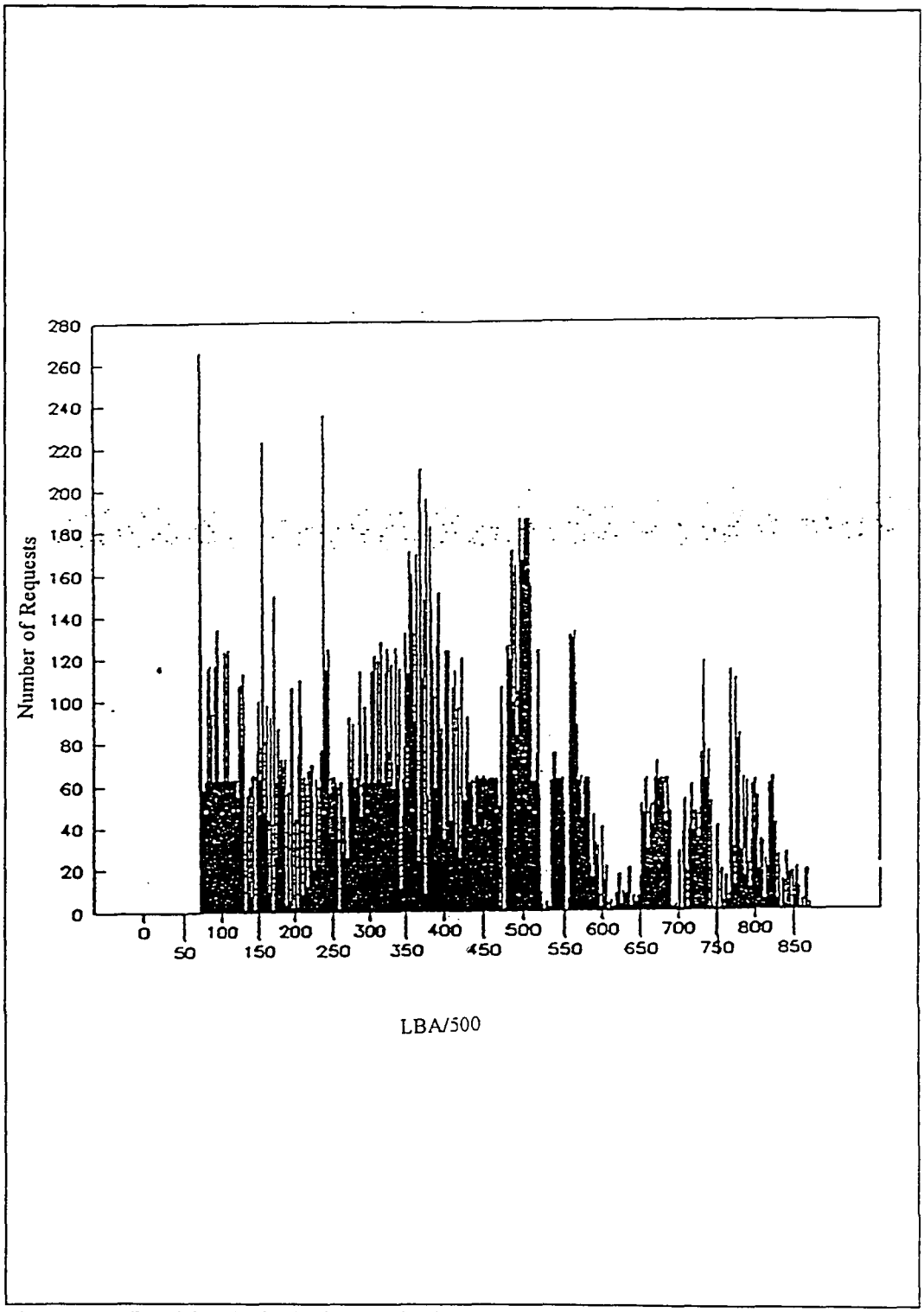

Figure 3.7: Read Logical Block Distribution on day-one (Drive 0).

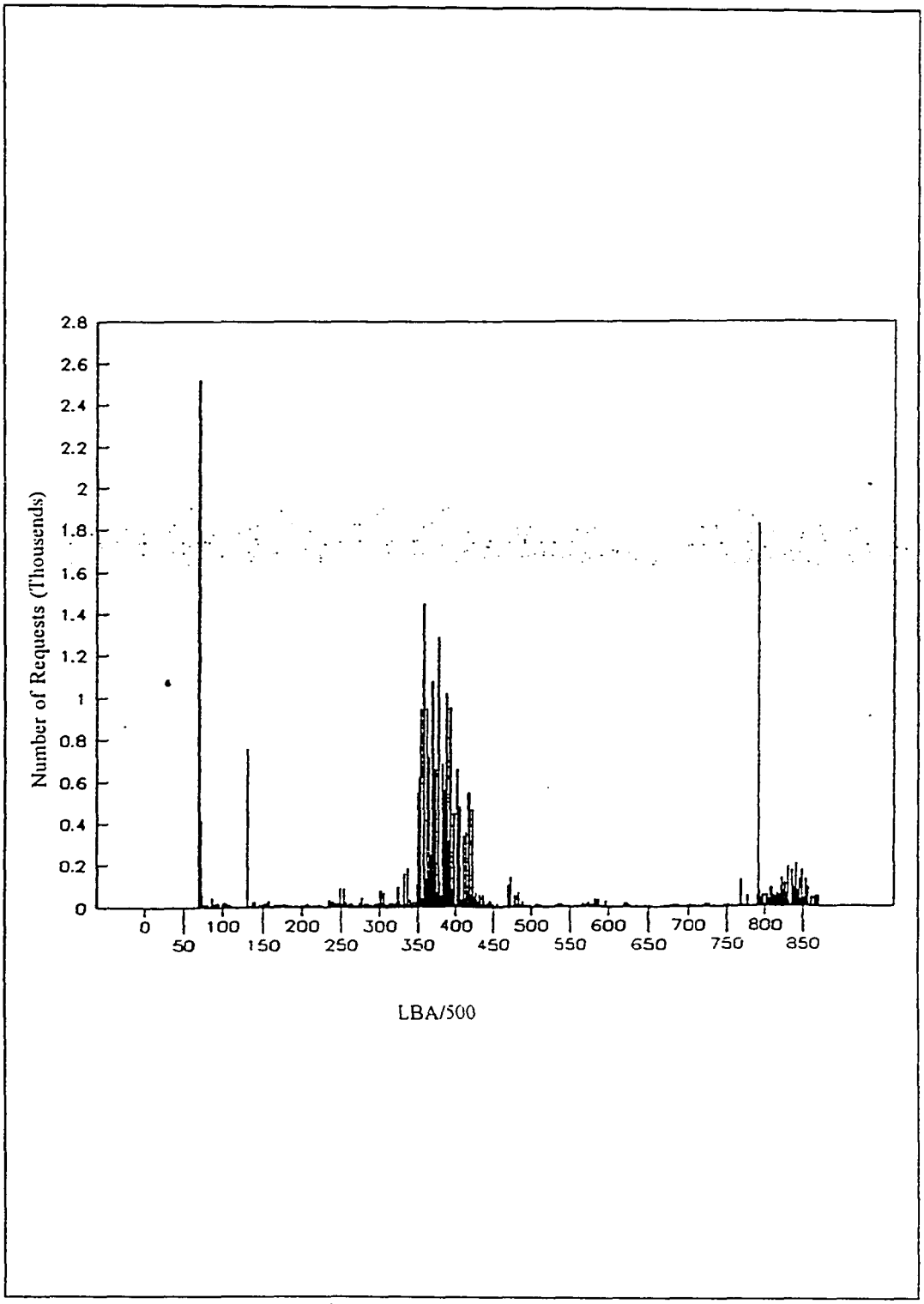

Figure 3.8: Write Logical Block Distribution on day-one (Drive 0).

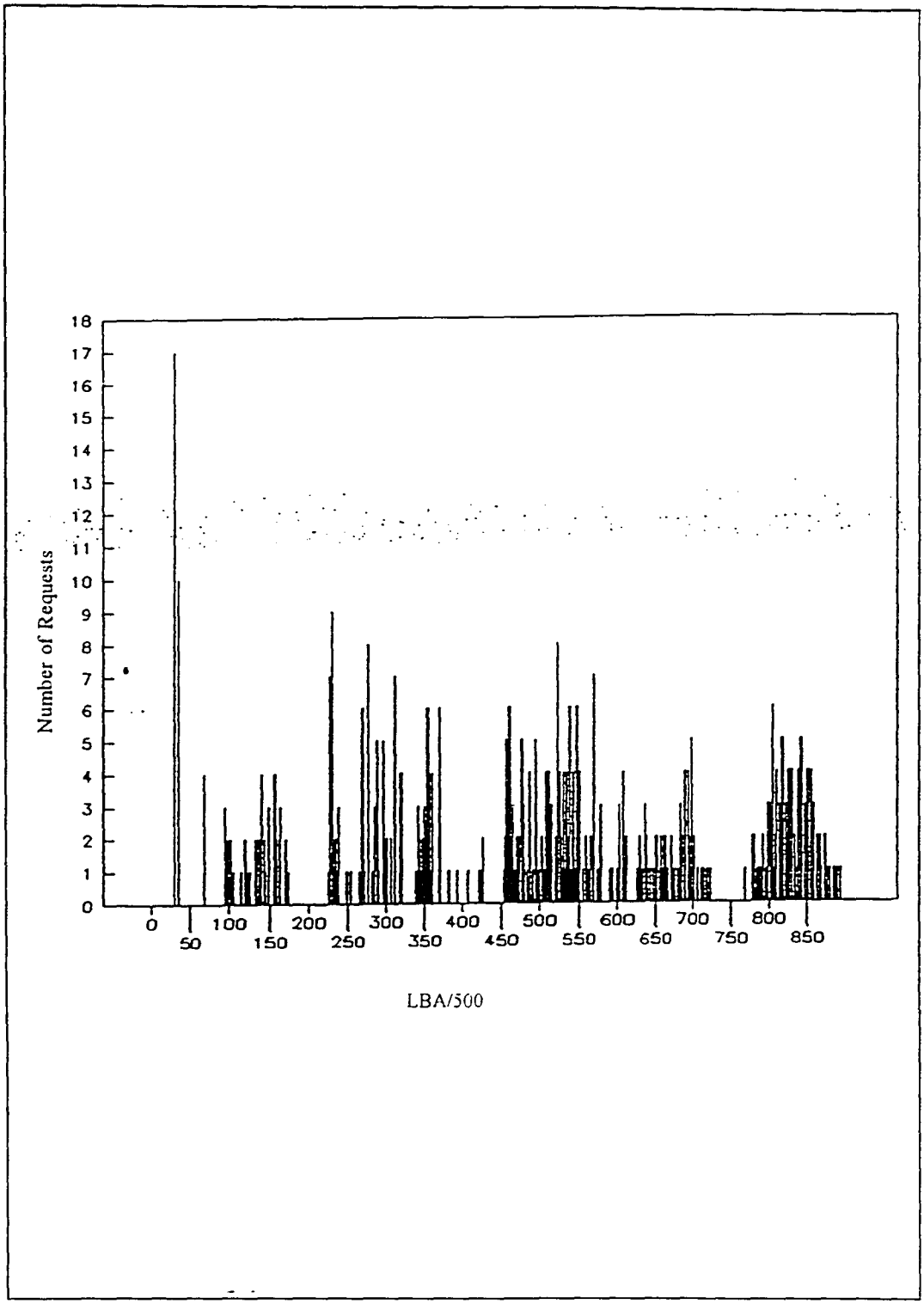

Figure 3.9: Read Logical Block Distribution on day-one (Drive 1).

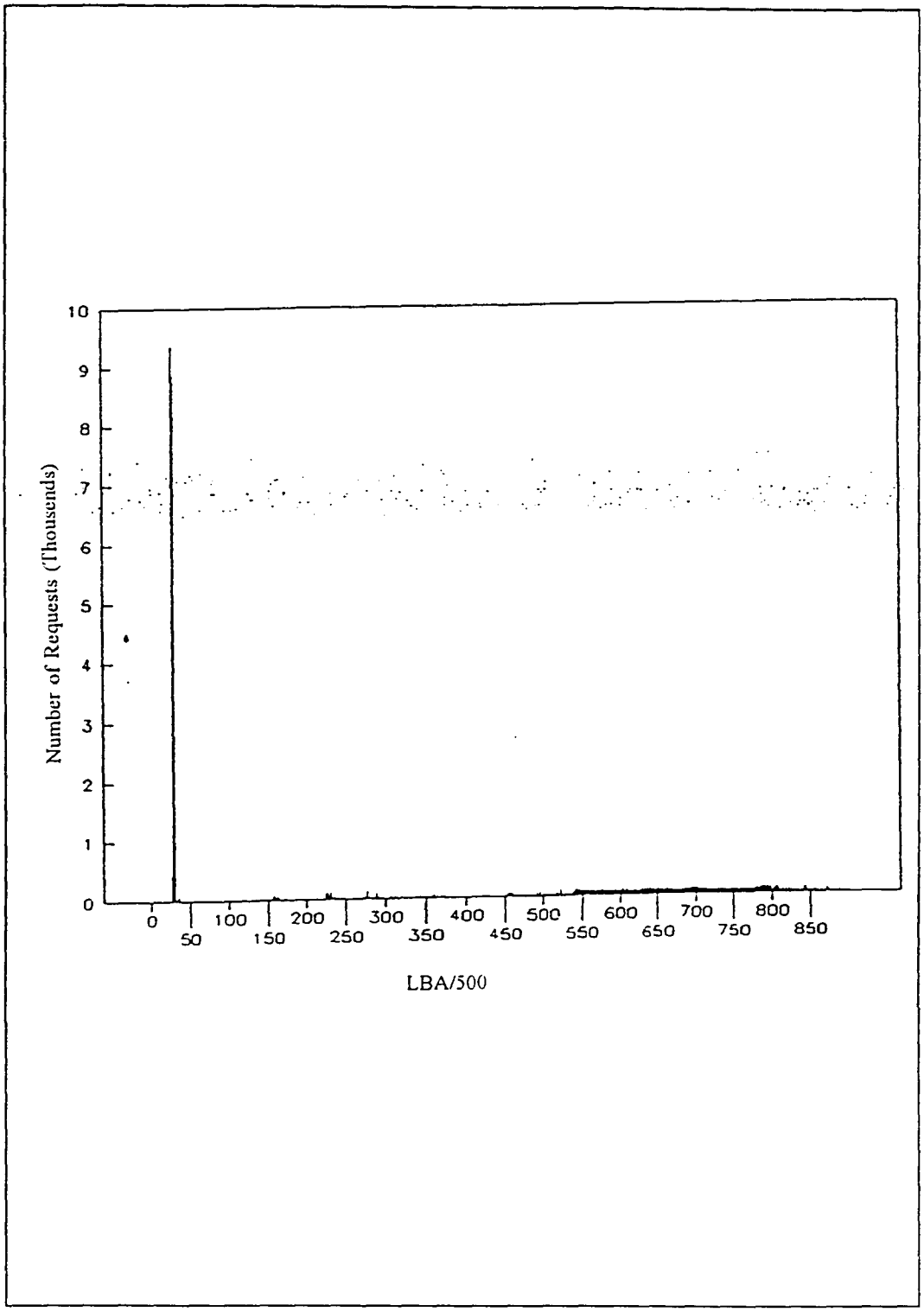

Figure 3.10: Write Logical Block Distribution on day-one (Drive 1).

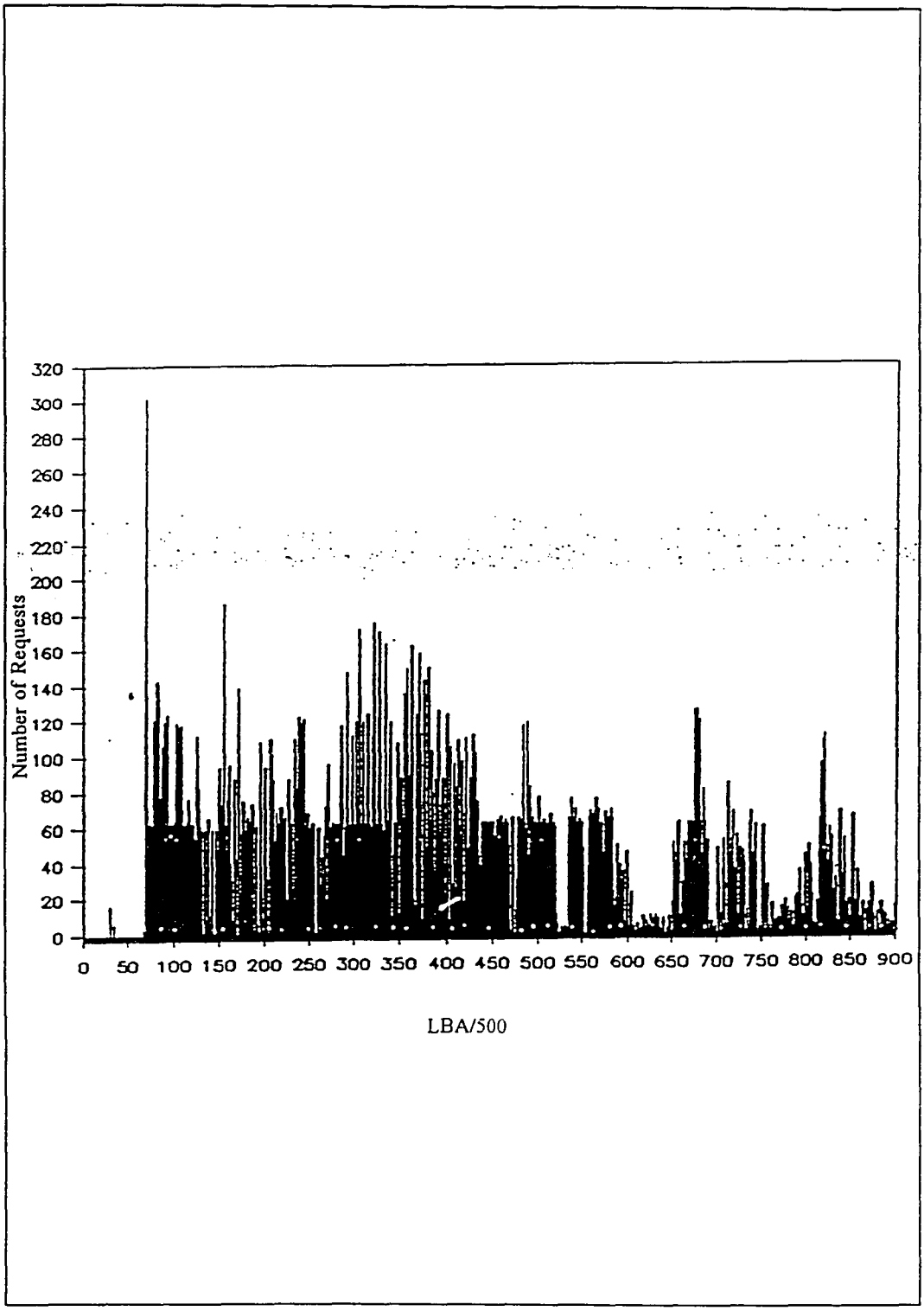

Figure 3.11: Read Logical Block Distribution on day-two.

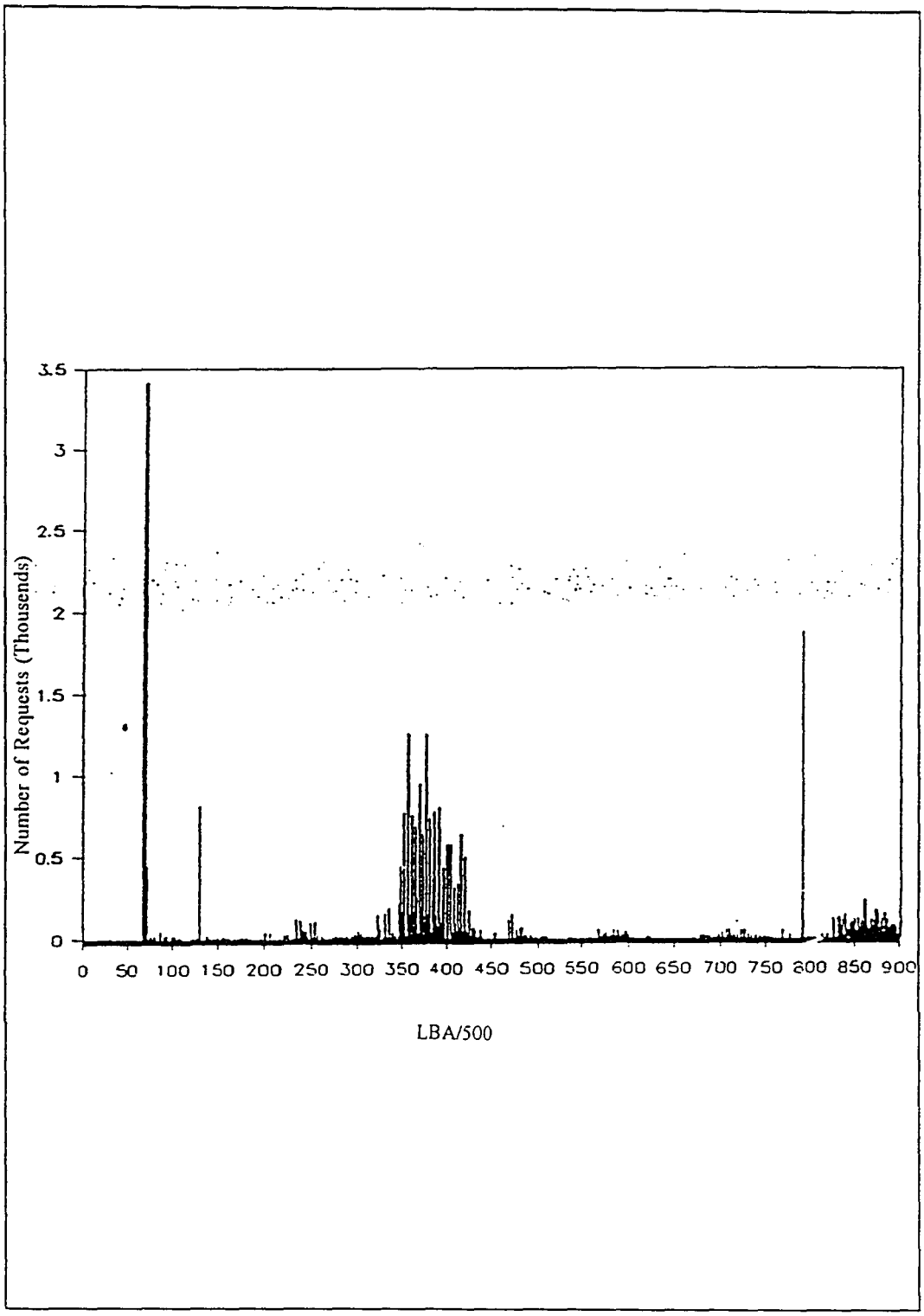

Figure 3.12: Write Logical Block Distribution on day-two.

