Input/Output Optimization and Disk Architectures: A Survey *

Alan Jay Smith **
University of California, Berkeley, CA 94720, U.S.A.

The file system, and the components of the computer system associated with it (disks, drums, channels, mass storage, tapes and tape drives, controllers, I/O drivers, etc.) comprise a very substantial fraction of most computer systems; substantial in several aspects including amount of operating system code, expense for components, physical size and effect on performance. In this paper we survey the state of the art in file and I/O system design and optimization as it applies to large data processing installations. In a companion paper, some research results applicable to both current and future system designs are summarized.

Among the topics we discuss is the optimization of current file systems, where some material is provided regarding block size choice, data set placement, disk arm scheduling, rotational scheduling, compaction, fragmentation, I/O multipathing and file data structures. A set of references to the literature, especially to analytic I/O system models, is presented. The general tuning of file and I/O systems is also considered. Current and forthcoming disk architectures are the second topic. The count key data architecture of current disks (e.g. IBM 3350, 3380) and the fixed block architecture of new products (IBM 3310, 3370) are compared. The use of semiconductor drum replacements is considered and some commercially available systems are briefly described.

Keywords: Disk, Drum, Rotational Scheduling, Arm Scheduling, Input/Output, I/O, Prefetching, Load Balancing, Compaction, Fragmentation, Data Structures.

* Partial support for this research has been provided by the National Science Foundation under grant MCS77-28429, and by the Department of Energy under Contract DE-AC03-76SF00515 (to the Stanford Linear Accelerator Center).

** Computer Science Division, EECS Department, University of California, Berkeley, California, 94720, USA. The author is also on the staff of the Lawrence Berkeley Laboratory and is a visitor at the Stanford Linear Accelerator Center.

1. Introduction

We shall refer in this paper to the part of the memory hierarchy beyond the main memory interface as the 'file system'. It consist of disks, drums, tapes, mass storage, I/O controllers, channels, and large parts of the operating system. These components comprise a large fraction of the expense and physical size of a computer system, and account for a large fraction of the operating system code and execution time. For these reasons, the efficient and effective use of the file system is important to the overall efficient use of the computer system. We shall survey the state of the art in file system design and use it as it applies to large data processing installations; some relevant research results are summarized in [1].

Our concern here will not be with a general software overview of the file system structure. There are several papers in the literature on this subject; operating systems discussed include Multics [2,3], MTS [4], OS [5], and others [6]. The reader is assumed to have a general background in this area.

Section 2 of this paper will survey the standard optimization and tuning problems in current computer systems. Throughout, our target is the large data processing installation, although some of the

Alan Jay Smith was born in New Rochelle, New York, USA. He received the B.S. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, and the M.S. and PhD. degrees in computer science from Stanford University, Stanford, California, USA, the latter in 1974.

He is currently an Assistant Professor in the Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California, USA, a position he has held since 1974. His research interests include the analysis and modeling of computer systems and devices, operating systems, computer architecture and data compression. He has published in these areas, and received an award for the best paper in the IEEE Transactions on Computer Systems in 1979.

Dr. Smith is a member of the Association for Computing Machinery, the Institute of Electrical and Electronic Engineers, the Society for Industrial and Applied Mathematics, Eta Kappa Nu, Tau Beta Pi and Sigma Xi.

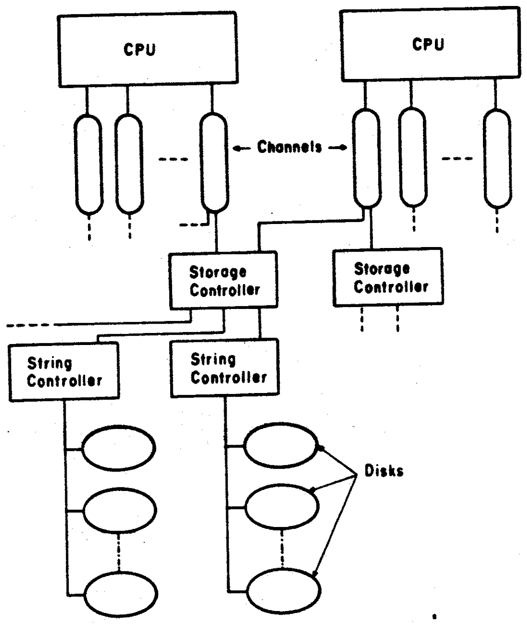

Fig. 1. Portion of large data processing system configuration.

material presented applies to smaller sites. Further, we shall stress IBM operating systems and devices because of their prevalence, due to the large amount of information available, and because many IBM devices have become industrial standards for which there are several (PCM) vendors. Specific concerns will be: block size choice, data set placement, arm scheduling, rotational scheduling, compaction, fragmentation and file structure. A set of references to the file system and the analytic model literature is presented and referenced where appropriate. Further references may be found in [7]. Changes in device architecture that are occurring are discussed in Section 3, where some of the differences between the traditional count key data disk and the new fixed block disk are presented. The new electronic drums are also discussed. A companion paper [1] summarizes some research results of the author on the topics of file migration and disk caches.

For purposes of illustration and reference, we have provided Fig. 1. Fig. 1 is intended to diagram part of a 'typical' large system IBM configuration, and will serve as a concrete example for later discussions.

2. Standard optimization techniques and topics

2.1. Reasons for optimization

The need for the optimization of the file system comes from two simple observations. First, the time to do a reference to the main or cache memory in most computer systems ranges from 50 nanoseconds to 1.0 microsecond, whereas the time to read or write from any sort of secondary storage (disk, drum, tape) is at least 10 milliseconds; thus the ratio of access times is from 10000 to 1000000. This very large ratio in access times is known as the access gap and implies that frequent I/O operations could result in the CPU (and most of the rest of the system) remaining idle while I/O operations complete. Second, I/O operations are in many systems quite costly as well in terms of CPU time. The portions of the operating system related to input/output are large and complex, and even in a fast machine (e.g. IBM 3033 or Amadahl 470) as much as 1–5 ms of CPU time may be required for an I/O. This includes all of the related operations such as: building a channel program, page fixing, setting up any necessary control blocks, starting the I/O, executing the dispatcher, processing the I/O interrupt and executing the dispatcher again. This second reason therefore also suggests that the frequency and cost of I/O operations must be minimized.

2.2. Characteristics of current systems

In order to have an understanding of the optimization issues involved, in this subsection we summarize some previously published information [8] describing the installation characteristics for a number of large IBM MVS user sites. This survey is only a sample of IBM installations, and the sites were chosen primarily on the basis of willingness to participate. Nevertheless, it does suggest the characteristics of large data processing centers (but not necessarily minicomputer systems).

