Disk Response Time Measurements

Thomas D. Johnson†
Jonathan M. Smith††
Eric S. Wilson†

Columbia University Computer Science Department
450 Computer Science, Columbia University, NY, NY 10027

1. Introduction

This report describes measurements made in the course of a disk performance study. These measurements deal with the response time for access requests made to the DEC RA81 disk, as measured on a system used for general-purpose time-shared computing. This differs from some previous work[1] [2], much of which is directed towards measuring and improving file system throughput. The paper assumes a knowledge of UNIX System V internals as discussed by Bach[3].

1.1 Equipment

The study is of a specific machine configuration, which consists of a DEC VAX-11/785 with 8 megabytes of main memory; mass storage devices are attached to a DEC UDA50 controller[4]. The associated disk hardware is the DEC RA81 disk drive; typical systems have four drives. The RA81 is a random-access, moving-head disk drive with non-removable media. The drive has a data storage capacity of 456 megabytes. The performance of the RA81 hardware is encapsulated in the following table, derived from the *RA81 Disk Drive User Guide*[5]:

DEC RA81 Disk Drive Characteristics	
Cylinders	2516
Transfer Rate	2.2 megabytes/second
Rotational Speed	3600 rpm
Average Rotational Latency	8.33 milliseconds
Head Switch Latency	6 milliseconds
Average Seek	28 milliseconds
One Cylinder Seek	7 milliseconds
Maximum Seek	50 milliseconds

Up to four RA81 drives can be connected to the UDA50 controller; the UDA50 is a UNIBUS device. The DEC UNIBUS subsystem interfaces to the Synchronous Bus Interconnect (SBI) through a UNIBUS Adapter (UBA). The UBA can support a maximum data transfer rate of only 1.35 megabytes/second[6]. Thus, the peak data transfer rate of the drives is not realizable with this configuration.

1.2 Software

This section describes system and performance measurement software available previous to our study.

1.2.1 Operating System

Our VAX Systems operate UNIX System V Release 2. Virtual memory is achieved via *swapping*; *paging* is not used in this Release. As supplied by AT&T Bell Laboratories, UNIX does not have a device driver for the UDA50/RA81. Our configuration was originally operational using a device driver adapted from a 4.2BSD driver for the UDA50; the driver's derivation from 4.2BSD led to the omission of features for reporting performance via the SAR[7] [8] system. SAR disk performance reporting was added by a few changes in the driver.
Each of our systems are configured with 1000 buffers.

---

† Bell Communications Research, Inc.

†† This work was performed at Bell Communications Research, Inc.

1.2.2 Measurement Tools

The measurement tools which existed previous to this study consisted mainly of the various pieces of the SAR package. The information provided by SAR was inadequate for the purposes of determing the causes of poor response times from our disks. Porting the disk activity reporting portion of SAR would require extensive driver updates, and we felt that some better measurements were necessary.

1.3 Tracing: Initial Configuration

Several goals were clear at the outset:

1. We needed a way to measure *actual* disk activity, rather than logical reads and writes.
2. The data gathering should not contaminate the measurements; that is, it should not cause extraneous data to be generated[1].
3. While it may add a few instructions to the service routines, the data gathering technique should not significantly alter the performance of the subsystem; otherwise timing data are meaningless.
4. It should give us enough auxiliary information for further detailed analysis with other tools.
5. The observations should be independent of the driver software.

Our approach was to implement a circular trace buffer of disk requests. It was used to track requests made to the driver through the *udstrategy()* routine[2]. The data gathered are a set from which we believe much can be derived; they are packed into a pair of 32 bit words; the inclusion of each datum is justified as follows:

1. R/W flag - High order bit of the first record. This was originally not included, but after preliminary data gathering was complete, it was clear that the direction of the I/O was important (e.g. for determining swapping characteristics)
2. Device number - 15 bits of a 16 bit quantity. In practice, all we need here is the 8 bit *minor* device number[3], plus a bit to distinguish between character special and block special accesses; but it is easier the way we do it.
3. Timestamp - implemented by using the last 16 bits of the **lbolt** software clock. **Lbolt** is a long integer which in incremented upon each clock interrupt ("tick"); it is incremented at 1/60 second intervals. This clock value provides the smallest granularity with which we can measure events in software[4]. Thus we can determine the rate at which requests are queued, and the intervals between events such as bursts of writing. 16 bits seem sufficient; we could give up a few in the device number field if more seem useful.
4. Size of the request - this is not meaningful for buffered I/O, since all requests will use the file system block size. It is useful in determing the amount of I/O performed through the character special interface. We used 8 bits because it is all we had to spare in the second 32-bit word; but since MAXBLK[5] is 125 on our systems, it is not a problem. Requests larger than MAXBLK are the result of abnormal conditions (such as a system administrator using a huge blocking factor with **dd(1)**) and are therefore not relevant.
5. The block number - we used 3 bytes (24 bits) because that's sufficient to store UNIX file system disk addresses. This block number is the block number relative to the beginning of the file system; the block number on the device can be determined off line by using information kept in the **ra_slices[]** structure defined in *sys/io.h*.

---

1. E.g., by using buffers which could otherwise be allocated.
2. This routine is called to place a buffer header onto the drive queue for the device. Once a buffer header is placed on the drive queue, the associated access will be performed. Thus, placing the tracing code in *udstrategy()* ensures that all disk accesses will be detected.
3. Indicating an instance of the device type determined by the first 8 bits, the *major* device number.
4. We could, of course, have modified the system's clock routines in order to achieve a finer granularity of measurement, but this would have required many modifications of the kernel.
5. The maximum size request made by the swapper in moving processes to and from the disk device.

1.3.1 Buffer Implementation

The information described above requires two 32-bit words; we allocated a 2048 entry integer array, giving us 1024 records. Given the observed transfer rates of the block I/O subsystem, this was sufficient to capture all of the data. In addition, an integer was used to track the current index of the array.

