The Three Bears of IMAGE

Fred White

Adager
Apartado 248
Antigua
Guatemala

The Three Bears of IMAGE

Fred White

Adager
Apartado 248
Antigua
Guatemala

INTRODUCTION

Software designers, whatever the product, hopefully provide a variety of features which they believe are important to user acceptance of the product.

In many cases, the implementation of a feature is optimized for the use envisioned by the implementers. Conversely, the implementation may be sub-optimized for use other than as intended.

Traditionally, product manuals seldom (if ever) include motivational discussions of product features so that users are not warned about sub-optimal uses of the product features.

In some cases the sub-optimal use of features may have no noticeable effect on throughput or response time. In others the effect may be disastrous.

Two features of IMAGE/3000 whose sub-optimal use can be disastrous are "integer keys" and "sorted paths". For the purposes of this paper, these two represent, respectively, PAPA BEAR, and MAMA BEAR. Each is a very deep pitfall and extricating yourself from either can be very expensive.

BABY BEAR is represented by "paths", another feature whose misuse, while normally not disastrous, may have a negative effect on response time and/or throughput. A discussion of the use of paths is included to justify the title and because it should be of general interest.

BACKGROUND

"Detail" datasets were intended as repositories for records having generally no unique identifying characteristic (field value) and for which the primary access method would be sequential.

Each detail dataset starts as an empty file of a size large enough to meet its capacity requirements. IMAGE keeps track of the highest record number (initially zero) assigned to any record of the dataset, as a result of a DBPUT. This serves as a "high-water-mark" and is analogous to the file system's EOF (end-of-file).

Stated another way, a detail dataset is similar to an ordinary MPE file in that each new record is assigned an address calculated by adding 1 to the high-water-mark. When this is done to an MPE file, MPE adds 1 to the current EOF pointer and appends the new record.

IMAGE, however, provides for the automatic re-use of space which results whenever a record is deleted. It keeps track of the reusable space by means of a push-down stack. It maintains a pointer to the newest member of this stack and each member points to an older member deeper in the stack. DBPUT always (for detail datasets) assigns the address of the newest member of this "delete chain" to the new record being "put" unless the "delete chain" is empty, in which case DBPUT increments the high-water-mark and assigns the new value of the high-water-mark as the address of the new record.

"Master" datasets were intended as repositories for records having a unique identifying characteristic (field value) and for which the primary retrieval technique would be dependent on this unique value. The IMAGE manual refers to this as calculated access.

After much discussion it was decided that two distinct "flavors" of calculated access be provided: one over which the user would have (essentially) no control and which would calculate record addresses via a hashing algorithm whose objective was to achieve a nearly uniform distribution of addresses in the face of random or non-random key values, and another over which the user would have (essentially) absolute control in that the low-order 31 bits of the key value would determine the desired address (modulo the capacity).

For those of you familiar with "direct access" methods, this latter capability can be viewed as a generalized "direct access" method. Generalized in the sense that addresses greater than the capacity are not considered invalid, but, instead, are reduced modulo the capacity. IMAGE does this by (a) subtracting 1 from the 31 bit key value, (b) dividing the result by the capacity to obtain the positive remainder and (c) adding 1 to this remainder.

It was further decided that this "direct access" technique would be used whenever the search field was defined as an item of type I, J, K or R (all of which are of binary format) while "hashing" would be used whenever the search file was defined as an item of type U, X, Z or P (none of which are of binary format).

For all of the "direct access" type keys, IMAGE treats the low-order (right most) 31 bits as a positive integer in calculating the record address. For this reason, these keys have been dubbed "integer" keys as a way to distinguish them from "hashed keys".

Space allocation from master datasets is completely different from that described for detail datasets. In effect, a master dataset starts out with the high-water-mark equal to the capacity and DBPUT never appends records. Instead, the record space starts out as entirely re-usable. No "delete chain" is maintained for master datasets. Instead, IMAGE relies on a "bit map" which is maintained at the front of each block of each dataset. For master datasets, DBPUT calculates the primary address (as described above) and, after verifying that the key value is unique, attempts to place the new record at the primary address.