Forty-seven IBM customers were queried. Programs were run to read the on-line VTOCs (volume tables of contents) for the 2314, 2305, 3330 and 3350 disks and the on-line VSAM catalogs. Statistics were gathered on DASD volumes, data sets and block sizes. The customers had business in the following areas: finance (3), government (2), insurance (16), manufacturing (7), IBM internal (2), service bureau (3), transportation (1), utility (3), distribution (3) and process (7).

There were 2154 DASD volumes, 250 VSAM catalogs and 265 613 data sets. 80% of the customers had 50 or fewer volumes (i.e. disk spindles). 6% of the volumes were 2305, 2314 or 3340, 12% were 3330, 38% were 3330-II and the 3350's accounted for 44%. The 2305's (drums) tended to be almost fully allocated (99%); the disks generally had about 75% of their space in use.

Data sets were most frequently sequential (67%) although sequential data sets accounted for only 39% of the disk space. Direct data sets (2.1% of data sets, 8% of space), partitioned data sets (18%, 23%), ISAM data sets (2%, 7%) and VSAM (2.5%, 14%) were also common. A significant fraction of the sequential data sets were small, whereas the other files were usually larger.

Record formats were distributed as follows: 71% of the files had fixed length records, 16% had variable length records and 10% had undefined block formats. Block sizes tended to be small; the average block sizes, by file organization were: sequential (1861 bytes), direct (1682 bytes), partitioned (961 bytes) and ISAM (1925 bytes). Those files with fixed block sizes and larger average block size per file (1745 bytes) than the variable block size files (1226 bytes). The overall average was 1530 bytes. The number of blocks above 8 K was very small, which suggests that full track blocking is rare. 24% of the block sizes were under 100 bytes (usually 80 bytes).

The data for this study was collected in two stages: the first was early 1977, and the second, late 1978. Six of the customers collected data both times. For these six customers, the following was observed: the number of DASD volumes increased 33% but there were about 30% fewer data sets per volume; thus the number of data sets in total remained about the same. There was little change in data set access methods, other than a 26% increase in VSAM data sets and an 18% decrease in ISAM data sets.

2.3. Block size optimization

There are some tradeoffs between large and small block sizes. Small block sizes have the following two advantages: they require small buffers and are quickly transferred, once located on disk. Conversely, to process a large amount of data, a large number of small blocks will have to be fetched, and with each fetch is associated operating system overhead and extra disk latency. Further, small blocks make inefficient use of physical space on the disk, due to interrecord gaps. These disadvantages for small block sizes outweigh the advantages in most circumstances, most especially for sequential data processing, and therefore large block sizes are usually preferred. Using some realistic assumptions, in [9] it is shown that block sizes should be at least 2–4 K. The use of block sizes larger than the optimum is usually almost equally good. Despite this, in [8] it was noted that very small block sizes are common. For this reason, many systems could significantly improve their operation by reblocking their files. Such reblocking might be best managed by moving block size choice from the user to the system as is done by many non-IBM operating systems.

Among the papers that consider block size choice are [10–12].

2.4. Data set placement and I/O balance

Data set placement can affect system efficiency in two ways: the placement of data sets within a single volume and the location of data sets on different volumes. If different data sets are in use on the same volume, efficiency can be improved by locating them near each other; thus seek distances are minimized. This issue is discussed briefly in [13] where the so-called 'organ pipe' arrangement is suggested. In that case, the most frequently used files are placed near the center of the disk and the least used near the inner and outer edges.

It is much better to locate data sets that are used concurrently on different volumes; that tends to reduce congestion and improve access time. The access time is improved particularly by the following phenomenon: if a sequentially allocated data set is the only one in active use on a volume, most I/O's will find the arm already positioned at the correct cylinder [14,15], which eliminates most seeks. In any given system, measurements can be made to determine which files are likely to be open and used concurrently. A cost function then can be set up which assigns a cost every time two files are assigned to the same spindle. Math programming methods can be used to select a desirable or optimal file assignment.

In [16] some figures are given which may be used to detect when I/O congestion may require moving data sets. In particular,

(a) more than 30% channel busy,

(b) DASD utilization above 40%.

(c) a device activity count above 15 I/O's per second, or

(d) a mean DASD queue length of 0.05 or more.

Some systems have disks of varying performance characteristics. In that case, it is not correct to use each DASD equally frequently; in [17] it is shown that the faster units should have a load that is higher by an amount that exceeds the speed ratio. I.e. the utilization of the higher speed units should be higher.

A difficulty with attempting to dynamically track the use of various data sets and disk spindles is that in a multiple CPU system, one CPU may not have an accurate idea of the aggregate rates of reference. There are several possible solutions to this problem: among them are to maintain a common data base or to have each CPU periodically (randomly) sample for device or control unit busy. This issue should always be considered.

2.5. *Arm scheduling*

A large component of disk access time is the arm seek time (see Fig. 2). For example, the average seek time on a 3350 disk is 25 ms [18] whereas the latency averages 8.4 ms. Therefore, to the extent that seek time can be minimized, performance improves. One method to reduce seek time is to consider all of the I/O requests outstanding in the queue to a given device and select an order to process them that minimizes the total seek time. Typical algorithms are SSTF (shortest seek time first), SCAN (an 'elevator like' algorithm that scans back and forth across the disk surface servicing requests as it passes the target cylinder), and CSCAN (which scans the disk, much like SCAN, but only in one direction. When a scan is finished, the arm returns to its starting position). The objectives of these algorithms are several:

(a) minimize mean access time,

(b) minimize the variance of the I/O access time and

(c) avoid discriminating against certain data sets in unfavorable locations (e.g. at edge of disk). The classic study of this problem is [19] (see also [20]); more recent results (one of which recommends SSTF) are in [21,22]. These later papers observe that SSTF reduces the mean access time enough that even though it has a large coefficient of variation, the magnitude of the variance is still small. Other papers which consider aspects of this problem are: [23–31].

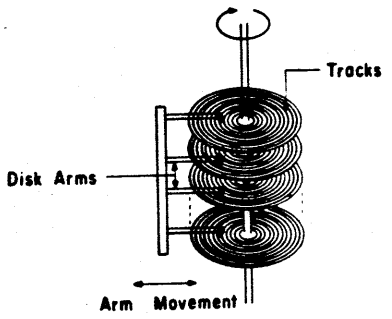

Fig. 2. Disk module.

Despite the large amount of research on arm scheduling, it has been found that it is very seldom necessary to schedule the arm at all [32], since

(a) queue lengths at the disk are usually short (one or less) and

(b) the disk arm seldom has to move.