1.3.2 Gathering

We gathered the records using the interface to the kernel's address space provided by **/dev/kmem**. The addresses of the trace buffer and the current index were obtained using the *nlist()* library routine. Data gathering consisted of monitoring the index variable for any changes; any records between changed values were dumped to the standard output. This output was redirected to the character special file interface for a magnetic tape device in order to minimize interference with the file system.

1.3.3 Formatting

The records from the tape device were formatted in a manner similar to the entries in **/etc/passwd**; that is, as colon-delimited strings, one line per record. This facilitated the use of tools such as **grep** and **sed** in analyzing the data. For example, records destined for the first slice of the root device via the raw interface could be selected using **grep**.

1.4 Tracing: Final Configuration

The remainder of this paper discusses the results derived from adding further tracing code to the UDA50 driver; this code traces both the *requests* made to the drive and the *responses* received from it. The methodology is discussed further in the section entitled "Data Gathering". We can analyze the results of this tracing to determine the behavior of the drive under UNIX file system activity.

2. Data Gathering

This section takes a "bottom-up" approach in describing how the data was gathered. By "bottom-up", we mean that the data gathering is discussed beginning at the lowest level (device driver) and ending at a discussion of the statistical analysis performed on formatted and processed data.

2.1 Tracing

In addition to the tracing described above, a circular trace buffer was added to the *udrsp()* routine, which handles responses from the drive[6]. We did not attempt to correlate the responses with the requests at this point as we anticipated that this would be costly in terms of driver performance[7].

2.2 Gathering Trace Data

Gathering the trace data required a slight modification of the data gathering programs. The modification consisted of checking both request and response circular traces for new data and writing out any that were found. One problem with this unsophisticated approach is that the data generated does not retain its ordering with respect to time[8]; this has to be solved later when resolving the times to generate response statistics. A new flag was added to the output in order to indicate whether the record was a "Send" to the device, or a "Receive" from the device. The modified program determines this by the trace buffer it is examining, and outputs an integer flag which the formatting program uses to generate the appropriate record. The records are formatted:

```
send/receive flag:(major,minor):size:r/w flag:block number:timestamp
```

---

6. When the drive has completed an operation, it signals the CPU via the Unibus Adapter (UBA). The UBA is located at a known interrupt vector; when an interrupt occurs causing a processor trap, the _Xuba routine handles the trap by passing control to the *udintr()* interrupt handler for the UDA50. This routine examines the message header associated with the interrupt, and if it is a response, passes control to the *udrsp()* routine. Note that this processing is occurring at an extremely high priority level, and thus must be rapid.

7. On drives such as the RP06, correlation is trivial, as only one request can be outstanding, while we can have up to 14 requests outstanding on the RA81. The correlation would also be trivial if we used buffer headers to contain our trace data, but this approach would be undesirable for several reasons.

8. That is, a response might show up in the data stream ahead of the request which caused it.

The sizes are in units of 512 byte *blocks*; the time stamp is provided by the **lbolt** system clock, mod 65536. Some sample output (it is actual gathered data, but abridged for the sake of presentation) follows:

```
S:(32,0x10):2:R:1984:22127
S:(32,0x10):2:R:1992:22128
S:(32,0x6):2:R:486502:22129
S:(32,0x11):2:R:62782:22129
R:(32,0x10):2:R:1976:22123
R:(32,0x10):2:W:2152:22124
R:(32,0x10):2:W:2144:22124
R:(32,0x10):2:R:1984:22128
R:(32,0x10):2:R:1992:22130
R:(32,0x11):2:R:62782:22131
R:(32,0x6):2:R:486502:22132
R:(32,0x6):2:R:337822:22157
R:(32,0x6):2:R:339738:22158
S:(32,0x6):2:R:337822:22156
S:(32,0x6):2:R:339738:22157
```

This output can be used to illustrate several of the problems one encounters when trying to generate response times from the raw data, as discussed in the following section.

2.3 Generating Response Times

It turns out to be fairly easy to process such data visually: find a record preceded by an 'S', for a "Send", and then find the matching 'R', or "Receive", record. The difference between their timestamps gives the response time for the request. For example, the first line of the sample output is an 'S' record, for block number 1984. The corresponding 'R' is on the eighth line, and we compute the response time as (22128-22127) *ticks*, where a *tick* is 1/60 second.

There are, however, some potential problems in the data stream. For example, there are no 'S' records corresponding to the first two 'R' records, and the 'R' record for block 337822 on device (32,0x6) precedes the 'S' record. It turns out that some heuristics are necessary to get around the corrupted time-ordering of the data stream; these heuristics were incorporated into a program to analyze the data and produce response time figures on a per-request basis.

2.4 Response Time Statistics

There are several output formats available to the user of the response time analysis program. One is used to generate data for analysis with the **S** system[9]. Raw data were collected to obtain counts which could be used in later analysis. The first few lines of such an analysis are:

```
7480 responses took 0 ticks.
47922 responses took 1 ticks.
35755 responses took 2 ticks.
17120 responses took 3 ticks.
7821 responses took 4 ticks.
4734 responses took 5 ticks.
2803 responses took 6 ticks.
1667 responses took 7 ticks.
1203 responses took 8 ticks.
833 responses took 9 ticks.
```

The formatting of the data helped us in this task as it had in earlier analyses, because UNIX tools such as **grep** and **sed** could be used to aid the analysis. For example, inserting such a filter into the pipeline containing the response time analyzer allows us to analyze reads and writes separately.

3. Data Analysis

The amount of data gathered was tremendous; we monitored a system in our computer center for 3 hours on an afternoon in July, 1986. The gathered records took up about 6 megabytes of space; at 12 bytes per record, this was about 1/2 million records. It is clear that automated analysis of the data is not only preferable, it is necessary. Further, single statistics such as the mean and median[10] do not carry enough information to be useful in any non-trivial analysis of the system.

