IMAGE: An Empirical Study

B. David Cathell
Hewlett-Packard

Background

Last April, Jorge Guerrero addressed the Performance Specialist Class that I was attending. He made a rather bold assertion that "There is no evidence that a prime capacity for an IMAGE master data set gives a better distribution than a non-prime capacity." In light of the fact that we had all been trained to believe that a prime capacity is decidedly better, we students were shocked to hear Jorge's contention. I began design and implemenation of a program to read any data base and test the key values contained in a specified data set using various capacities. Although the initial goal was to prove or disprove Jorge's assertion, the program was also intended to be a tool to evaluate potential capacities should the assertion prove to be true.

Image Theory

The following section deals with the theory upon which the IMAGE design is based. Readers who are already familiar with this theory may wish to jump forward to the discussion of the program itself.

IMAGE, like any data base system is an organized collection of data. The method of organization is intended to speed the retrieval of particular data that a user requires. There are many methods of organization but the designers of IMAGE chose two types of sets of data which we call master data sets and detail data sets.

Master data sets contain key data by which the rest of the data is retrieved. Names and identifying numbers (such as social security number) are common keys with which we are probably all familiar.

Detail data sets usually contain the bulk data but are organized with pointers to maintain linked lists of information with common key (called search items) values. Data for a particular person, for example, can be obtained by finding the head of the chain for that person's name and following the chain.

Because accessing the data in an IMAGE data base usually requires initially accessing a master data set, this paper will only deal with questions pertaining to masters.

When a user attempts to access a master data set, he or she provides a key value. Somehow, IMAGE must be able to transform that key value into a unique address where the desired record (data entry) resides. The easiest way to do this is to assign a unique number to every possible key value and have a correspondingly numbered data entry location reserved for it. Unfortunately, even a short data item may have a huge number of possible values; for example, the number of possible file names in MPE is over two trillion. Although, your HP sales representative might like to sell you another disc drive, two trillion records would require a good sized building full of disc drives.

Therefore, IMAGE has to transform that key value to some reasonable number of possible record locations. Whatever method is used must be repeatable. That is, once the user places information in some location, a subsequent request for that same information must be directed to the same location. Secondly, the method must produce a reasonably good distribution of record numbers. If the transformation algorithm produces the same record number from two different keys (the keys are called synonyms), we have a problem in that only one data entry can occupy a record location, but two entries wish to reside there.

Thus, in addition to selecting a good transformation algorithm, the designers of IMAGE also had to contend with a method of handling synonyms. The method chosen allows a synonym to reside in a location other than the one determined by the transformation algorithm. Such a synonym is designated a secondary entry, whereas an entry which resides in the correct location is termed the primary entry. The primary and secondary entries which are synonyms of each other are linked together in a linked list, with the head of the chain in the primary location. In order to find a secondary entry, IMAGE must first find the primary entry and follow the synonym chain until it finds the desired entry.

Importance

With this brief discussion of the theory of IMAGE, one now has to ask the question "Why would anyone care if the capacity of a master data set is prime or not?" Actually, one has to go back to the transformation algorithm mentioned above.

IMAGE actually has two parts to the transformation. The first part applies only to byte type keys. It is called the Bale-Estes-White hashing algorithm and attempts to fold the entire key into a two word value. (The exact algorithm is given below.) The second part applies to all keys and consists of simple modulus arithmetic whereby the right most two words of the key (or two word hash) are divided by the capacity. The remainder plus one is the record location to be used for that key. It has been asserted (although I don't know who made the original assertion) that a prime divisor produces a better distribution of record numbers.

It should be apparent that a poor distribution would result in an excessive number of synonyms and thus a greater number of secondary entries. This results in additional overhead in accessing the master data set because IMAGE must more frequently follow a synonym chain in order to produce the desired data entry.

