# Saving the Precious Resource -- Disc Accesses

by
Jim Kramer
HP3000 Performance Specialist
Hewlett-Packard Co.
St. Louis, Missouri

## I. Introduction

Disc accesses is, along with CPU and memory, one of the three major resources on a computer system.

On a 3000 it is the most likely of the three to be the critical resource -- the one in short supply. This is because the 3000 is used primarily for business processing, which is usually I/O intensive.

This paper presents some ways of saving this resource when it is appropriate to do so.

## II. What is a Disc Access

By disc access I mean the entire process of reading a block of data from disc or writing it to disc. This includes such activities as software setup, disc head seek to the required track, wait for the required block to pass beneath the disc head (latency), and transfer of data between disc and main memory.

As a rule the longest part of a disc access is the seek; the following figures which pertain to an average access on an HP 7925 disc drive show why:

| | | |
|---|---|---|
| Software setup | -- | small value -- varies with CPU |
| Seek | -- | 25.0 ms |
| Latency | -- | 11.1 ms |
| Transfer time | -- | 1.4 ms/kbyte |
| Total | -- | about 40 ms |

The total of about 40 ms is what leads to the rule of thumb that the 3000 can maintain an average of about 25 disc accesses per second. It is important to notice the assumptions behind such a statement, however. One assumption is that only one disc

drive at a time is active, a situation that may not hold under MPE IV for systems with multiple discs. Another assumption is that an average access takes takes 40 ms, whereas the true average depends on the actual distribution of data and the sequence in which it is accessed. This assumption is probably close to being true for a time-sharing multi-user environment because of the randomness of access in such an environment, but unlikely to be true for a stand-alone batch environment.

III. What Resource is Critical

Before any optimization is done to save accessess it should be determined that accesses is indeed the limiting resource. The reason is that most techniques of saving accesses are trade-offs -- their price is the increased use of some other resource (usually main memory). If that other resource is the limiting resource then saving accesses may actually undermine performance.

Unfortunately there is no easy method to determine the limiting resource other than to purchase performance consulting from HP. An HP performance specialist has access to measurement tools which are not normally available to customers. If this is done, great care should be taken to assure that measurements are taken during a period which is typical of the problem to be solved.

There are some tools available to customers which could probably be used to do a fairly reliable diagnosis by someone experienced in their use. I will mention these tools here, but a discussion of their use is beyond the scope of this paper.

Some tools are; the busy lights on the disc drives; the current instruction register on the Series I, II, and III; the status display on the console of Series 30's and 33's; the SHOWQ command; the contributed program SOO; the log files.

One tool which should be mentioned specifically is the contributed library program FILERPT written by Chuck Storla, an SE from the Rolling Meadows office. FILERPT can show which files are being accessed most and therefore where optimization should begin once it has been decided to optimize for disc accesses.

IV. Some Design Suggestions

A few things should be kept in mind during the design stage of an application.

Load balancing is often an effective way to alleviate performance problems. The idea is to redistribute the work load of the computer system to reduce the load during the problem period.

One form of load balancing in a transaction processing environment is the technique of batch updating -- collecting

transactions into a batch for updating of the master files at a later time by a batch program. This technique has one major disadvantage -- the data in the master file is not completely up to date. Among the many advantages are:

1. The high-overhead posting operations can be done at a non-peak period.

2. The sharing of the master files for transaction processing is read only -- a more efficient sharing environment than update (no locking is required).

3. Transaction logging and recovery is generally a much simpler problem.

Another major design consideration is the amount of structure -- keys, sort items, etc. -- to use in files. Structure requires considerable machine work to establish during the writing of data, and justifies itself only by the work it saves during reading. Therefore the file must be relatively static, i.e. there must be more reading than writing of it, with respect to whatever structure it has. In terms of a catch phrase:

"A Static File Supports More Structure".

Every piece of proposed structure should be examined in this light. Further, when balancing work saved against work expended it must be considered whether either the expended or saved work is during a peak period. For example a key intended to save work during batch reporting may not be justified only because it adds to the peak load during daytime transaction processing. Or a sort item may justify itself by speeding up transaction processing even though it adds significantly to the nightly batch load.