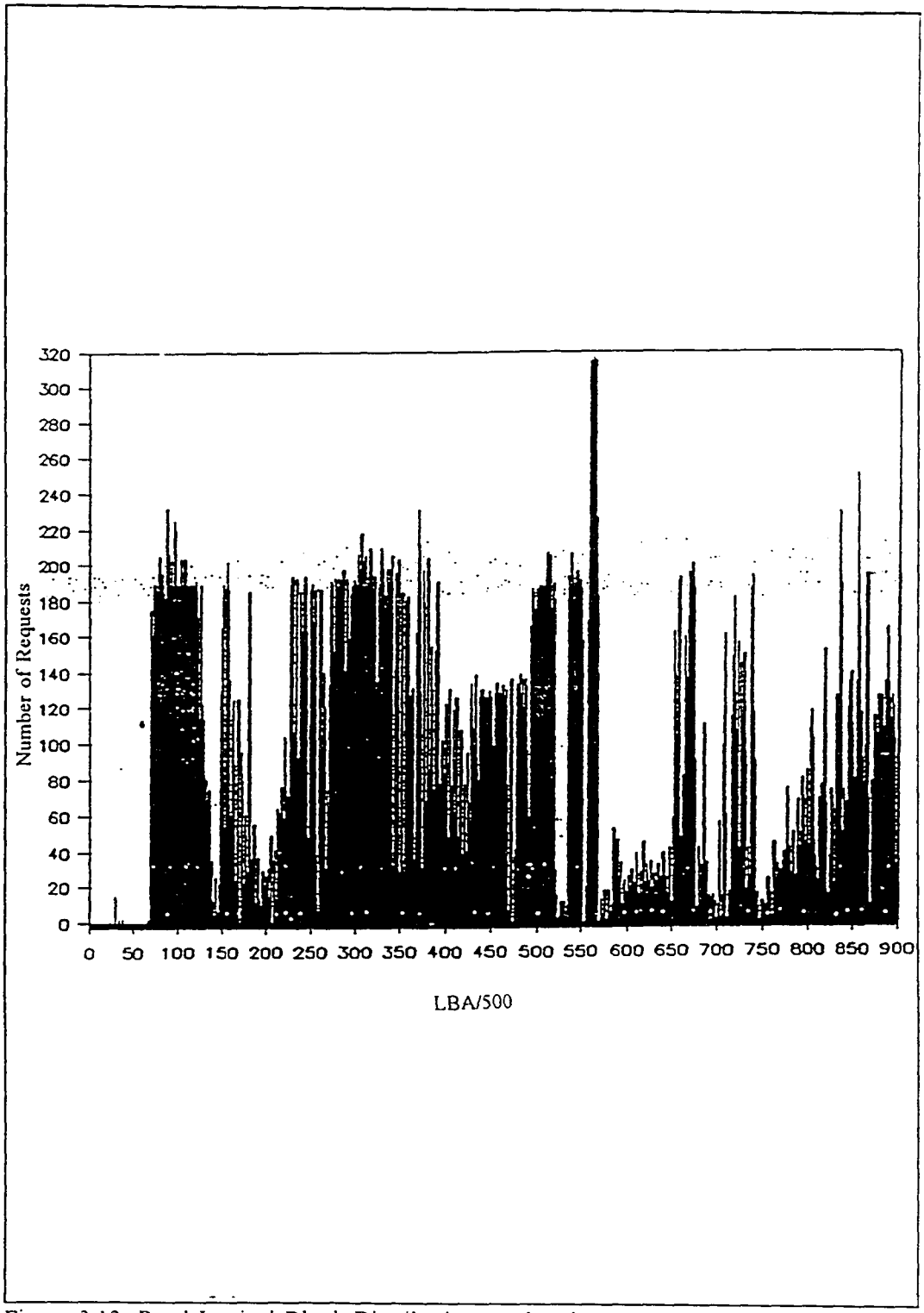

Figure 3.13: Read Logical Block Distribution on day-three.

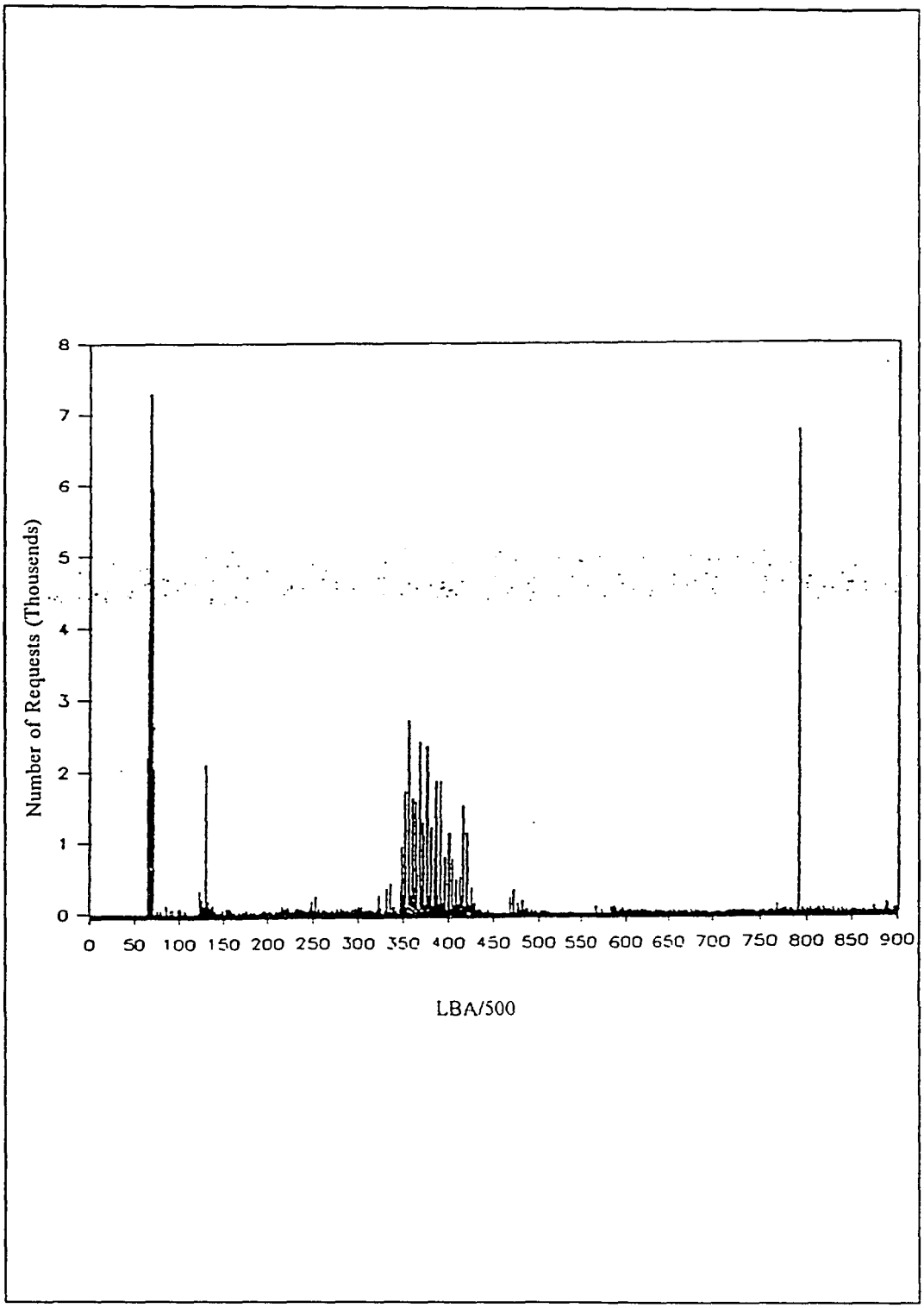

Figure 3.14: Write Logical Block Distribution on day-three.

If most of the client requests are 16 sectors , 8-sector file system block size would generate almost twice the traffic to the storage subsystem as a 16-sector file system block size. However, if most of the time client read requests size is less than 16 sectors and file system block size is set to 16-sectors, the host will always issue 16-sector read requests to the subsystem. It will increase data transfer time over the SCSI bus and at the same time destage more blocks from the host cache memory to make space for the blocks being read. If most of the client write requests are less than 16-sectors where file system block size is set to 16-sectors , the performance does not degrade since write I/O requests are issued to the storage subsystem during host cache lazy write or cache flushout and the size of these (less than or equal to 16) write requests depend on the sectors that need to be updated on the disk. In fact it is possible to observe some performance improvement in 16-sector file system block over 8-sector file system block due to reduced number of I/Os when write requests are less than 8-sectors.

We studied the sequentiality of consecutive 8-sector requests to see if the storage subsystem performance would improve by setting up the file system maximum block size to 16-sectors. Figures 3.15 and 3.16 show the probability mass function of number of 8-sector sequential read and write I/O requests on day-one. In these figures abscissa represents the numbers of sequential 8-sector requests. From figure 3.15 it is clear that most of the read requests (more than 85%) are one 8-sector read. However, from figure 3.16 we see that out of all 8-sector write requests (around 24% of all write requests) more than 50% are one 8-sector write and more than 45% are two or more 8-sector write

requests. Clearly if file system block size is changed to 16 sector, approximately half of these 45% write requests can be eliminated. Similarly, from figure 3.17, on day-two we see that there are few two or more 8-sector read requests, and figure 3.18 shows that out of all 8-sector write requests (around 8.5% of all write requests) around 40% are two or more 8-sector requests. In day-three (Figure 3.19) read requests are similar to day-one and day-two. However, write requests (Figure 3.20) are different from day-one and day-two as two or more 8-sector sequential requests drop to less than 20% of all 8-sector write requests. In day-three, 8-sector write requests are around 4.7% only.

3.1.5: SUBSYSTEM LOAD VARIATION WITH RESPECT TO TIME

Figures 3.21, 3.22 and 3.23 show instantaneous I/Os per hour with respect to hours of operation on day-one, day-two and day-three, respectively. Note that abscissa in these figures represent numbers of hours in operation and not the time of day. In day-one (Tuesday), trace data collection started (zero hour) around 6:00 pm. In day-two (Wednesday), the data collection started around 5:00pm in the evening. In day-three (Thursday), data collection started around 4:30 pm and data collection was completed on following Monday in evening around 8:30.

On day-one evening (first two hours of operation) shown in Figure 3.21 we see activity of mostly read operations. These heavy activities, apparently, represent the routine backup of the day. After the backup operation, server is almost inactive until next morning when

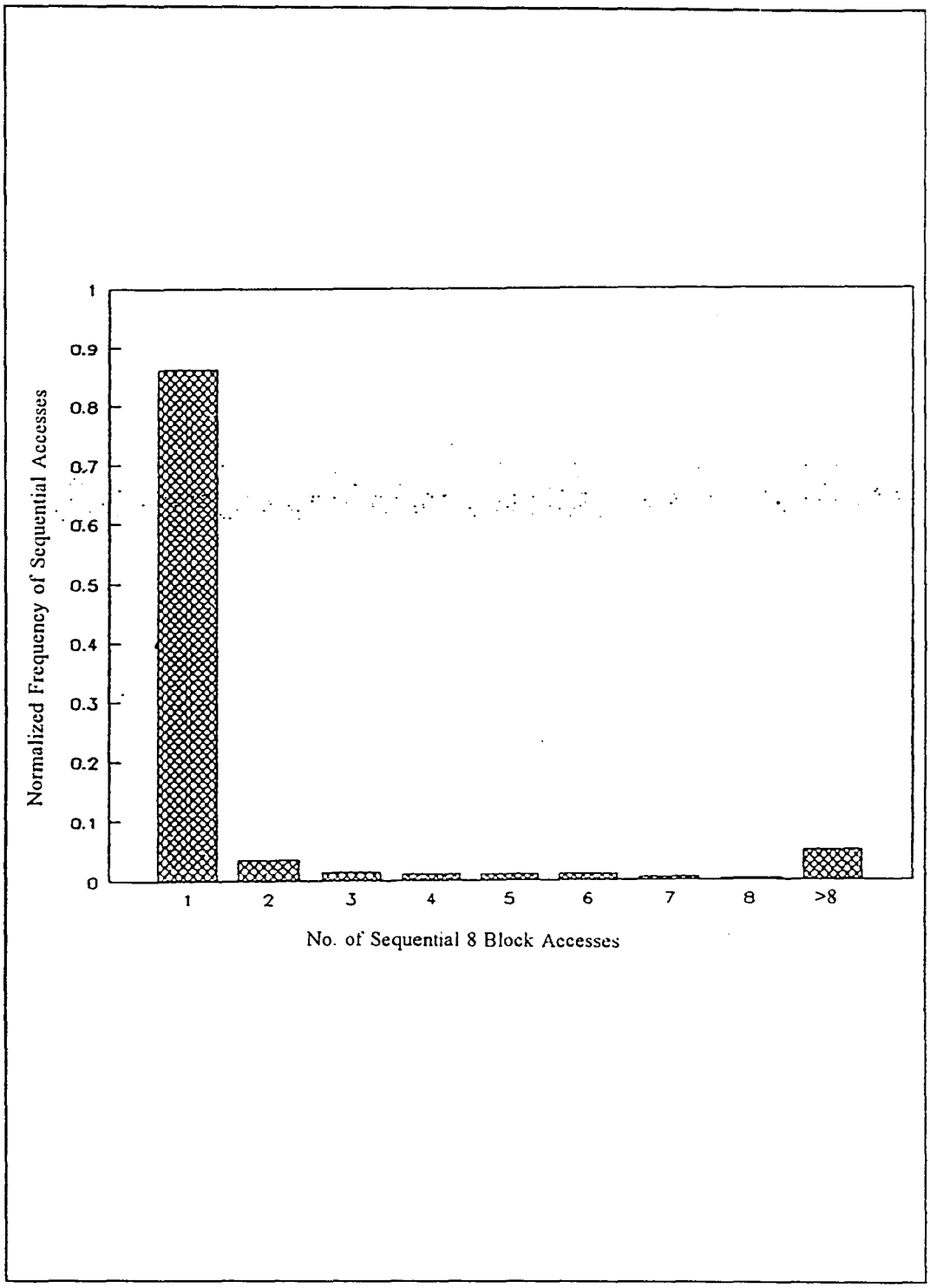

Figure 3.15: Probability mass function of number of 8-sector sequential read requests on day-one.

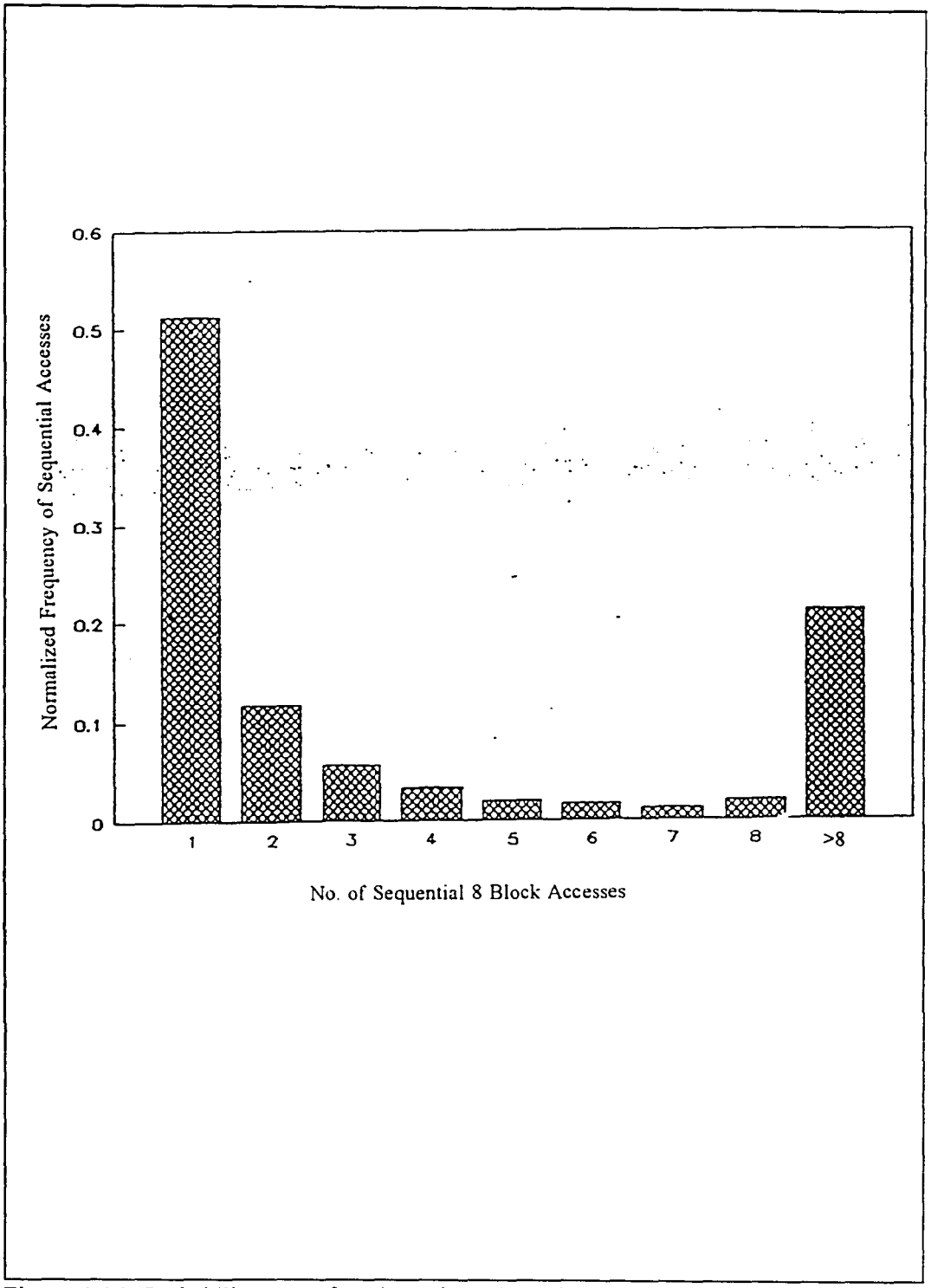

Figure 3.16: Probability mass function of number of 8-sector sequential write requests on day-one.

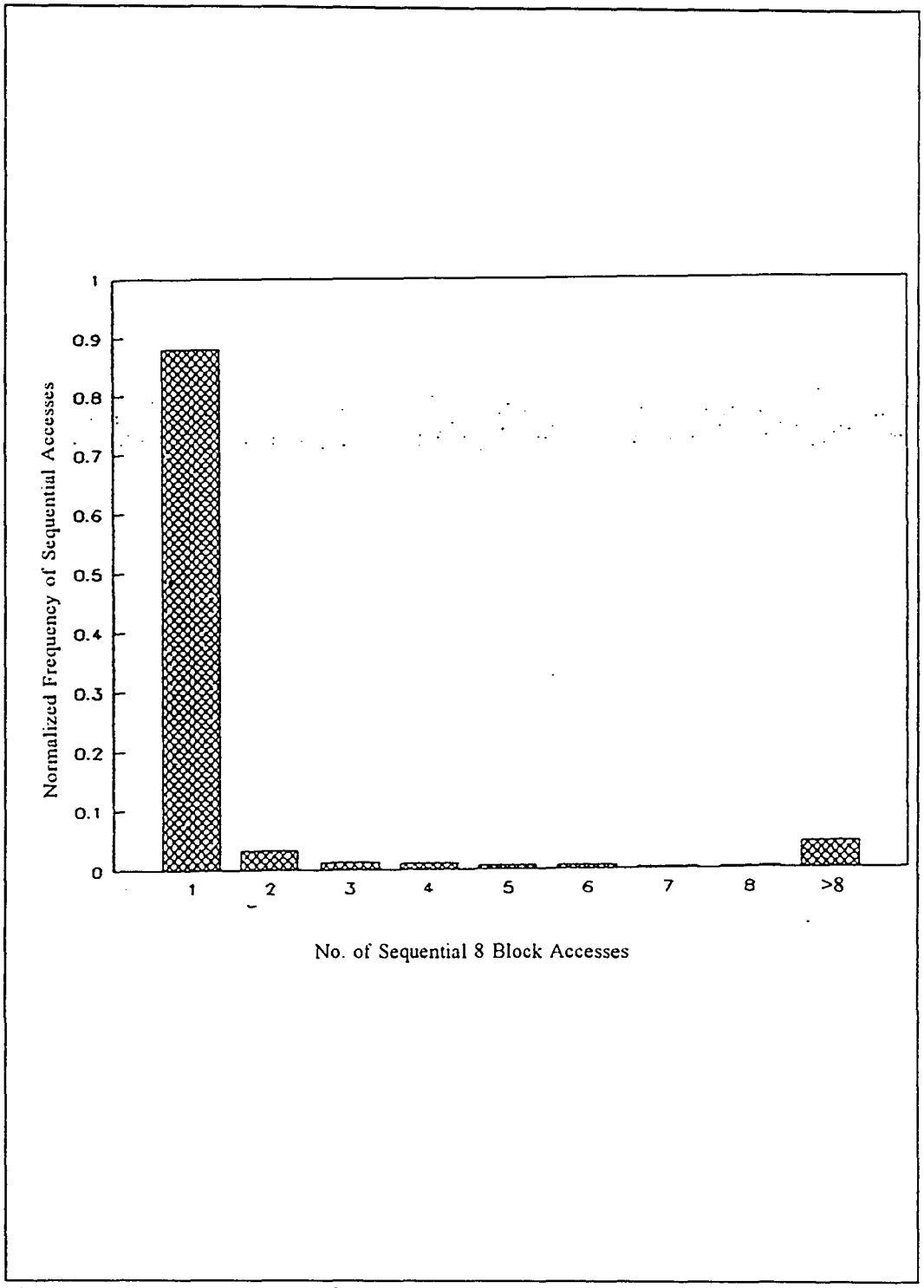

Figure 3.17: Probability mass function of number of 8-sector sequential read requests on day-two.