Graphical techniques, however, provide us with a methodology which is both compact and sufficiently informative. Once a subset of our data[9] was entered into **S**, we began the analysis.

3.1 Simple Counts

The first plotted data, presented in **Figure 1**, is the number of responses which took a specified number of ticks.

---

9. Unfortunately, due to memory limitations, we are forced to use only the first 2000 of our gathered records to perform analysis other than simple counting. This sample had no responses taking longer than 200 ticks, unlike the larger sample examined for counting statistics. Examination of more data leads us to the conclusion that our sample is representative, and thus our conclusions are statistically valid.

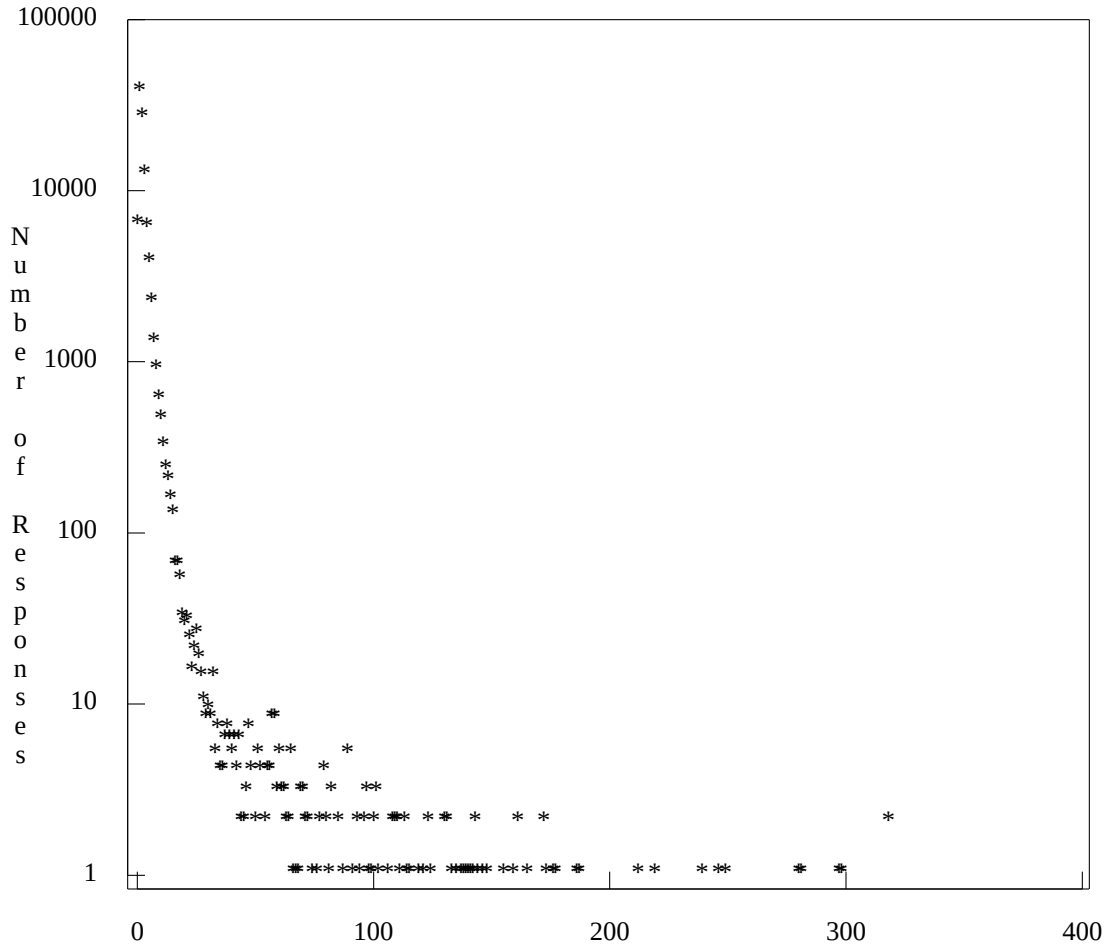

Figure 1. # Responses versus Response Time

We can see from this data that most of the responses were received in 5 or less ticks (note that the *y* axis is logarithmic). However, there are outliers in the greater than 300 tick range, indicating that *some* requests are seeing very bad service. While this won't affect the response time of a process performing writes[10], reads will be affected, as the process cannot continue until the data is delivered. For example, if one of the blocks which required 300 ticks was in the middle of an **a.out** file being executed by some user, that command, however trivial, would require at least five seconds to begin executing.

3.2 Response By Drive

When performing disk load balancing, it is useful to know which drives are more heavily or lightly used. Thus information about the response time as a function of the drive may tell you which drives are candidates for exchanging some file systems.

10. UNIX writes are asynchronous; the write call returns as soon as the data is transferred into a system buffer. The contents of this system buffer will be written at some later time, either when the system needs the buffer or when a request to **sync** the disks is issued.