This attempt will succeed if and only if this new record has no synonyms. Otherwise, DBPUT assigns a secondary address physically near (hopefully) the primary address. It finds such a hole by means of a sequential (and cyclical) search starting with the block containing the current end of its synonym chain. In a master dataset which is not too full and where existing records are not "clustered" (i.e. nearly uniformly distributed) and where the "blocking factor" is not very small, this search might require zero, or only a few, disc reads.

This technique assigns synonyms to the same block or to neighboring blocks thus minimizing I/O during DBPUT's, DBFIND's, and keyed DBGETS.

Having covered the pertinent differences between detail and master datasets, let us proceed to a discussion of the path feature.

Under IMAGE, a path is a relationship between a master dataset and a detail dataset. The relationship is 1-to-N (where N varies from zero to 64535) in the sense that each master record is related to N records of the detail dataset and that each record of the detail dataset is related by this path to exactly one record of the master dataset.

The N detail records related to a common master record are referred to as a chain since IMAGE links them together with backward and forward pointers. One end is referred to as the "begining-of-chain" and the other is referred to as the "end-of-chain". New records are added to the "end-of-chain". IMAGE maintains a chain length count and pointers to the beginning- and end-of-chain in this common master record.

The common master serves as a locator record (via a DBFIND) to the corresponding detail chain. This is analogous to using the card catalog in a library to locate all books written by a particular author.

The fact that a detail dataset can have paths to more than one master dataset is analogous to the books in a library being referenced by other card catalogs such as Titles or Topic.

This, together with the fact that IMAGE permits master datasets to have paths to more than one detail and have more than one path to any detail, make IMAGE (along with the AUTOMATIC master feature) a very flexible 2-level network structure data base management system.

=====================================

PAPA BEAR...the INTEGER KEY pitfall

=====================================

My first live encounter with a misuse of integer keys arose in 1978.

One Friday in 1978 I received a phone call from an insurance firm in the San Francisco Bay Area. I was told that their claims application was having serious performance problems and that, in an attempt to improve the situation, they had, on the previous Friday, performed a DBUNLOAD, changed some capacities and then started a DBLOAD which did not conclude until the early hours of Tuesday morning!

They were a $100,000,000-plus company which couldn't stand the on-line response they were getting and couldn't afford losing another Monday in another vain attempt to resolve their problems.

Investigation revealed that claims information was stored in the two detail datasets with paths to a shared automatic master. The search fields for these three datasets was a double integer key whose values were all of the form YYNNNNN (shown in decimal) where YY was the two-digit representation of the year (beginning with 71) and where each year NNNNN took on the values 00001, 00002, etc. up to 30,000.

Although the application was built on IMAGE in late 1976, the earlier claims information (from 1971 thru 1976) was loaded to be available for current access. I do not recall the exact capacity of the master dataset but, for purposes of displaying the nature of the problem (especially the fact that it didn't surface until 1978) I will assume a capacity of 370,000.

Although the number of claims per year varied the illustration will also assume that each year had 30,000.

The first claim of 1971 was claim number 7100001 which, using a capacity of 370,000 IMAGE would assign a primary address of 70,001. This is because 7,100,001 is congruent to 70,001 modulo 370,000. The 30,000 claims of 1971 were thus assigned the successive addresses 70,001 through 100,000.

Similar calculations show that the claims for each year were stored in groups of successive addresses as follows:

YEAR	CLAIM NUMBERS	ASSIGNED ADDRESSES
1971	7100001-7125000	70,001-100,000
1972	7200001-7230000	170,001-200,000
1973	7300001-7330000	270,001-300,000
1974	7400001-7430000	1- 30,000
1975	7500001-7530000	100,001-130,000
1976	7600001-7630000	200,001-230,000
1977	7700001-7730000	300,001-330,000

Note that no two records had the same assigned address and thus that there were no synonyms and that all DBPUT's, DBFIND's and keyed DBGET's were very fast indeed!