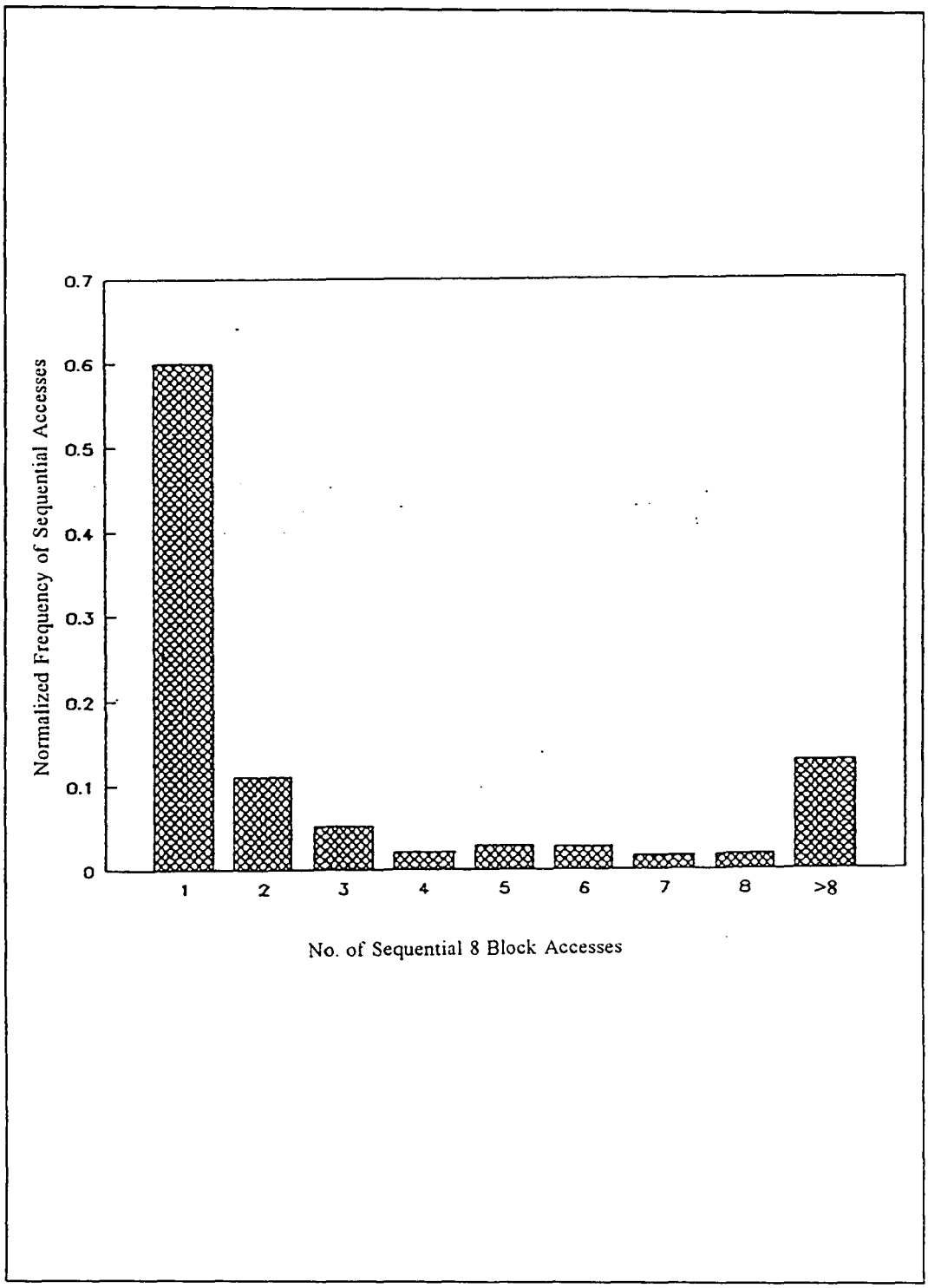

Figure 3.18: Probability mass function of number of 8-sector sequential write requests on day-two.

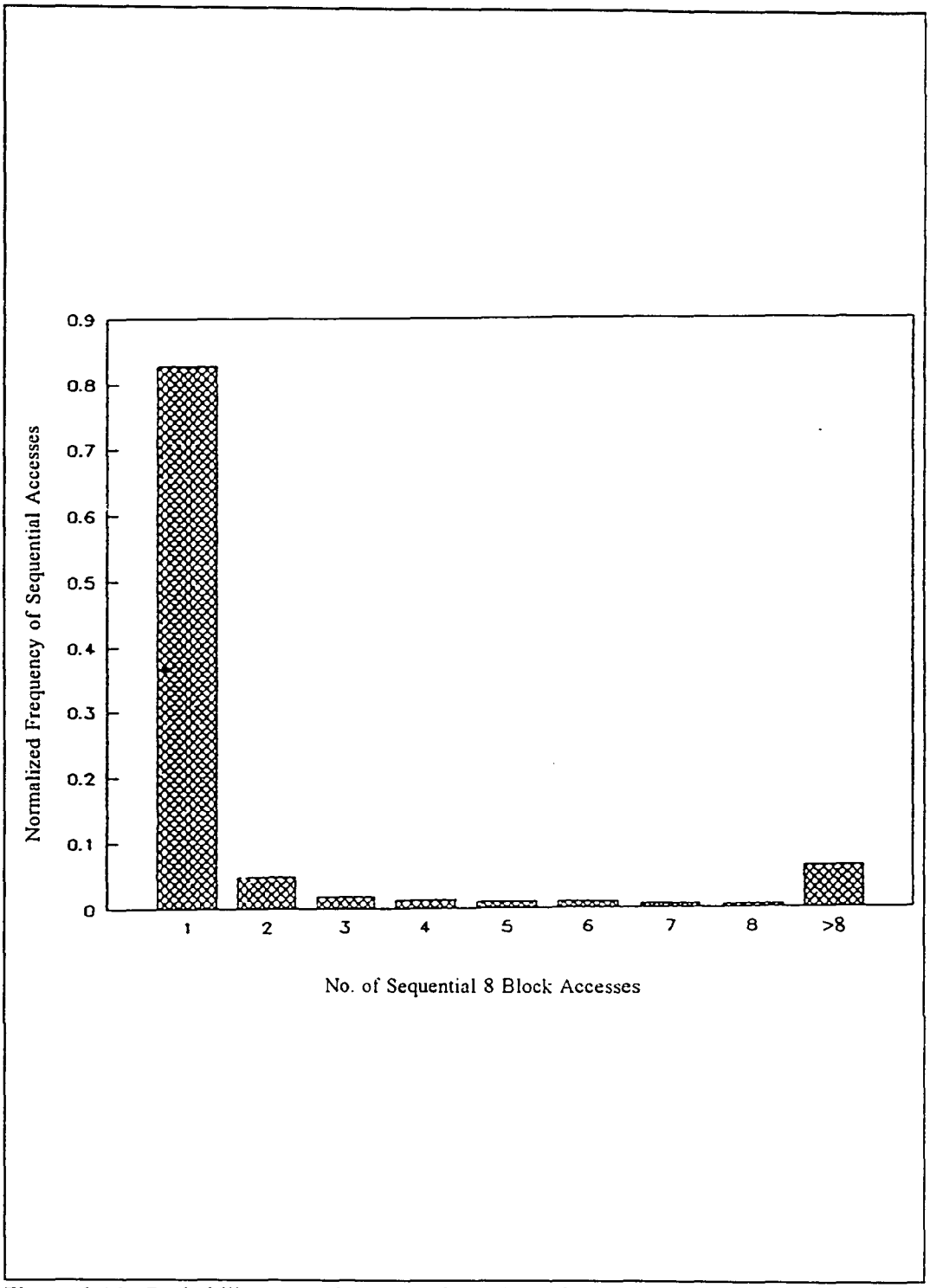

Figure 3.19: Probability mass function of number of 8-sector sequential read requests on day-three.

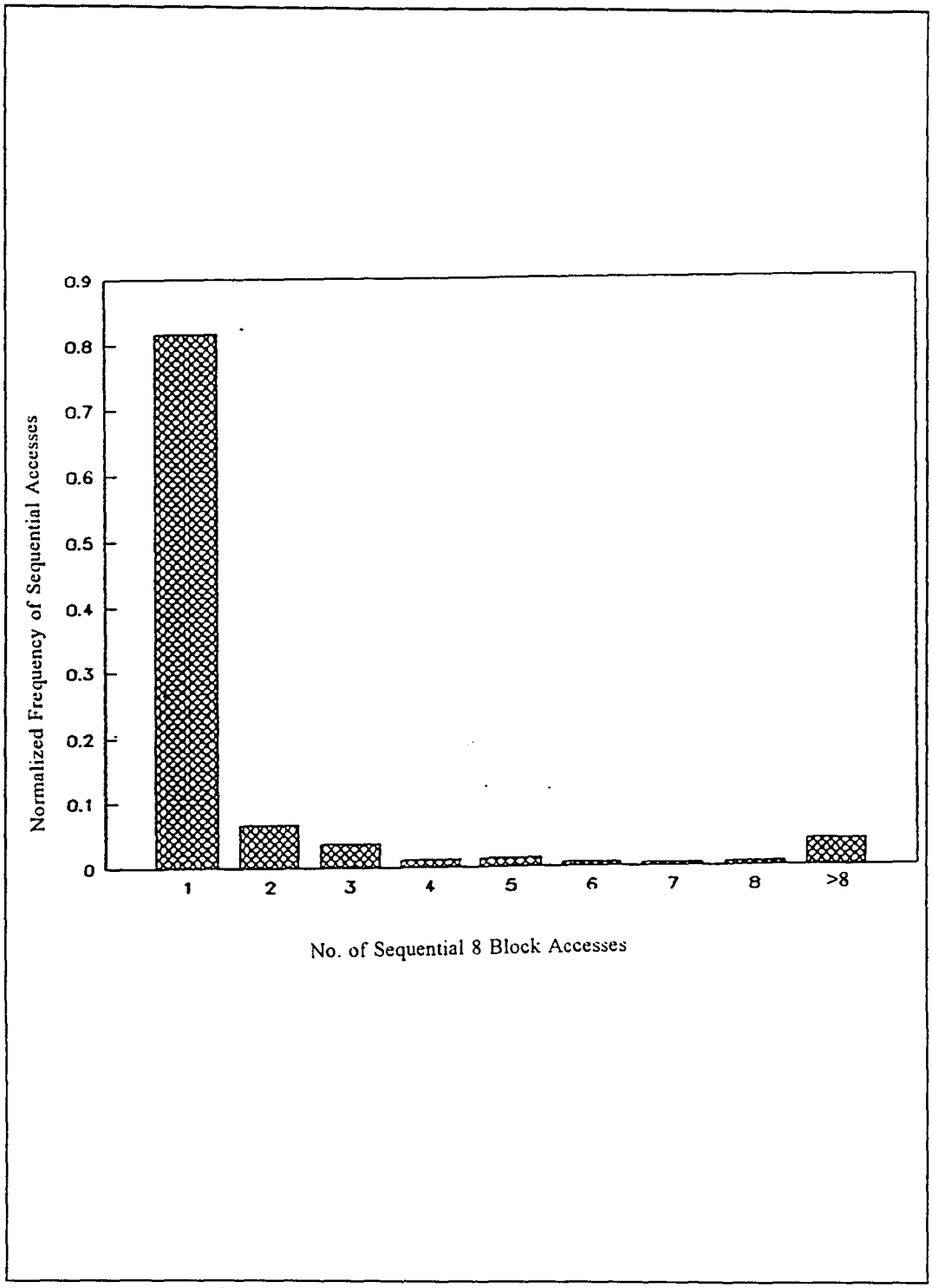

Figure 3.20: Probability mass function of number of 8-sector sequential write requests on day-three.

load starts to peak around the 12th hour of the operation. We also see a tall tower (representing heavy activity) around the 16th hour of operation (around 10 o'clock in the morning). We then see a sharp decrease in load in the 17th and a valley around the 18th hour of operation during the lunch time. Load starts to grow gradually after lunch up to the 22nd hour of operation which represents 4 to 5 o'clock in the afternoon. Figure 3.22 shows continuation of the activity of figure 3.21 where we again see a drop on the load as they represent very late afternoon hours, 5 and 6 o'clock in the evening. We see a sharp rise in the load around the 3rd hour of operation in day-two. This sharp rise in the load is due to routine backup done probably at 7 o'clock in the evening. There is almost no activity during rest of the evening and night. We again see some activity around the 13th hour of operation which is around 5 o'clock in the morning on the day-three. We see a sharp increase in activity in the following couple of hours up to lunch time which is around 20th hour of operation. We again see load going up due to late afternoon activity. In day-three (figure 3.23) we again see a sharp rise in mostly read load representing routine backup at early evening. Server was mostly inactive during the night of day-three. Activity during Friday (day-three period) morning and afternoon was similar to the load seen on day-two. There was no routine backup on Friday evening. There was almost no activity until Sunday evening, except some write operations on Saturday morning around 38th hour of operation (around 6:30am).

We see some intense activity on Sunday evening around the 73rd hour of operation (around 5:30pm). We first see large number of write I/O requests followed by a large

number of read I/O requests which strongly suggest that there was some kind of file maintenance operation (e.g. backup/restore). We then see usual Monday morning activity, a dip during lunch, an increase after lunch, and a gradual drop in load towards late afternoon. We also see increase in load due to routine backup around the 99th hour of operation (around 7:30pm).

3.1.6: SUMMARY OF WORKLOAD IN THE TRACED NETWARE ENVIRONMENT

Detailed analysis of the six day long trace data totaling 372,271 I/O requests in a PS/2 file server running Novell Netware is presented. Analysis showed that peak subsystem instantaneous load (when averaged over one second interval) on day-one, day-two and day-three were 100 I/Os per sec, 60 I/Os per sec and 64 I/Os per sec, respectively. However, when averaged over one hour period they drop to around 14 I/Os per sec, 7 I/Os per sec and 9 I/Os per sec, respectively. Although, subsystem performance under the peak transient load (i.e. averaged over one second) can be improved by increasing the number of subsystem components (e.g. adapter and drives), the peak steady-state load (i.e. averaged over one hour) is well below the existing subsystem throughput capacity. It is also observed that on day-one, both drives were active in contrast to day-two and day-three when mostly one drive was active. Some response time reduction can be easily achieved by spreading data over both drives such that they are accessed uniformly.

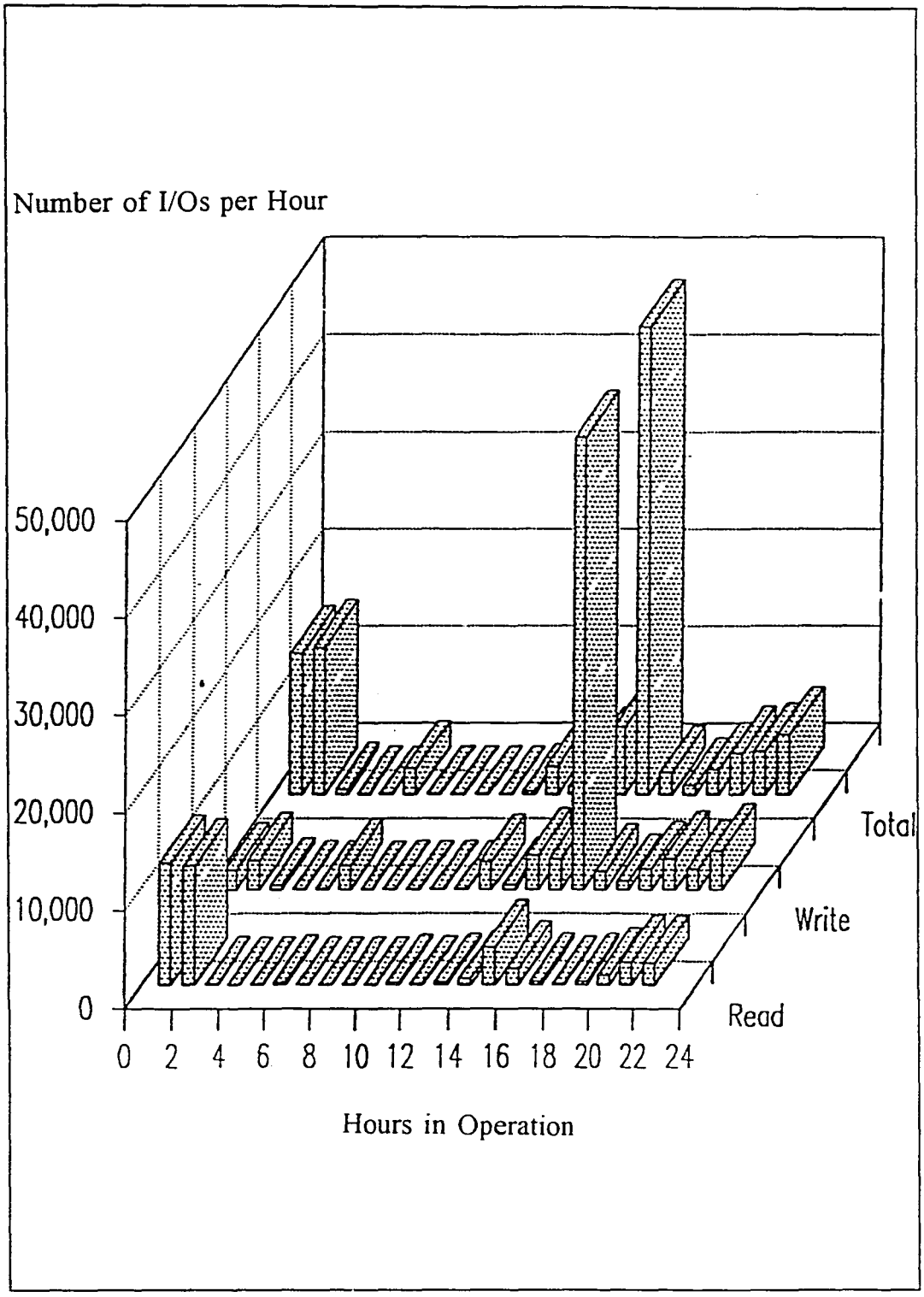

Figure 3.21: Instantaneous subsystem load with respect to time on day-one.

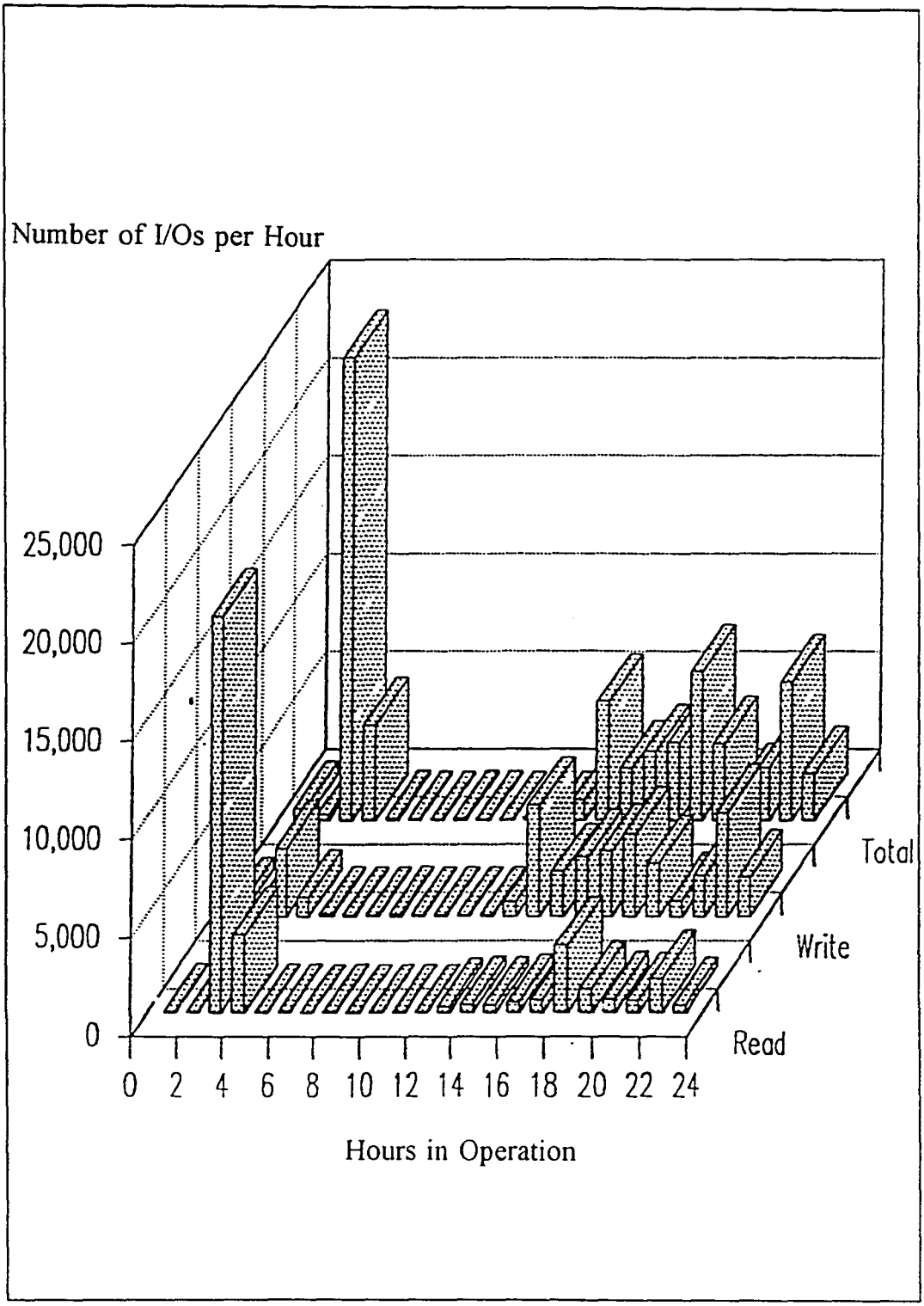

Figure 3.22: Instantaneous subsystem load with respect to time on day-two.