(See also [14,15,33].) The latter is because most disks have only one open file, and that file is usually allocated and accessed sequentially. In fact, only about 30% or 40% of the time is a seek required. Thus arm scheduling is rarely an interesting problem, and any reasonable arm scheduling algorithm (e.g. FCFS) is likely to be satisfactory. The increasing density of disks (see Section 3 and [1]) may indicate, however, that the probability of multiple open files on one spindle will increase. (The author has no appropriate data, but believes that there has been only a small increase thus far. In most large installations, the number of disk spindles has grown almost as quickly as the CPU power over the last few years.)

In addition to the papers noted which deal with arm scheduling, there are additional articles which model disk systems. The following may be of some interest: [34–37].

2.6. *Rotational scheduling*

Rotational scheduling has to do with scheduling the processing of I/O requests at the same seek address on rotational storage media, i.e. disks and drums. For disks, rotational scheduling is usually unnecessary, since it is rare to have more than one I/O request outstanding for the same cylinder. The issue is more important for drums and fixed

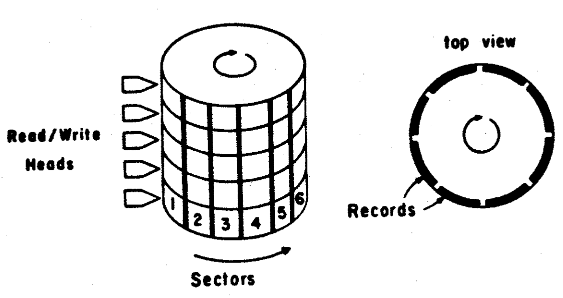

Fig. 3. Paging drum.

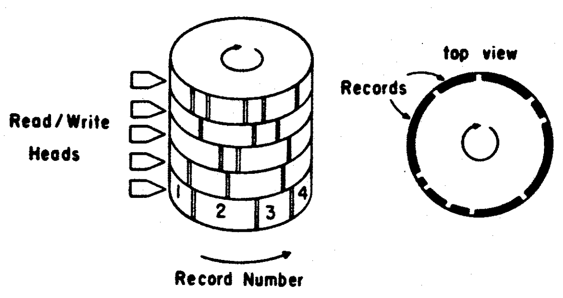

Fig. 4. File drum.

head disks (see Figs. 3 and 4). We will refer to fixed head disks henceforth as drums.

2.6.1. *File drums*

Drums can be either 'sector drums' or 'file drums'. File drums have arbitrary length records (blocks) placed at arbitrary locations (angular positions). (Throughout most of this paper, the terms record and block are used interchangeably, as is done frequently in the literature. When necessary, we will distinguish between logical records and physical blocks.) When a queue of requests exists to such a drum, the requests in the queue can be scanned and an optimal algorithm for processing them can be constructed. It has been shown [38–40] that simply using SSTF (Shortest Seek Time First; i.e. select the first record to pass under the read head position) always performs within one revolution of the optimum. It seems clear that in practice, SSTF should be satisfactory.

2.6.2. *Sector or paging drums*

Sector drums, also called paging drums, are simpler. These drums are organized so that records are fixed length and begin at only certain prespecified angular positions. In the simplest case, sectors do not overlap, and all records starting at the same angular position constitute a sector. In this case, a separate queue is set up for each sector, and is serviced first come, first serve. By inspection, this is optimal. When sectors overlap (as they do on the IBM 2305 fixed head disk [41]), the solution is not quite as simple, but sector queueing and FCFS service are still efficient.

2.6.3. *Hardware implementation*

There are two types of hardware implementation for rotational scheduling that the author is aware of. IBM has a primitive method known as 'RPS' or rotational position sensing. For RPS, the string and storage control units and channel become free while the disk is rotating; when the correct angular position is reached, the disk, storage controller and string controller attempt to reconnect. (See [42] for some analysis.) If the reconnect fails because one of those units is busy, another full rotation must occur. Depending on block size and specific record location, this scheme can lead to poor performance for files with small block sizes.

The other implementation is one by Burroughs, which uses a 'disk file optimizer'. The optimizer is a single hardware controller which can control several fixed head disks. It maintains a queue in hardware of up to 32 outstanding requests. The queue is kept sorted in SLTF order, and the requests are handled accordingly [43].

There is a large body of literature that considers rotational scheduling. Among those papers are [20,25,31,44–52].

2.6.4. *Skip sector allocation*

Three other optimization methods can also be included in the category of rotational scheduling. First, we note that a process which is sequentially processing a file will generally have some minimum time between I/O requests, and this time is usually larger than the time to pass the inter-record gap on a disk. The result is that such processes experience latencies that are frequently close to a full disk or drum rotation, rather than the expected one-half. The solution is 'skip sector' allocation, by which the sectors (from 1–9) might be numbered 1, 6, 2, 7, 3, 8, 4, 9, 5. Thus, if a process reads a block and a very short time later issues a request to read the next sequential block, with high probability the next block to start to pass under the read head will be the desired one. In many cases, this will cut the mean latency substantially [53].

2.6.5. Track offset for head switching

A related problem is that of head switching when a process reads the last block on a track and then the first block on the next track. The head switching time is usually significantly larger than the time to pass the interrecord gap; thus if the track start position is aligned on all tracks in the cylinder, rotational latency will often be almost a full rotation time. The IBM 3310 disk (discussed below) solves this problem by offsetting the start of track position in consecutive tracks by eight sector positions ($\frac{1}{4}$ track). Therefore, a read can continue on the next track without missing a rotation. (See also [54].) A similar scheme is used for the 3370 [55].

2.6.6. Folding

Finally, a third rotational optimization is 'folding' [56,57]. This is a means of trading space for time. Frequently used pages on a drum are replicated at different angular positions, so that a request is serviced by the first copy to pass a read head. Now that main memory has become so cheap, such pages can almost always be kept memory resident.

2.7. Look ahead fetch and allocation

Because many or most files (certainly most user files) are read sequentially, look ahead fetches to files being read can reduce or eliminate the need to wait on I/O operations. This idea is particularly valuable for very fast machines which run very large programs (e.g. the Cray I), since the degree of multiprogramming usually cannot be increased sufficiently to overlap most I/O waits.

Two different and interesting schemes for look ahead fetching are described in the literature. In [58] data for the IBM IMS data base system was studied. See also [59,60]. Information was collected on sequences of references to consecutively located blocks on the disk. It was found that disk blocks tended to be read in sequential runs, with the run length distribution being highly skewed. Therefore, a prefetch algorithm could be designed to fetch a variable number of blocks ahead, depending on the length of the current sequential run. Dynamic programming was used to develop an algorithm which prefetched the correct number of blocks, such that the costs associated with both demand fetches and unnecessary prefetches were minimized.