It can also cause additional overhead in adding or deleting entries in the master data set. This condition is termed migrating secondaries and results from a rule that IMAGE enforces; a data entry has first priority over its home (calculated) location. If in adding a new data entry, IMAGE discovers that a secondary entry (a synonym of some other key) already resides in the this data entry's home location, the secondary entry must be moved (migrated) first to some other empty location. Then the new primary key may be placed in its rightful home location.

The second case of migrating secondary is caused by the deletion of the primary entry which has a synonym chain. The second synonym in the chain is moved (migrated) to the primary location.

It should be obvious that it is very desirable to minimize the number of synonyms in order to avoid excessive overhead in both accessing and changing a master data set. If choosing a capacity which is a prime number (or any other specially computed number) produces significantly fewer synonyms, then it is certainly worth the effort.

The DBCAPCK program

In an effort to answer the question of prime capacities, I decided that one could argue the theory of transformation algorithms forever, but the proof is contained in real data bases with real user data. Although my primary objective was to prove or disprove the prime capacity assertion, I also intended that the design of my program would be flexible enough so that if the assertion proved to be true, the program could be used to determine good capacities for a given data set.

The first step was to obtain the IMAGE transformation algorithm and adapt it to my program. I had already decided to use PASCAL for the program and had to either interface to the actual SPL code or simulate it. Perhaps it was intellectual curiousity or perhaps some personality flaw, but I decided to simulate. For those who wish to share this experience, the hashing algorithm follows:

IMAGE Hashing Algorithm

1. Obtain the left most two words (4 bytes) of the key.
2. If the key length (in words) is an odd number, shift the two word value to the right by 16 bits, introducing leading zeros.
3. Shift the two words left by 1 bit, with an end around carry of the sign bit.
4. If all the words in the key have been used in the hashing; go to step 9.
5. Get the next two words in the key (beginning at the right) and divide (unsigned) by 31 and add 1 to the remainder.
6. Use the above result as a shift count and shift the hash accumulated thus far, to the left with an end around carry of the sign bit.
7. Perform an unsigned addition of these next two words in the key with the just shifted accumulated hash (saved as the new intermediate hash value.
8. Go back to step 4.
9. Shift the accumulated hash to the right by 1 bit, introducing a leading zero bit.

I can assure you that I had no confidence that I accurately translated the algorithm (written in SPL but actually every line is part of an ASSEMBLE statement). Therefore, the program contains a check, comparing the record number computed by the program versus the actual record number of the primary key (aha! Finally a use for the mode 8 DBGET) when the user scans a data set specifying the original capacity.

Modulus Calculation

The actual calculation of the relative record number is very straight forward; one only needs to perform unsigned division of the hash (byte key) or right most two words by the capacity, using the remainder plus one as the relative record number. Or so I thought!

After the program was beginning to run, I visited one of my accounts who is a heavy data base user. They kindly consented to test out my program on one of their data bases. All was fine until we examined a data set that had a capacity of 150,000 entries (since it was non-prime, it looked like a good candidate to analyze). The program began issuing an error message indicating that the relative record number computed by the program differed from the data base. There were so many miscalculations that it couldn't have been an end case; in fact, we had the impression that every entry was miscalculated. But a pattern emerged, every calculated value was exactly one less than the value IMAGE had generated. Later testing revealed that IMAGE has a bug in the modulus calulation (I even suspected PASCAL) when the capacity is greater than 65535. Of course, the "bug" can never be fixed because of the requirement for repeatibility of the transformation algorithm.

General program description

When the program is run, it initially prompts the user for a data base name and password. It then opens the data base in mode 5 (shared, read only) and using DBINFO lists the master data sets with information about each. The user is prompted for a data set, a proposed capacity and a proposed blocking factor. The program reads the indicated data set, calculates the hash (if a byte key) for each data entry, stores it in an extra data segment, calculates the relative record number and records it in a statistics file. Once all the data entries have been read, it counts the number of keys which had no synonyms, the number with exactly one synonym and so on. It then produces a table of these results on the terminal display. The user may also request a graph of the distribution of entries. The user may then select to repeat with the same data set changing the proposed capacity. In the second pass of a data set, the program uses the hash from the extra data segment rather than re-reading the data set.