## V. Increased Blocking

### General Discussion

Using a larger blocking factor is a simple and effective technique for reducing accesses if access to the file has "good locality", i.e. if the next record required is likely to be near within the file to the one just fetched. This is true when the file is being accessed sequentially, and rarely true otherwise. With MPE files it is obvious when access is physically sequential; we will discuss below when access is sequential for Image and KSAM.

It is hard to over-emphasize the importance of the technique of using larger blocks: it is generally very easy and very effective. The only price paid is increased memory requirements for buffering the larger blocks.

With MPE files it is usually possible to double the block size -- thereby cutting disc accesses in half -- without paying a memory price. This is done just by reducing the number of buffers from two (the default) to one by putting "BUF=1" on the file equation. The function of the second buffer -- allowing overlap of processing and I/O -- is usually obviated by MPE's multiprogramming environment. Even if it is not, halving the number of accesses should be a greater benefit than the second buffer.

## Examples of Sequential Access

Sequential access is very common, and thus the opportunities for applying this technique are many. Examples of sequential access are: most batch applications; sorting; the text and keep operations of file editing; source program compilation; and MPE's reading of UDC files at logon and SETCATALOG time.

## Image

Image does not allow control over the blocking of each data set separately -- the BLOCKMAX parameter of DBSCHEMA sets blocking for all data sets. Therefore it may be difficult to take good advantage of the technique. However if it is determined that a large proportion of access to a data base is physically sequential, it may be worthwhile to increase BLOCKMAX above the default.

In Image the most important instance of sequential access is backward or forward serial access to a data set. If the data base has been reorganized by doing a chained unload followed by a load, then chained access on primary keys is also sequential. If there are a large number of secondaries in a master, then accesses can be reduced by larger blocks; however it is probably more appropriate to investigate why the hashing scheme is not working well.

## KSAM Data Files

Chronological access to a KSAM file, including addition of new data to the file, is by definition physically sequential. A very important mode of KSAM access -- keyed sequential -- will be physically sequential if the file was originally loaded in key sequence. Because of this, accesses can be saved during keyed sequential access by using large data blocks and loading in key sequence.

### KSAM versus Image

If a KSAM file is being used only for keyed access, not keyed sequential, Image should probably be used instead. This is because Image's hashing techniques generally fetch a record in a single access, whereas KSAM's B-tree technique requires a search through the multi-level key tree before the data can be read. (On the other hand KSAM is very fast for keyed sequential access which Image cannot do at all).

### KSAM Key Files

KSAM key files can benefit from larger blocks. Perhaps surprisingly the greatest benefit is for random keyed access rather than keyed sequential. The reason is that keyed sequential access is very sparing in its use of the key file regardless of key blocking.

However the opposite is true for random keyed access. Each fetch of a record by key requires a top to bottom search through the key tree. This results in H-1 disc accesses, where H is the height of the tree. (Unless there is locking, the root block does not have to be read -- it is already in the buffers). The fetch of the data record requires another disc access, making H accesses the total number required.

The possible benefit of larger key blocks is that the height of the tree may be reduced. A reduction of tree height from three to two would save one third of the accesses.

Altough it is possible to calculate whether a larger block will actually reduce tree height, it is probably just as simple in most cases to simply try it. The KSAM manual discusses how to specify key block factors.

## VI. Increased Buffering

### General Discussion

Increased buffering is of doubtful value in a sequential access environment, but can under certain circumstances be very helpful in a random access environment. These circumstances are that there are repeated accesses to the same blocks of the file and these blocks can be held in memory. As with increasing block size, the price paid for increasing buffering is main memory.

It some situations it may be possible to use MPE's buffering to implement this technique. Suppose for example that an application makes thousands of accesses to a file which is only 100 records long, each record being 100 bytes long. The total length of the file is only 10,000 bytes. Since an MPE buffer can be as large as 16K bytes (8K words)

it is possible to hold the entire file in a buffer. It doesn't make too much difference how the file is blocked, but it may as well be blocked 100 -- the entire file will be brought into the buffers with a single access, and that is the only access required for the entire run.