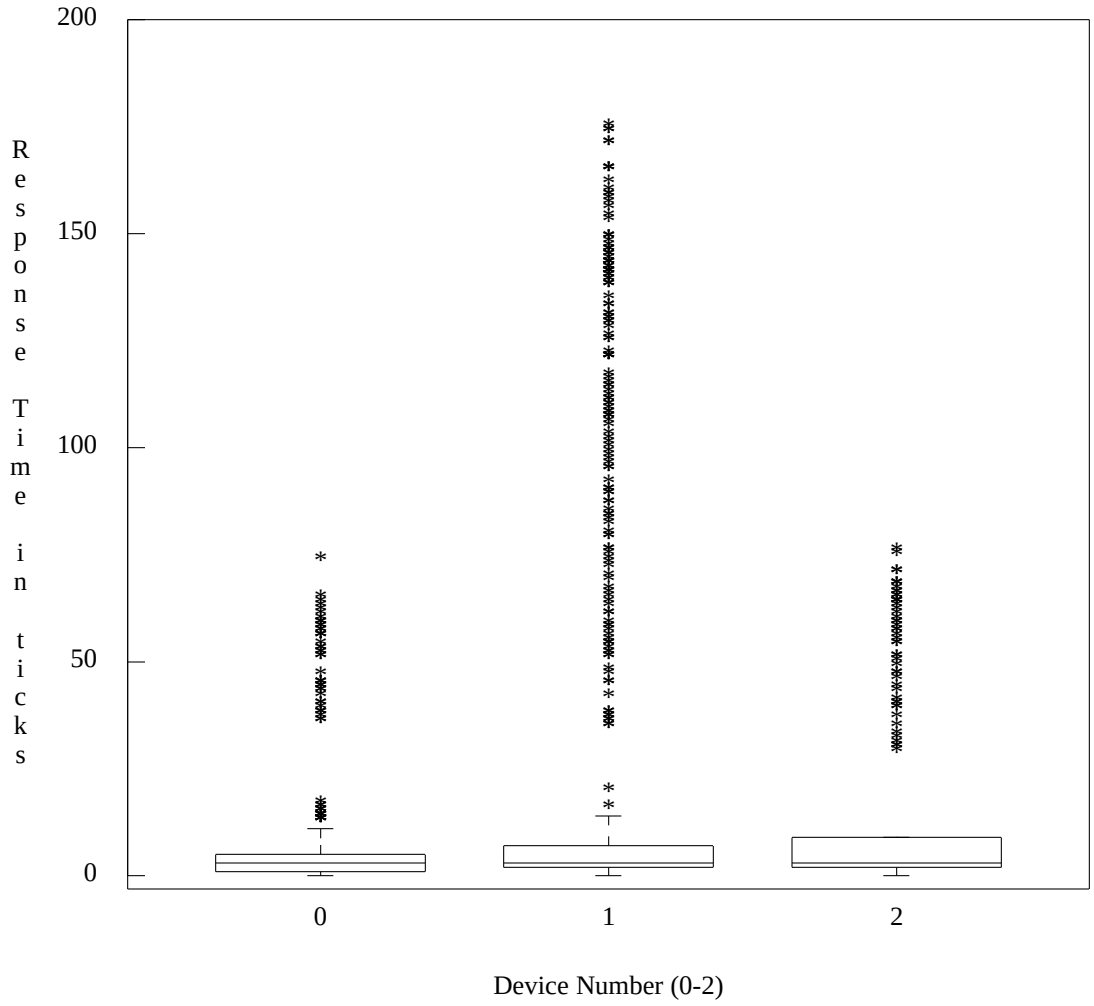

Figure 2. Response Time versus Device Number (0-2)

This graph[11] shows that although the worst response times are experienced by requests on drive #1, the medians of drive #0, drive #1, and drive #2 are fairly evenly matched. This evenness of the medians indicates that for this kind of file system activity, the loads seen by the drives is "balanced", indicating that the busy file systems are apportioned well between the three drives. Note, however, that the somewhat longer response times for the top 50 percent of requests on drive #2 may indicate some slowness; this information is lost if only the median is examined. If a particular drive or drives show unusual inequities in response times, it may prove useful to determine which slice(s) of the drive(s) are experiencing large numbers of accesses, and perhaps move some of that activity to another drive.

---

11. This particular presentation is called a *boxplot*; it provides much more information than an **X-Y** plot for this type of data. For a given **x** value, the box defines the middle 50 percent of the data, the horizontal line inside the box is the median, and the bar at the end of the dashed line marks the nearest value not beyond some standard range (in this case, 1.5*(inter-quartile range)) from the quartiles. Points outside these ranges are shown individually. Details of *boxplot* presentation can be found in Chambers, Cleveland, Kleiner, and Tukey[11].

3.3 Response by Block Number

Response by block number can tell us what areas of the disk are "hot spots" - places where there is a great deal of activity.