The following is a sample of the dialogue from the program.

DBCAPCK Version 1.0 (c) Hewlett-Packard Company, Inc.

```
                           Data Base = dummy
                            Password = <cr>

No. Name            Type  BF     Entries   Capacity
 1  PEOPLE           M    14         166        200
 2  INT              M    68          36        200
 3  ASCINUM          M    38          25        200
                     Data set number = 1
                         Search item = NAME
                           Item type = X
                Capacity (      200) = <cr>
              Next larger prime? (N) - <cr>
                Blocking factor (14) = <cr>
                         Entry count = 166 (83.0%)
            Entries with  0 synonyms -         62   37.3%
            Entries with  1 synonyms -         46   27.7%
            Entries with  2 synonyms -         45   27.1%
            Entries with  3 synonyms -          8    4.8%
            Entries with  4 synonyms -          5    3.0%
                Overflow block count = 3
                   Total block count = 15
         Would you like a graph? (N) - <cr>
                         Repeat? (Y) - n
```

Most of the above is pretty straight forward, but a few items should be discussed. Why is blocking factor important? In all of the previous discussion, we simplified the explanation so that it would appear that each data entry has its own physical location on the disc. In fact, data entries reside in blocks whose blocksize is determined by the BLOCKMAX control option in DBSCHEMA. The number of data entries that will fit is a function of the size of the data entries and the number of paths to associated detail data sets. As we discussed earlier, there is overhead associated with synonym chains, but the amount of overhead is much greater when the synonym chains span multiple blocks. The time required to perform the physical I/O to obtain a second (or even third) block in following a chain is considerably greater than if the chain is contained only within one block.

It is also for these reasons that the Overflow Block Count is reported. This statistic represents how many blocks in the data set could not contain all the data entries which have their home location in that block. In each of these blocks there will be at least one synonym chain which spans into another block. In many cases, this statistic may be as important as the actual synonym counts themselves.

Are prime capacities better?

I have used this program to examine many data bases with many types of keys and I cannot find any benefit to prime capacities. I realize that many of us find this hard to accept and may question the conclusion of one individual.

The following is a compilation of the statistics reported by DBCAPCK making numerous passes through a data set containing a six-byte key (TRACKER PICS id consisting of three character month abbreviation followed by three ASCII digits). The number of entries was 1243 with a blocking factor of 20. The results are representative of the patterns that I have observed.

	%	Percentage with synonym count of						Blocks	
Capacity	Full	0	1	2	3	4	5+	Ovrflo	Total
1380	90.1	40.9	36.8	15.9	5.1	1.2	0.0	19	69
1381*	90.0	43.4	34.6	15.9	4.8	0.8	0.5	20	70
1382	89.9	39.8	37.3	16.4	4.8	1.6	0.0	20	70
1383	89.9	40.5	38.9	15.9	4.2	0.4	0.0	18	70
1384	89.8	41.6	36.8	15.2	5.1	1.2	0.0	15	70
1385	89.7	41.4	37.8	13.3	6.8	0.8	0.0	15	70
1386	89.7	39.9	35.1	20.0	4.2	0.8	0.0	22	70
1387	89.6	40.2	35.4	18.8	3.5	2.0	0.0	18	70
1388	89.6	41.2	34.8	15.7	6.8	1.6	0.0	18	70
1389	89.5	41.5	36.5	15.7	3.9	2.4	0.0	16	70
1390	89.4	39.3	37.7	14.0	5.8	3.2	0.0	18	70
1391	89.4	40.1	37.7	16.4	5.5	0.4	0.0	19	70
1392	89.3	40.0	39.7	14.5	4.5	0.8	0.5	15	70
1393	89.2	40.6	34.6	17.1	6.4	1.2	0.0	20	70
1394	89.2	40.8	38.0	15.2	4.8	1.2	0.0	18	70
1395	89.1	43.3	37.0	14.7	4.2	0.8	0.0	16	70
1396	89.0	41.4	35.1	16.9	6.1	0.0	0.5	22	70
1397	89.0	41.8	36.8	13.8	5.5	1.6	0.5	18	70
1398	88.9	43.0	35.2	17.1	4.2	0.4	0.0	17	70
1399*	88.8	41.5	37.2	17.1	3.2	0.4	0.6	18	70
1400	88.8	43.5	36.5	13.3	5.8	0.4	0.5	19	70
1550	80.2	44.3	38.3	12.8	4.2	0.4	0.0	7	78
1551	80.1	45.7	38.0	11.3	4.5	0.0	0.5	10	78
1552	80.1	44.8	35.9	13.5	5.8	0.0	0.0	10	78
1553*	80.0	45.2	37.2	12.1	5.1	0.4	0.0	12	78
1554	80.0	46.3	36.0	12.6	3.5	1.6	0.0	10	78
1555	79.9	44.0	36.5	13.5	3.9	0.5	0.0	10	78
1775	70.0	50.1	36.7	10.6	2.6	0.0	0.0	6	89
1776	70.0	49.2	36.7	11.3	1.9	0.8	0.0	5	89
1777*	69.9	51.7	31.7	12.6	3.2	0.8	0.0	5	89
1778	69.9	51.6	36.5	9.7	2.3	0.0	0.0	6	89
1779	69.9	49.1	37.3	11.3	2.3	0.0	0.0	1	89