Different considerations motivated the work done by Powell [6] on the DEMOS operating system for the Cray I. For that machine, it is typical to have one program or a small number of programs at a time running on the machine in batch mode. Frequently, one or more sequential files are being processed. The buffer space that is available must be dynamically allocated among all of the files that are being read or written, and the space may also be used for other operating system functions. The problem therefore is to dynamically allocate just enough buffer blocks to a file to reduce to a minimum the probability that it will run out of data, without over allocating buffer blocks and therefore wasting them.

There is some other literature relating to look ahead fetch and allocation. The interested reader might look at [61].

2.8. Compaction and fragmentation

There are several ways to allocate space on disk for files. A simple method is to divide the disk surface into fixed size sectors, and to assign these sectors randomly as space is required; this method is used in UNIX [62]. Unfortunately, since most files are read and written sequentially, this results in a disk seek being required for almost every I/O, even though seeks might not have been required had the file been allocated sequentially. Further, it makes look ahead fetch very difficult, since the next block in the file can be located anywhere on the disk surface, and must be found (via tables or links) before it can be accessed. Therefore, even though this scheme has no problem with external fragmentation or compaction, it is not very desirable for large scale machines, and doesn't even work very well for small ones. Fixed block allocation has the additional problem of internal fragmentation, by which the last block in the file is only partially in use. If the blocks are large and the files small, this can be significant effect.

Much more efficient is the idea of allocating data sets in extents. That is, when a file is being written, a block of space is allocated to it; usually several tracks or cylinders. If that amount of space is insufficient, another block (extent) is allocated; if possible, the next sequential block, otherwise wherever convenient. If a block of the desired size (e.g. 10 cylinders) is not available, two or more smaller blocks may be supplied to fill the space request. The only constraint is that some systems limit the number of extents that can be recorded. (E.g. in IBM systems, the limit is 16 extents.) When the file is deleted, the space that it occupied (one or more extents) is returned to the free space list.

When allocating by extents, external fragmentation can become a problem; i.e. after a period of time, the space available for file allocation becomes fragmented into a large number of small blocks (separated by space in use) and large extents are not available. Since files can be allocated in several segments, most files can still be placed. The problems that arise are the following:

(a) some files cannot be allocated because the desired amount of space cannot be obtained with the limited number of extents permitted, and

(b) when the file is allocated in noncontiguous segments, there is significantly reduced physical sequentiality and the optimizations mentioned above (arm scheduling, block size selection, look ahead fetching) are no longer as effective.

Internal fragmentation can also be a problem in two different ways when using extents. First, a user may specify too large an extent. If that extra space isn't released, significant amounts of disk space may be wasted. Similarly, if fixed physical block sizes are in use, the last physical block may also contain empty space.

There are several steps to be taken to minimize the fragmentation problem. One of the more interesting is discussed in [6]. In the DEMOS system, file blocks can be allocated one at a time as the file is written, much as in Unix. Powell suggests that when a pattern of sequential writing is observed, a number of blocks ahead of the point of writing is observed, a number of blocks ahead of the point of writing be reserved (to be released later if necessary). IBM operating systems, conversely, ask the user to specify an extent size, all of which is initially reserved (to be optionally released later).

The most general solution to fragmentation is compaction; that is, the partial or total reorganization of the information on the disk, so that fragmentation is temporarily eliminated. Compaction is undesirable for several reasons:

(a) It is slow and time consuming, thus using system resources.

(b) It will usually prevent access to the area being compacted, thus disrupting system use.

(c) It is necessary to reset any pointers that refer to the file (or to locations within the file) by physical address; this may not always be possible.

Despite these problems with compaction, it is practical, especially when used on a partial basis. The author has been informed that the algorithms suggested in [63] have been operational on a daily basis in more than 50 installations for almost two years. Apparently five minutes per volume per day keeps 3330-11 or 3350 volumes essentially free of fragmentation problems.

A general discussion of the compaction/fragmentation problem is available in [64]. More recent work relating directly to file systems is in [65] and [66].

2.9. I/O congestion and multiple paths

It is not intuitively obvious, but it is the understanding of the author that congestion in the I/O system most frequently manifests itself not at the device, but at the channel, storage controller or string controller. This happens because one channel may interface with two or more storage controllers, each of which will control two or more strings, each string having a head of string controller and several disk spindles. A significant decrease in such congestion can be obtained by allowing multiple paths; i.e. a storage controller might interface to two channels, and a string might have two string controllers. Two published performance analyses of these strategies exist, [67,68], and show that multiple pathing can yield significant benefits.

2.10 File structure

A very important aspect of file system optimization is the structuring (and searching) of files in a way that reflects the observed or anticipated use of that file. Tree structured files (e.g. ISAM, VSAM, B-Trees, etc.) are particularly useful for files with random insertion or deletion. Other structures are used under other circumstances. A good discussion of the general data structure issues is provided in [13]. More recent research relating specifically to file systems may be found in [69].

2.11. Other file system tuning

Additional methods for improving I/O and file system performance are discussed in [16] and we summarize some of them here:

2.11.1.
There should be several paging data sets and those data sets should be spread across as many devices, controllers and channels as possible to minimize congestion.

2.11.2.
Other frequently used data sets (user catalogs, spool data sets, scratch data space and user data bases) should also be widely distributed.

2.11.3.
Frequently used system modules should be either made main memory resident or should be placed in a special paging data set, such that the modules can be immediately located. Modules should also be packaged (as densely as possible) on page boundaries. Modules used together should be stored together.

2.11.4.
Pages used just frequently enough to be faulted on frequently (e.g. timing routines) may be fixed in memory.

2.12. Paging I/O

It should be pointed out that paging I/O differs from user initiated I/O in that it is almost totally under the control of the operating system. Thus each of the optimizations described above can be employed in the desired manner by changing only a small amount of supervisor code rather than by forcing users to change their habits and individual programs. In particular, if devices, controllers and channels are dedicated solely to paging, congestion can be controlled and minimized.

2.13. I/O for real time systems

Real time systems are those that must respond in 'real' or wall clock time. Typical examples are process control, robots and military applications. Such real time systems are almost always run on dedicated minicomputers, and hence we mention them only briefly. Most important, we note that certain I/O events may require very fast response; the usual scheme is to hardwire a priority interrupt mechanism with a large number of priority levels. A discussion of some of the considerations and some implementations appear in [70].

3.0. Current and forthcoming device architecture

An important influence on both the design and performance of the file and I/O systems is the architecture of the I/O devices. Since Direct Access Storage Devices (DASD) such as disks and drums are the primary components in most large machine I/O configurations, we shall concentrate on their architecture and future development. Further, since IBM is the dominant force in large data processing installations, we shall also discuss primarily IBM or IBM compatible devices, and aspects of the IBM I/O system architecture. We also note that a large number of IBM I/O devices have become industry standards, in that they are available from several vendors other than IBM [71–74].

