VARIATIONS ON A TUNE - ANOTHER LOOK AT THE NEVER-ENDING STRUGGLE TOWARDS OPTIMAL PERFORMANCE

Steven M. Cooper
Allegro Consultants, Inc.
Redwood City, CA, USA

Research sponsored by
ADAGER, Antigua GUATEMALA

Introduction

Back in the Dark Ages (seven or eight years ago), we had no choice but to grope in the dark in terms of tuning our computers and databases for maximum performance; monitoring tools did not exist and the general guidelines that have become common knowledge had not yet been discovered and published. Relief began to come shortly thereafter, though, as articles on performance and tuning, many now considered classics, were written, providing us with a set of guidelines to follow.

Once the articles were written and the information disseminated, nobody seemed to give it much thought, considering the topic adequately covered. However, a lot has happened in the past three or four years in the HP3000 world and perhaps we ought to take another look at some of these issues. Comparing a Series III with one half megabyte of memory to a largish Series 68 running MPE V, the newer machine has a CPU that is five times faster, has up to 16 times more memory, and is capable of five or six times the effective disc I/Os per second (assuming caching). Most systems will also support four times the number of users. Given that we now have a much larger and differently shaped beast to deal with, some of the assumptions of those original articles must have changed, and therefore, so must the guidelines.

Unfortunately, we must all suffer through a period of semi-darkness again as we experiment with caching and its parameters, and learn by trial and error what is good or bad for these new machines. In an effort to encourage the re-opening of these topics, this article presents some observations and results of experiments done to date. I do not have all of the answers, and can't even explain all of the results that I've gotten, but perhaps combined with your own experimentation, we can continue to evolve these guidelines to keep pace with the maturing world of the HP3000.

CACHECONTROL

Caching is a wonderful thing if you've got the extra memory and are I/O-bound. The designers of caching left us with two dials and a switch to play with, so it is our duty as system managers and system tuners to turn and flick and see what happens. The "dials" are the RANDOM and SEQUENTIAL options of the CACHECONTROL command. They control how much is actually read from disc into a cache domain when we issue a read from a cached disc. (This is a simplification. We will discuss some of the complications later.) The RANDOM value tells how many sectors should be read when we do an FREADDIR call. (In COBOL, this happens for files with ACCESS IS RANDOM clauses.) The SEQUENTIAL value corresponds to FREAD calls. (In COBOL, ACCESS IS SEQUENTIAL.) Both parameters can be set to values between 1 and 96 sectors, indicating cache domains between 256 bytes and 24,576 bytes. By default, the system initializes RANDOM to 16 and SEQUENTIAL to 96.

It is important to note that no matter what type of DBGET or DBanything we do, IMAGE always issues FREADDIRs and FWRITEDIRs. (This is also true for KSAM files.) So for the typical IMAGE-based system, the vast majority of I/Os will be affected by the RANDOM value and not at all by the SEQUENTIAL value.

The MPE folks at HP chose the default RANDOM value of 16 because the HP7933 and HP7935 disc drives have a buffer size of 16 sectors. But you will need an HP7933/5 on the same GIC as another in-use device before you will ever use this buffer. In actuality, this buffer doesn't seem to have much affect unless you have two or more HP7933/5s on a GIC, these discs have firmware revision levels of 5.1 or greater, you have a version of MPE that supports rotational position sensing (most releases after and including MPE V/E), and you RPS is enabled on all of those drives. If you meet all of these requirements, a RANDOM value of 16 is probably the right value for your system.

However, if you don't meet all of these requirements, chances are that your system will perform better with a higher value for RANDOM. The bigger the value, the more will be read each time, and, assuming there is locality to your reads, the less you will have to actually go to disc. Furthermore, if we leave the value at 16, the number of domains tends to get very large, often over 2500. This causes extra work for the cache manager, since every time someone does a read, the manager must first see if the record is already in one of the 2500 domains. This suggests that the RANDOM value be increased, perhaps to 32, 64, or even 96.

But life is never that simple. Another thing to consider is the setting of BLOCKONWRITE. Let's assume that BLOCKONWRITE, the "switch" next to the two "dials", is set to NO. In this mode, when we issue an FWRITE or an FWRITEDIR, control returns to our program when the cache domain is updated, not when the actual disc I/O completes. The cache manager does, however, request an I/O at that time to flush that cache domain to disc. If we attempt to write to that same cache domain again before the I/O to disc actually completes, we will then get blocked and control will not return to our program until the following occur: the first I/O completes, our process gets launched again, and the cache domain is updated with our write request. (It is for this reason that some people say that we want to minimize the WRITE HIT statistic. This is only partially true; we want to minimize write hits to domains that are being written out at that moment. Write hits to "clean" domains are a good thing. Unfortunately, the cache statistics do not separate the good hits from the bad.)