Prime capacities are marked with an asterisk (*).

General Conclusions

Are there especially good capacities?
Of the data sets that I have examined, there have not been any instances of a capacity that was overwhelmingly better than others of approximately the same value.

Are there especially bad capacities?
I have seen only one class of values that are clearly to be avoided; that is the powers of two. These capacities produce large numbers of synonyms, yet values one greater or less are perfectly acceptable.

What about percent full?
There does not appear to be a hard rule that some percentage is acceptable and one percent greater is unacceptable. Generally, a capacity between 70% and 80% seems to be satisfactory, especially in the overflow block count statistic.

So do I just pick a number out of the air?
I would suggest that you use a capacity that will keep your data set between 70% and 80% full and then if possible check it with DBCAPCK. If you feel comfortable with a prime capacity (or if you are forced to use a prime by third party software which will allow you to change the capacity of a data set), by all means, use it. Prime capacities don't appear to be bad, they just aren't demonstrably better than non-prime.

How do I get to run DBCAPCK?
I have sent a copy of DBCAPCK to the performance library at CSY in Cupertino. Contact the Performance trained SE in your area to obtain use of the program. Please do not try to contact the factory directly.

Conclusion

I think the real lesson to be learned by all this, is that as users we should all maintain a certain amount of skepticism when we are asked to accept "truths" which are not grounded in empirical evidence. And if you have your doubts, construct an experiment to test the assertion. You might overturn the next IMAGE myth. Why you might even discover that integer keys are all right after all!

B. David Cathell Born 1946 in Showell, Maryland. Raised in Baltimore, Maryland. Graduated from Fort Lauderdale (Florida) High School in 1964. Graduated from Purdue University in 1968 with a Bachelor of Science Degree in Mathematics with specialization in Computer Science, minor in Physics. Employed by Control Data Corporation in Arden Hills, Minnesota, as a Programmer Analyis. Designed and implemented Operating System software for the CDC 3000L Series Computer System. Transfered with Control Data to Sunnyvale, California in 1976. Designed and implemented Operating System software for the CDC STAR computer system. Employed by Hewlett-Packard Company in Cupertino, California beginning 1978. Designed and implemented Factory Automation software on the HP 1000 for internal use. Transfered to Neely Sales Organization in Santa Clara, California in 1980 as a Commercial (3000) Systems Engineer. Areas of specialization include Data Communications, Laser Printing, Performance, Personal Computing and Office Automation. Transfering to the Monterey Bay Sale Office when it opens, expected January, 1984.