First, we shall consider current IBM device and system architecture, its problems, and the new fixed sector disks (3310, 3370) which solve some of these problems. Other problems are addressed by the semiconductor drums that are being made by STC and Intel. Finally, we shall discuss some of the future directions for the I/O and file system as they affect device architecture and I/O system operations.

There are some topics related to the material in this section which we shall not discuss in any detail. IBM channel architecture is covered to some extent in [75]. Disk technology is discussed in [76]. For control units, see [73,77–79].

3.1. Count-key-data disk architecture and use

Up until recently, all IBM large disk products (i.e. excluding things such as floppy disks) were of the 'count key data' (CKD) architecture. By this, it is meant that data blocks on the disk are composed of three parts (Fig. 5). The first part of each block is the COUNT, which contains the physical address of its own block and defines the size of the

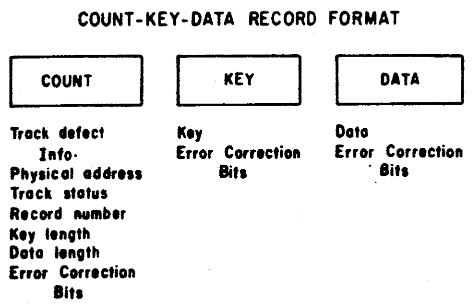

Fig. 5. Count-key-data record format.

key and data areas of the block. The KEY area is optional, and contains a key associated with the block. It is possible to search a track, or more general portions of a file, for a specific record by using search commands that look for certain key values [80]. Finally, the DATA area contains the actual programmer defined data in the file. There is a gap between each of the three components of a block and between blocks. Because of the existence of these gaps, the storage capacity of a disk is a function of the block length; short blocks result in far more gaps which occupy disk area without containing data. For example, the maximum track capacity of an IBM 3350 disk [18,74] with one block is 19069 bytes; this shrinks to 5840 bytes when using 80 byte blocks, as is common (see above). (Our example for standard CKD disk architecture shall be the IBM3350. At the time this is being written, the 3380 [81] has been announced, but no details are available. The 3340 [82] is not typical and is a poor example.)

An I/O operation takes place as the result of the execution of a channel program. A channel program is constructed by the operating system as the result of some I/O request. A typical channel program might appear as in Table 1.

A given channel program can repeat the last two commands several times so that several adjacent blocks can be read or written, or it can even repeat the whole sequence so that a large number of scattered blocks may be transferred.

In the IBM 370 architecture, a channel program may be initiated with either of two instructions: Start IO (SIO) or Start IO Fast Release (SIOF). The SIO instruction initiates a channel program,

Table 1
Channel program

Command	Parameters	Purpose
Seek	Cylinder and track address	Move read/write head to correct track
Set sector	Sector number	Release I/O path during rotation
Search	Block identifier or key	Find identifier of the desired block
Read or write	Buffer address	Read or write data

and then keeps the CPU waiting while it is determined that the channel, control unit and device are all available. The SIOF instruction basically waits only to determine if the channel is available; if the rest of the I/O path is not free, the channel program will have to be restarted. The SIOF instruction is generally a lot faster, since an SIO can take on the order of 100 μs to complete.

A discussion of some of the material above may be found in [18,80].

3.2. Shortcomings of current architectures

The description in Section 3.1 of the nature of IBM device and I/O system design should immediately suggest some problems. First we note that the sequence of commands for most CKD channel programs does not permit much flexibility or optimization. Because each command contains very little information, it is very difficult to look ahead and sequence requests, nor can buffering be set up in the correct direction, since it is not known in advance whether the operation is read or write.

The record format also inhibits many kinds of optimization. First, since records are of variable size (and may span several tracks, up to a maximum record size of 32768 bytes), it is difficult to provide general purpose fixed size buffers. (This is different from some other computer systems; may other manufacturers have long since chosen to implement only fixed size blocks located at fixed sector addresses.) Second, when doing a read or write, the actual data address may not be known, since a key search may be needed to determine which record is actually desired. It is possible to set up a channel program that searches a large number of tracks looking for the desired record. The need to examine many blocks before locating the correct target may interfere with buffering strategies.

The way in which the CPU starts channel programs is inherently inefficient. Even with the SIOF instruction, the CPU must wait to determine whether the channel is free; if the rest of the path is not free, the start IO must be reissued. Ideally, an I/O operation could be unconditionally initiated, and would then be queued by the hardware until it either completed or failed in such a way that software intervention was required.

Similar complaints may be made about current small disk designs.

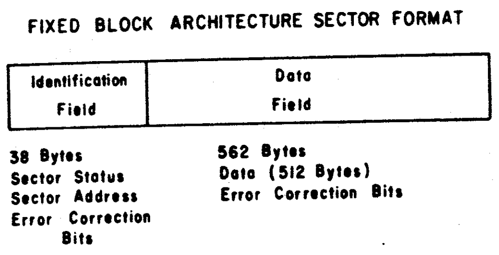

Fig. 6. Fixed block architecture sector format.

3.3. Fixed sector disks

Within the last year, IBM has begun shipping two new disk products, the 3310 disk [83] and the 3370 [84] disk, which have what is called the fixed sector (FS) architecture. Because these new disks have certain advantages over the older designs [85], and because they represent a significant change over the older disks, we believe that they indicate the future direction of IBM disk architecture. The recently announced 3380 [81] and 3375 [86] are believed by this author to be the end of the line, rather than indicating a continuation of the CKD architecture for computer systems beyond the System 370.

We concentrate here on the 3310 disk which was the first to be shipped. The 3310 is designed with exactly 32 blocks per track, each block of which has exactly 512 data bytes and no key field. There are also 88 bytes per block for the record address, error correcting codes, etc. (See Fig. 6.) The sectors are numbered in such a way that on each successive track, the sector numbers are offset by 8. This is so that a multiblock transfer which spans consecutive tracks can continue without interruption despite the time to switch heads, which is never more than the time to rotate 8 sectors.

The commands accepted by a 3310 disk are significantly different from those used for the 3350. All reading or writing occurs in what is referred to as an extent (not the same 'extent' as noted above in Section 2.8). The first command in all channel programs is 'define extent' which specifies a sequence of consecutive blocks. Typically, the next command is 'locate' which indicates the starting block, the block count (number of blocks) and the operation to be performed. Locate does not, however, actually initiate the operation. The final command is 'read' or 'write' with a buffer address. We note that the all of the relevant information is received by the time of the second command; that is, the locate specifies the operation, i.e. which blocks are to be transferred and whether they are to be read or written. There is no uncertainty such as occurs when there is a keyed search.