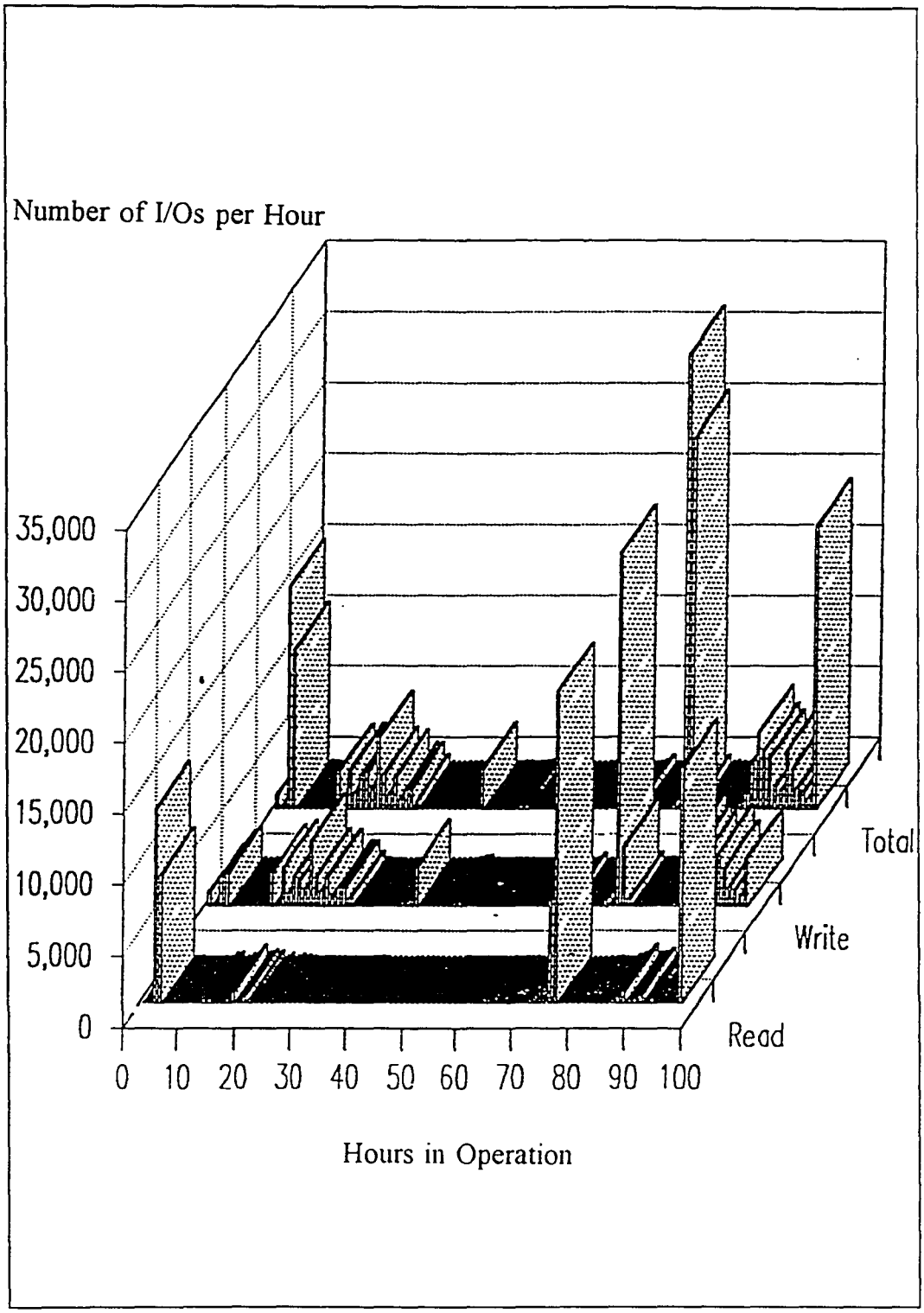

Figure 3.23: Instantaneous subsystem load with respect to time on day-three.

All read request sizes were 8-sector and most of the write request sizes (63% to 75%) were one sector. Analysis further showed that more than 85% of the read requests were only one 8-sector read requests. On day-one, day-two and day-three, one or more 8-sector write requests were 24%, 8.5% and 4.7%, respectively. It is concluded that with current workload, changing file system block size from 8-sector to 16-sector would not significantly improve the overall performance. The fraction of I/Os with two or more 8-sector requests is very much sensitive to future workload changes and should be closely monitored. If this fraction becomes significant, the file system block size should be changed to 16-sector.

3.2: CHARACTERISTICS OF WORKLOAD TRACED IN OS/2 ENVIRONMENT

In this section, we present the characteristics of the storage subsystem workload in PS/2 database servers running OS/2 Extended Edition version 1.2 Database Manager. The storage subsystems of database servers, running under OS/2 version 1.2 were traced, on seven different sites of the same company, one site at a time. On each site, traces were collected for one business day (about eight hours) using the SSTT. The collected traces are processed and workload is analyzed using tools described in chapter 2. The system profile for each site is same. A general site profile is shown in table 3.5. Each database server has four drives which are assigned letter C, D, E and F.

Application Business: Database/Transaction. Representative Cycle Time: Generally majority of activity is during 8am to 5pm business day.
Server Configuration System: IBM PS/2 model 8580 (386/20MHz). Hard Drives: 4 320MB IBM SCSI drives, each connected to a separate SCSI adapter.

Table 3.5: General site profile.

Table 3.6 shows the relative load distribution across all drives for each site. On each site drive F got 5% or less of total requests for the day. While drive C and D, together, got more than 80% of requests. This skewed load distribution shows under utilization of resources (drives E & F), which may probably be impeding the overall system performance. For each site the read-to-write ratio has been computed and request size, LBA and interarrival distributions have been computed and plotted.

Site Code name	Drive C (%)	Drive D (%)	Drive E (%)	Drive F (%)	Total Requests
Site1	46.78	35.37	12.75	05.10	55637
Site2	57.21	30.44	08.95	03.40	48400
Site3	14.61	67.06	17.11	01.21	439312
Site4	21.54	69.81	06.24	02.41	107154
Site5	50.24	34.29	11.26	04.23	52875
Site6	61.32	25.00	09.52	04.17	35467
Site7	44.71	39.68	11.69	03.91	99195

Table 3.6: Relative load distribution across drives for each site.

Since, we found that the workload characteristics on all sites are very similar, in this section we have chosen to present workload characteristics of only one site. The following is a brief study of storage subsystem workload on the Site3.

Traces were collected starting at 11:45 am. We see all drives getting most requests per hour in two hours after 2pm. Figure 3.24 shows comparison of load on the four drives. Figure 3.25 shows load for each drive and comparison of read and write requests for individual drives. Table 3.7 shows the percentage of read and write requests and controllers' cache performance in terms of read hit and write hit percentages for each of the four drives.

Drive: C	Drive: D
Total transactions: 64196 Read Operations: 71.18% (45696) Adapter Cache hit in Read: 0% Write Operations: 28.82% (18500) Adapter Cache hit in Write: 1.46% (935)	Total transactions: 294614 Read Operations: 98.56% (290361) Adapter Cache hit in Read: 0% Write Operations: 1.44% (4253) Adapter Cache hit in Write: 0.03% (77)
Drive: E	Drive: F
Total transactions: 75176 Read Operations: 95.18% (71550) Adapter Cache hit in Read: 0% Write Operations: 4.82% (3626) Adapter Cache hit in Write: 0.34% (259)	Total transactions: 5326 Read Operations: 60.76% (3236) Adapter Cache hit in Read: 0% Write Operations: 39.24% (2090) Adapter Cache hit in Write: 0.51% (27)

Table 3.7: Summary of storage subsystem workload for Site3.

From table 3.7 it can be noted that there is no adapter cache hit on read requests. For write requests there is a very small number of adapter cache hits (less than 2%). When further investigated, it was found that the SCSI device driver used by the operating system is turning off the read caching by adapter.

Figures 3.26 through 3.29 show the request size distributions. Request sizes range from one block to 127 blocks. On all drives most of the read request are of size four block. About 78% of all read requests on drive C, about 80% of read requests on drive D, about 90% of read requests on drive E and about 83% of read requests on drive F are 4 block requests. Other observed read requests sizes are 124-blocks (≈6%) on drive C, 16-blocks (≈14%) on drive D, 16-blocks (≈8%) on drive E and 16-blocks (≈8%) & 127-blocks (≈5%) on drive F. Similarly, for write requests, about 20% of all write request drive C, about 88% of all write requests on drive D, about 95% of all requests on drive E and about 8% of all write requests on drive F are 4 block requests. Other observed write requests sizes are 1-block (≈24%) & 2-block (≈14%) on drive C and 1-blocks (≈27%) & 12-blocks (≈9%) on drive F.

Figures 3.30 through 3.33 show the LBA distributions and inter arrival time distributions. For inter arrival delays greater than 40 millisecond an approximately equal distribution is noticed except on drive C where we have another small peak between 45 and 65 milliseconds. LBA accesses are clustered around different points over the DASD address space.

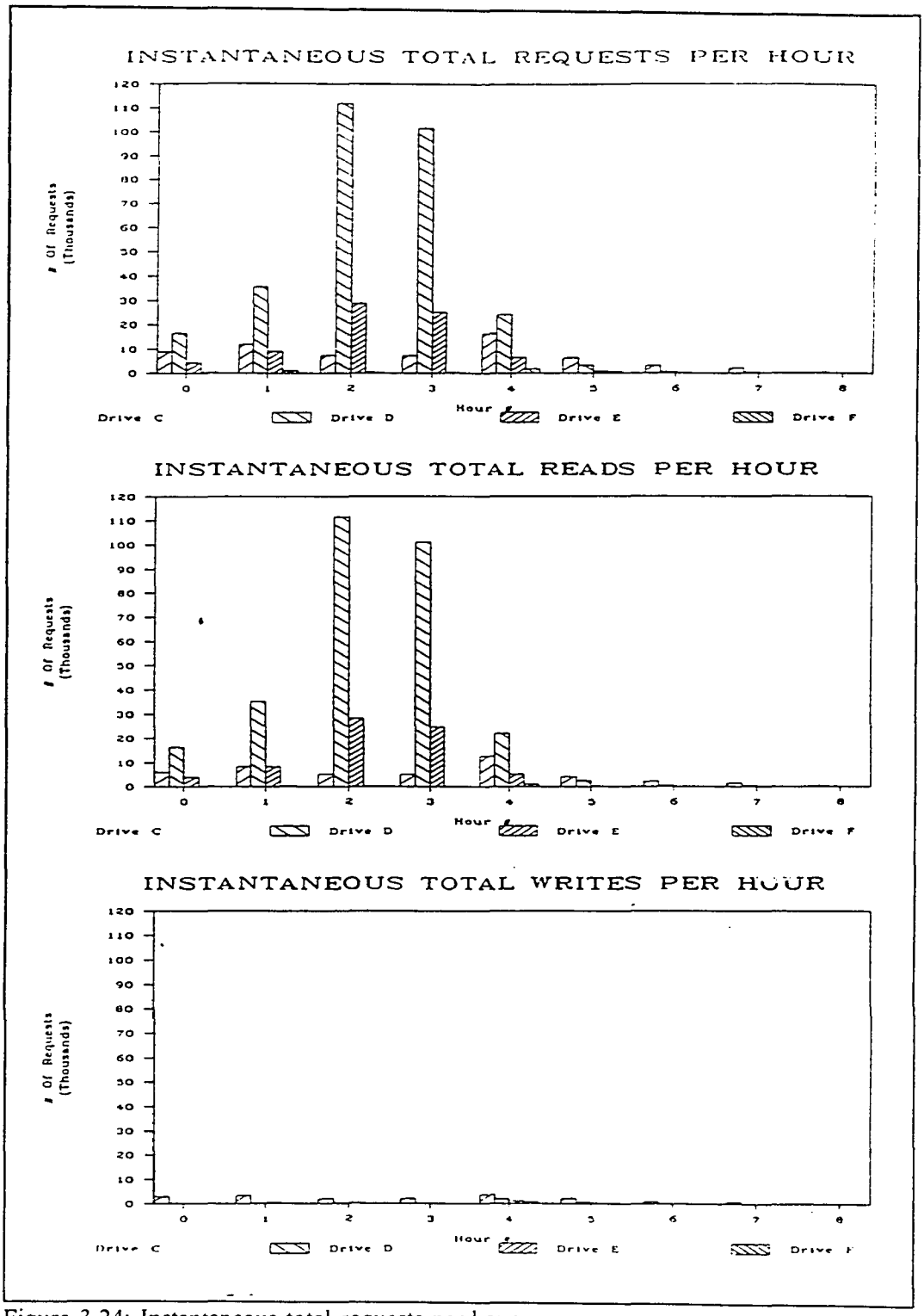

Figure 3.24: Instantaneous total requests per hour.

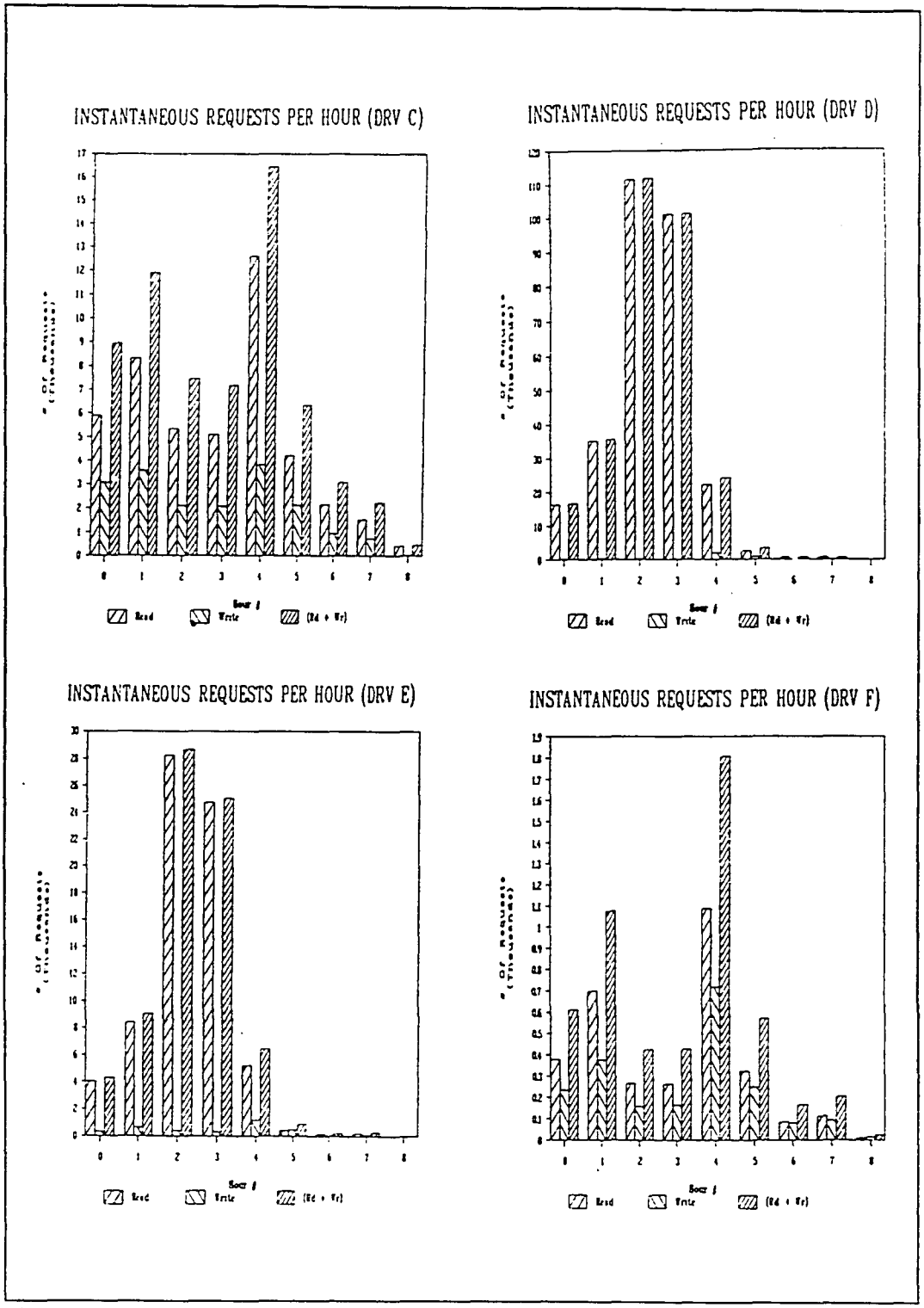

Figure 3.25: Instantaneous requests per hour for each drive.

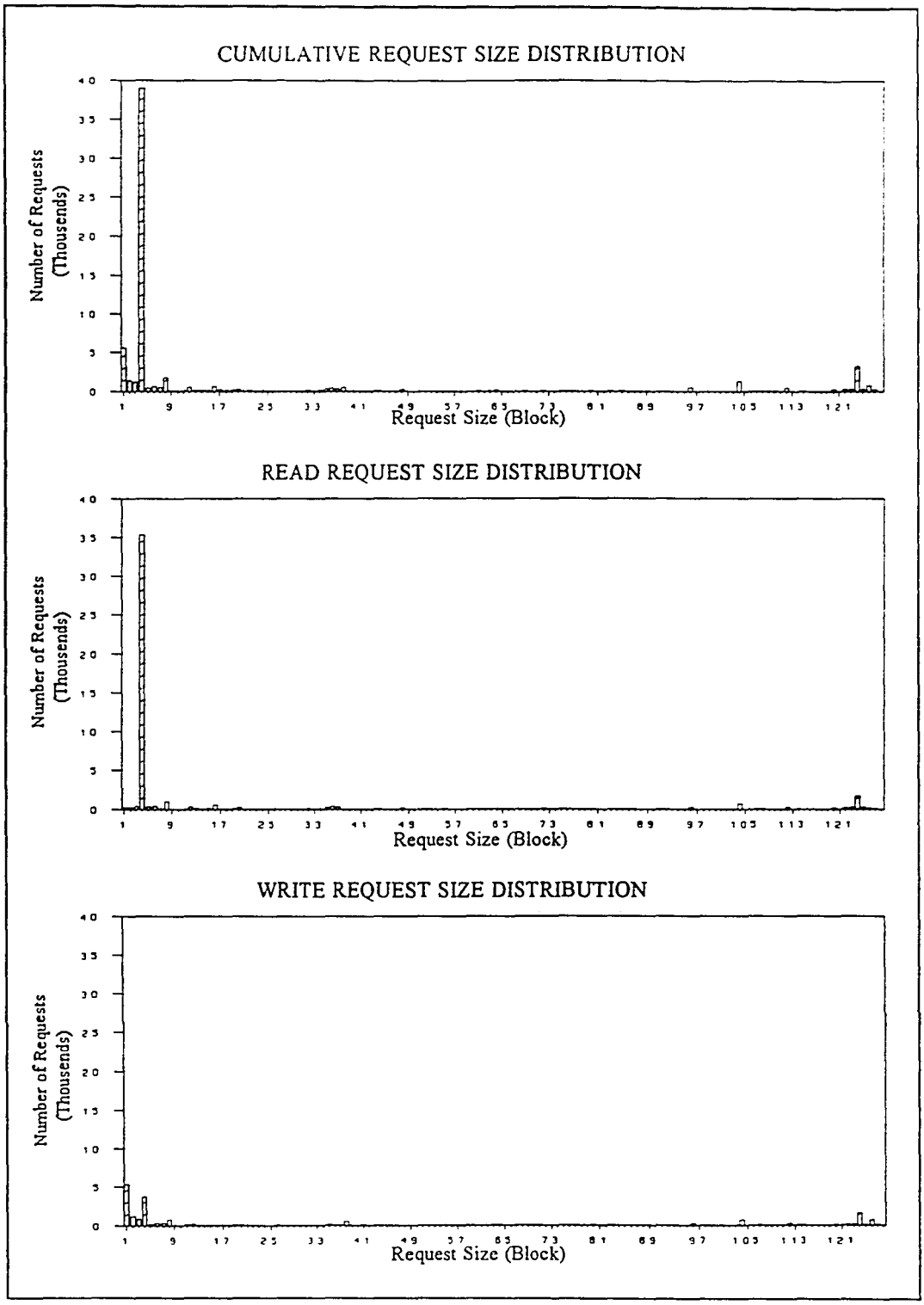

Figure 3.26: Request size distribution on first drive.

