ELSEVIER

Information Sciences 112 (1998) 239–266

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

Personal computer storage subsystem workload: Measurement and characterization

Imad Mahgoub *, Ahmad Ali

Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431, USA

Received 1 June 1997; accepted 29 April 1998

Communicated by Ahmed Elmagarmid

Abstract

This paper discusses the design and implementation of an operating system independent tracing tool for storage subsystem workload in personal computers. This tool measures the workload at the storage subsystem level, and due to hardware assistance it has very little overhead. The workload traces are collected in real-time on a data collection station executing a special data collection program. We use the tool to collect workload traces in real-life Netware-based personal computer environments. We then develop a scheme to analyze and characterize the traced workload.

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,

* Corresponding author. E-mail: imad@cse.fau.edu

0020-0255/98/$19.00
PII: S0020-0255(98)10036-1

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 [4–6] 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 include subsystem data transfer channel, drive head assembly, look-ahead buffer in drive.

For valid conclusions to be drawn from performance evaluations, based on simulation, 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. Fig. 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 and more helpful for storage subsystem hardware designers.

There is a 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

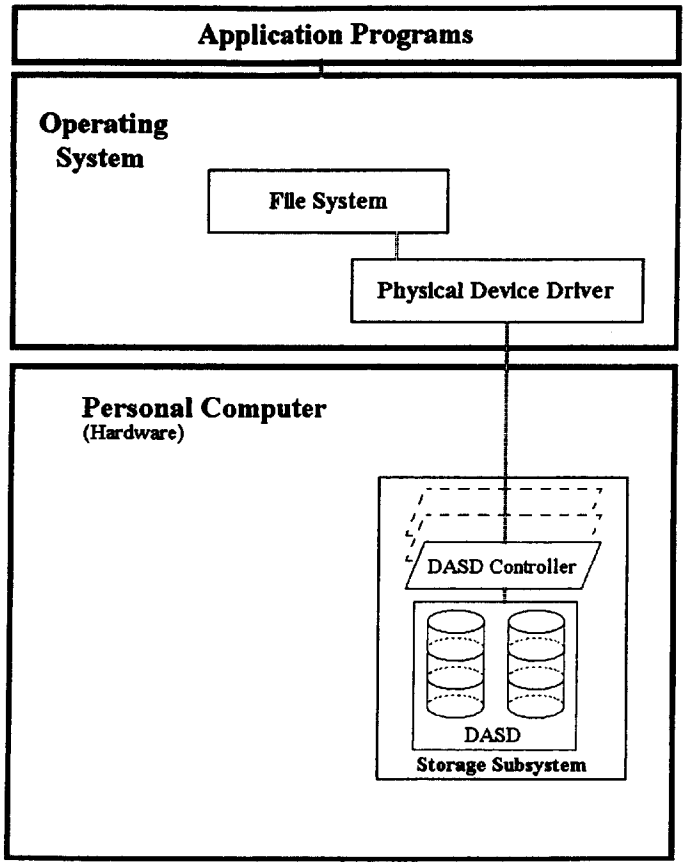


Fig. 1. A simplistic view of access path to the storage subsystem in personal computer.

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 motivated this research. This paper discusses the design and implementation of an operating system independent tracing tool for the storage subsystem workload in personal computers. This tool measure the workload at the storage subsystem level. Due to hardware assistance, this tool has very little overhead. This is achieved by incorporating the tracing routine in the firmware of an IBM PS/2 SCSI adapter. The workload traces are collected in real-time, on a PS/2, executing a special data

collection program, connected to the modified SCSI adapter via an asynchronous link. We also discuss a storage subsystem workload characterization scheme by studying the parameters of the traced workload. These parameters include LBA distribution, interarrival time distribution, request size distribution, ratios between read requests and write requests, adapter's cache performance. This tool was used to trace the storage subsystem of Novell Netware fileserver in a real-life environment. The paper also presents characterization of this workload. Section 2 describes the tools developed for workload tracing and post processing. Workload characterization is presented in Section 3. Finally, conclusions are included in Section 4.

2. Description of tools developed

In this section, 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 present tools developed to post process 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 [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.

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

2.1. Storage subsystem tracing tool

The Storage subsystem tracing tool (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.