The advantages of this simplified architecture are several. The fixed block size significantly minimizes data management problems; that is, all buffers can be a fixed size. External fragmentation on the disk surface is eliminated, since there is always an integral number of blocks per track. The management of any sort of I/O cache [1] is simplified, since all blocks are of fixed size. The number of interrecord gaps is limited to 32; thus a certain level of storage utilization is guaranteed.

The nature of the commands accepted by the 3310 permit future changes in the structure of the I/O system. Specifically, only the final command (read or write) requires a response; thus the entire channel program could be accepted and queued in the channel or control unit without any attention from the CPU.

Some of the features of the 3370 disk are also notable. In order to minimize arm contention, the disk has two arms, each serving a different set of disk surfaces. The data rate is 1.859 mbytes/s which is higher than can be accommodated on a single byte wide standard IBM channel. Finally, the data density on the disk is much higher than on earlier disks.

3.4. Electronic drums

Disks have an inherent shortcoming: they are mechanical and there are limits to the performance of mechanical systems. For example, the rotation time of the 3350 disk, which is designed for high performance machines, is 16.7 ms, and for the much newer 3310 disk, it is 19.1 ms. The performance of the 3370 is only slightly better than that for the 3310. Even the 3375 and 3380 are not significantly faster. It is generally known [76] that improvements in the mechanical access times of disks and drums are difficult to obtain and changes will be small. Therefore, if fast I/O response is required, mechanical delays must be eliminated.

Two companies produce electronic replacements for the IBM 2305 fixed head disk. Intel [87] is making its Fast 3805 electronic drum, which is in several customer sites undergoing tests at the time of this writing. It is about 10 times faster than the 2305 [41] and offers greater data capacity (12–72 mbytes) at lower cost and higher transfer rate. It is composed of MOS RAM chips. Storage Technology Corporation is making the 4305 Solid State Disk. Originally designed with CCD's, it has been redesigned using MOS RAM, since CCD's were not available in the desired quantity. Although random access memory can normally be used more effectively as main memory, it may still be cost effective for electronic drums. There are several reasons why main memory is not simply expanded; some of them appear in [1]. Most significant is the fact that major changes in the operating system would be required.

Both of the electronic drums (or electronic disks) discussed here are likely to sell in good volume, since many high end machines have heavy paging loads which are currently serviced by a combination of 2305 fixed head disks, also referred to as drums, which are expensive and small, and 3330 and 3350 disks which are slow. There are not yet many of these devices in the field, and their success in improving system performance has not yet been shown. The importance of electronic drums is that their high speed and data transfer rates may avoid I/O bottlenecks [1] that are likely to occur as increasing CPU speeds result in higher levels of I/O traffic than can be accommodated by current or future mechanical I/O systems.

3.5. Future directions for I/O device and system architecture

Several changes are likely in I/O device and system architecture in IBM-like systems over the next few years. These changes will be the result not only of improvements in the technology directly applicable to the I/O system, but are also due to changes in the rest of the system which make I/O system improvements important.

Currently the CPU in an IBM operating system requires a detailed knowledge of the I/O transaction and spends a significant amount of time ensuring that the I/O operation occurs as intended. Specifically, it writes the channel program, translates virtual to real addresses, fixes pages, provides physical device addresses (track, cylinder), receives I/O condition codes (often after every step), etc. It is reasonable to expect that the CPU will eventually be able to issue a command of the form: {(virtual) buffer address, logical data address (i.e. byte range within file), read/write} and the rest will all occur asynchronously. There will be no delay for channel, controller or device busy; the command will be accepted and queued in either the channel, controller, or device. All I/O interrupts possibly excepting the final one will be suppressed. In fact, the I/O interrupt could be placed on a memory resident queue which would be periodically examined by the CPU rather than actually interrupting the CPU.

Current physical protocols between the channels and the controllers are too slow for upcoming systems. Even a double wide data path permits only a 3 mbytes/s transmission rate because of the handshaking required. (The IBM channel interface protocol is of the form: 'please', 'here it is', 'thank you', 'you're welcome' [88].) Faster CPU's will mean new and faster data transfer protocols [89]. Further, higher data densities on the disk surface means that data will have to be transmitted faster. This has already happened to some extent with the IBM 3380 which due to its high bit density has to transmit at 3 mbytes/s. This is accommodated by bypassing the standard channel protocol and wiring the storage controller into the channel director rather than to the standard channel interface.

I/O transfers could benefit from additional buffering and parallelism. Controllers and channels should be able to transmit more than one I/O data streamat a time, especially since a controller may be shared by more than one CPU. (The new IBM 3880 Storage Controller [78] contains two 'storage directors', each of which is really an independent controller. We are suggesting that each effective controller, whether it is called a director or controller, have the capacity to pass multiple data streams.) High data rates suggest that more buffering at each end of each transmission line could improve performance. In particular, the limited buffering currently (of only a few bytes) means that all events must be responded to quickly or the operation aborted. A significant amount of buffering already exists in the Cray disks, which can buffer an entire disk sector and then transmit it as needed at a much higher data rate than the inherent disk data rate. This feature is used advantageously in the DEMOS operating system for the Cray [6], in which the next physical sequential block is buffered.

It might be useful to permit larger I/O block sizes. Current 32 K limitations are smaller than might sometimes be desired for high density tapes. (The author has been informed that the basic 370 I/O architecture allows up to 64 K records with a single channel command word. The 32 K restriction is imposed by the access methods.)

We therefore expect that the following changes will take place:

(a) Controllers and/or channels will become a lot more intelligent. They will be able to take simple commands and execute long and complex sequences of operations from them. They will be able to execute simultaneously several different I/O operations and will be able to perform commands out of order when that is desirable.

(b) Much more buffering will be introduced. Not only will I/O streams be transfer buffered to a greater extent, but cache buffers will appear [1].

(c) Devices will have higher transfer rates, more local buffering, both for I/O transfers and to hold data while data paths are busy, and will have substantial local error correction facilities [90].

4. Summary

In this paper we have surveyed the state of the art in large I/O system architecture and optimization. Section 2 considered most of the possible I/O system optimizations and provided a survey of the relevant literature, including analytic modeling papers. Section 3 examined disk and I/O architecture. The nature of current count-key-data disks and the new fixed block disks was discussed. Shortcomings of IBM I/O architectures were examined, and some likely changes were indicated. In the companion paper to this [1], we go one step further, and summarize some of our research results which will lead to improved I/O system performance.

References

[1] A.J. Smith, Optimization of I/O systems by cache disk and file migration: a summary, in Performance Evaluation (1981) to appear.

[2] R.C. Daley and P.G. Neumann, A general purpose file system for secondary storage, Proc. FJCC (1965) 213–229.

[3] R.J. Feiertag and E.I. Organick, The multics input/output system, Proc. Third SIGOPS, Oct. 1971, Stanford University, Stanford, CA, pp. 35–41.