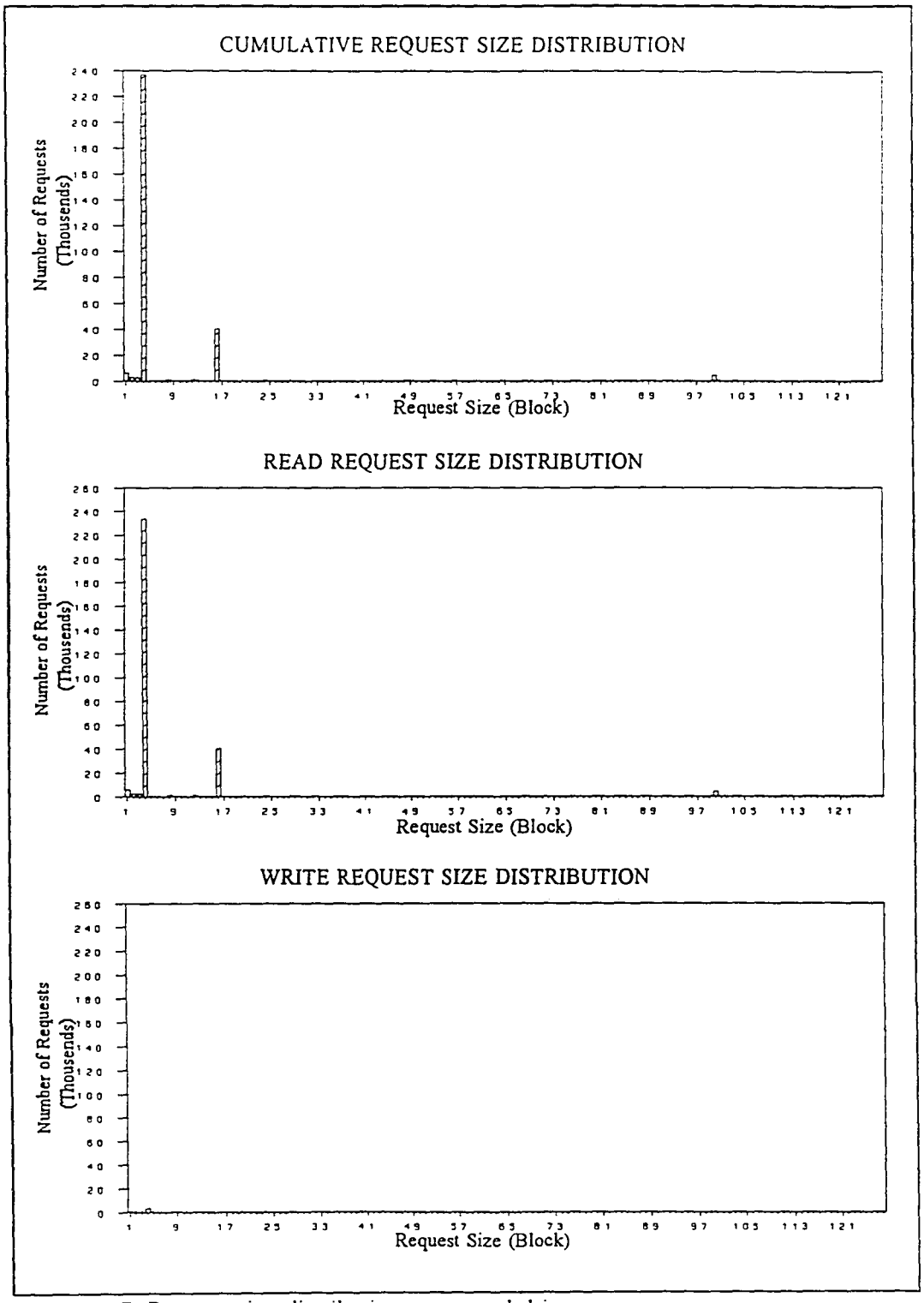

Figure 3.27: Request size distribution on second drive.

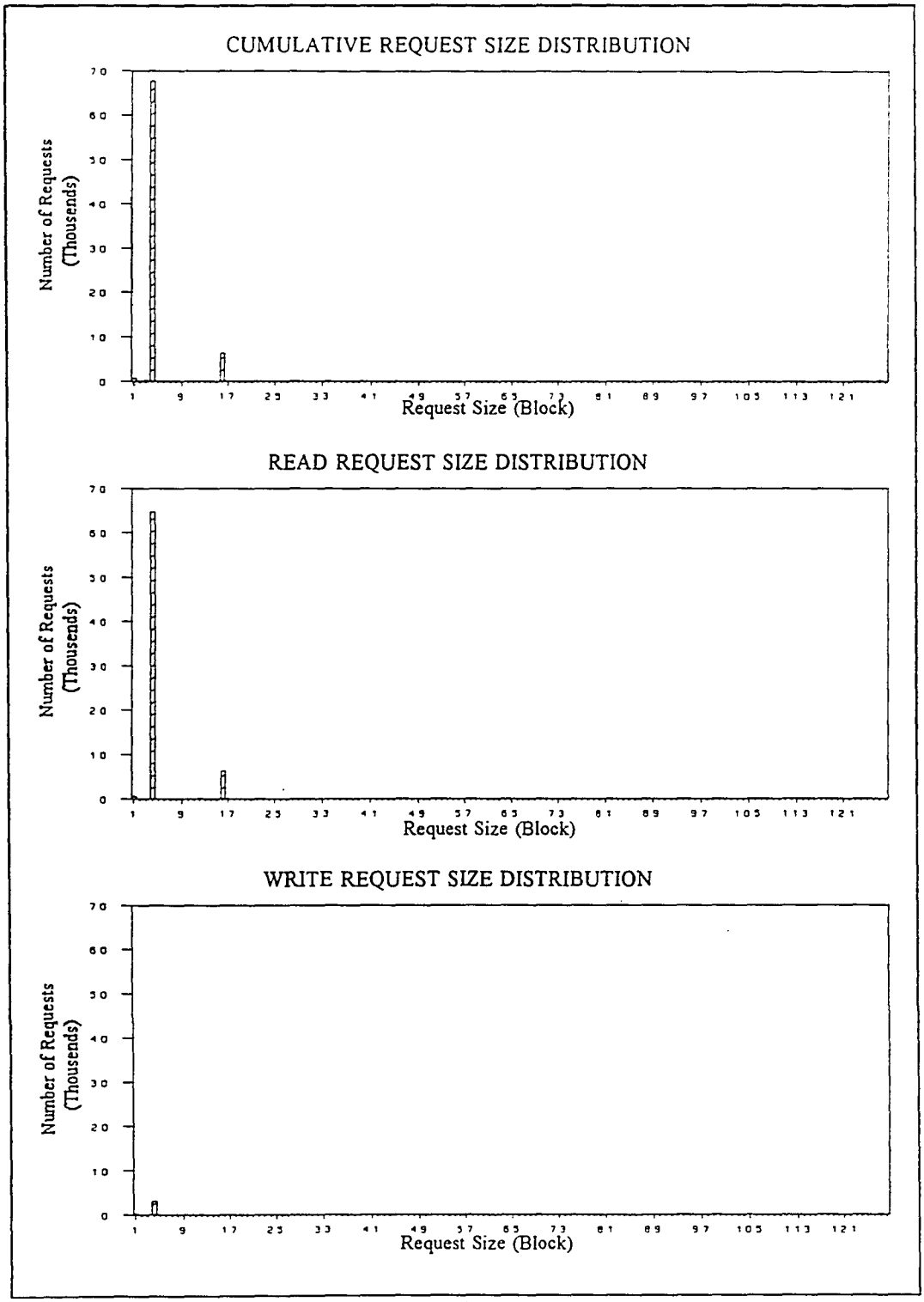

Figure 3.28: Request size distribution on third drive.

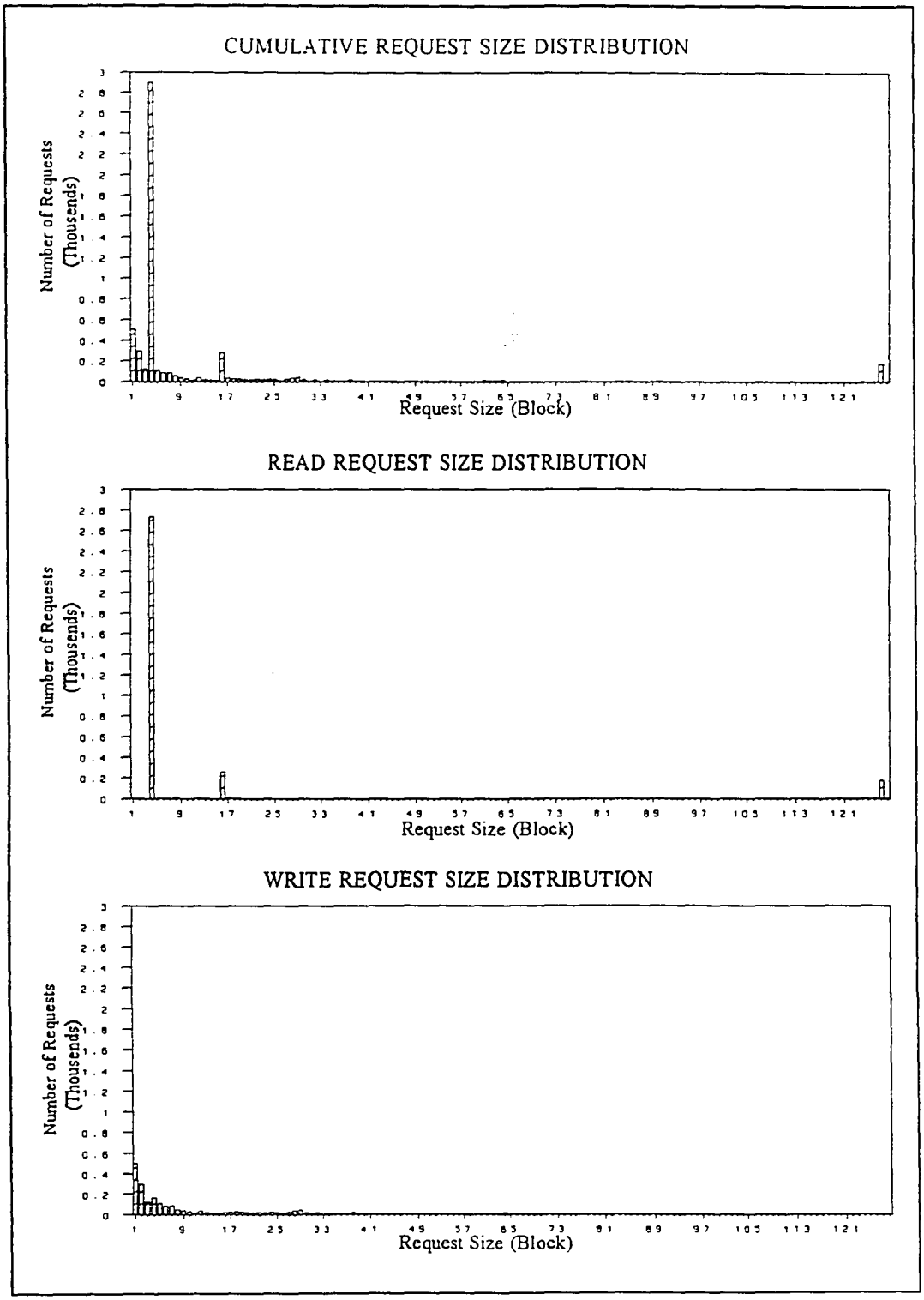

Figure 3.29: Request size distribution on fourth drive.

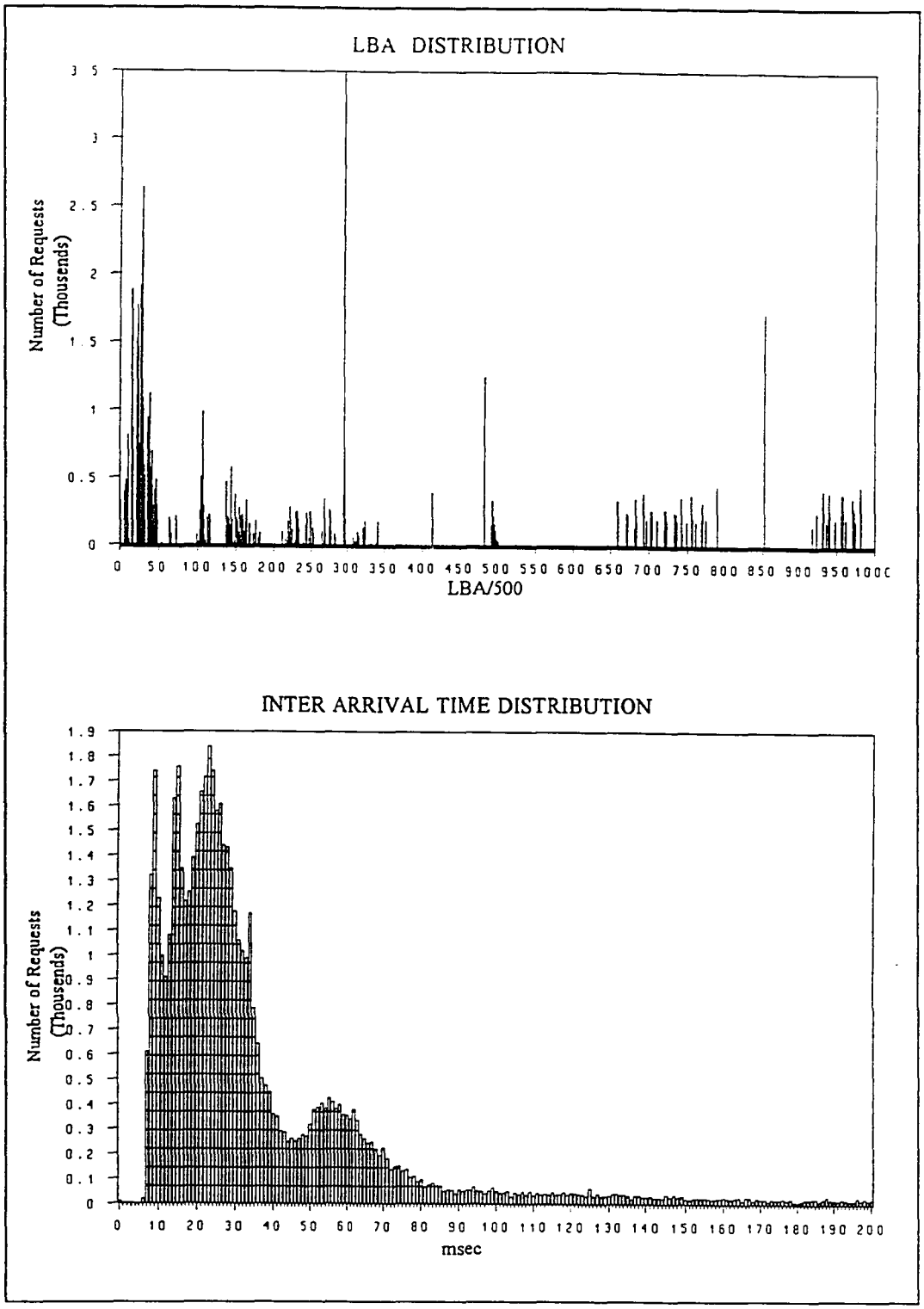

Figure 3.30: LBA and Inter arrival time distributions on first drive.

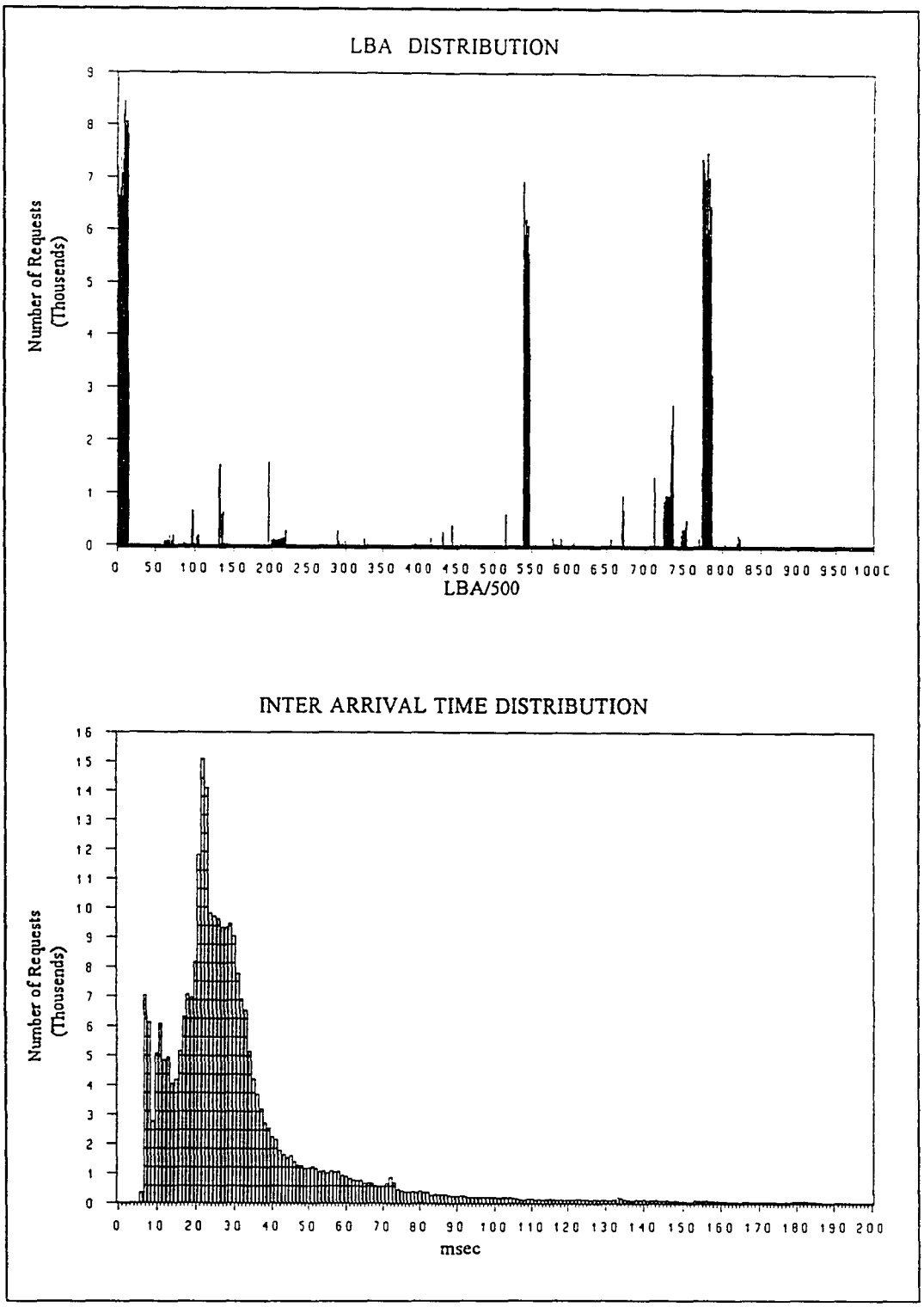

Figure 3.31: LBA and Inter arrival time distributions on second drive.

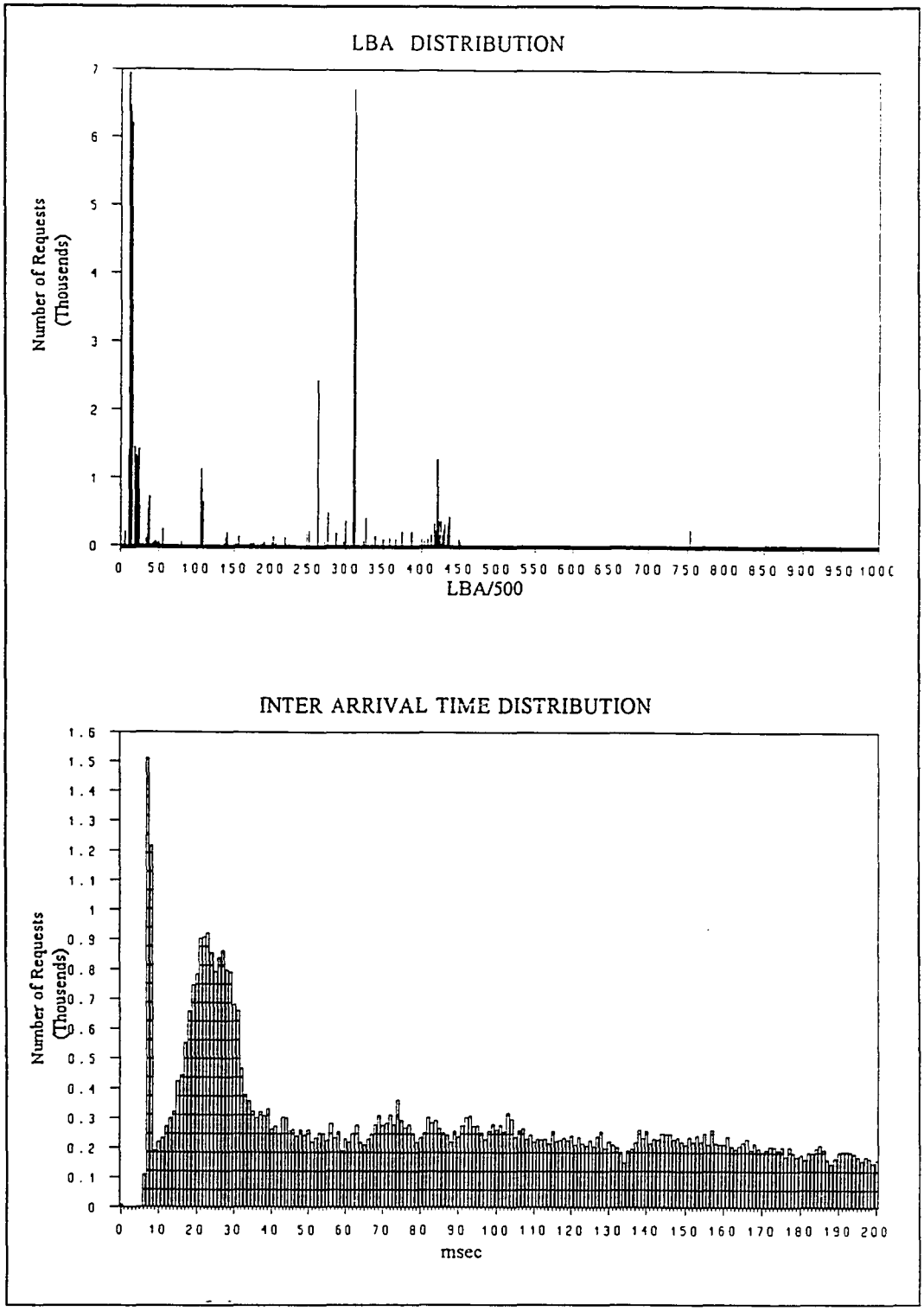

Figure 3.32: LBA and Inter arrival time distributions on third drive.