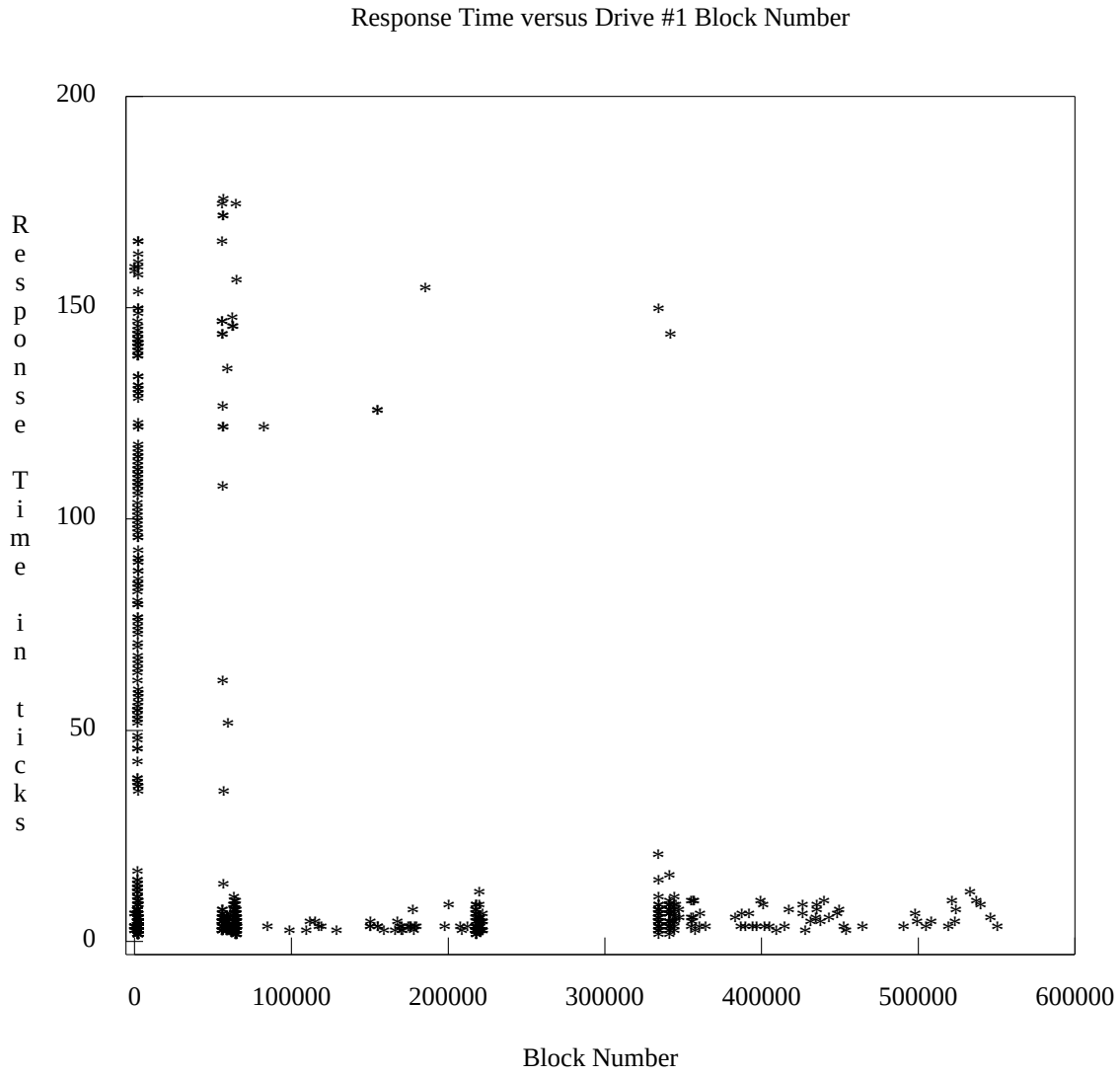

Figure 3. Response Time versus Block Number

For example, the above graph shows that certain areas of drive #1 experience very long response times, and that these areas are well-defined[12]. Some examination led to the deduction that these spikes are located at the *i-lists* of the file systems on drive #1. The spikes in response time result from a "sync" which causes many of these requests to occur in a large cluster.

3.4 Device Type

The character special or "raw" interface provides a means by which transfers of an arbitrary size can be accomplished; for example, swap I/O. The response times seen by raw I/O requests can tell you whether the disk

12. The size of a slice is 55692 blocks.

transfer rate[13] or the time spent in the drive queue is the major source of delay experienced by processes being swapped in.

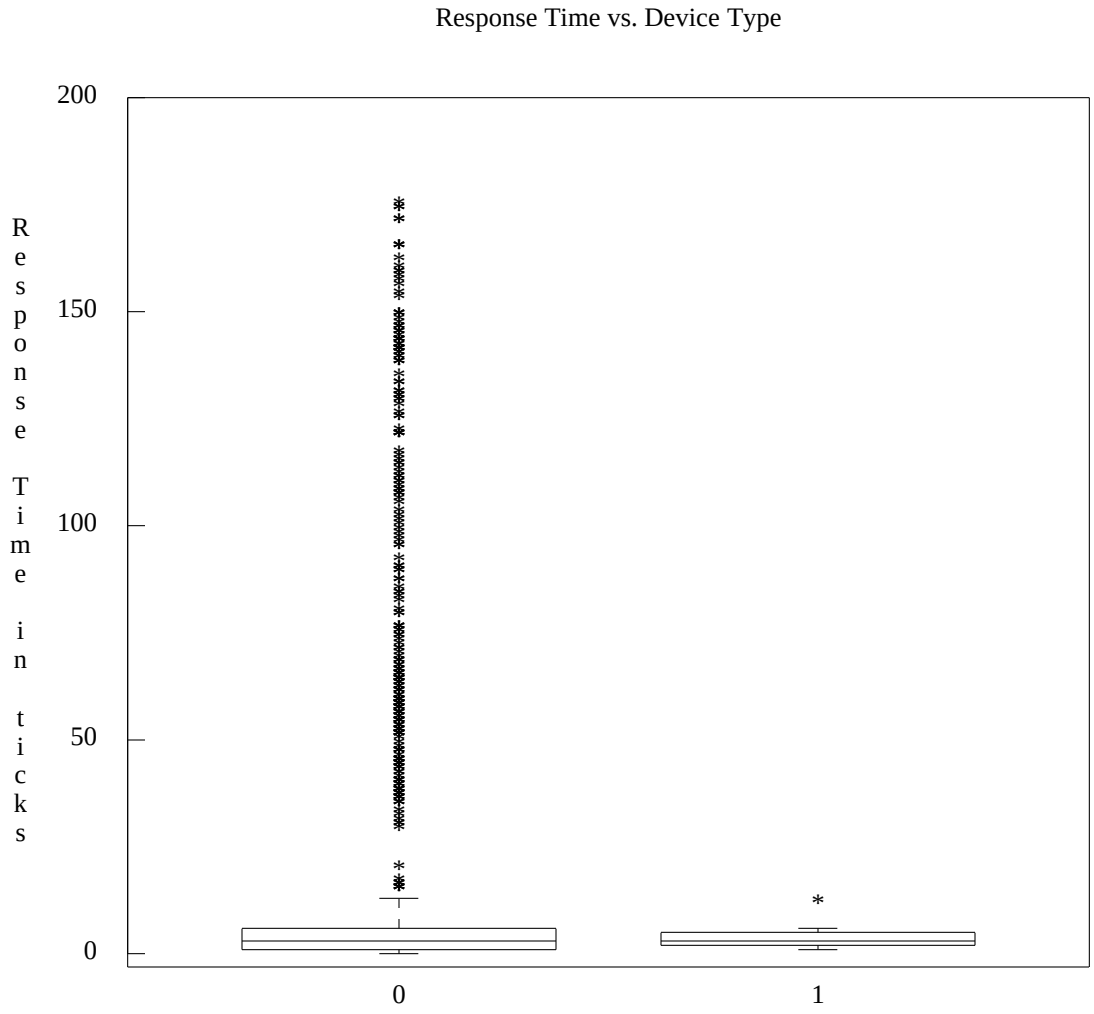

Figure 4. Response Time versus Device type (block=0,raw=1)

We see from the data that transfers to and from the raw device experience good response times, and thus swaps occur relatively rapidly. Due to the rather small size of this data sample, we did not experience a swap request intermingled with data transfer caused by a "sync", where the queued buffers which have been written into are actually transferred to the disk[14]. Such a request may have shown extremely poor response time.