[4] G. Pirkola, A file system for a general purpose time sharing environment, Proc. IEEE 63 (6) (1975) 918–924.

[5] W.A. Clark, The functional structure of OS/360, Part III, Data management, IBM Systems J. 5 (1) (1966) 30–51.

[6] M. Powell, The DEMOS file system, Proc. Sixth ACM Symp. on Operating Systems Principles, Nov. 1977, pp. 33–42.

[7] A.J. Smith, Bibliography on file system and input/output optimization and related topics, submitted for publication.

[8] S. Berbec, A. Shibamiya, S. Togasaki and H. Yoshida, Use of direct access storage devices by MVS customers—guide survey results, Proc. Guide 47 Conference, Nov. 1978, Chicago, IL, pp. 1121–1138.

[9] T. Peterson, Criteria for optimal blocksize selection: Computer measurement and evaluation, selected papers from the SHARE project, Vol. III, Dec. 1973–Mar. 1975, pp. 454–465.

[10] R.E. Paulhamus and G.E. Ward, A study on the effectiveness of data blocking in an MVS environment, Proc. Computer Performance Evaluation Users Group Meeting, Oct. 1977, New Orleans, Louisiana, pp. 143–158.

[11] O. Nevalainen and M. Vesterinen, Determining blocking factors for sequential files by heuristic methods, Comput. J. 20 (3) (1977) 245–247.

[12] C. Lazos and A. Vafladis, A method to estimate the I/O buffer size in a computer system, Comput. J. 22 (4) (1979) 323–327.

[13] D.E. Knuth, The Art of Computer Programming, Vol. 3, Sorting and Searching (Addison-Wesley, Reading, MA, 1973).

[14] A.J. Smith, A locality model for disk reference patterns, Proc. IEEE Comput. Soc. Conf., Feb. 1975, San Francisco, CA, pp. 109–112.

[15] A.J. Smith, Analysis of a locality model for disk reference patterns, Proc. 1976 Conf. Information Sci. and Syst., Mar. 1976, The Johns Hopkins University, Baltimore, MD, pp. 593–601.

[16] T. Beretvas, Performance tuning in OS/VS2 MVS, IBM Systems J. 17 (3) (1978) 290–313.

[17] W. Piepmeier, Optimal balancing of I/O requests to disks, Comm. ACM 18 (9) (1975) 524–527.

[18] IBM Corp., Reference manual for IBM 3350 direct access storage, GA26-1638, IBM Corp., Armonk NY.

[19] T. Teorey and Tad B. Pinkerton, A comparative analysis of disk scheduling policies, Comm. ACM 15 (3) (1972) 177–184.

[20] E.G. Coffman and P.J. Denning, Operating Systems Theory (Prentice-Hall, Englewood Cliffs, NJ, 1973).

[21] E.G. Coffman and M. Hofri, On scanning disks and the analysis of their steady state behavior, Proc. Conf. on Measuring, Modelling and Evaluating Computer Systems, Oct. 1977 (North-Holland, New York) 251–263.

[22] M. Hofri, Disk scheduling: FCFS vs. SSTF revisited, Proc. Second Colloque International Sur Les Systems D'Exploitation, IRIA, Oct. 1978. Republished, Comm. ACM 23 (11) (1980) 645–653.

[23] J. Abate, H. Dubner and S. Weinberg, Queueing analysis of the IBM 2314 disk storage facility, JACM 15 (4) (1968) 577–589.

[24] E.G. Coffman, L.A. Klimko and B. Ryan, Analysis of scanning policies for reducing disk seek times, SIAM J. Comput. 1 (3) (1972) 269–279.

[25] P. Denning, Effects of scheduling on file memory operations, Proc. SJCC 1967, pp. 9–21.

[26] D.W. Fife and J.L. Smith, Transmission capacity of disk storage systems with concurrent arm positioning, IEEEEC 14 (8) (1965) 575–582.

[27] H. Frank, Analysis and optimization of disk storage devices for time sharing systems, JACM 16 (4) (1969) 602–620.

[28] C.C. Gotlieb and G.H. MacEwen, Performance of movable head disk storage devices, JACM 20 (4) (1973) 604-623.

[29] T.J. Teorey, Properties of disk scheduling policies in multiprogrammed computer systems, Proc. FJCC (1972) 1-11.

[30] N.C. Wilhelm, An anomaly in disk scheduling: a comparison of FCFS and SSTF seek scheduling using an empirical model for disk accesses, Comm. ACM 19 (1) (1976) 13-17.

[31] C.K. Wong, Minimizing expected head movement in one-dimensional and two dimensional mass storage systems, Comput. Surveys 12 (2) (1980) 167-178.

[32] W.C. Lynch, Do disk arms move?, Performance Evaluation Review 1 (4) (1972) 3-16.

[33] A.J. Smith, On the effectiveness of buffered and multiple arm disks, Proc. Fifth Annual Symp. on Computer Architecture, Apr. 1978, Palo Alto, CA, pp. 242-248.

[34] M. Franklin and A. Sen, An analytic response time model for single- and dual-density disk systems, IEEE Trans. Comm. 23 (12) (1974) 1269-1276.

[35] D.L. Stone and R. Turner, Disk throughput estimation, Proc. ACM Annual Conference, Aug., 1972, pp. 704-711.

[36] P.H. Seaman, R.A. Lind and T.L. Wilson, On teleprocessing system design, Part IV, An analysis of auxiliary storage activity, IBM Systems J. 5 (3) (1966) 158-170.

[37] N.C. Wilhelm, A general model for the performance of disk systems, JACM 24 (1) (1977) 14-31.

[38] S. Fuller, An optimal drum scheduling algorithm, IEEE Trans. Comm. 21 (11) (1972) 1153-1165.

[39] S. Fuller, Minimal total processing time drum and disk scheduling disciplines, Comm. ACM 17 (7) (1974) 376-381.

[40] H. Stone and S.H. Fuller, On the near optimality of the shortest—Latency time first drum scheduling discipline, Comm. ACM 16 (6) (1973) 352-353.

[41] IBM Corp., Reference manual for the IBM 2835 storage control and the IBM 2305 fixed head storage module, GA26-1589, (IBM Corp., San Jose, CA, 1972).

[42] A. Rafii, Study of the performance of RPS, Performance Evaluation Review 5 (4) (1976) 21-38.

[43] Burroughs Corp., Burroughs B6700 information processing systems reference manual, (Burroughs Corp., Detroit, MI, 1978).

[44] C. Adams, E. Gelenbe and J. Vicard, An experimentally validated model of the paging drum, Acta Informat. 11 (1979) 103-117.