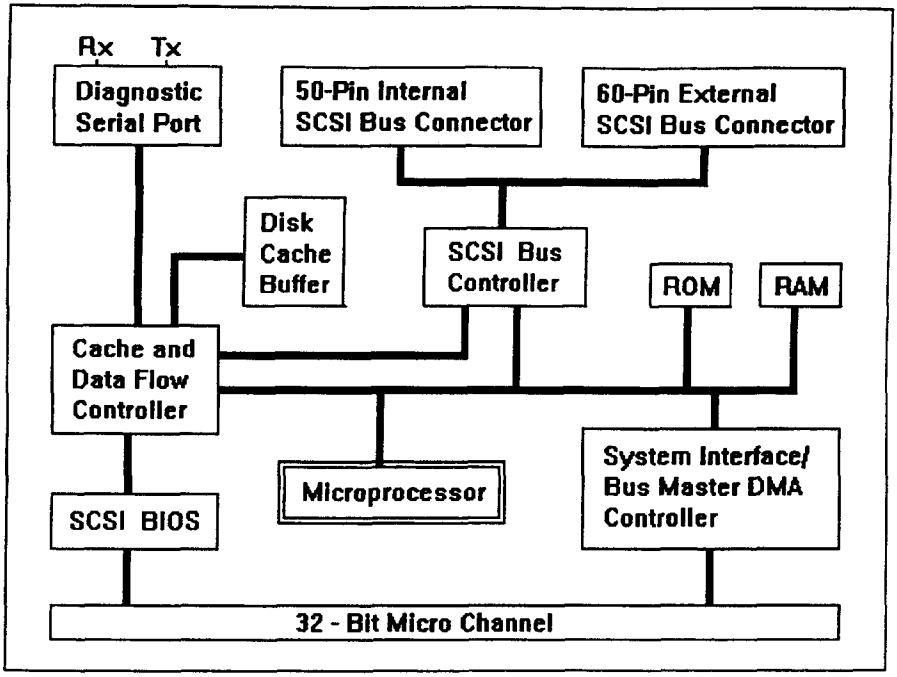


Fig. 2. Block diagram of the micro channel SCSI adapter with cache.

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]. Fig. 2 shows a simplified block diagram of such a card.

This card has 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.

Fig. 3 shows a Generalized Stochastic Petri Net (GSPN) Model [12,13] of the card's firmware, and Table 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 38 400 baud, 8 data bits, 1 start bit and no parity bit. Fig. 4 shows the structure of this information packet. The packet consists of the following fields.

- *Sync Mark* (1 byte): Declaring beginning of a new packet.

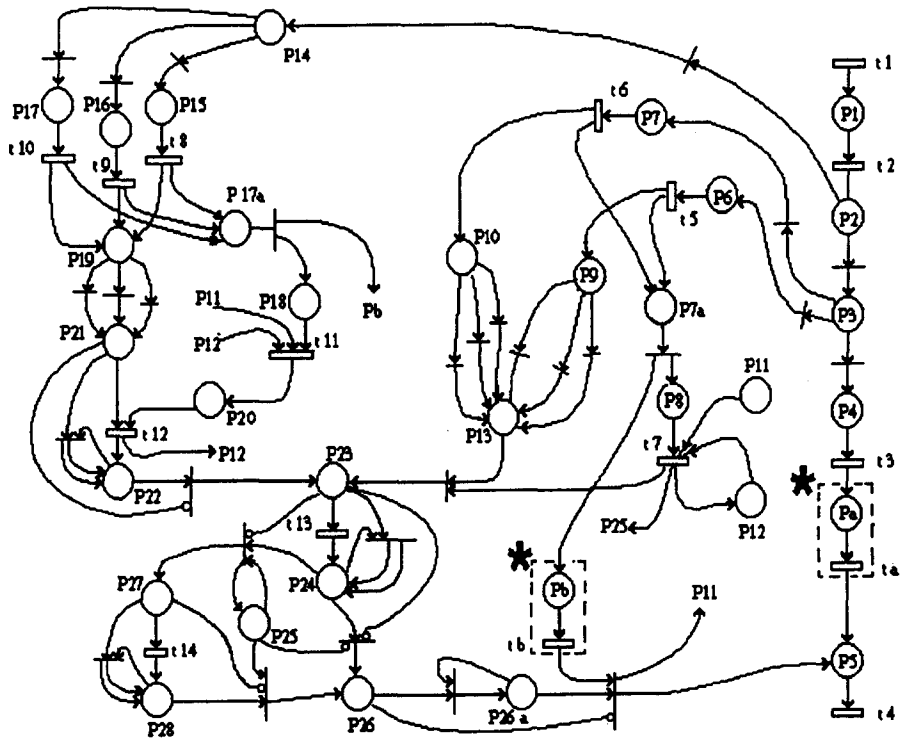


Fig. 3. GSPN model of IBM SCSI attachment controller.

Table 1
Description of the GSPN model in Fig. 3

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

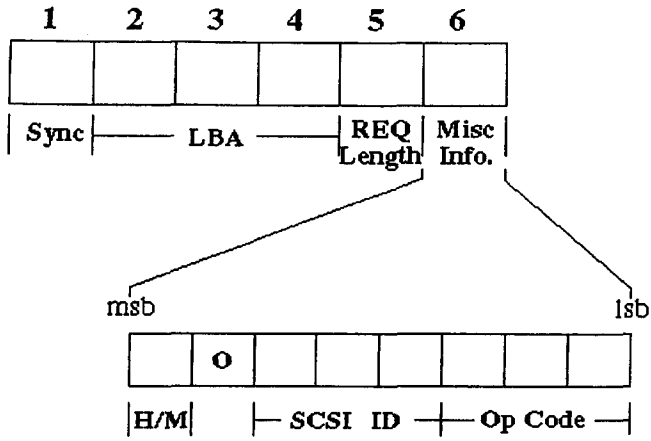


Fig. 4. Structure of a workload trace packet.

- *Logical Block Address* (3 bytes): Showing beginning LBA of the request. The 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 follows:

$$\text{Transmission time per packet} = 10N/R.$$

From above, the transmission time for a 6 byte packet at 38 000 baud is 1.56 ms. 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 Fig. 3). Therefore, in a worstcase situation (i.e., read hit in controller cache) the modified/added code introduces a maximum 1.56 ms overhead. Depending upon SCSI bus contention and DASD response, other scenarios introduce overhead values of much less than 1.56 ms.