3.5 Transfer Size

The transfer size (in blocks) plotted against the response time gives some measure of the contribution of the device bandwidth versus the queueing delay experienced by a request. If the device speed was responsible for much of the delay, large requests should take much longer than smaller requests.

---

13. Although more about the disk transfer rate can be deduced from the transfer size, as discussed in the next section.

14. **Update()** is the kernel routine called when the system call **sync()** is executed by some process. **/bin/sync** is typically this process.

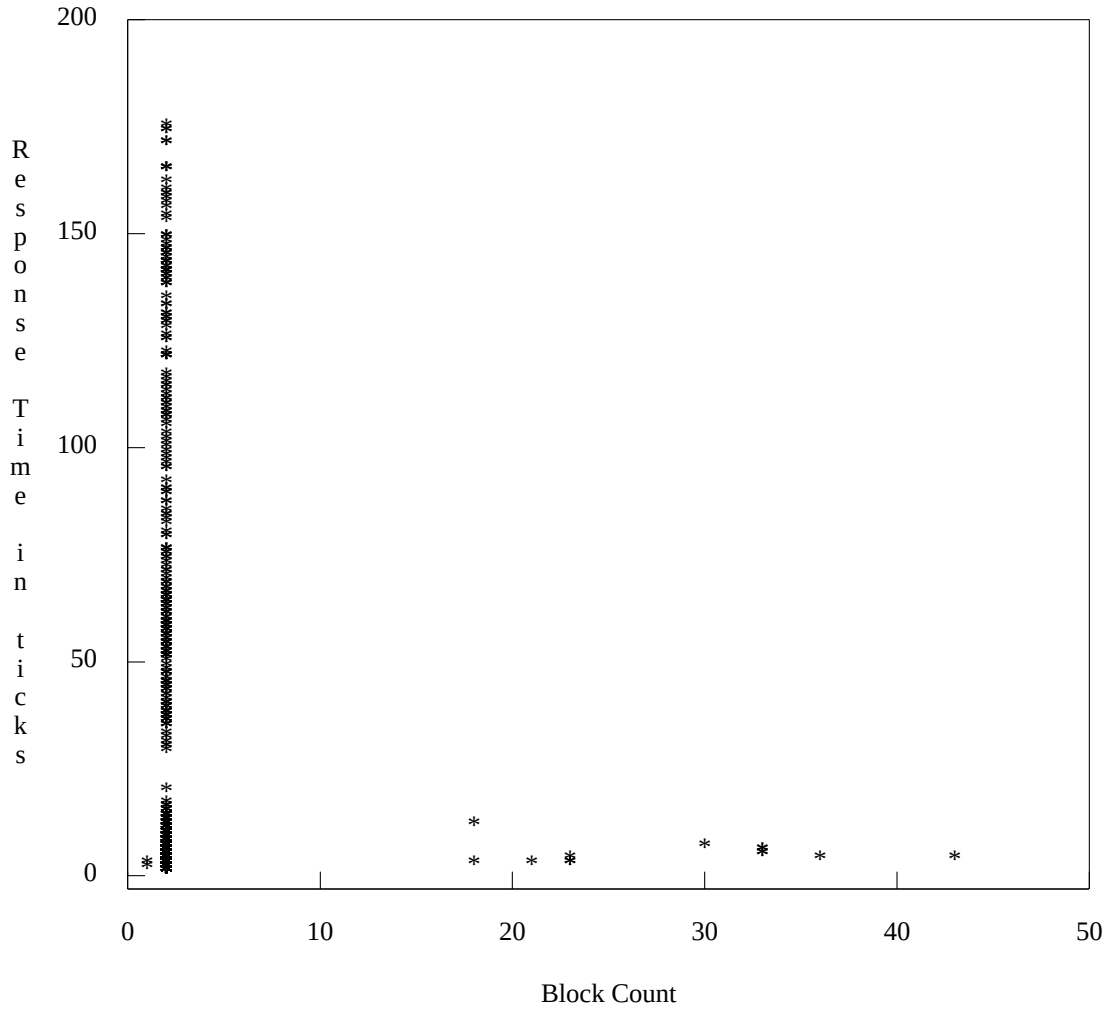

Figure 5. Response Time versus Transfer Size (blocks)

As can be seen from the graph, there seems to be little, if any, relation between transfer size and response time. This indicates that the time spent in the drive queue is more significant in its effect on response times than the size of the transfer (which is directly related to disk bandwidth).

3.6 Transfer Type

There are two transfer types, **read** and **write**. It is interesting to see what response times are experienced by these different types of requests.