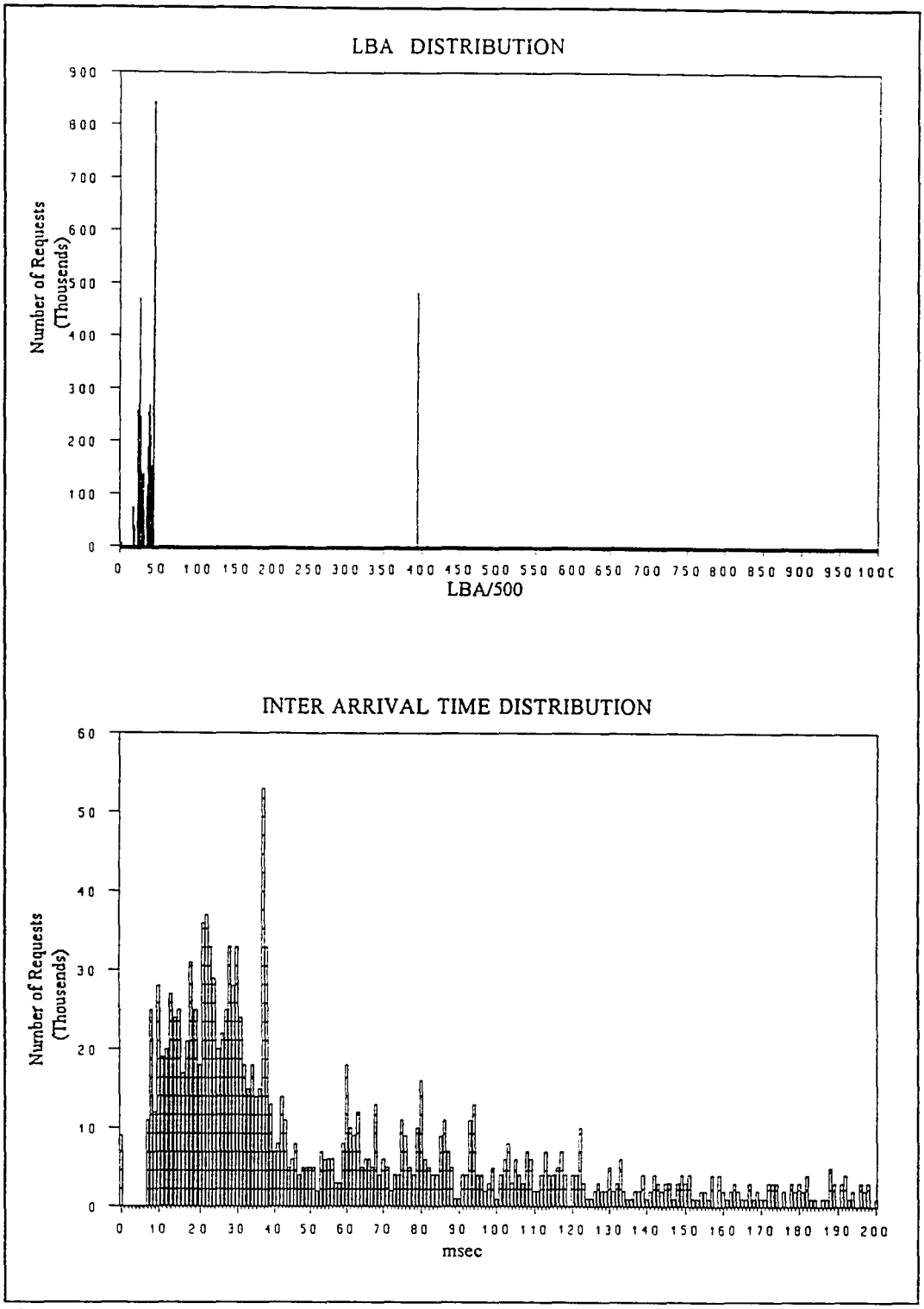

Figure 3.33: LBA and Inter arrival time distributions on fourth drive.

3.2.1: SUMMARY OF WORKLOAD IN THE TRACED OS/2 ENVIRONMENT

Analysis of trace data totalling 838,040 I/O requests in identical PS/2 database servers running OS/2 Database Manager Version 1.2 is performed. These traces were collected on seven sites, one business day at a time. Following is a summary of workload analysis.

o On all the sites drive F is the least accessed drive (getting less than 5% of load for the respective site) and drives C and D got more than 80% of total requests for that site.

o No DASD controller cache hit on a read request is observed. For write requests, the highest hit ratio (2.69%) is observed in drive C's controller at Site6. When investigated, it was found that the SCSI device driver is turning off the read cache. We recommended to upgrade the system to next version of OS/2 and performance was significantly improved as software allowed hardware read caching.

o For all sites, the ratio of read requests to total requests ranged from 73.9% (Site2) to 93.5% (Site3). This read to write ratio falls into the generally expected ratio range for a typical system. Apparently, no significant disk caching is done by the file system. Comparing this ratio for each drive on all sites it was observed that they follow an approximately same pattern. Table 3.8 shows the ratio of read requests to total requests going to a drive. This table (last column) also shows the average of this ratio for each

drive.

Sites → Drives ↓	Site1 (%)	Site2 (%)	Site3 (%)	Site4 (%)	Site5 (%)	Site6 (%)	Site7 (%)	Avg. (%)
'C'	69.38	62.18	71.18	70.31	73.25	66.26	64.32	68.12
'D'	96.38	95.87	98.56	98.81	96.01	94.08	97.10	96.69
'E'	87.65	80.44	95.18	83.21	88.36	80.21	85.58	85.80
'F'	61.64	56.60	60.76	55.86	54.70	56.63	52.49	56.97

Table 3.8: Read requests for each drive on each site.

- On all sites, no traced request is observed to have a size of 128 blocks or greater.

- For read requests, four block is a dominant request size (>60%). Other observed read request sizes are 8, 16, 100 and 127 blocks. It is also observed that read requests are mostly multiples of 4 blocks.

- On all sites, most of the write requests have sizes that range from one block to nine blocks. Dominant request sizes are one block and four blocks. It is observed that on drives C and F, one block is the most frequently (>30%) requested write request size; while on drives D and E at least 75% of read requests submitted to these drives are of size four blocks.

- On each site it was observed that requests going to drive C and D are covering

80% of the LBA space but I/O requests to drives E and F are accessing only initial third of the DASD address space.

- On most of the sites we see a peak-load window of about 2 hours in the afternoon.

- Generalizing above observations it can be concluded that shifting some load from drives C and D to drives E and F may help distribute the load about evenly. This may help improve the overall system response time during peak hours.

3.3: CHARACTERISTICS OF WORKLOAD TRACED IN AIX ENVIRONMENT

In this section, we present the characteristics of the storage subsystem workload in PS/2 working as AIX host. The site profile is given in Table 3.9. The storage subsystem traces containing detailed information about the I/O requests were collected at the storage subsystem level using the SSTT as described in chapter 2. These traces were later processed using tools described in chapter 2 to characterize the workload. The characteristics of the workload at the storage subsystem are summarized in Table 3.10. A total of 1,989,261 I/O requests were traced.

Applications Software development related tools like compilers, linkers, etc. Representative Cycle Time: Unknown.
Server Configuration System: IBM PS/2 model 8595 (486/33MHz) with 32MB RAM. Hard Drives: 2 1.2GB SCSI drives connected to a single SCSI adaptor. Network Adapter: IBM 16/4 Token Ring/A.

Table 3.9: The Site Profile.

Total Transactions:	1,989,261 (Drive 0: 433,210; Drive 1: 1,556,051)
Read Operations:	82.06%
Adapter Cache hit in Read:	28.59% (38.4% of total read)
Write Operation:	17.94%
Adapter Cache hit in Write:	6.66% (37.1% of total write)

Table 3.10: Summary of the storage subsystem workload.

Figures 3.34 and 3.35 show the request size distribution. More than 95 % of requests are 8-block requests. It can also be observed that request sizes are 1 or 2 blocks or a multiple of 8 blocks. Figure 3.36 shows the LBA frequency distribution. The smallest LBA accessed is 0 and the largest LBA accessed is 2,936,584. In the figure, the LBA is divided by 500 and truncated to accommodate the LBA range in the limited space available for a figure. It can be observed that except for the very beginning of the address space (where 12% of the requests have accessed an LBA < 17,500) the LBA accessed has been quite uniformly distributed over the address space.

Figures 3.37 and 3.38 show the characteristics of the inter-arrival time in the traced storage subsystem in the AIX environment. Most (80%) of the requests are within 42 msec of the previous request. About 25% of the requests have an inter-arrival time of 6±2 msec. Another 21% of the requests arrive within 25±4 msec of the previous request. This reflects an almost continuous load on the storage subsystem with very few long idle times.

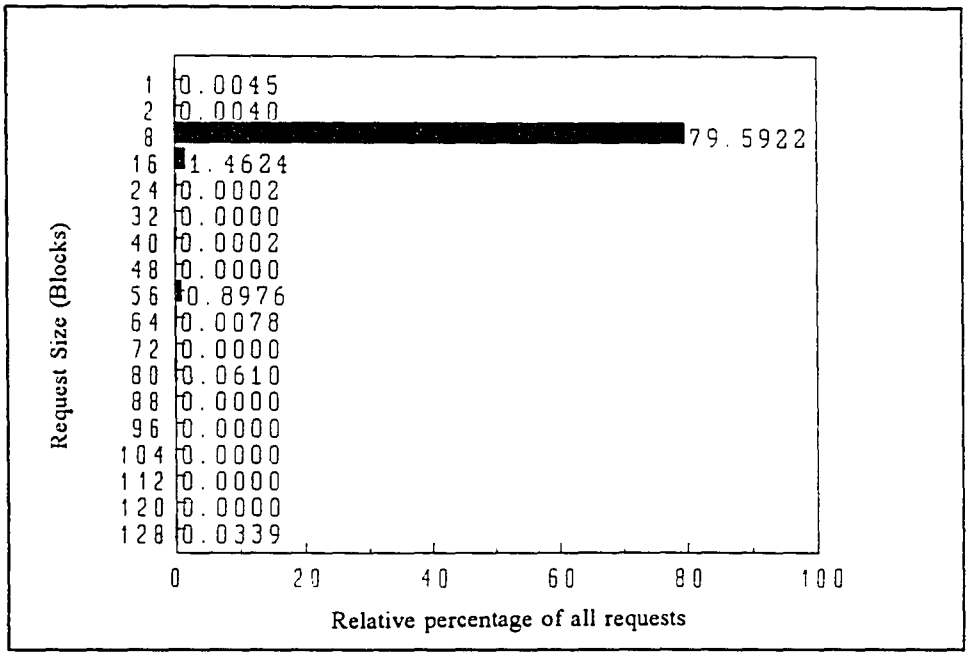

Figure 3.34: Read Request Size Distribution.

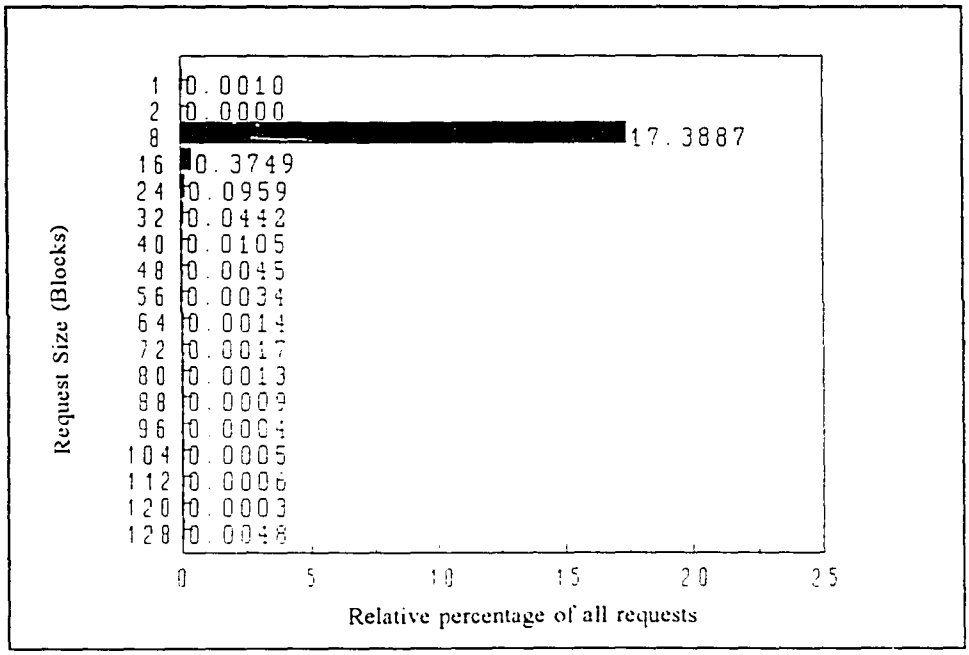

Figure 3.35: Write Request Size Distribution.

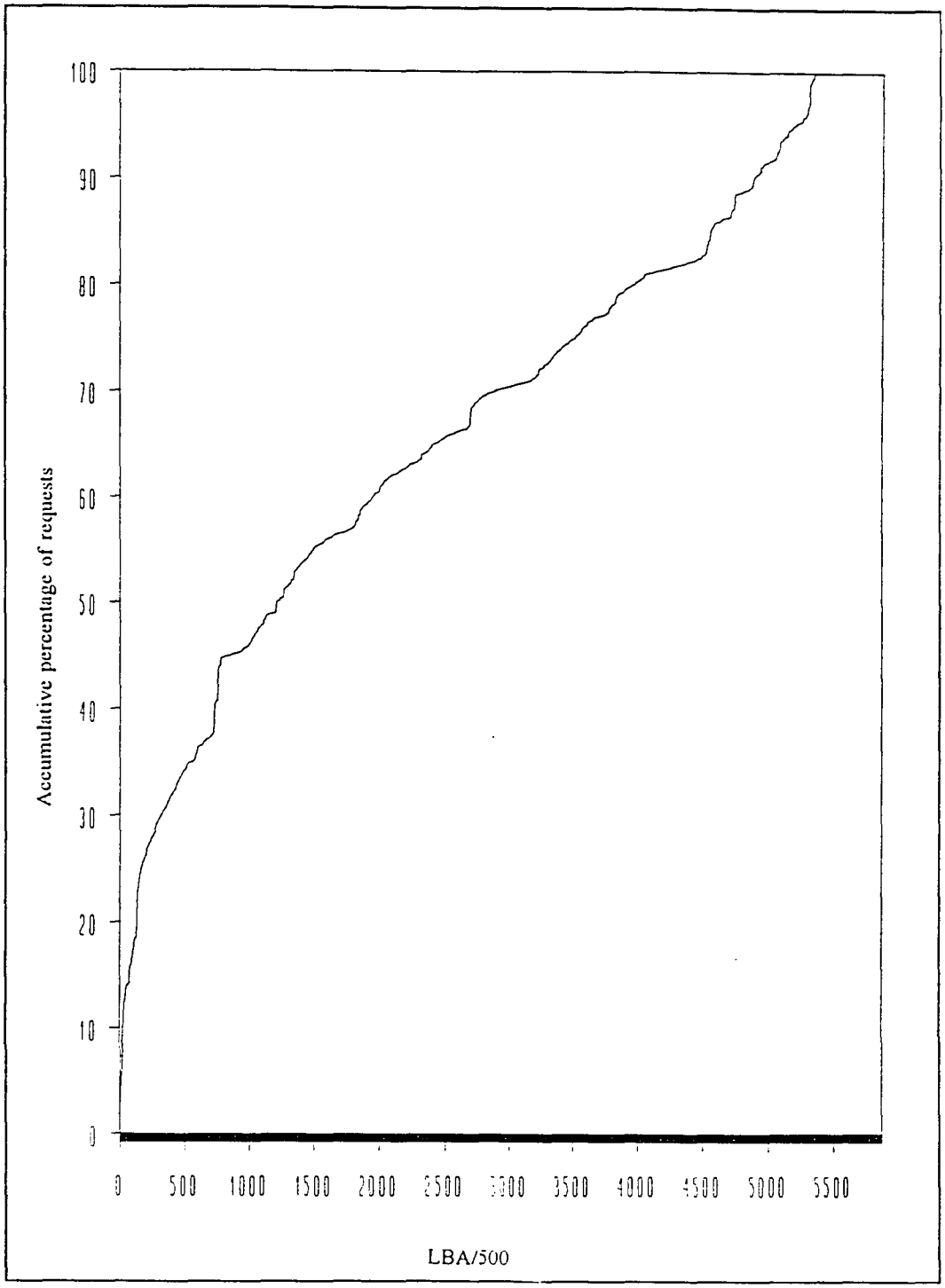

Figure 3.36: LBA Frequency Distribution.

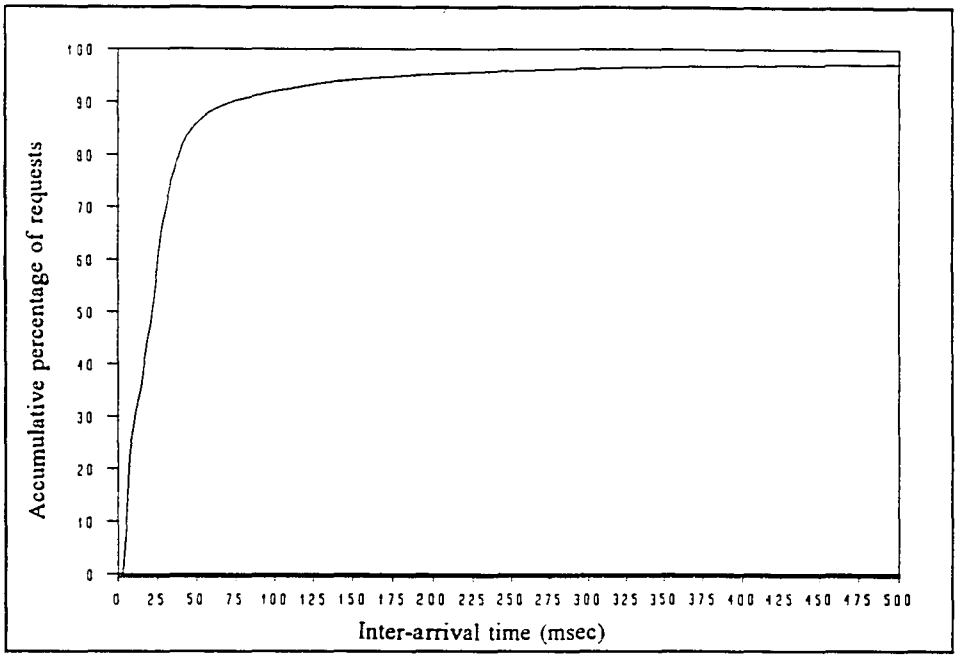

Figure 3.37: Inter Arrival Time Frequency Distribution.

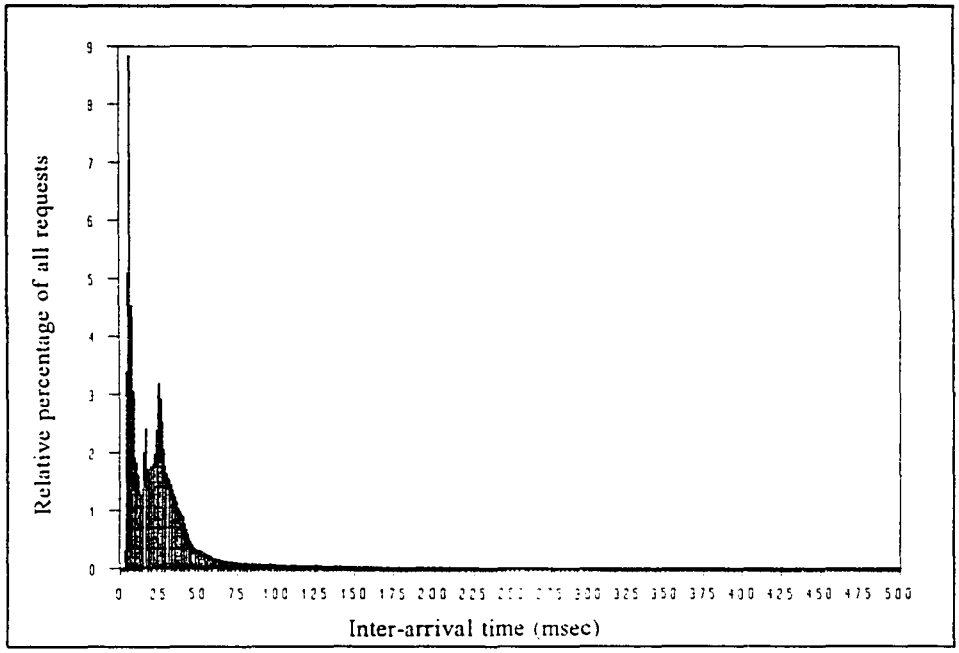

Figure 3.38: Inter Arrival Time Frequency Density.

CHAPTER 4

GENERATION OF SYNTHETIC WORKLOAD IN THE NETWARE ENVIRONMENT

In the previous chapter, we have discussed workload characterization using workload traces collected by the SSTT. These traces can also be used to evaluate the performance of storage subsystems which are still on the drawing board or in some development stage. The workload traces as collected by SSTT can be used to drive a simulation model of the storage subsystem. The performance results of a simulation model, thus driven, would be more realistic than the estimated workload driven model. Using simple workload traces for running simulations may involve a huge processing time due to large number of trace in a representative piece of workload. The synthetic workload retains important characteristics of the original workload while its size is reduced significantly. The synthetic workload, due to its reduced size, can help save processing time when running workload driven storage subsystem simulation models during design validation cycle. We also can map the workload for newer subsystems or environments during the synthesis of workload. Finally, the synthetic workload (or even the originally traced workload) can be used to drive prototypes of new storage subsystems during engineering validation

cycle.

In this chapter, a method of generating a drive workload for storage subsystem, representative of a real workload in the NETWARE environment, is described. This method is based on a technique of data reduction using data clustering. The following is an overview of existing data clustering techniques.

4.1: CLUSTERING

Data clustering [17,18] is a technique which subdivides a set of data points into groups so as to make the differences among the members of a group smaller, according to a certain criterion, than those among members of different groups.

4.1.1: HIERARCHICAL METHODS

AGGLOMERATIVE METHODS:

These methods construct a tree by grouping the closest points initially, then the points closest to the cluster so generated, and so on, until all points have been grouped in a single class. In the beginning the number of clusters is equal to the number of data points and at the end there is only one cluster. Depending on the size and requirements the process of clustering can be stopped at any level during this tree construction.