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 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 five iterations).

Table 2 shows computed results for drive 'C' and Table 3 shows results for drive 'D' as reported by 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 (37.86–44.63%) 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 writer 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 of clock used by the benchmark. The benchmark reports results only up to 2 digits after the decimal and the difference in this case is 0.01.
- 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 up to 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 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%).

Table 2
Disk 'C' performance as reported by the benchmark

	Disk performance tests, for drive 'C'		
	With modified code (s)	Without modified code (s)	% of overhead
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 (16/16 384)	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 and large records)	49.83	47.11	5.77

- 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.

Table 3
Disk 'D' performance as reported by the benchmark

	Disk performance tests for drive 'D'		
	With modified code (s)	Without modified code (s)	% 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 (16/16 384)	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 and large records)	39.06	36.32	7.54

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

Table 4
Disk performance as reported by the SCSI Exerciser

	With modified code (ms)	Without modified code (ms)	% of overhead
No cache bypass (small requests)			
Read same location			
512 byte requests	51.04	36.23	40.88
4096 bytes 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 bytes 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 bytes 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 bytes requests	195.09	193.6	0.77
8192 byte requests	196.88	193.6	1.69
Cache bypass (16384 bytes 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

of the time the overhead due to the modified/added code would be less than 0.33 ms.

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 ms) is also required for time-stamping.

In our implementation of a DCS we found that a 386-20 MHz-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) than 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 ms. It has enough processing power to receive the traces at a high rate, time-stamp them and store them on the hard disk. Description of the data collection program (called COLLECT) running on the DCS follows.

The serial port (COM1) is programmed to give an interrupt ($0 \times 0C$) after receiving each byte of the information from the card. The COM1 is programmed at 38 400 baud, 8 data bit, 1 start bit and no parity bit. It is achieved by loading different registers of the serial port controller as listed below:

```
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.
```

The existing clock in a PC gives a resolution of about 54.9 ms ($=3\ 600\ 000 \div 65\ 536$). In an IBM PC (or compatible) a timer/counter chip issues an interrupt (timer tick) every 54.9 ms. The interrupt controller forwards it as interrupt number 8 to the system processor, and the BIOS handler [14] services it as follows.

- Keeps a 4-byte count of timer ticks at memory location 0004:006C.
- Wraps it to zero after 24 h (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 65 536 [15] to give an output frequency of about 18.2 Hz ($1/18.2 = 0.0549$ s). For a 1 ms (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 ms 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 $0 \times 0C$ 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 gives 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 Fig. 4).
- 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 ms 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 min, 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 the following.

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

(ii) serial communication line errors (voltages drives by the card are between 0 and 5 V), 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.

2.2.1. XTRACT

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

- time stamps that were placed after data bytes that are same as sync mark,
- erroneous bytes introduced due to serial communication line errors,
- 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.

(1) Find next sync looking byte. Assume that the byte found is byte1 (see Fig. 4) of the packet.
(2) Next four bytes constitute the present time stamp, skip them.
(3) 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.
(4) 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.
(5) 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.
(6) 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 TRACT 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.

2.2.2. BNJ

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

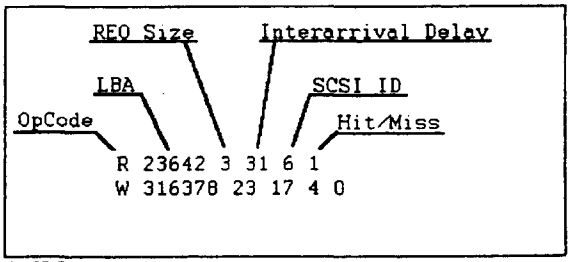
1. OpCode.
2. LBA accessed.
3. Request size in blocks.
4. Inter arrival time in milliseconds.
5. SCSI id of the target device.
6. Controller cache hit/miss.