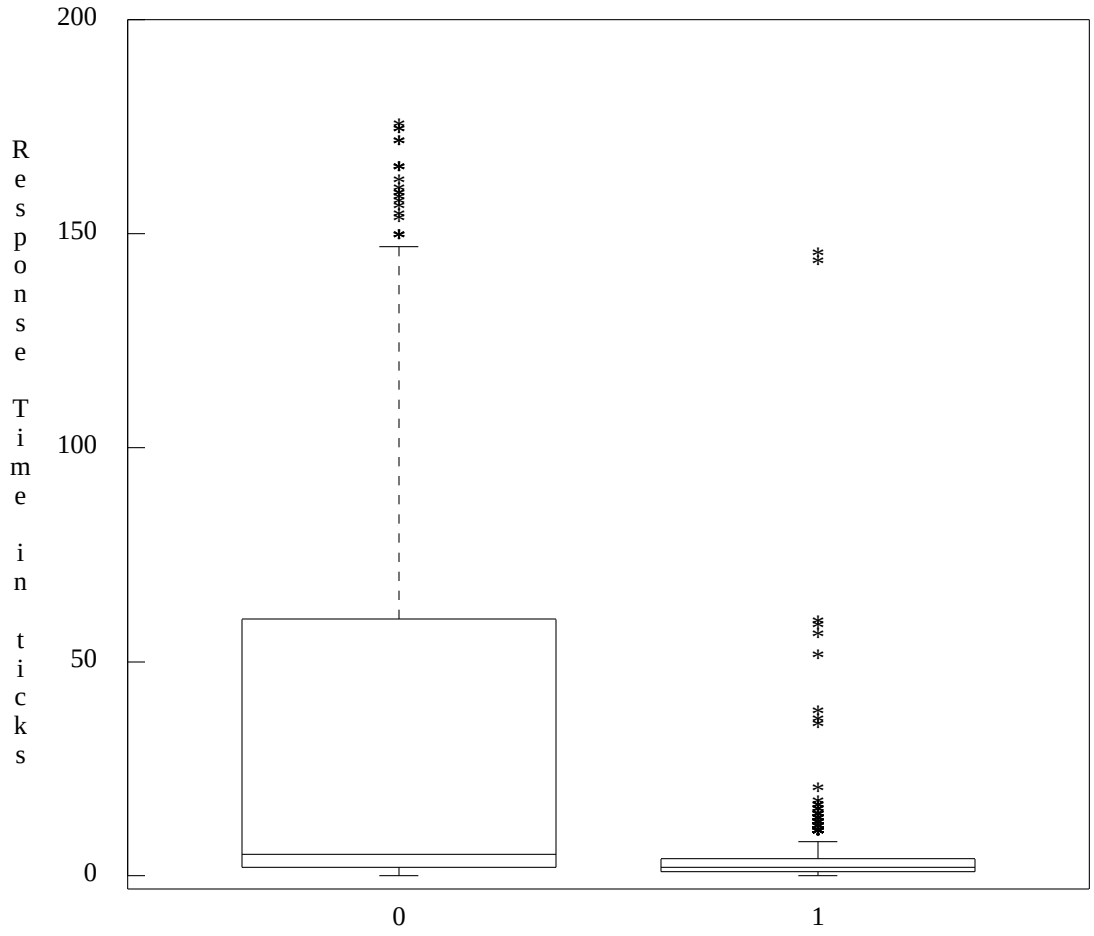

Figure 6. Response Time versus Transfer Type

It seems clear from the graph that writes in general seem to experience a longer response time than reads, although there are a few reads which take a long time to complete. The explanation for this is quite simple: writes to block devices only happen as the result of buffers being flushed due to a shortage or due to a "sync" being executed. These both result in a large number of write requests being seen by the disk driver in a short time period. The congestion caused by this behavior results in many write requests exhibiting an extremely large response time. Some further analysis has shown that the read requests which exhibit large response times are those which are queued for the drive in a particular interval. The interval occurs shortly after a "sync" has been initiated. Thus, reads queued in this interval must wait for the large preceding queue of writes to be flushed[15].

3.7 Start Time

The response time of a request plotted against the start time can tell you if there are time-related events occuring which can affect the response time seen by disk requests.