Now comes 1978!!!

Unfortunately 7,800,001 is congruent to 70,001 so that the first DBPUT for 1978 creates the first synonym of the master dataset. It is, in fact, a synonym of claim 7100001. Recalling that DBPUT finds an alternate location by means of a serial search, DBPUT then searches the next 60,000 records before it finds an unused address at location 130,001! Even with a blocking factor of 50, this would require 1200 additional disc reads which would make each DBPUT up to 200 times as slow as those of previous years!!

Note that the next claim if 1978 (with claim number 7800002) is congruent to 70,002 so is a synonym of 7100002 and also lead to a serial search which ends at location 130,002! Thus each successive DBPUT results in a search of 60,000 records 59,999 of which it had inspected during the preceding DBPUT!!

PAPA BEAR had claimed another victim!! The designer of this system had unknowingly laid a trap which would snap at a mathematically predictable time, in this case 1978. After struggling with this problem for months, the user ultimately escaped from PAPA BEAR by converting to "hashed keys" (in both the database and the application modules); a very expensive conversion!

Note that the problem was not a synonym problem in the sense that synonym chains were long nor was it a "fullness" problem since the master dataset was less than 57% full when PAPA BEAR struck. The problem was due to the fact that the records were maximally clustered whereas DBPUT's space searching technique for masters is optimum only under (nearly) uniform distribution assumptions.

Note that the performance of DBFIND and DBGET was excellent since the maximum synonym chain length was 2.

Another much shallower pitfall would have been designed if, in the above example, the claim numbers had been of the form NNNNNYY with the same capacity of 370000. In this case, the performance of DBPUT's ,DBFIND's and keyed DBGET's would all degrade over time but would never reach the disastrous level of the DBPUT's of the example. In this case, the degradation would arise due to the length of synonym chains and due to local clustering.

Note that this modest pitfall could be eliminated by changing the capacity, for example, to 370010.

Note however that this problem would still arise if the capacity were merely changed, for example, to 370001.

It should be apparent by now that designers may avoid the clutches of PAPA BEAR by carefully (mathematically) inspecting the consequences of the values of their choice of master dataset capacity.

====================================

MAMA BEAR...the SORTED PATH pitfall

====================================

My first live encounter with a misuse of sorted paths arose in 1975.

The facts surrounding this incident were told to me by Jonathan Bale who was still on the IMAGE project. Neither one of us remembers the exact numeric details so I have used poetic license by making up numbers which seem to be reasonably close to the actual ones involved in the incident.

The user had created a database containing one automatic master data set and one detail dataset related by a 2-character key and where the resulting path was sorted by some long forgotten field(s).

The user had written a program which read a record from an input file, added two blank characters to serve as the search field and then performed a DBPUT to the detail dataset. This was repeated for all records of the input file.

At the time Jon received a phone call, the tape had not moved for around 10 hours and the program had already been running(?) for at least 30 hours.

On inquiry, Jon learned that the input file contained over 40,000 80-character records and that the user was using IMAGE to sort these records!

This is an extreme example of a sub-optimal use of sorted paths. To see this, it is important to know that when adding a new record to a sorted path, DBPUT starts its search for the appropriate point of insertion at the end of the chain and then searches the chain backward until it encounters a record whose sort field(s) value is not greater than that of the record being added.

For input records whose sort field values are randomly ordered, the expected number of records to be searched is one-half of the length of the chain. When the chain is 20 records long, the search will cover 10 records on the average. When it becomes 30,000 long, the search will cover 15,000 records on the average!

For a file with 40,000 records to be sorted into one chain the expected number of reads to cover all searches is approximately 400 million with the last record alone expected to take 20,000!

The blocking factor of the input tape was 200. No wonder the tape hadn't moved for 10 hours!!

