PERFORMANCE OPTIMIZATION FOR IMAGE

by Bernadette Reiter
S.E. Neely Englewood

Many discussions in earlier days of improving system performance were slanted toward CPU utilization optimization. Today however, in the age of faster CPU speed, cheaper memory, and more memory available on CPU's (including the range of so called mini computers), optimization strategies have had to take a different approach. Currently on a one to two megabyte systems, memory constraints no longer are the prime contributor of system degradation. IO constraints are now a potential bottle-neck on most CPU'S.

The HP3000 allows for IO optimization strategies when dealing with files handled by the file system, using methods such as multi record IO, NOBUF IO, and NO WAIT IO. These few file system parameter options allow for user IO optimization strategies in their applications (dealing strictly with sequential or direct access methods). These optimization techniques became unapplicable with software technology moving toward structured data organization spanning beyond sequential and simple direct access file organization. This thrust toward data organization launched technology into the era of DATABASE MANAGEMENT.

Image being a friendly DATABASE MANAGER, allowing for extremely involved and sophisticated interrelationships of data, lets the user inadvertently misuse IMAGE'S power to the degradation of preformance. This paper is designed to give the user an insight into the different aspects of IMAGE, as well optimization techniques to take advantage of IMAGE's sophistication and capabilities of outpreforming other data structures. The suggestions made are based on using IMAGE in interactive environments, performance analysis of IMAGE processes, internal structure information, and the IMAGE reference manual.

A.) OPTIMIZATION FOR MASTER DATA SETS

A MASTER DATA SET uses a hashing algorithm as its method of of finding the address of a particular key item within the master file. A hash is a calculation imposed on the key itself and results in giving a relative record number within the file. From an efficiency standpoint, using a hashing algorithm has always been considered a tremendously effective means of retrieving data in strictly random fashion.

No hashing algorithm is perfect. Based on the algorithm used and the nature of the data, several key items could hash to the same location. These collisions are chained together in a linear linked list by IMAGE. The key to the success of a hashing algorithm is to optimize for an even distribution of the keys, over the allocated file space (the capacity), minimizing the amount of collisions.

IMAGE determines the hashing algorithm used on a particular master by using a function of the modulous of the capacity chosen by the user. To optimize for an even distribution of keys, studies have shown that the modulous of a prime number or a multiple of prime number is typically a good algorithm. It is for the above reason that the IMAGE reference manual recommends that a prime or a multiple of prime be chosen as the capacity of a master.

The key values typically are extremely random. This makes is difficult to find an algorithm which will gaurantee a unique address for every key. There are going to be incidences where a collision will occurr. Image will take that collision and put it into the first available free space. If the MASTER is getting filled to above 80% of capacity, IMAGE will potentially have to do several IO's to find the first available free record into which to place a synonym, and will have to do additional IO's to retrieve those synonyms (should the synonym be in a different block than its predecesor in the chain). Therefore, to optimize, for IO, for the retrieval and placement of synonyms, it helps considerably to keep at least a 20% pad of free space in your MASTER DATA SETS.

B.) OPTIMIZATION FOR DETAIL DATA SETS

1.) Detail data sets are kept in a logical order by doubley linked lists (with the head and tail of the chain stored with the corresponding master entry). Due to the chained entry sequence being reflected in a logical oreder (by forward and backward chains) rather than a physical order, the entries of a particular chain could all reside in different blocks. This would require an IO per record. Therefore, if chains are long, and the detail is heavily accessed, it is to the users advantage, from an IO preformance standpoint, to periodically reload the database, putting the detail in primary key sequence. This means that when IMAGE rebuilds the detail, he will physically put the entries together, as well as logically. In this physical/logical order, when reading down a particular chain , chances are that the next X records to be read in the chain will already be in the block in memory, minimizing the amount of IO necessary to read down a detail chain.

2.) In a DETAIL DATA SET, IMAGE also allows for the entries to be chained in a sorted fashion by a predefined sort key. Since the data in detail is kept in a doubley linked list, there is some overhead involved in maintaining a chain in a sorted sequence. When adding a particular entry to a detail set, where sorted chains have been specified, IMAGE could potentially read the entire chain in order to determine where the entry to be added belongs within the sort sequence. If the chains are long this could require several IO's to determine where in the sort sequence the new entry belongs. From a performance standpoint use sort chains with care, when designing your data-base, they incurr more IO's in adds and deletes than a regular chain.

C.) IO OPTIMIZATION WHEN DESIGNING A DATA-BASE

When designing a data-base structure, you may want to think of some IO performance considerations. IMAGE allows you the flexibility of having 1 to 16 MASTER DATA SETS pointing to a DETAIL DATA SET, or 1to 16 DETAIL DATA SETS pointed to by a MASTER DATA SET. Based on how the data-base structure is defined, it will have different IO implication. When adding or deleting entries from a DETAIL DATA SET, that detail change gets reflected in the corresponding master entry (in chain entry counts and potentially head and tail pointer information). This means that to reflect the change in a detail pointed to by X number of master sets the following amount of IO will occur:

(X * number of masters pointing to the detail) + number of IO's to reflect chain change in detail

X= (Z * read of master + write of master)

Z= number of IO's required to find a master entry if it is a synonym.

