How to Compute Master Data Set Capacities

by

Kurt Sager

SWS SoftWare Systems AG
Schönauweg 8
CH-3007 Bern
Switzerland

Summary

Defining optimal capacities for IMAGE master data sets is not just a matter of getting a number from a prime number table. Many other factors influence the access performance of master data sets. A high percentage of synonyms not always means long DBPUT transaction times, 0.1% synonyms however may indicate severe performance degradation if other conditions are met. A method is presented which allows to compute master set capacities such that *perfect hashing* (no synonyms at all) is achieved in many practical cases.

References:

[1] IMAGE Data Base Management System Reference Manual HP-32215-90003

[2] The IMAGE/3000 Handbook / R.M. Green, F.A. Rego, F. White, D.J. Greer, D.L. Heidner. - Seattle : WORDWARE, 1984. - ISBN 0-914243-00-4

[3] IMAGE: An Empirical Study / B.D. Cathwell. - Proceedings HP3000 IUG 1984 Anaheim Conference. - p. 4-1 to 4-5

Introduction

In appendix C of HP's IMAGE Reference Manual [1], the following statement is written:

> *"A master data set capacity equal to a prime number or the product of two or three primes generally yields fewer synonyms than a master data set capacity of many prime factors."*

The appendix in the same manual shows a table of selected prime numbers from 101 to over 8 million.

Participants of IMAGE training classes are told to use *prime numbers* where possible to reduce the number of *synonyms* in master data sets.

ADAGER, the well-known data base utility package, refuses any number but a prime number as a master data set capacity to be changed. And other IMAGE utilities make similar propositions.

These statements and practices may raise the questions below:

> *Where is the mathematical proof which demonstrates the superiority of prime numbers for master data set capacities?*
>
> *What is the real impact of synonyms on system performance?*
>
> *If system throughput and response times suffer from synonyms, how should the capacity be changed to cure the problem?*
>
> *Are prime capacities really the solution?*

Findings from practical experience

Ten years of practical experiences with many IMAGE data bases prove that best performance is not guaranteed by just using prime numbers as master data set capacities.

Other factors have much more influence on the number of synonyms and possible performance degradation. It is generally known, that the load factor (the ratio of the actual number of entries over the capacity) of master sets should not exceed 70 to 80 percent. Although IMAGE allows master sets to be filled up to 100% severe performance degradation can be observed if entries are added to a master set already more than 80 to 90% full.

But in many cases adding entries to a nearly full master set shows no performance degradation. Sometimes however terminal response times become inacceptable due to synonym problems with master sets less than 50% loaded. Why?

It seems that the key item value distribution, or perhaps some specific value patterns, are much more important whether or not a significant amount of synonyms is generated, rather than choosing a prime or non-prime as capacity.

How Hashing Works

A very common task in today's data processing applications is to store an information entity on a secondary mass storage device (disc) for later retrieval. Each information block shall be identified by a unique key item value.

In computer science several methods are known to store an entity of information to a file and retrieve it at a later time, based on the value of the chosen key item.

On the HP3000 two techniques have been implemented and are available to programmers as part of the fundamental operating software package (FOS):

- the *B+ Tree technique*, used in KSAM files

- the *Hashing method*, used in IMAGE

The *B+ tree technique* implements an ordered sequence of the key item values as a tree in natural order (numerical or alphabetical order). The main advantage of this technique: processing in key order sequence is easy. The time to insert or to locate a given entity grows with file size.

With *hashing*, a defined algorithm computes a relative file address using the search item value as argument. Insert and retrieval times are independent of the file size. Ordered sequential processing is not possible, however.

IMAGE uses two different *hashing algorithms* to compute a file address, depending on the search item type.

Binary Type Search Items

If the the search item is of type I, J, K, or R (binary type), then a quite simple method is used:

$$a := (v - 1) \text{ modulo } c + 1$$

where
v is the search item value
c is the capacity of the data set
a is the computed data set address
and modulo means: "remainder of the division of $(v - 1)$ by c"

The few examples below show how this formula works:

search item value v	capacity c	computed address a
10	1000	10
100	1000	100
777	1000	777
1001	1000	1
1010	1000	10
1234567	1000	567
10	500	10
100	500	100
777	500	277
1001	500	1
1010	500	10
1234567	500	67

The mechanism of this technique can be visualized by mapping the domain of the key item values to a circle representing the address space of the master data set. Of course, the circle has a circumference of lenght c, the capacity.