15. Requests are not reordered by the driver software, as the hardware provides seek optimization.

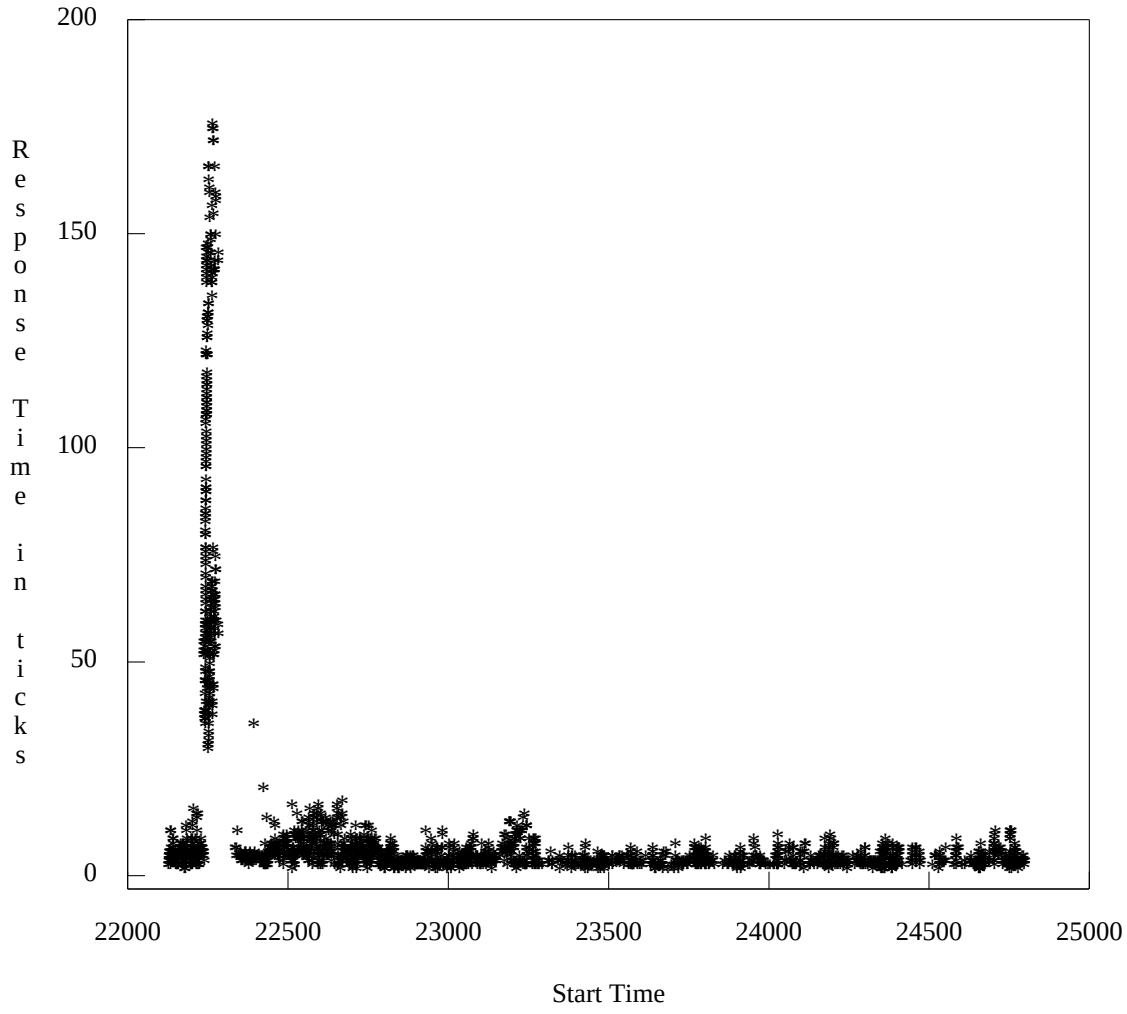

Figure 7. Response Time versus Start Time

This particular plot shows a window of about 3000 ticks of time; the time values were obtained from the least significant 16 bits of the system **lbolt** timer.

It seems obvious that there is some event which is causing the large "spike" in response time seen in the graph. This event is a "sync". Any requests which are queued in the same time frame as the "sync" are affected by it; in particular **read** requests are postponed until the preceding **write** requests are satisfied.

3.8 Conclusions

The data which we have analyzed have shown that the devices themselves are not particularly slow[16]. What seems to most affect the response time of **read** requests is their timing with relation to the large blocks of **write** requests generated by a "sync". Since writes are already asynchronous, and interactive response time is heavily dependent

16. The observation must be made, however, that a timer with a resolution on the order of 20 milliseconds is inadequate for logging transactions taking place at the speeds of these disk transfers. For example, a relatively large percentage of requests exhibited response times of 0 ticks; it is clear that we are losing information.

on **read** responses, it would seem that processes trying to **read** data when such a block of **write** requests appears will suffer with respect to response time. Perhaps some schema such as "dribbling" the writes at the disk[17], rather than bunching them up, could alleviate the problem to some degree. Other suggestions include prioritizing **read**s, but this may involve a great deal of processing in order to preserve data consistency[18]. However, any strategy must be examined by carefully testing response time characteristics of the *system* both before and after the strategy is implemented, for example in the manner of McKusick, et al[12].

4. Summary

We have demonstrated an effective methodology for tracing disk requests, and deriving response time data from this trace information. The response time data has been analyzed and correlated with other trace data to yield a further understanding of the behavior of the UNIX System V Rel. 2 file system with respect to the DEC RA81 disks.

In particular, we have seen that the mechanism used to flush the file system buffer cache ("sync") can have a great effect on the response times of **read** operations, which are synchronous. This can, in turn, increase the variance seen by a user in the system's response time.

We made several suggestions for further exploration.

5. Acknowledgements

L.E. Bonanni, A.M. Gross, J. Ashmead, and T. Liu made suggestions which greatly improved an earlier version of this paper. K.J. Busch's suggestions lead to several improvements in this paper. Remaining faults are ours alone.

---

17. For example, since the buffer cache is maintained as an LRU list, we can easily select some number of the "least recently written" blocks for queueing to the drive. This could be done with some mechanism such as the callout table which provides a way to schedule periodic activities inside the UNIX kernel.

18. Inconsistency can easily be generated by reordering disk requests, in particular, by putting a **read** in front of a **write** for the same block

REFERENCES

1. *A Fast File System for UNIX*
Marshall Kirk McKusick, William N. Joy, Samuel J. Leffler, Robert S. Fabry
Computer Systems Research Group Technical Report #7,
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
July 27, 1983

2. *Performance Effects of Disk Subsystem Choices for VAX Systems Running 4.2BSD UNIX*
Bob Kridle and Marshall Kirk McKusick
Computer Systems Research Group Technical Report #8,
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
July 27, 1983

3. *The Design of the UNIX Operating System*,
Maurice J. Bach
Prentice-Hall
1986

4. *UDA50 User Guide*
EK-UDA50-UG-003
Digital Equipment Corporation

5. *RA81 Disk Drive User Guide*
EK-0RA81-UG-001
Digital Equipment Corporation

6. *VAX Hardware Handbook* (1982-1983)
Digital Equipment Corporation,
Order Code EB-21710-20

7. *The UNIX System Activity Package*
Tsyh-Wen Pao
Bell Telephone Laboratories 3644-800313.01MF
March 13, 1980

8. *The UNIX System Activity Package - UNIX 4.0*
Tsyh-Wen Pao
Bell Telephone Laboratories 45173-810106.01MF
January 6, 1981

9. *S - An Interactive Environment for Data Analysis and Graphics*
Richard A. Becker
John M. Chambers
Wadsworth Statistics/Probability Series
1984

10. *Introduction to Statistics*,
Herbert Robbins and John Van Ryzin,
SRA, 1975

11. *Graphical Methods for Data Analysis*,
John M. Chambers, William S. Cleveland, Beat Kleiner, and Paul A. Tukey
Duxbury Press
1983

12. *Measuring and Improving the Performance of Berkeley UNIX*
Marshall Kirk McKusick, Samuel J. Leffler, Michael J. Karels, Luis Felipe Cabrera
Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
November 30, 1985