DIVISIVE METHODS:

These methods start with one group of data and all data points belong to this group. In

the first step it splits the data in two parts and then goes on by dividing them further into smaller groups. The grouping does not have to be exactly on the center of each subgroup, rather it is a user defined. The splitting can be done along naturally existing gaps within subgroups.

CLUSTERING USING EUCLIDIAN DISTANCE MATRIX:

A matrix is constructed based on euclidian distance between data points. Then points within a certain distance of each other are clustered. At this point one data point may be assigned to two different clusters. Such data points are assigned to the cluster whose center is closest to the data point.

MONOTHETIC CLUSTERING:

Data points are grouped along each variable axis, one axis at a time. Distance limit along each variable axis has to be predefined. For a given set of data and parameters there is one unique grouping.

4.1.2: NON HIERARCHICAL METHODS

In non-hierarchical methods, the number of clusters 'K' is predefined. 'K' points, called centeroid, are selected at equal distances over the range of the data set. Data points are assigned to the nearest centeroid. A data point belongs to only one cluster.

4.2: NETWARE WORKLOAD SYNTHESIS

The synthetic workload retains important characteristics of the original workload while its size is reduced significantly. The real workload of the storage subsystem in the NetWare environment can be characterized by a mix of the following:

1) Request type (read or write),
2) Request size,
3) Logical Block Address (LBA) of the request, and
4) Inter-arrival time distribution.

The Synthetic Workload is generated in three steps:

1) Data Clustering
2) Computing a Transition Matrix
3) Synthesis of traces using transition matrix

In this section, we develop a data clustering and workload synthesis technique to generate synthetic workload for netware environment.

4.2.1: DATA CLUSTERING

Clustering on the basis of euclidian distance matrix seems to be the best option, but this method is very expensive in terms of computer resources if the number of data points exceeds few hundreds. In simpler terms this method requires at least $3n^2$ bytes of memory

space for n data points. Even if we use hard drive as part of main memory the computing time is enormous when using a 80386 PC (assuming the number of data points in a representative piece of workload to be at least five thousand, the memory requirement for the distance matrix is at least 75 million bytes). After considering the data size and computing requirements of different clustering techniques we chose monothetic clustering. In monothetic clustering data is clustered along one variable axis at a time. We have clustered the data along three dimensions, the three dimensions being request type, size and LBA. Clustering along request-type axis gives us two clusters (ie. read and write). Then clustering along size axis we divide these traces into more clusters, each cluster having requests of same type and size. Then these clusters are further divided along LBA axis keeping all those traces which are within a distance limit from each other in the same clusters.

In NetWare file system all read requests have same size which is equal to the maximum allowed in the system. By default NetWare sets this number to eight blocks. Clustering along request type axis and request size axis generates a maximum of nine types ie. write one block, write two blocks, write three blocks, write four blocks, write five blocks, write six blocks, write seven blocks, write eight blocks and read eight blocks. Using the nearest neighbor distance limit as a variable we can control the number of clusters generated.

Following is a complete description of our clustering method:

1) Convert raw traces into BNJ format and assign an integer sequential case number to each transaction (trace) in the BNJfile.

2) Split BNJ file into nine files as follows:

File 1: Transactions requesting 8 block read operation

File 2: Transactions requesting 1 block write operation

File 3: Transactions requesting 2 block write operation

File 4: Transactions requesting 3 block write operation

File 5: Transactions requesting 4 block write operation

File 6: Transactions requesting 5 block write operation

File 7: Transactions requesting 6 block write operation

File 8: Transactions requesting 7 block write operation

File 9: Transactions requesting 8 block write operation

3) Sort each file in ascending LBA of the request

4) Cluster requests which are close to each other (LBA wise) and assign a unique number to each cluster (cluster number). The closeness is defined by some minimum distance threshold in numbers of sectors. It should be mentioned that,

the number of resulting clusters is very sensitive to this threshold. A smaller distance limit can result in a larger number of final clusters and vice versa.

5) Each request of the BNJ file in step 1 gets a corresponding cluster number from step 4.

4.2.2: COMPUTING TRANSITION MATRIX

A square matrix $A=[a_{ij}]$, called transition matrix, is formed such that the number of rows is equal to the number of clusters in step 4 of clustering. Element a_{ij} (transition element) defines conditions of transition from the ith cluster to the jth cluster.

Fields of a transition element are:

Pr : Probability of going from cluster (row) 'i' to cluster (column) 'j'.

LBAmin: Smallest LBA requested by a transaction preceding i-j transition.

LBAmax: Largest LBA requested by a transaction preceding i-j transition.

DeltaMin: Smallest sojourn time for i-j transition.

DeltaMax: Largest sojourn time for i-j transition

Pop: Population of the transition element, i.e. number of transitions from ith cluster to jth cluster in original trace.

4.2.3: SYNTHESIS OF TRACES

In this section, we develop an algorithm for trace synthesis. Using the transition matrix generated in the previous section we can generate synthetic workload traces. The size of generated synthetic workload can be controlled by introducing two variable parameters, i.e. reduction factor and discard factor. Reduction factor is used to reduce the original workload by this factor. Discard factor is always less than reduction factor and it is used in case when the population of a transition element is so small that we want to ignore it.

We start from a randomly selected cluster. Using random number generator a number 'x' is selected ($0 <= x < 1$). With the help of 'x' and 'Pr' field of the transition records in the present cluster the next cluster is selected using following test:

1- If population of the transition record is zero then a new 'x' is selected.

2- If population of the transition record is not less than the reduction factor then it is reduced by the reduction factor. Next cluster number is equal to the present column number.

3- If population of the transition record is less than reduction factor but not less than discard factor than population is made zero. Next cluster number is equal to the present column number.

4- If population of the transition record is nonzero but less than the discard factor then population is made zero and it is added to the next nonzero transition record population in the same row

5- If population of the present cluster (row of the transition matrix) has vanished or it is less than the discard factor then a randomly selected cluster is made the present cluster.

When a transition is made from cluster '*i*' to cluster '*j*' then :

1- Knowledge of present CLUSTER # determines the OpCode and Request Size of the synthesized trace.

2- A random LBA is selected within the range of LBAmin and LBAmax of that a_{ij} (transition record).

3- Similarly, sojourn time can be selected, but in NetWare environment our assumption of uniformly distributed sojourn time within DeltaMin and DeltaMax is not valid. Therefore, we are adding this information during second pass using normalized frequency function of interarrival time of original trace.

4.3: VALIDATION OF SYNTHETIC WORKLOAD ON THE BASIS OF STATISTICAL CHARACTERISTICS

We have validated the synthetic workload by comparing the statistical characteristics of the original and synthetic workload. We synthesized workloads with different reduction ratios and then we compared block size distribution, LBA distribution and interarrival time distribution. Figure 4.1 shows the read/write block size distribution of the original and synthesized workload where the workload size has been reduced by a factor (reduction factor) of 30, i.e. original and synthetic workload sizes have a ratio of 30:1. The difference is within about 10%. Figure 4.2 shows same comparison but with a reduction factor of 15. The difference is within about 5%. The highest discrepancy (5%) appears for three block write requests otherwise it is well within 3%. Similarly, in figure 4.3 where original workload is same as in figure 3.2 but the reduction factor is 17, we see a difference of less than 5%. Figure 4.4 shows comparison of request distribution between real workload of day-one and synthetic load generated with a reduction factor of 19. The differences are within 8% except in case of write requests of eight blocks. Earlier analysis (section 3.1) has pointed towards unusually high number of write requests concentrated during a short span of time, all going to close proximity on the DASD.

Figure 4.5 (real workload) and figure 4.6 (synthetic workload) show a comparison of LBA distributions (x-axis) between original of day-three synthetic workload with a reduction factor of 30. Similarly, figures 4.7 and 4.8 show comparison with reduction factor of 15 for workload of day-two. Figures 4.9 and 4.10 show a comparison of LBA distributions

for original workload of day-one and its synthetic representative with a reduction factor of 15. In these three comparisons, we see that major spikes in the LBA distributions of original traces are preserved in the respective synthetic workload.

Figures 4.11 and 4.13 show interarrival time distributions for original workload of day-three and day-two respectively; and 4.12 and 4.14 show interarrival time distributions for synthetic workload representation of day-three and day-two with a reduction factors of 30 and 15 respectively. Here again the synthetic workload retains the characteristic interarrival time distributions as in original workload.

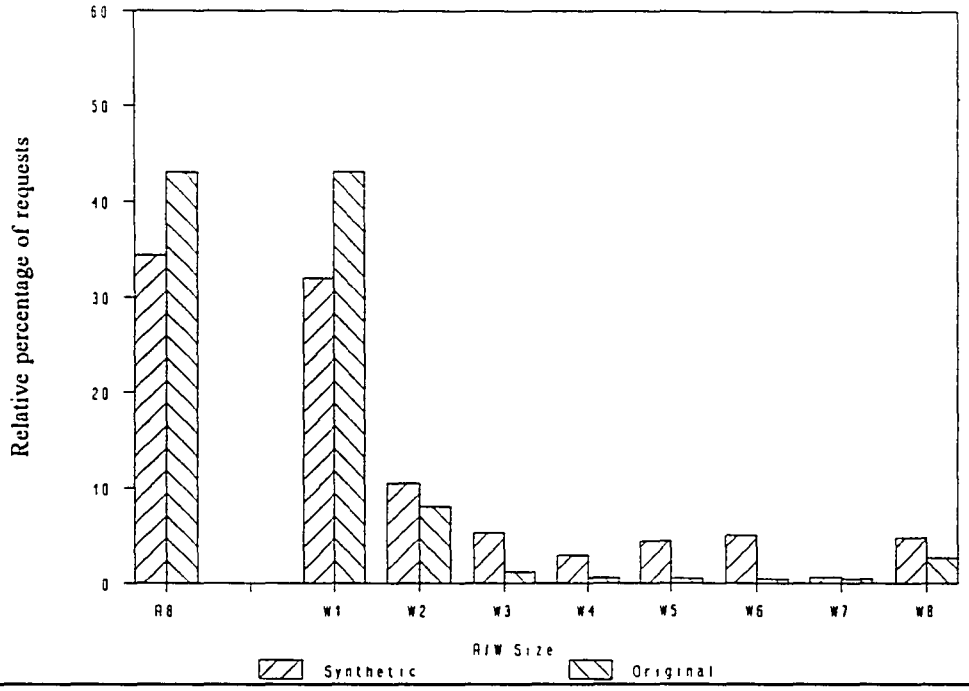

Figure 4.1: Comparison of Request Size Distribution. Original from day-three.

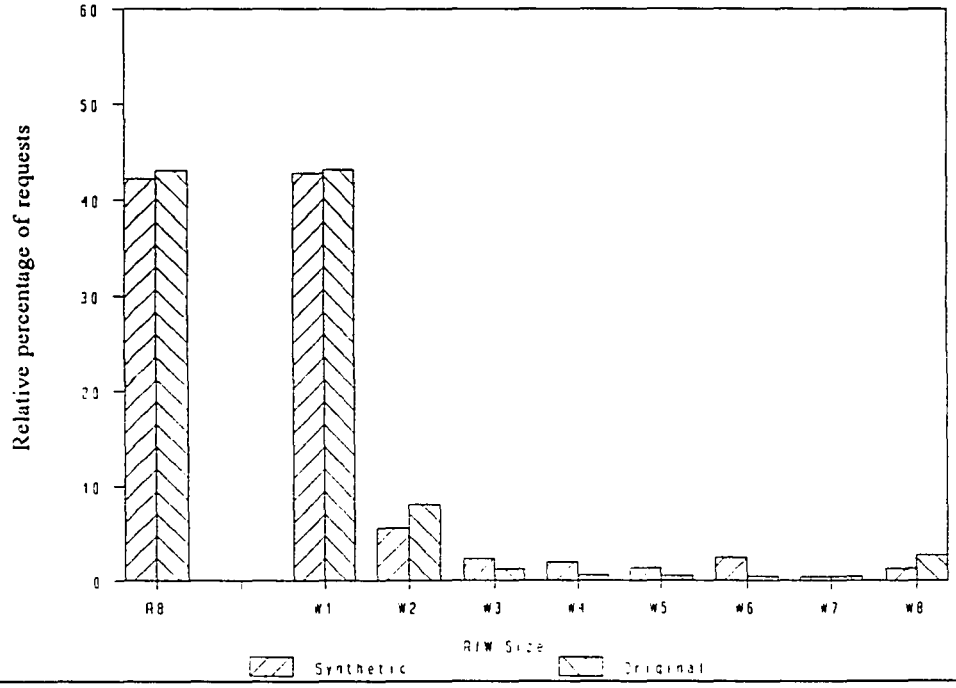

Figure 4.2: Comparison of Request Size Distribution. Original from day-three.

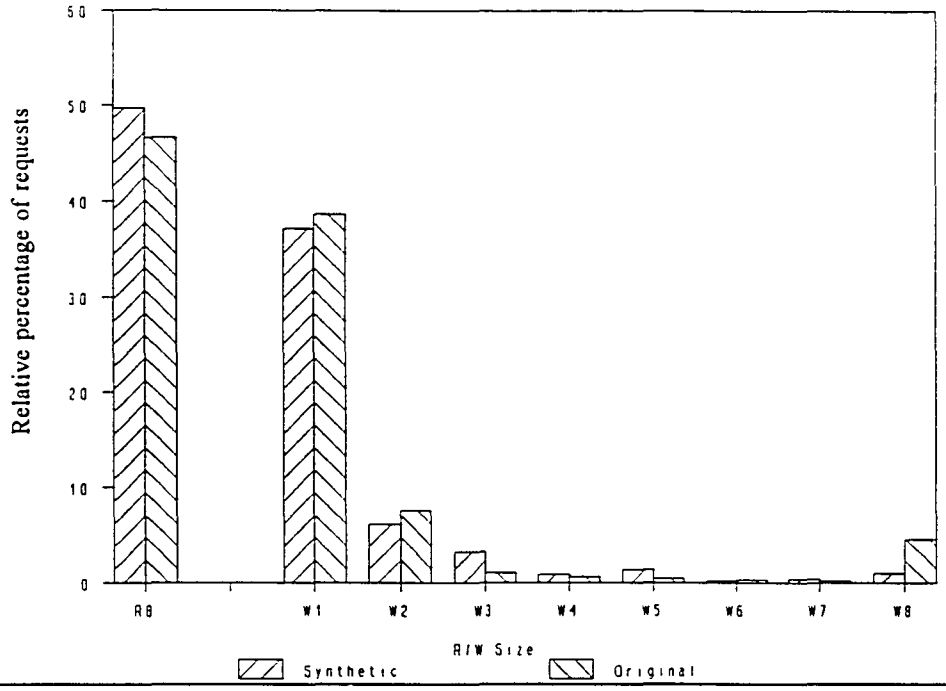

Figure 4.3: Comparison of Request Size Distribution. Original from day-two.

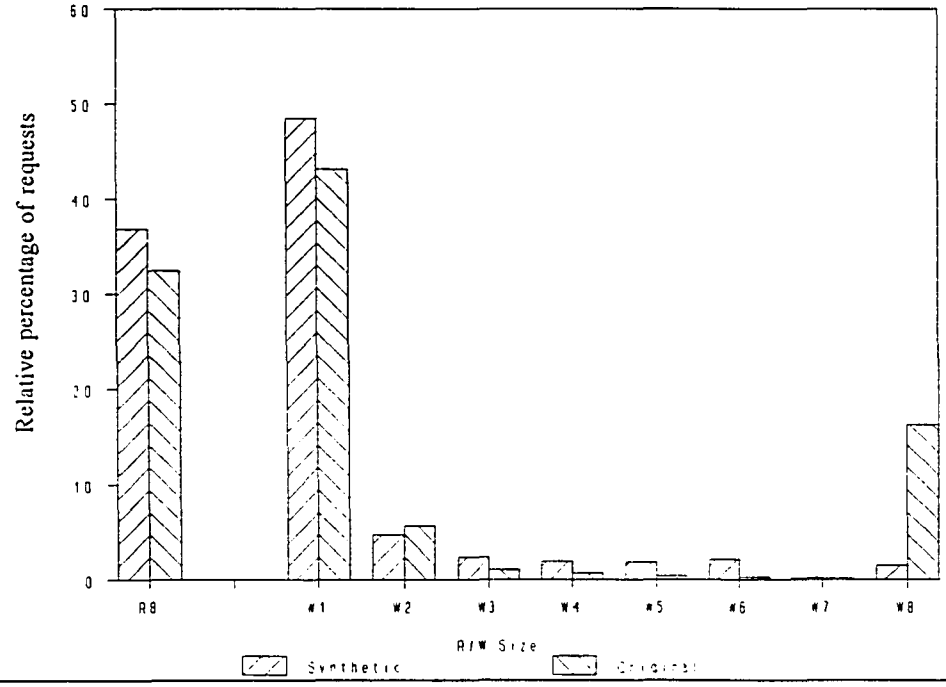

Figure 4.4: Comparison of Request Size Distribution. Original from day-one.

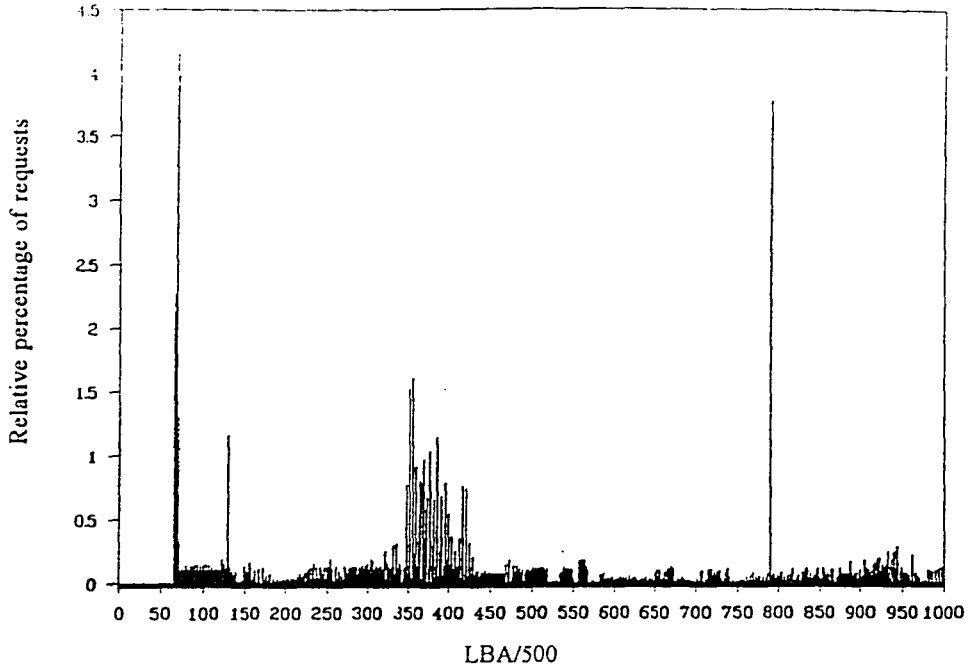

Figure 4.5: LBA distribution for day-three workload.

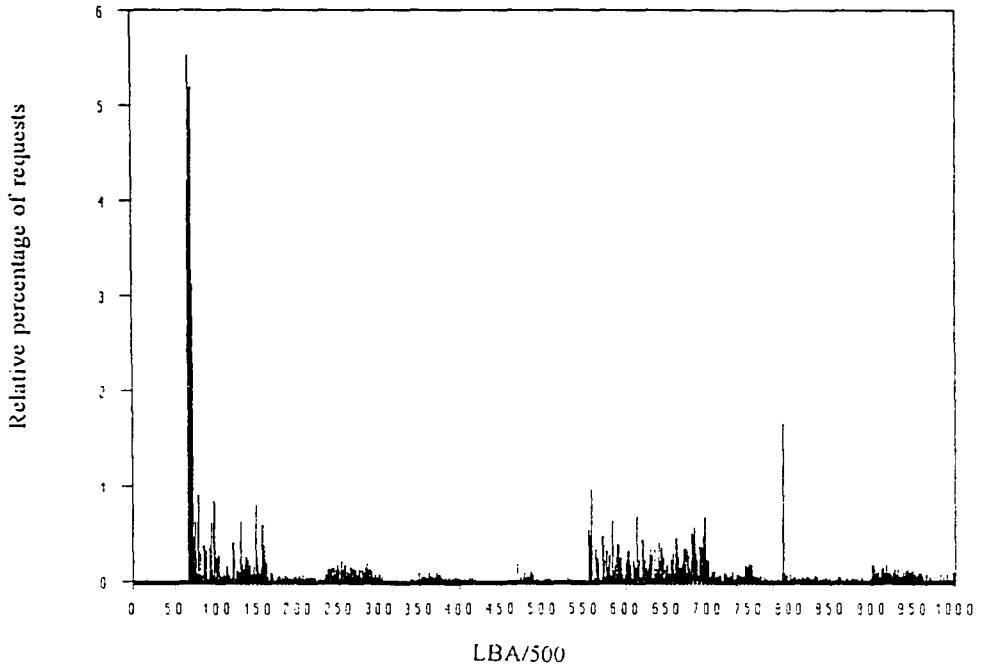

Figure 4.6: LBA distribution for synthetic (day-three) workload.