To avoid the clutches of MAMA BEAR, avoid using sorted paths if the chains are very dynamic or very long. The more dynamic they are, the shorter they should be, and, the longer they are ,the less dynamic they should be. The term dynamic is used here to refer to the relative frequency with which entries are added and deleted.

Contrary to the many warnings you may read against using sorted paths, there are occasions when their use is infinitely better than any other option.

HP's Corporate Parts Center in Mountain View used a sorted path in its back-order dataset. The search field was the part number and the sort field was a priority assigned by order-entry personnel in such a manner that the highest priority back-orders were at the front of the chain.

When new parts were received, a clerk at the receiving dock would enter the part-number and quantity at a terminal. The program would then perform a DBFIND with that part-number on the back-order dataset followed by a sequence of chained reads. For each record in the chain, a packing slip would be printed showing the quantity and destination and the record was then deleted. This process was repeated until the chain was empty or all received parts were accounted for. In the former case, an additional shipping slip was printed so that the remaining parts would be delivered to inventory.

This on-line technique eliminated unnecessary shipment of parts to inventory, minimized parts handling, facilitated shipments and minimized errors.

Even though the chains were sorted, most back order chains were either empty or had only a few entries so that adding new entries was never really slow.

Another, even more outstanding, use is available to order processing systems where each sub-system (or part) in a master dataset is related to its components in a detail dataset by the part number of the subsystem (or part). The component numbers in each detail record are also present as part numbers in the master dataset and each of these in turn may be related to other components in the detail dataset. In other words the "parent-child" relation implicit in the concept of "component" is recursive.

The detail dataset here is related to the master via a parent-number field and is sorted by component-number. The fields of the record are ordered to take advantage of IMAGE's extended sort to include component option and quantity.

This "clever" design together with a recursive procedure enables the application to provide on-line, single- or multi-level, fully indented, bill-of-material explosions with the components at each level in component-number and component-option order. No sorting is required and the performance of the explosion is limited by terminal speed.

Although many people may recommend that you avoid sorted paths, try implementing either of these applications without them. Response time would be somewhere between bad and disastrous!

There really is a place for network databases and sorted paths.

==================================

BABY BEAR...a discussion of PATHS

==================================

As illustrated in the examples, sorted paths can provide benefits critical to some applications.

For instance, the application may not have to search the entire chain or it may simply be easier to program and/or marvelously faster as with the bill-of-material example mentioned above.

The overhead for paths mentioned in reference to DBPUT's and DBDELETE's is also proportional to their frequency of use. In other words, this overhead is less of a consideration for relatively static datasets than for relatively dynamic datasets. So additional paths for static datasets have less DBPUT and DBDELETE performance costs than on dynamic datasets.

S U M M A R Y

In general, the rule for a path is: "when in doubt, leave it out". If leaving it out proves to be a mistake, you can be sure that someone will call it to your attention and then (with the help of ADAGER) you may add it without impact on any application module. On the other hand, if providing it proves to be of little benefit, no one will tell you and removing it will undoubtedly have dire consequences on some application module(s).

======================
Biographical Sketch
======================

Fred White began his Programming career in 1957 as a scientific programmer. He programmed for 8 years in various user environments expanding into commercial, management system and system programming areas primarily on IBM and Burroughs computers.

On August 1,1965, Fred took a programming position with IBM where he worked on a multiprocessing text processing system which ran on modified 1440's or 1460's. During his 4 years with IBM he furthered his knowledge of file sharing and concurrency control.

On August 1, 1969, Fred joined Hewlett-Packard as Project Manager of what later became the MPE File system. He designed the account/group/user structure, capability classes, file codes and other features of that system.

Fred is best known in the user community for his involvement with IMAGE/3000 where he served as Designer, Programmer, and Project Manager.

Fred left HP and has been working with Adager since November 1,1981.

Note: Sorry about being so wordy. 26 years are hard to encapsule (at least for me).