ASCII Type Search Items

If the search item is of type X, U, Z, or P, then a much more complicated algorithm is used (see the *IMAGE Handbook* [2] for detail information and example calculation).

Synonyms

Starting from a key item value the hashing algorithm calculates an address between 1 and capacity c. The entry is then stored at this address if it is not yet occupied by an earlier added entry.

Only for a few special cases hashing algorithms perform so well that every (different) key item value maps to a different address, however. Thus sometimes a key item value generates the same address as an entry added earlier. This is called a *collision.* IMAGE then has to find the next free space to store the new entry. This new entry is called a *synonym* to the previous one, and is linked to it by forward and backward pointers. All different key items values hashing to the same address build-up a so-called *synonym chain.*

Hashing is said to perform well if few synonyms are found in a master data set relative to the number of entries. A good hashing algorithm spreads the entries as uniformly as possible into the available address space. As the master data set is filled up, the number of free locations diminishes, and therefor the probability of collisions will increase.

The programs *DBLOADNG* from the *Contributed Software Library*, and *HowMessy*, a bonus program for customers of *Robelle* products, show the percentage of synonyms and the average/maximum synonym chain length of every master set in a data base. *HowMessy* runs about 10 times faster as *DBLOADNG.* It is well documented in the *IMAGE Handbook* [2].

Performance considerations

Does a high percentage of synonyms mean bad performance for data base operations?

Not generally, performance depends on many other factors too!

The processing time to *read* data from a master set by a keyed DBGET is only marginally dependent on the synonym percentage (see the column *Inefficient pointers* in the *HowMessy* report).

The case of *adding* entries to a master set is more critical. If IMAGE can find a free entry in the near vicinity of the collision point, hopefully in the same disc block, then it's just a matter of pointer settings.

If there is no free space in the same disc block then the next block must be read from disc, and so on, until a block with free entries is found.

The maximum number of entries per disc block is equal to the blocking factor, the ratio of block size over entry length.

For data bases with many online accesses the block size should not be increased over the default 512-word size. The entry size of masters however should be kept as small as possible. Bad examples are manual master sets with many or big data items, good examples are manual or automatic master sets containing the key item only.

Therefor: the smaller the entry size the higher the probability of a free entry in the same disc block.

Many fully occupied blocks side by side, called *clusters*, are very dangerous if a calculated hashing address falls in the beginning of this zone. IMAGE then has to read many disc blocks to find a free entry for this DBPUT. During this sequential reading the whole computer system is locked for any other processes, including MPE!

In the column *Max.Blocks* the program *HowMessy* shows the maximum number of contiguous completely filled blocks for every master set. If this number is very low, say zero or one, then even a high percentage of synonyms does no harm, and DBPUT's will process as fast as ususal.

A large number for *Max.Blocks* indicates a potential performance problem, which actually already may occur if there are *any* synonyms present.

Every production data base should be analysed by *DBLOADNG* or *HowMessy* at regular intervals, once per week for example. The resulting output (one line per search item and per data set) should be checked for possible problem figures, as decribed in the *IMAGE Handbook* [2], chapter 23.

Analysing and correcting critical hashing situations

Our experiences show that especially search items of binary type need attention. The following remarks apply to this type of search items only: the I, J, K and R types.

It is quite frequent that binary serach item values are forming a (nearly) continuous sequence of integers, such as 101, 102, 103, ...

We can say they belong to a (almost) dense interval *[a,b]*, where *a* is the lower limit, and *b* the upper limit.

The corresponding master set capacity is therefor at least $c := b - a + 1$. Applying the simple hashing algorithm to this case we easily see, that every search item value maps to a *different* master set address. The result is *perfect hashing (no synonyms at all), no matter how full the data set is!*

If the search item values belong to *two or more intervals*, it becomes more complicated to see what happens. In fact this case is usually the *reason for very long response times* of DBPUT's if the intervals are very dense and mapping them to the address space causes overlapping of the intervals.