Anyway, this seems to suggest that in order to minimize write hits to "dirty" cache domains, the RANDOM value should be set low, so only a smaller chunk of the file is "dirty" at any time. This directly conflicts with the advice of the first suggestion. "What now?", you ask? Well, this time we are in luck. The SHOWCACHE command tells us how effective caching is being and the CACHECONTROL command lets us tweak these parameters while the system is up and running. What we can do is STOPCACHE and STARTCACHE to zero out all of the statistics, issue our CACHECONTROL commands to adjust RANDOM, SEQUENTIAL, and BLOCKONWRITE, wait an hour or so, and then issue a SHOWCACHE command to see how we did. Try to find values that maximize "percent of user I/Os eliminated" and after that, minimize the "data overhead".

Let's focus on the BLOCKONWRITE parameter a bit more. If it is OFF, the system allegedly will run faster since our application programs will not have to wait for the disc outputs to complete. But we pay a price: unless we have asked for the file system to use the Serial Write Queue for the file, we will not know in which order our outputs will be flushed to disc. If the system should fail before all of the "dirty" cache domains have been flushed, we would have a higher likelihood of database corruption than we would have had with caching off or BLOCKONWRITE ON, assuming that we are not using the Serial Write Queue. How do you ask the file system to use the Serial Write Queue? For normal files, issue a call to FSETMODE. But for databases, there is no direct way to do this. IMAGE will issue the FSETMODE if and only if you are using Intrinsic Level Recovery (ILR) or Transaction Logging. If you use neither and have BLOCKONWRITE OFF, be aware of this new vulnerability.

It might be interesting to note some results we obtained on a machine with five megabytes. We ran ADAGER's DETPACK program in a stand-alone environment over and over, varying these parameters each time. This program runs through several IMAGE datasets and updates just about every record it looks at. Although this is a very different environment to one with 80 on-line users, it is similar to many shops' nighttime runs, with single-threaded report and update programs running. We found that the higher the value for RANDOM, the shorter the run times. And, much to our surprise, we had shorter run times with BLOCKONWRITE set to YES! This was probably due to the frequency of writing to "dirty" cache domains as described above. Hence, our best times were obtained with RANDOM = 96 and BLOCKONWRITE = ON. These were about 10% faster than with RANDOM = 8 and BLOCKONWRITE = OFF.

Extents

When we build a normal file, we can specify how many separate areas of disc we would like the file to be broken up into. We can also ask for these areas to be initially allocated or allocated the first time that a record in that area is referenced. IMAGE still uses its original algorithm for determining the number of extents that datasets should use. The algorithm always asks for the largest number of extents, 32, for medium size datasets or larger, and all extents are always initially allocated. This was fine in the old days; the larger the number of extents, the smaller each extent will need to be, and the easier it will be to find areas of disc that are the proper size.

But since the advent of caching, the number of extents can have a more important impact. (Now come the complications promised above.) When we do a random read to disc, the caching manager will read into a cache domain a chunk of the file as large as the larger of the RANDOM value and the number of words we requested, but not past the end of the extent. Hence, if a file has 32 extents in it, there are 31 brick walls built into the file, around which caching must work. We can improve the caching efficiency of nonIMAGE files, therefore, by lowering the number of extents in a file, to as low as one if enough contiguous free disc space is available. Larger numbers of extents are only justified when the file will grow over time and we want to minimize wasted disc space, or when disc is fragmented so that we cannot obtain large enough extents. (As an extra bonus, on tape drives newer than the HP7970, the fewer the number of extents that a file has, the quicker that file will STORE.)

For IMAGE datasets there is not much we can do until the algorithm used by IMAGE to build datasets is changed. One option, though, is to use ADAGER to create the datasets. Whenever an ADAGER function needs to create a new dataset, it uses an algorithm that will attempt to lower the number of extents in the dataset, without producing extents that are too large.

BLOCKMAX and BUFFSPECS

These are two tuning parameters that IMAGE allows us to set that are often ignored. Here again, it was safe to ignore them in the past, since the defaults were good values for the smaller system. But if we leave things alone now, we may not be happy with the results. For example, if you have a report program that runs stand-alone for three hours every night, IMAGE will be polite and use only 8,000 bytes of memory by default, even though megabytes are sitting unused. Caching minimizes this effect, since it will use all the extra memory it can find, but we would still be better off instructing IMAGE to be more aggressive in its use of memory resources.