The above example is an extreme case, but the principle applies for less extreme cases as well. Suppose that the file were 50000 bytes long -- five times as long as the previous file. Now only about one third of the file can be held in the buffers at one time, but this is still worthwhile: the chances of finding the required block in the buffers is one out of three, and we reduce our disc accesses by one third.

In situations where not all the data in each record is needed, it might be worthwhile to create a new file with smaller records, just so that a larger percentage of the file can be kept in buffers.

It is not necessary to accept MPE's restriction of a 16K buffer segment. A user can do his own buffering with extra data segments and fill as much of main memory as desired with blocks from his file. This requires some sophisticated programming but it might be the solution to some difficult performance problems.

## The Effects of Locking

It is very important to keep in mind that for both MPE and KSAM files file locking can negate the value of buffering entirely and drastically increase disc access requirements. The reason is that for these file types (but not Image), each user has his own set of file buffers. To assure that all users have an accurate view of the file, it is neccesary that a file lock cause clearing of the user's buffers (forcing him to go to the file) and that an unlock post any changed blocks to the file.

It is best to avoid sharing MPE and KSAM files at all, unless access is read only for all users. Otherwise locking is required for proper sharing of the file. If a locking environment cannot be avoided, its harmful effects can be minimized by doing as many operations as possible between each lock and unlock.

## Buffering and Image

Image performance can be helped by increased buffering in certain circumstances. An example is repeated random access to a master which is sufficiently small to allow a signficant part of it to be held in memory. Interestingly perhaps, sequential access can also be helped by additional buffers if they prevent the buffer containing the sequentially accessed block from being overwritten by another block.

In general it is very difficult to know how many buffers is "right" for an Image application. Probably the best advice is to try different numbers of buffers and check the effects on disc accesses by observing the file close log records.

For a batch application the right number is generally the maximum (specify 255). This increases the memory requirements for the batch job but gets it out of the way sooner. In many cases batch jobs are run at night when there is not much competition for memory anyway. Batch jobs which do posting should generally run with buffer posting deferred (invoked by calling DBCONTROL with mode = 1). This causes Image to postpone the posting of a block in a buffer until the buffer is needed for some other block. In many cases the reduction in the number of disc accesses is dramatic. If the system should fail during the running of the batch job, the data base is almost guaranteed to be invalid -- it will be necessary to restore the data base and rerun the job in its entirety. Of course this is the usual tactic for any failure of a batch job.

DBLOAD, by the way, will by default run with the maximum number of buffers and with buffer posting deferred.

### Buffering and KSAM

The buffers for both the key and data files of a KSAM file are in a single data segment. There is always exactly one buffer for the data file, but the user has control over the number of key block buffers. KSAM allows as many as 20 key block buffers, but if key blocks are close to the maximum allowed length of 4K bytes, the maximum number of buffers allowed may be less than this -- as few as 15. Please note that to specify the number of buffers with a file equation, the DEV= keyword is used; the parameter is the one which specifies number of copies when writing to a spooled device. Thus to specify 5 key block buffers, add ";DEV=,,5" to your file equation. (I know this sounds strange, but what reason would I have to lie about it?).

Increased buffering has little if any value for keyed sequential access but may reduce disc accesses dramatically for random keyed access if a significant proportion of the key file can be held in the buffers (assuming no locking, of course). This can be determined easily from the KEYINFO command of KSAMUTIL, which tells the number of blocks in the key file.

## VII. Increasing the Resource

All of the above suggestions have been oriented toward reducing the demand for disc accesses. It is often possible as well to increase the supply of them, by reducing the time it takes to do them, in particular by reducing seek time.

The easiest tactic is a reload. This consolidates the files to one edge of the disc and brings the extents of a file together, thus reducing average seek times.

Another major tactic is to reduce head movement be avoiding the situation in which a head is forced to move back and forth between two or more files on the same disc. Here are some suggestions based on this approach:

1. Avoid putting any frequently accessed files (including spool files) on the system disc, which contains the directory and the swapping area, both of which are frequently accessed.

2. Keep the input and output files of batch activities (including sorts) on separate discs.

3. Keep heavily accessed data sets from a data base on separate discs, especially if they tend to be accessed together.

4. Keeping the key and data file of a KSAM file on separate discs.