All the fields are separated by single space. The program computers 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 ms 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 23 642, and there is a controller cache hit (1) and 31 ms 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 ms going to drive 4 at an LBA of 316 378 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
```



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.

3. Workload characterization

Workload characterization for any subsystem involves measuring the workload as seen by the subsystem and then searching for some quantifiable 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 Section 2, provides the information about all the requests coming down to the storage subsystem. We can characterize the storage subsystem workload by computing the following.

- 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 deman 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.

The storage subsystem workload in Netware operating system environment is traced using the tracking tool (SSTT) described in the Section 2. The traced workload is then characterized by obtaining the above mentioned distributions.

3.1. Characteristics of traced workload

In this section, we present the characteristics of the traced storage subsystem workload in a PS/2 file server running Novell Netware version 3.11. The site profile is given in Table 5. 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 Section 2. These traces were later processed using tools described in Section 2 to characterize the workload. The characteristics of the workload at the storage subsystem are summarized in Table 6. Traces were collected for 99.09 h.

3.1.1. I/O request size distribution

Fig. 5 shows read and write I/O request size frequency distribution. 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. I/O request size frequency distribution is presented in Table 7.

The read request to write request ratio is observed to be 43:57. This is almost the 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

Table 5
The site profile

Applications
Business: Database/transaction
Office support: E-mail, Wordprocessor, and Spreadsheet
Representative cycle time: Generally majority of activity is during 8 am to 5 pm business day
Server configuration
System: IBM PS/2 model 8595 (486/33 MHz) 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 6
Summary of the storage subsystem workload

Period	Day-three
Total transactions	180 321 (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 ms (St. Dev.: 10228.42 ms)
Mean I/O request size	4.39 Sectors (St. Dev.: 3.38 Sectors)

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 7) that more that 60% of write requests are small (1 block) in size while 44%–50% of read requests (8 blocks each)

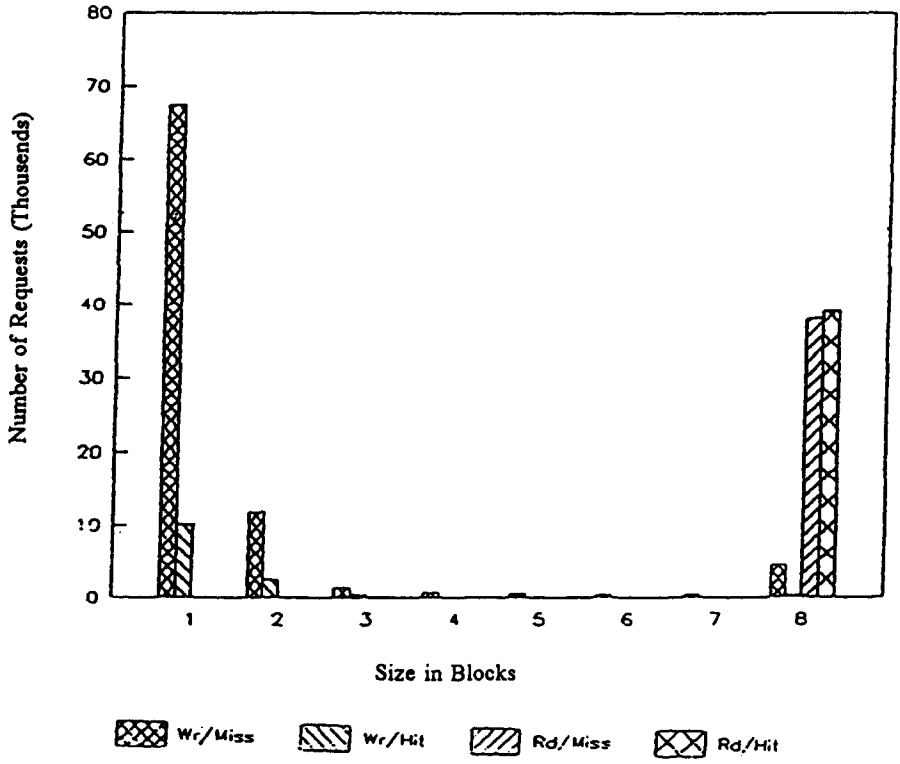


Fig. 5. I/O request size distribution on day-three.

Table 7
I/O request size frequency distribution

	Sector							
	1	2	3	4	5	6	7	8
Write	77 792	14 604	2105	1052	902	718	705	4847
Read	0	0	0	0	0	0	0	77 596
Total	77 792	14 604	2105	1052	902	718	705	82 443

resulted in adapter cache hit. From Table 7, 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 78.5:21.5.

3.1.2. Interarrival time distribution

Interarrival time between successive requests are shown in Fig. 6. The figure shows two peaks, one around 20 ms and another around 70 ms. It was also

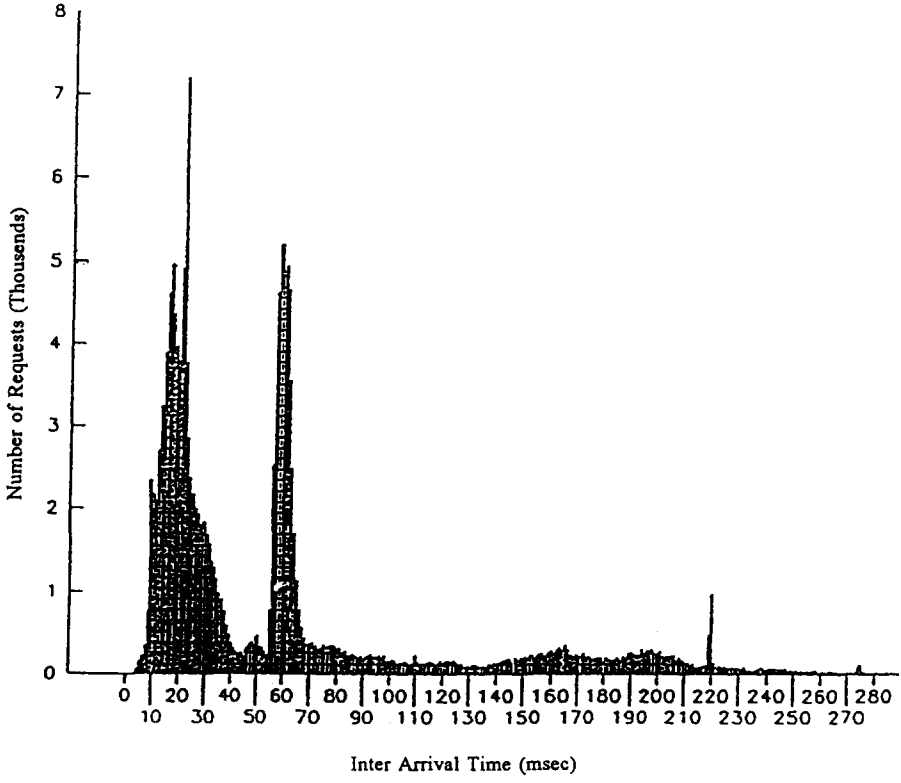


Fig. 6. I/O request interarrival time distribution on day-three.

observed that drive 1 was practically inactive and received less than 0.4% of the subsystem load (see Table 6). 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 that the mean and standard deviation of the interarrival time around peaks is as presented in Table 8. As explained in previous paragraphs, the first peak around 20 ms represents the subsystem response time which is summation of delays in the SCSI adapter and drive(s).

However, the second peak at the 60 ms 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 standard deviation of interarrival time around 60 ms ($\geqslant 55$ and $\leqslant 70$ ms) is also small due to sequential read requests which get a large number of hits in the SCSI adapter cache and almost deterministic interval between successive requests from the backup device.