The BLOCKMAX parameter is specified in the Schema. This specifies the largest size that we will accept for the block size for this database. Remember, IMAGE will decide what the blocking factor for each dataset will be. Then, the largest block size among all of the datasets will become the size of all of the buffers whenever the database is accessed. The default value is 512 words. We can increase this number up to 2048 words.

Look closely at the blocking factor for each dataset at the bottom of the DBSCHEMA listing. Sometimes IMAGE has a choice of a blocking factor that saves a bit of disc space, but does not pack as many entries into each block as will fit. IMAGE will let us override this choice. We can specify our own blocking factor for a dataset by putting the desired number in parentheses between the capacity and the semicolon in the schema. E.g.

```
Capacity: 40000 (20);
```

would request a capacity of 40,000 entries and 20 entries per block.

The BUFFSPECS for a database can be interrogated by using the SHOW command in DBUTIL. The number of buffers is dependent upon the number of times the database is opened. The default BUFFSPECS are:

```
BUFFSPECS = 8(1/2),9(3/4),10(5/6) ... 17(19/120)
```

which means that with one or two users in the database, 8 buffers will be allocated, three or four users will get 9 buffers, and so on until 19 or more users open the database, at which time 17 buffers will get allocated. To go back to our example, that stand-alone report program would run with eight buffers, since when it runs, it runs alone.

If you have many databases on your system, then all of this may be all right. But if you have one or two central databases, you might be better off giving more memory to these database to work with. In other words, change the BLOCKMAX and the BUFFSPECS. If you have ADAGER, the best way to change the BLOCKMAX is with the REBLOCK function.

This program will help you select a proper value, giving you a little lesson on the complexities of blocking factors along the way, and then will reblock your database. Choose a large value for a central database. It is bad enough that all users and all datasets must share the same set of buffers. The least we can do is make these buffers as large and as plentiful as is reasonable.

Now that we've adjusted the size of each buffer, we need to decide how many buffers we want to allocate. A good rule seems to be that the number of buffers should not be dependent upon the number of users. The number should be set to nine plus the largest number of paths going to any one detail dataset. If a detail dataset has five paths from master datasets, for example, set the BUFFSPECS as follows:

```
BUFFSPECS = 14(1/120)
```

To do this, use the SET command of DBUTIL when no one has the database opened.

Cabling

Larger machines with two or three IMBs (InterModule Bus) and four or more GICs (General Interface Channel) can be cabled in many different ways. How things are cabled can have a greater impact on performance than just about any other factor. Do not assume that HP cabled you up optimally! Your discs should be spread over all of the IMBs and GICs that they can. If the system is accessing two files, performance will be best if the files reside on discs hooked to two different IMBs, next best if they are on two differ-

ent GICs on the same IMB, next if they are on two different master drives on the same GIC, next if they are on a slave drive and its corresponding master drive, and worst if they are on the same disc.

Review your own IMB/GIC/disc drive configuration and fix things if they are not optimal. Then, with a tool such as FILERPT in the Contributed Library, identify the ten or so most used files. (We have contributed an updated version of FILERPT to the Contributed Library that will support both MPE IV and MPE V log formats.) Spread these files out among the discs according to the rules of the previous paragraph. We have found that disc file placement can have a major impact on performance, as long as we are concerned with the placement of the correct files.

Data Distribution

As databases get larger and larger, the way in which the entries are loaded in the database becomes ever more important in terms of performance. But since these large databases take so long to reload, many sites cannot afford the time it takes to reorganize their databases. This can have a dramatic affect on performance though. For example, a detail dataset with a blocking factor of ten has had so many puts and deletes over time, that its free entry chain points all over the place. When ten line items are placed on the same chain into this detail, they end up in ten different blocks. If this detail were packed along this path, all ten entries could reside in the same block. Instead of ten I/Os to read the chain, it will then just take one! A major improvement.

Programs such as DBLOADNG in the Contributed Library or HowMessy from Robelle Consulting will tell how badly your datasets need repacking. Then if called for, we can either take the time to reload them or use ADAGER's DETPACK and MASTPACK.

Conclusion

Just when we think we understand the rules and get our systems tuned accordingly, something comes along that changes the rules. Ignoring the changes can waste time and money. Tuning is a never-ending process that usually pays for itself over and over again. Give some thought to these guidelines, take a look at your system, and experiment. And when you discover something, share it!

Biography

Steve Cooper is a member of the Adager Research and Development Laboratory. He has had nine years of experience on the HP3000. He holds a BA degree in Computer Science from the University of California, San Diego and an MBA degree from the University of California, Los Angeles.