To overcome this quite frequent problem, the program *MASCAP* has been written and contributed to the *Madrid SWAP TAPE*.

For search items values belonging to several intervals *MASCAP* cpmputes master data set capacities such that *perfect hashing* will be achieved. The program aks for the lower and upper limits of every interval, the maximum allowed capacity, and then shows ranges of perfect hashing capacities.

The examples below show how *MASCAP* works (user input is written in *italics*):

```
:RUN MASCAP

MASCAP / Version 1.0 / (C) 1982 SWS SoftWare Systems AG, Bern

MASCAP computes Perfect Hashing Capacities for IMAGE Master
Sets with binary type search items (In,Kn,Jn).
This algorithm assumes that the search item values belong to
several relatively dense intervals.

The program will ask you to enter the number of intervals,
then for each interval the lower and upper limits.


Number of intervals: 2
Limits (min,max) of interval  1    ? 8400001,8412000
Limits (min,max) of interval  2    ? 8500001,8502000
Minimum capacity allowed:  14000
Minimum capacity: 14000
Maximum capacity: 25000
Perfect Hashing Capacities from 14572 (96.1 %) to 14666 (95.5 %)
Perfect Hashing Capacities from 17000 (82.3 %) to 17600 (79.5 %)
Perfect Hashing Capacities from 20400 (68.6 %) to 22000 (63.6 %)

Number of intervals: 3
Limits (min,max) of interval  1    ? 8400001,8413000
Limits (min,max) of interval  2    ? 8500001,8512000
Limits (min,max) of interval  3    ? 8600001,8602000
Minimum capacity allowed:  27000
Minimum capacity: 27000
Maximum capacity: 50000
Perfect Hashing Capacities from 28858 (93.6 %) to 29000 (93.1 %)
Perfect Hashing Capacities from 37334 (72.3 %) to 37400 (72.2 %)
Perfect Hashing Capacities from 40400 (66.8 %) to 43500 (62.1 %)

Number of intervals: 0

END OF PROGRAM
```

The figures above show that *perfect hashing* can be achieved using even numbers as master set capacities, for example 17000 from the first case above. When choosing a capacity outside the reported ranges catastrophical situations may occur.

In a real case observed at a customer site, a capacity of 15013 (a prime number of course!) and a load factor of about 70% one day produced a sudden jump in response time from 1 second previously to 45 seconds for every DBPUT to this master set. *Decreasing* the capacity to 14600 cured the problem!

An other author (see [3]) reports that the benefits of prime capacities are rather a myth than a miracle!

Program *MASCAP* needs the interval limits as input for each run. Sometimes however the number and the limits of each interval are not known in advance. To help in such situations the program *MASANAL* has been developped. It takes a specified master data set as input and then reports all relatively dense intervals found. The output can then be used by the program *MASCAP* to compute optimal capacities.

Conclusions

To obtain best master data set capacities in specific situations the following rules are proposed:

If the search item is of binary type and all values are contained in *one* relatively dense interval between *a* (smallest value) and *b* (largest value) then use a capacity of $c := (b - a + 1)$. *The resulting hashing will be perfect (no synonyms) independent of the load factor.*

If the search item is of binary type and all values are contained in several relatively dense intervals then use the program *MASCAP* to compute *perfect hashing capacities*. If the interval limits and their number are not known then the program *MASANAL* should be used first to extract these figures from actual search item values stored in the master set.

In both cases the performance of existing data bases can be substantially improved without any changes in existing application programs.

If the numeric search item values are distributed more or less randomly then use a prime as master data set capacity and define the search item as binary type. In fact, the theory on pseudo random number generators explains that prime numbers are important in algorithms of "modulo type" to obtain good random distributions. Is this the origin of the prime number myth in IMAGE?

In all other cases define a search item as ASCII type and use *any* suitable number as master data set capacity, provided that it is not a power of 2 and that the load factor does not exceed 70 to 80%.