3.1.3. Distribution of the logical block address of I/O requests

Figs. 7 and 8 show the distribution of the LBA on day-one for read and write I/O requests for drive 0 (there was almost no activity in drive 1 as shown in Table 6). In Fig. 7, read requests are distributed approximately uniformly over the first 70% of the storage address space. However, the write requests (Fig. 8) are mainly concentrated into several LBA regions.

3.1.4. Sequential I/O requests

The Novell Netware file system block size was set to eight sectors during volume definition. Therefore, the size of all read I/O requests to storage subsystems is eight sectors and write requests size vary from one to eight sectors. When an application program running in a client system requests more than

Table 8
Mean and standard deviation of interarrival time

Only first peak ($\leqslant 40$ ms) Mean/St. Dev.	Only second peak ($\geqslant 55$ and $\leqslant 70$ ms) Mean/St. Dev.	Both first and second peaks ($\leqslant 70$ ms) Mean/St. Dev.
20.37/7.35	60.20/3.09	33.74/19.51

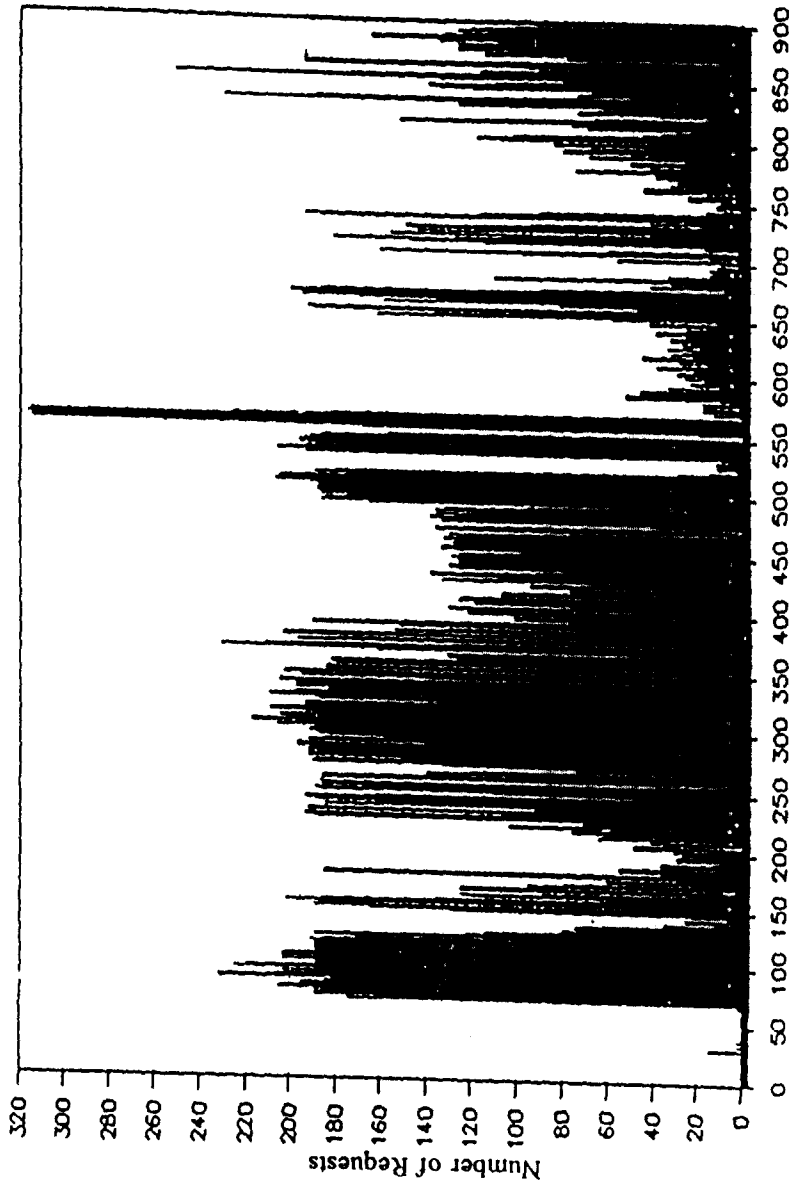


Fig. 7. Read logical block distribution on day-three.

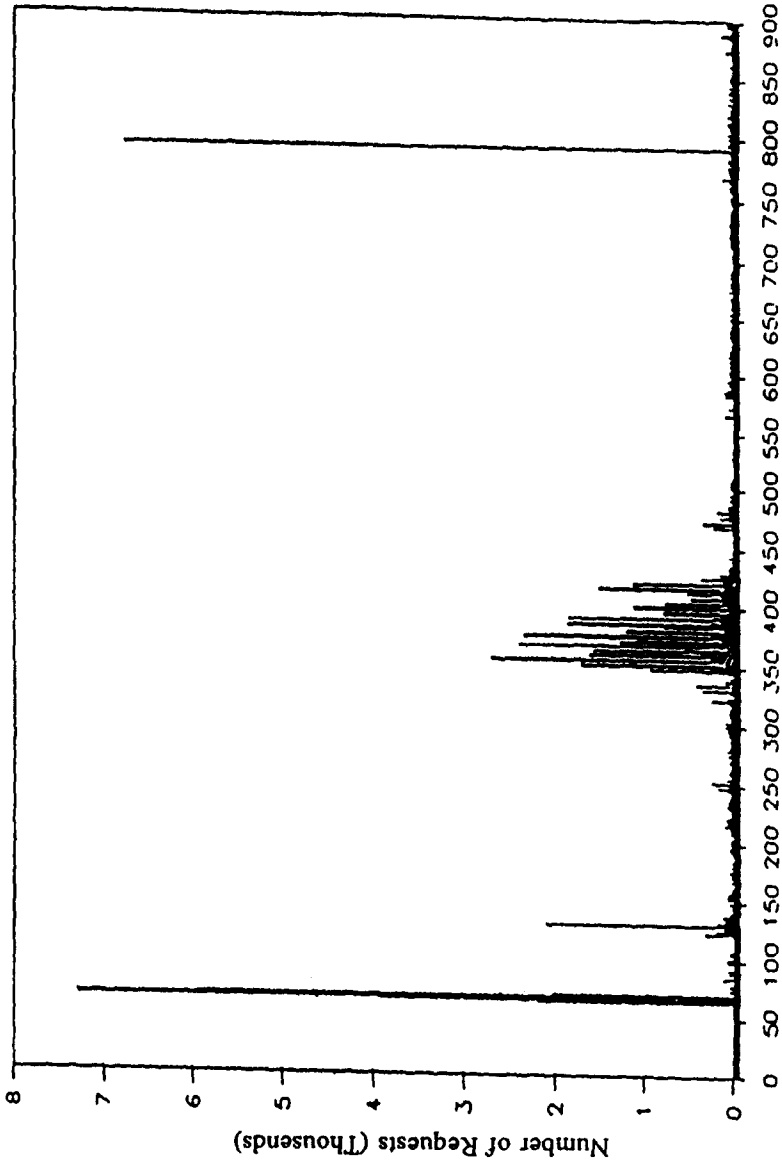


Fig. 8. Write logical block distribution on day-three.

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. 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 then 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. Figs. 9 and 10 show the probability mass function of number of 8-sector sequential read and write I/O requests. In these figures abscissa represents the numbers of sequential 8-sector requests. From Fig. 9 it is clear that most of the read requests (more than 85%) are one 8-sector read. From Fig. 10 we see that out of all 8-sector write requests less than 20% of these sequential requests are accessing contiguous 8-sector data blocks.

3.1.5. Subsystem load variation with respect to time