Therefore, when designing a data-base structure keep in mind the number of IO's necessary to reflect a change in a detail.

D.) RUNNING DEFERRED FOR BATCH UPDATING

When making a change to a record in an IMAGE data-base, IMAGE will immediately post that record to DISC. If you change X number of records in a block, then the block is posted X number of times. IMAGE does allow you to, in an exclusive access mode, to do a DBCONTROL with a deferred mode. This means that the block is only posted when it is full, or the buffer is needed, rather than every time a record is changed, added, or deleted in the block. This option is extremely beneficial in data processing environments where on line transactions are accumulated all day long, to be posted against the data-base, in a batch environment at night. You can decrease substantially the amount of IO done by the batch update program by running deferred , making run time for that process significantly less, if it needs to post a large number of transactions.

* CAUTION If the system crashes while running deffered you can potentially lose much more data since each transaction wasn't posted right away. However, in the above environment chances are that if you crash you will have to restore your data-base and restart the batch process again.

E.) PROGRAMMING OPTIMIZATION CONSIDERATION

1) DELETE and ADD vs. UPDATE against a detail.
It requires less IO to do an update to a detail record (as long as the sort or search item is not altered) than a delete and an add. An update only requires posting the detail record since the chain information didn't change. A delete and an add however requires reflecting the deleted record change in the corresponding masters as well as reflecting the absence of the record in the detail chain. Now to re-add that record requires doing the IO against the master again and reflecting the add in the detail also. For IO optimization, when doing a modification of a detail record, use the update function versus the delete and add to reflect the detail change.

2) LOCKING strategies.
Use the locking option which will allow the most amount of IO to go against the data base. If the transaction to be locked around is lengthy in time, consider record level locking. If the transaction is short consider data-set or data-base locking (from a CPU standpoint they are a faster lock).

3) Setting up DATA-BASE security.
IMAGE allows you to use security options at the set and individual item level. This allows for very complex security options. The more complex the security structure the more overhead is involved in interregating the users access capabilities against the data-base. Where possible allow a user to have write access to the entire set, therby IMAGE doesn't check the individual items in the set to determine the read/write access capabilities.

4) LIST "*" option on DBGETS.
When doing successive reads against a particular DATA-SET, read the first record with the list of items you are interested in retrieving. Then make all successive calls with the "*" as the list parameter. This allows IMAGE to bypass security for the subsequent reads on the premise that if you passed security on the first record, you will also pass security on subsequent calls. This can cut down significant processing time since the security checking overhead per read has been reduced to only having to be done on the first record.

F.) DISTRIBUTED DATA BASE AND REMOTE DATA BASE ACCESS

The new surge in data processing is to ditribute data bases and processes across several CPU's and be able to access the data bases on all the CPU's in the network. In environments such as these, IO optimization will also give you the most significant performance improvements. However in the distributed environment, optimization for IO over the communications lines between CPU's, especially over phone networks, will give you the most gains.

In a DS3000 environment we allow the user 3 types of remote database access:

1) Remote command access (user needs knowledge of DS)
   a) user logs on to local system
   b) user logs on the remote system with remote HELLO
   c) commands are executed on remote system
   d) IO communication requirements- all commands, all data, all output from remote command $stdlist

2) Remote file access
   a) user in DBUTIL defines where remote data base is, along with remote log on
   b) when IMAGE issues open against the remote data base definition, IMAGE establishes communication with remote system (if dial connection user still has to dial and respond to dial message on console, however, user doesn't need remote log on).
   c) processes are executing on the local system
   d) IMAGE commands, any records retrieved by IMAGE are sent over communications lines. However, any reports generated from that data is generated on local system having no communications demands.

3) Program to program communications (PTOP)
   a) master program runs on local system
   b) master program start execution of a slave program on remote system
   c) user PTOP programs controls data being transferred across communication lines.

Guidelines to choosing remote access techniques:

1) The most effecient for IO and the most difficult to implement is PTOP. The user has total control over the transmission of data. Performance-wise significant gains can be made using this technique.

2) Whether to use remote command or remote file access?
   a) If the report generated by the retrieval of remote data is greater than the amount of data necessary to retrieve the data, use remote data base access to minimize the data communication IO (requires no knowledge of ds).

   b) If the report generated is less than the amount of data required to retrieve the data (from a command standpoint) then use remote command access. (this does require that the user knows how to use DS to establish communication with the remote CPU)

CONCLUSION:

IMAGE does allow for several optimization techniques. The afore mentioned suggestions primarily address optimization for IO on the premise that the most significant gains in performance can be made by improving IO strategies. When designing the data-base optimize for DISC IO. When accessing remote data-base's also optimize for data communication IO.