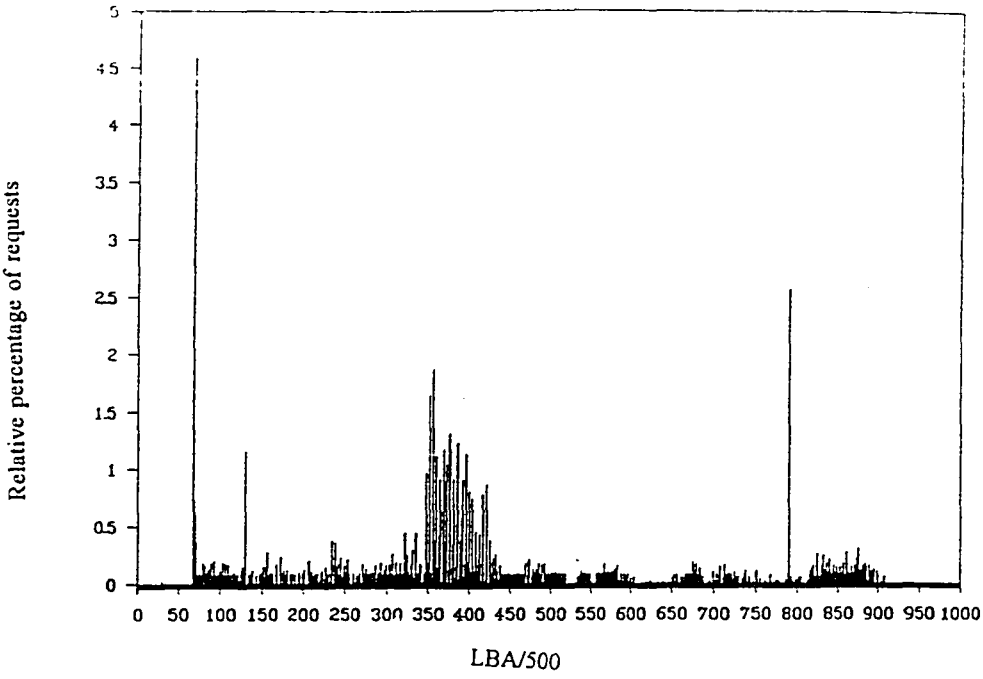

Figure 4.7: LBA distribution for day-two workload.

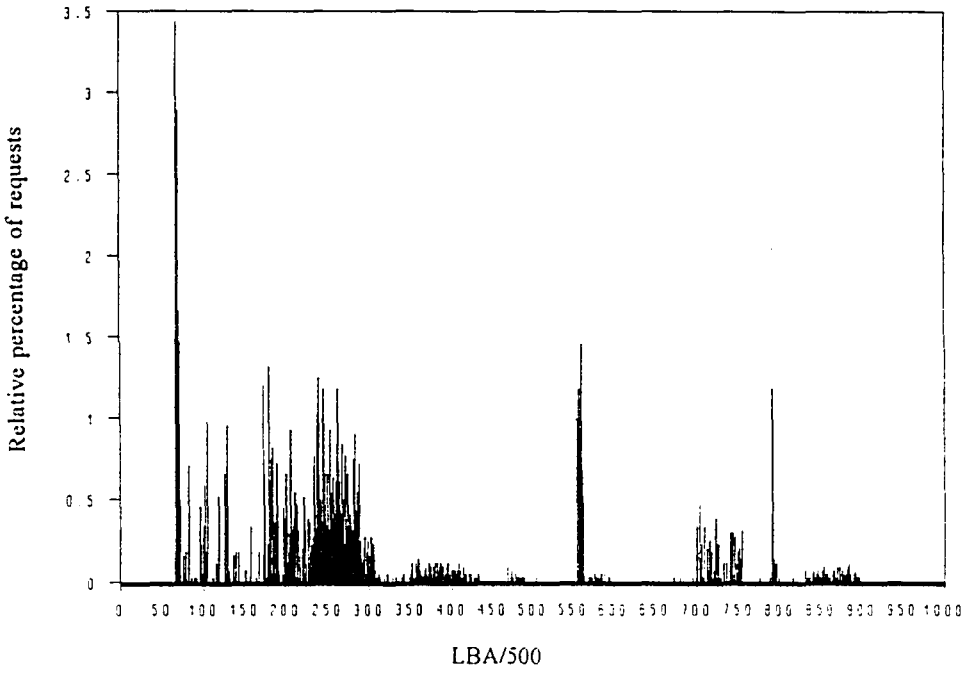

Figure 4.8: LBA distribution for synthetic (day-two) workload.

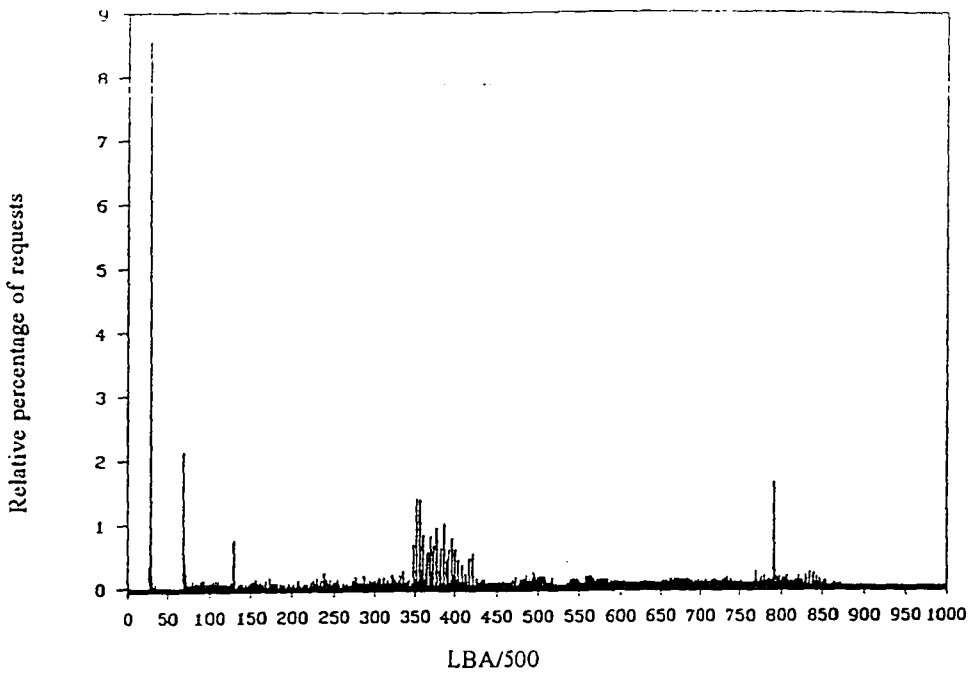

Figure 4.9: LBA distribution for day-one workload.

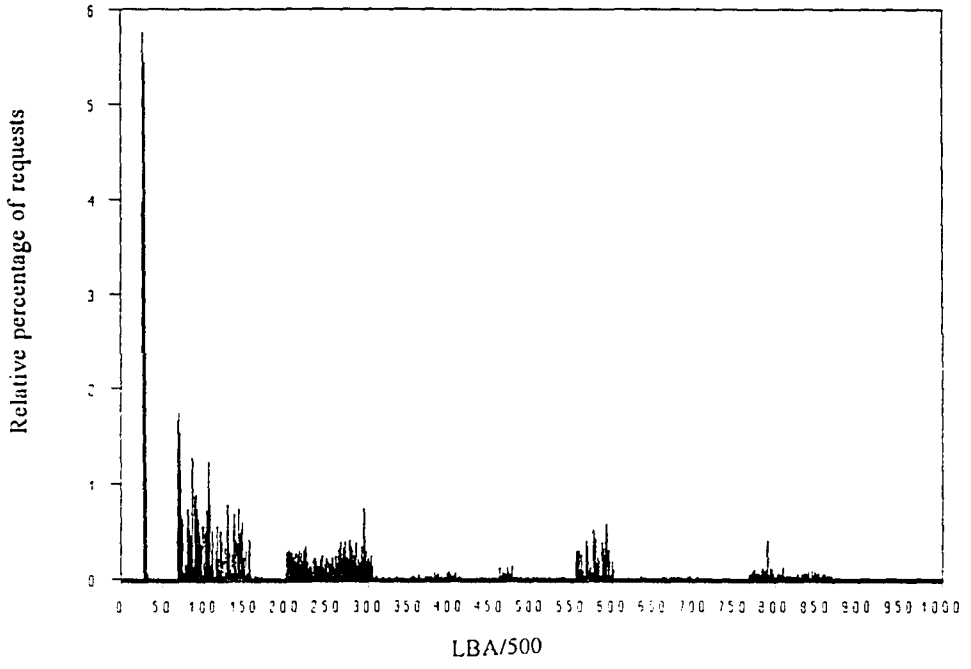

Figure 4.10: LBA distribution for synthetic (day-one) workload.

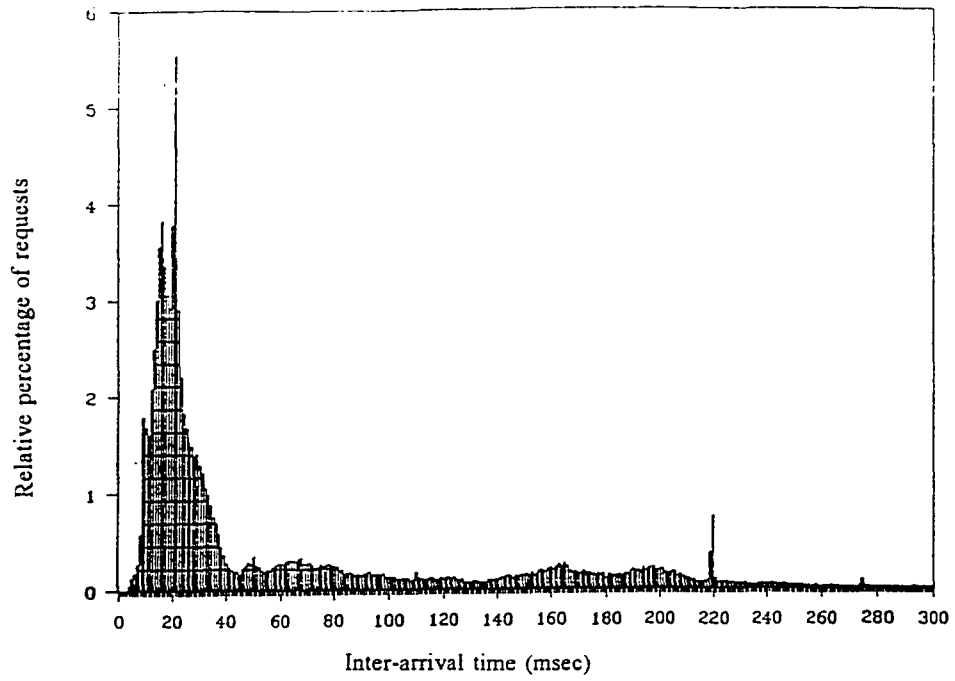

Figure 4.11: Inter-arrival time distribution for day-three workload.

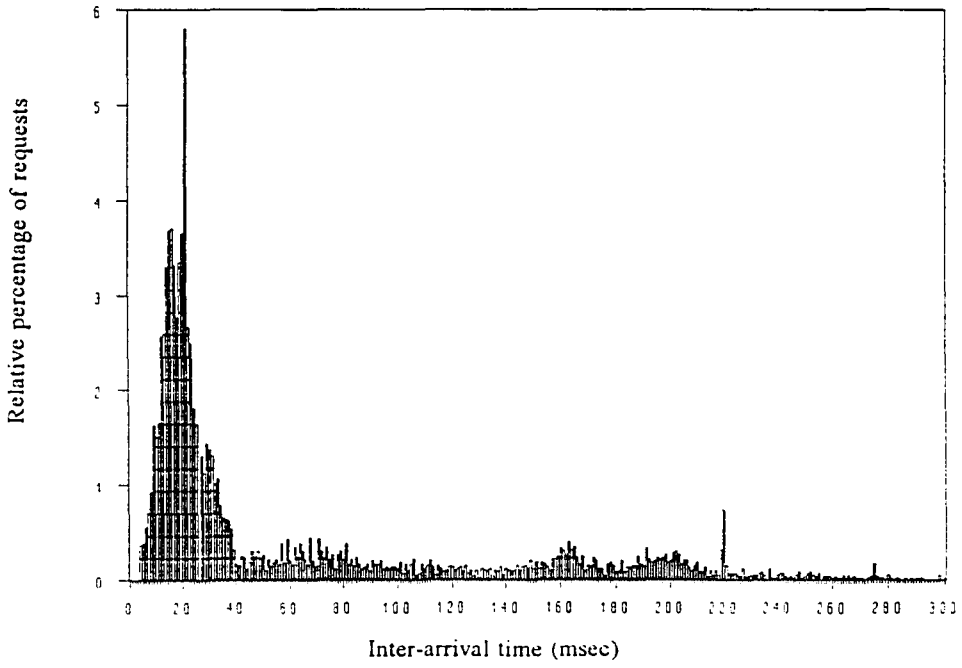

Figure 4.12: Inter-arrival time distribution for synthetic (day-three) workload.

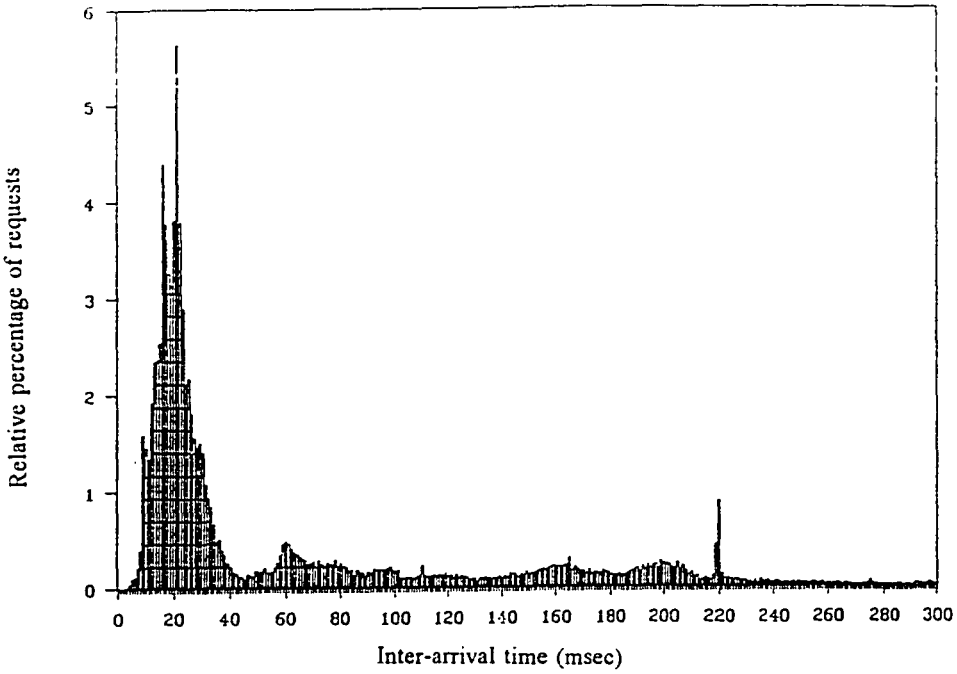

Figure 4.13: Inter-arrival time distribution for day-two workload.

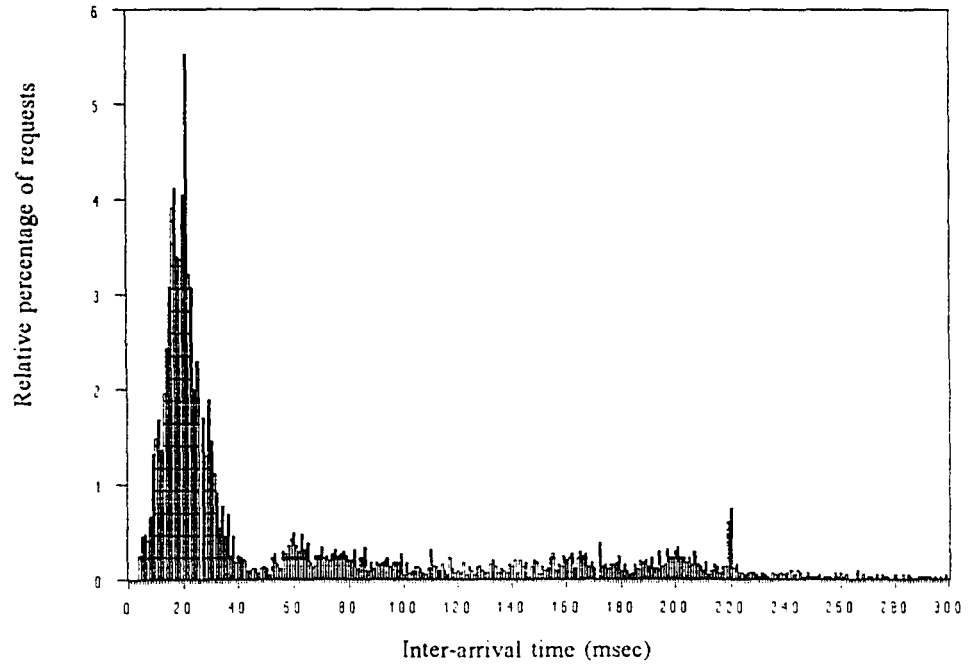

Figure 4.14: Inter-arrival time distribution for synthetic (day-two) workload.

CHAPTER 5

CONCLUSIONS AND FUTURE DIRECTIONS

In this study, an operating system independent tool has been developed which traces the storage subsystem workload at the subsystem level. Due to hardware assistance, this tool has very little overhead. This is achieved by incorporating the tracing routine in the firmware of the IBM SCSI adapter. The I/O traces are collected, in realtime, on a PS/2 connected to the modified SCSI adapter via a special asynchronous (serial) port. These traces are then processed and converted into formats useable by other specially developed software tools.

We proposed a workload characterization scheme and developed tools to implement it. One set of tools developed characterize the workload by studying statistical parameters of the traced workload. These parameters include LBA distributions, interarrival time distribution, size distributions, ratios between read requests and write requests, adapter's cache performance. In this study, workload characterization of storage subsystems in NetWare, OS/2 and AIX environments was performed.

The study of the NetWare environment shows that the file system block size is set for

eight blocks. It is observed that all read requests are eight block requests and most of write requests (more than 63%) are one block requests. Further analysis shows that 85% of the read requests were only one 8-blocks requests. Which means controller cache with a threshold of more than 17-blocks (current threshold) may not buy much. We found, although, subsystem performance under the peak transient load (i.e. averaged over one second) can be improved by increasing the number of subsystem components (e.g. adapter and drives), the peak steady-state load (i.e. averaged over one hour) is well below the existing subsystem throughput capacity. The system has two drives (table 3.1) but one drive is rarely accessed. It is observed that some response time reduction can be easily achieved by spreading data over both drives such that they are accessed uniformly.

By studying the OS/2 systems (each with four drives) we found that system performance can be improved by distributing more load on third and fourth drives (i.e. drives E and F). It was also observed that in this environment the controller's cache had no hits on read requests and very few hits on write requests. Further investigation attributed this to the OS/2 device driver for SCSI card. The user moved to the next version of OS/2 where SCSI device driver did take advantage of cache on the SCSI adapter and a significance performance improvement was noticed. Most of the requests were four block requests and there were no requests of 128 blocks or greater. On each site three fourths or more of the requests were read requests. It may be concluded that for this particular environment a storage subsystem tuned for 4 block requests would enhance the system performance.

By studying the AIX system we found that 95% of the requests are eight block requests and more than 80% of the requests are read requests. About 25% requests have an interarrival time of 6±2 msec and about 80% of the requests have an interarrival time of 42msec. The LBA requests are uniformly distributed. Here again, a storage subsystem tuned for eight block read requests would probably have a better performance.

An algorithm to synthesize the workload is also developed and implemented as part of this study. The synthetic workload retains all important characteristics of the original workload but its size is reduced. We successfully synthesized storage subsystem workload for NetWare using traces collected in real NetWare environment. We then compared the synthetic workload with the original workload and found it to be within reasonable margin of 2% to 8%.

5.1: FUTURE WORK

In terms of future direction for this research there are various interesting issues to be studied.

- Characterization of workload based on information about requesting thread or process.
- Using interarrival time in the matrix for workload synthesis.
- Mapping workload across different DASD space. This would make synthetic workload independent of original DASD type and thereby be of more value during the design process of new technologies.

REFERENCES

1. Moad, J., "Relief for Slow Storage Systems," DATAMATION, September 1, 1990.

2. Smith, D., "The M212 - A Zero-Glue Single-Chip VLSI Winchester Controller with On-Board 10 MIPS Processor," Wescon/86 - Conference Record, November, 1986, Anaheim, California.

3. Grossman, C.P., "Cache-DASD storage design for improving system performance," IBM System Journal, v24, Nos.3/4, 1985.

4. Goyal, A. and Agerwala, A., "Performance Analysis of future shared Storage Systems," IBM Journal of Research and Development, vol. 28, no. 1, January, 1984.

5. Beretvas, T., "DASD Performance Analysis Using Modeling," CMG '85 Conference Proceedings, December, 1985, Dallas, Texas.

6. Lim, P.K. and Tien, J.M., "A Tool For Direct Access Storage Device (DASD) Performance Evaluation," Proceedings of the Urban Regional Information Systems Association Conference, August, 1989, Boston, MA.

7. Smith, B.J., "A Survey of the State of the Art and Practice In I/O Subsystem Modeling and Analysis," Proceedings of the Fall Joint Computer Conference, November, 1986, Dallas, Texas.

8. Houtekamer, G.E., Measuring and Modelling Disk I/O Subsystems, Ph.D. dissertation, 1989, Technische Universiteit te Delft, The Netherlands.

9. Zhou, S., DaCosta, H., and Smith, A.J., "A File System Tracing Package for Berkeley UNIX," USENIX Association Summer Conference Proceedings, 1985, Portland.

10. Bishop, D.A., DEKKO and PCPERF/VMPERF Performance Trace User's Guide, 1991, IBM.

11. PS/2 Micro Channel SCSI Adapter Technical Reference, IBM Corporation, Armonk NY, 1990.

12. Peterson, J. L., Petri Net Theory and the Modeling of Systems, Englewood Cliffs, NJ; Prentice Hall, 1981.

13. M. A. Marson, G. Balbo and G. Conte, "A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems." ACM Trans. on Computer Systems, Vol. 2 No. 4, May 1984.

14. The new Peter Norton programmer's guide to IBM PC and PS/2, Microsoft Press, Redmond, Washington, 1988.

15. Personal System/2 and Personal Computer BIOS Interface Technical Reference, IBM Corporation, Armonk NY, 1988.

16. Personal System/2 Hardware Interface Technical Reference -Common Interfaces, IBM Corporation, Armonk NY, 1990.

17. Everitt, B., Cluster Analysis, London; Heinemann Educational Books Ltd, 1974.

18. Ferrari, D., Serazzi, G., Zeigner, A., Measurement and Tuning of Computer Systems, Englewood Cliffs, NJ; Prentice Hall, 1983.