Fig. 11 shows instantaneous I/Os per hour with respect to hours of operation. Note that abscissa in this figure represent numbers of hours in operation and not the time of day. The storage subsystem workload data collection started around 4:30 pm on a Thursday and data collection was completed on following Monday evening around 8:30.

In Fig. 11 we see a sharp rise in mostly read load representing routine backup at early evening. Server was mostly inactive during the night. Activity during Friday 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 6:30 am).

We see some intense activity on Sunday evening around the 73rd hour of operation (around 5:30 pm). 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,

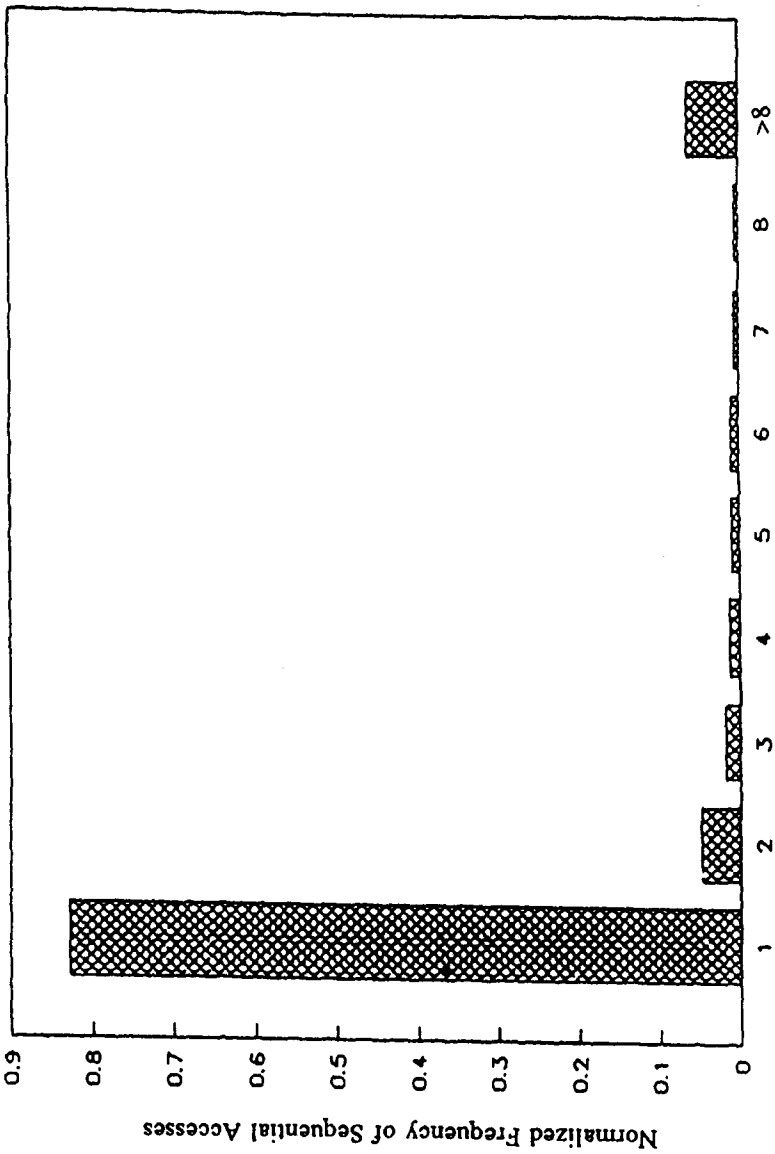


Fig. 9. Probability mass function of number of 8-sector sequential read requests on day-three.

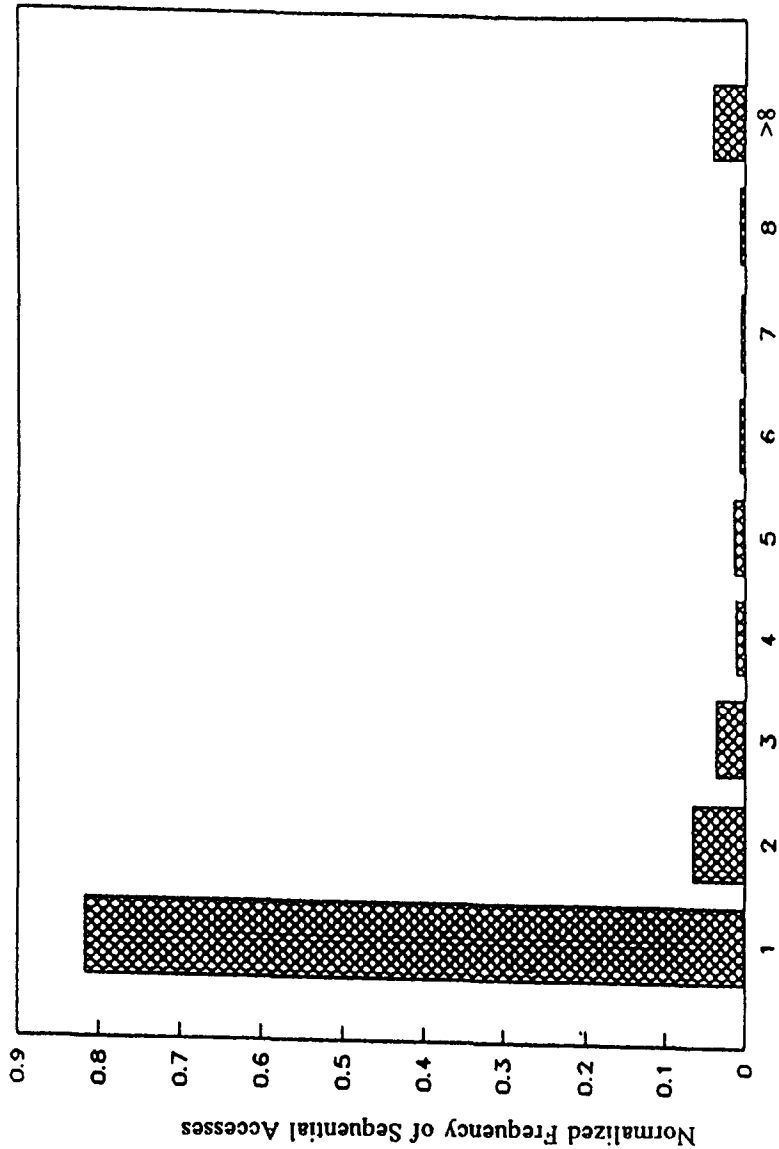


Fig. 10. Probability mass function of number of 8-sector sequential write requests on day-three.

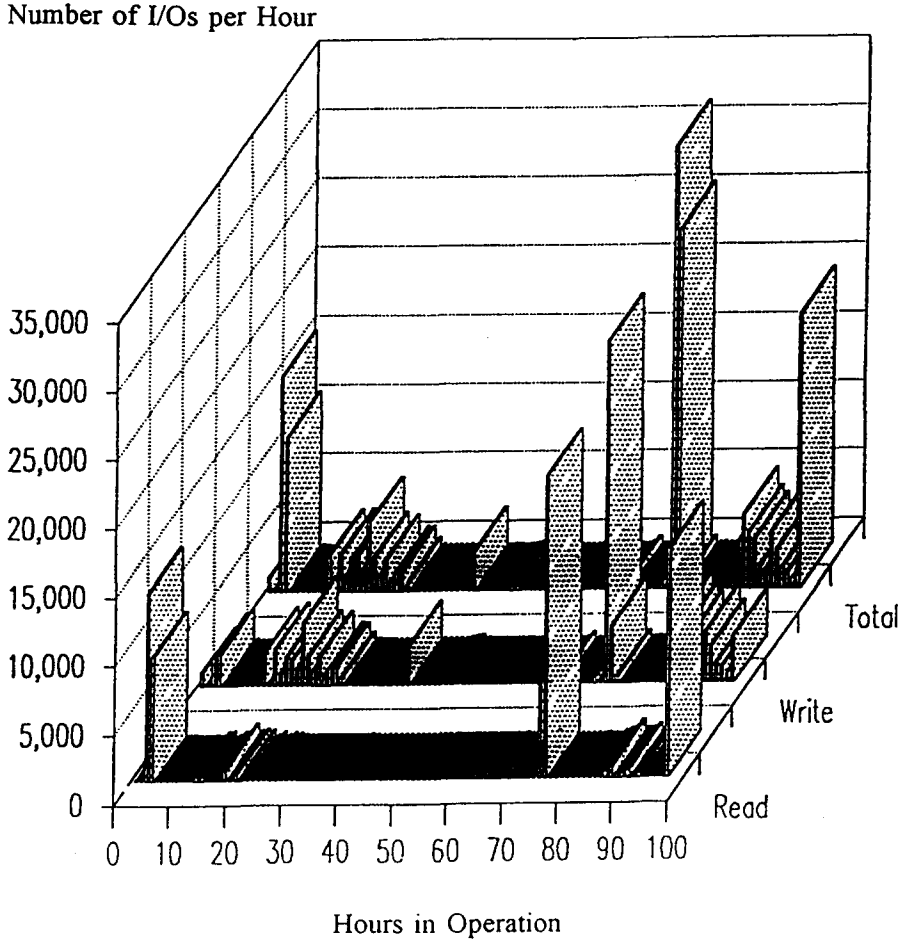


Fig. 11. Instantaneous subsystem load with respect to time on day-three.

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 7:30 pm).

3.1.6. Summary of workload in the traced Netware environment

Detailed analysis of the four day long trace data totaling 82 443 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) was 64 I/Os per s. However, when averaged over one hour period they drop to around 9 I/Os per s. 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 mostly only one of the two drives was active. Some response time reduction can be easily achieved by spreading data over both drives such that they are accessed uniformly.

All read requests' sizes were eight sectors and most of the write requests' sizes (63–75%) were one sector. Further analysis showed that more than 85% of the read requests were only one 8-sector read requests. One or more 8-sector write requests were 4.7%. 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.

4. Conclusions

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 real-time, 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.

Acknowledgements