[45] J. Abate and H. Dubner, Optimizing the performance of a drum-like storage, IEEE Trans. Comm. 18 (11) (1969) 992-997.

[46] W.H. Burge and A.G. Konheim, An accessing model, JACM 18 (3) (1971) 400-404.

[47] R.A. Cody and E.G. Coffman Jr., Record allocation for minimizing expected retrieval costs on drum-like storage devices, JACM 23 (1) (1976) 103-115.

[48] E.G. Coffman, Jr., Analysis of a drum input/output queue under scheduled operation in a paged computer system, JACM 16 (1) (1969) 73-90. Also see Errata, JACM 16 (4) (1969) 646.

[49] E. Gelenbe, J. Lenfant and D. Potier, Response time of a fixed head disk to transfers of variable length, SIAM J. Comput. 4 (4) (1975) 461-473.

[50] S. Ghosh, File organization: Consecutive storage of relevant records on drum-type storage, Information and Control 25 (1974) 145-165.

[51] P.J. Denning, A note on paging drum efficiency, Comput. Surveys 4 (1) (1972) 1-3.

[52] W. Oney, Queueing analysis of the scan policy for moving head disks, JACM 22 (3) (1975) 397-412.

[53] H. Chorosz, System allowing direct access to sequential files, IBM Technical Disclosure Bulletin 19 (6) (1976) 2105-2107.

[54] J.E. Guest, R.W. King and R.C. Kiscaden, Logical sector interleave, IBM Technical Disclosure Bulletin 17 (5) (1974) 1460-1463.

[55] IBM Corp., Disk storage technology, (IBM Corp., San Jose, CA, 1980).

[56] L.J. Scheffler, Optimal folding of a paging drum in a three level memory system, Proc. Fourth SIGOPS Conf. Yorktown Heights, NY, Operating Systems Review 7 (4) (1973) 58-65.

[57] R. Baird, Rapid access method for fixed block DASD record, IBM Technical Disclosure Bulletin 20 (4) (1977) 1565-1566.

[58] A.J. Smith, Sequentiality and prefetching in data base systems, ACM Trans. Database Systems 3 (3) (1978) 223-247.

[59] A.J. Smith, Sequential program prefetching in memory hierarchies, IEEE Trans. Computers 11 (12) (1978) 7-21.

[60] J. Rodriguez-Rosell, Empirical data reference behavior in database systems, Comput. J. 9 (11) (1976) 9-13.

[61] D.E. Gold and D.J. Kuck, A model for masking rotational latency by dynamic disk allocation, Comm. ACM 17 (5) (1974) 278-288.

[62] D. Ritchie and K. Thompson, The UNIX time sharing system, Comm. ACM 17 (7) (1974) 365-375.

[63] P.A. Franaszek and J.P. Considine, Reduction of storage fragmentation on direct access devices, IBM J. Res. Develop. 23 (2) (1979) 140-148.

[64] D.E. Knuth, The Art of Computer Programming, Vol. 1, Fundamental Algorithms, 2nd edition (Addison-Wesley, Reading, MA, 1973).

[65] J.P. Considine, A computable measure of fragmentation for direct access volumes, IBM Res. Rep. 6241 (1976).

[66] K. Maruyama and S.E. Smith, Optimal reorganization of distributed space disk files, Comm. ACM 19 (11) (1976) 634-642.

[67] Y. Bard, A model of shared DASD and multipathing, Comm. ACM 23 (10) (1980) 564-583.

[68] A.J. Smith, An analytic and experimental study of multiple channel controllers, IEEE Trans. Comm. C-28 (1) (1979) 38-49.

[69] P. Quittner and D. Kotsis, Comparison of different disk searching methods, Software-Practice and Experience 8 (6) (1978) 673-679.

[70] K.J. Thurber, R.C. DeWard, T.W. Petschauer, M.P. Fedde, and D.G. Kaminski, I/O concepts for a real time system, Proc. Compcon, 1976, Washington, DC, pp. 190-195.

[71] Computerworld, Plug-compatible drives: What to look for, Computerworld (1980) 10.

[72] M. Blumenthal, PCMs braving ups and downs in disk market, Computerworld (1980) 1.

[73] Memorex Corp., 3674 storage control unit theory of operations manual; PN 3674.21-01, (Memorex Corp., Santa Clara, CA, 1979).

[74] Memorex Corp., 3650 direct access storage subsystem, theory of operations manual, PN 3650.21-02, (Memorex Corp., Santa Clara, CA, 1978).

[75] D.T. Brown, R.L. Eibsen and C.A. Thorn, Channel and direct access device architecture, IBM Systems J. 8 (3) (1972) 186–199.

[76] K. Haughton, An overview of disk storage systems, Proc. IEEE 63 (8) (1975) 1148–1152.

[77] IBM Corp., Reference manual for IBM 3830 storage control and IBM 3330 disk storage, GA26-1592, (IBM Corp., Armonk,'NY 1972).

[78] IBM Corp., IBM 3880 storage control description, GA26-1661, (IBM Corp., San Jose, CA, 1980).

[79] G.R. Ahearn, Y. Dishon and R.N. Snively, Design innovations of the IBM 3830 and 2835 storage control units, IBM J. Res. Develop. 16 (1) (1972) 11–18.

[80] J. Buzen, I/O subsystem architecture, Proc. IEEE 63 (6) (1975) 871–879.

[81] IBM Corp., IBM 3380 direct access storage, IBM DPD Product Announcement (June, 1980).

[82] R.B. Mulvany, Engineering design of a disk storage facility with data modules, IBM J. Res. Develop. 18 (6) (1974) 489–505.

[83] IBM Corp., IBM 3310 direct access storage reference manual, GA26-1660, (IBM Corp., Armonk, NY 1979).

[84] IBM Corp., IBM 3370 direct access storage description, GA26-1657, (IBM, General Products Division, San Jose, CA, 1979).

[85] W.F. Jurist, FBA disks faster, cheaper, Computerworld 14 (28) (1980) 1.

[86] IBM Corp., IBM 3375 Direct Access Storage, IBM DPD Product Announcement (June, 1980).

[87] Intel Corp., FAST-3805 functional description, PN 19-1619-006, Aug. 1979, Intel Commercial Systems Division, Phoenix, AZ.

[88] IBM Corp., IBM system/360 and system/370 I/O interface channel to control unit original equipment manufacturers information, PN GA22-6974-3, (IBM Corp., Jan. 1976, Poughkeepsie, New York).

[89] D. Hillman, Intelligent buffer reconciles fast processors and slow peripherals, Electronics 53 (20) (1980) 131–135.

[90] M. Feller, Intelligent disk: The next generation, Proc. IEEE Comput. Soc. Conf. Sept. 1977, pp. 306–310.