This work was supported in part by Entry Systems Technology, IBM, under contract No. N-UN-270-00 task # 3.

References

[1] J. Moad, Relief for Slow Storage Systems, Datamation, 1 September 1990, pp. 22–28.
[2] D. Smith, The M212 – A zero-glue single-chip VLSI Wnnchester controller with on-board 10 MIPS processor, in: Wescon/86 – Conference Record, Anaheim, CA, November 1986.
[3] C.P. Grossman, Cache-DASD storage design for improving system performance, IBM System Journal 24 (3/4) (1985) 316–334.

[4] A. Goyal, A. Agerwala, Performance analysis of future shared storage systems, IBM Jornal of Research and Development 28 (1) (1984) 95–108.
[5] T. Beretvas, DASD performance analysis using modelling, in: CMG'85 Conference Proceedings, Dallas, TX, December 1985.
[6] P.K. Lim, J.M. Tien, A tool for direct access storage device (DASD) performance evaluation, in: Proceedings of the Urban Regional Information Systems Association Conference, Boston, MA, August 1989.
[7] B.J. Smith, A survey of the state of the art and practice in I/O subsystem modelling and analysis, in: Proceedings of the Fall Joint Computer Conference, Dallas, TX, 1986.
[8] G.E. Houtekamer, Measuring and modelling disk I/O subsystems, Ph.D. Dissertation, Technische Universiteit te Delft, The Netherlands, 1989.
[9] S. Zhou, H. DaCosta, A.J. Smith, A file system tracing package for Berkley UNIX, in: USENIX Association Summer Conference Proceedings, Portland, 1985.
[10] D.A. Bishop, Dekko and PCPERF/VMPERF Performance Trace User's Guide, IBM, 1991.
[11] PS/2 Micro Channel SCSI Adapter Technical Reference, IBM Corporation, Armonk, NY, 1990.
[12] J.L. Peterson, Petri Net Theory and the Modelling of Systems, Prentice-Hall, Englewood Cliffs, NJ, 1981.
[13] M. Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, Modelling with Generalized Stochastic Petri Nets, Wiley, New York, 1995.
[14] Personal System/2 and Personal Computer BIOS Interface Technical Reference, IBM Corporation, Armonk, NY, 1988.
[15] Personal System/2 Hardware Interface Technical Reference – Common Interfaces, IBM Corporation, Armonk, NY, 1990.