Reducing Migrating Secondaries

Earl Waller and Mel Pearce
INLEX, Inc.
P.O. Box 1349
Monterey, CA. 93942

If you are like us, you have read a lot recently about IMAGE and its 'myths', and you know that there can be performance problems associated with migrating secondaries in an IMAGE database. And like us, you follow the recommendations to use ASCII key structures and prime number capacities. And, finally, you expect and accept some performance degradation as the database fills up and schedule your database reloads over long weekends. But what do you do when the performance becomes unacceptable?

INLEX provides library automation solutions. Design considerations led us to use a combination of MPE flat files (accessed randomly via pointers), KSAM files (for partial key searches and sorted access) and IMAGE databases (to give us fast known key, multi-record access). As relative newcomers to IMAGE, we read all of the available information about IMAGE database design and wanted to follow all of the recommendations.

Due to several restraints, we found that we needed to use a binary pointer as a key. We discussed the problem with an IMAGE expert who suggested that we use a six character ASCII key as the binary pointer but store it as a binary value. He thought that this might give better results than using a three word binary key.

Our customers routinely experienced performance variability when loading bibliographic records. A typical bibliographic record requires two variable length flat file records, eleven KSAM records, and three IMAGE records. Load rates differed across our customer base from one to thirty records per minute and degraded severely as the databases filled up. This seemed to follow very closely the bibliographic record load rates of other vendors in our industry and, because we had been told that KSAM was very slow when rebalancing its trees, we didn't believe that migrating secondaries were a serious issue.

The real problem surfaced when we tried reloading one of our customer's IMAGE databases. Using a dedicated Series 52 computer, we initiated a database reload of an IMAGE database containing 141,000 records. We aborted the not yet completed computer run after two and a half weeks of dedicated time with the realization that we had some serious migrating secondary issues to resolve.

About this time there were numerous articles in the literature exposing myths about prime number capacities and integer keys. However, these articles failed to explain how to analyze a database, how to tune a database for performance, or even how to know that the database is optimum.

Once we were confronted with the problem we realized that we needed an analysis tool and a methodology to answer these questions. We searched for tools to combat the problem but were unable to find anything that promised sufficient insight or a clear solution. So, we decided to build our own tools.

We obtained the IMAGE hashing algorithms and implemented a program that would read an IMAGE master and chart the distribution of the hashed keys. The program also gave a measurement of the number of primaries with no synonyms, with one synonym, with two synonyms, etc.

Needing more disk to hold the customer's database for testing and realizing we would be making hundreds of disk intensive runs, we obtained a 132 megabyte ram disk from Imperial Technologies[1]. We were able to configure the device as an HP 7914 disk drive, allowing us to perform high speed disk i/o, eliminating seek and latency time.

With our customer's 141,000 record database and our new program, we cycled through the automatic master hundreds of times using hundreds of different capacities. Because the program only totaled secondaries and did not migrate them, it ran in a fraction of the time needed to reload a database. The program generated the calculated hash distribution for each run and displayed the results in a series of bar charts. Figure 1, on the next page, is an example of one of these bar charts.

Analysis of the charts showed that many of the keys hashed to the same address and/or the same general area in the automatic master. This created clusters and synonym chains, many of them quite long. When IMAGE computes a synonym it adds the entry to the end of a synonym chain or creates a new chain and attempts to place the new entry in an empty slot in the same block of the file. When the existing block is full, IMAGE places the new entry in another block, causing an overflow. The computed overflow block count showed that the empty slots in the hashed block were filling up and that IMAGE placed the entries in additional blocks, sometimes after very long serial searches for an empty slot.

Figure 1 - Sample Bar Chart Report

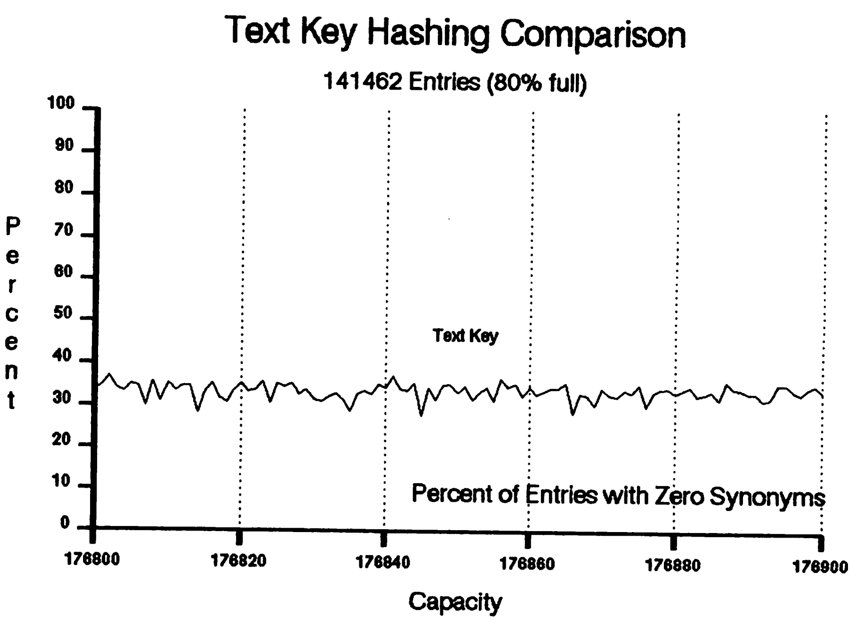
```
                              Data Base = AUTHOR.DBS
No.  Name                Type  BF   Entries    Capacity
1    AUTHOR-KEY           A    28   141462      324001
                         Data set number = 1
                             Search item = KEY-POINTER
                               Item type = X
                                Capacity = 190973
                         Blocking factor = 28
                             Entry count = 141462 (74.1%)
         Entries with  0 synonyms -        50960   36.0%
         Entries with  1 synonyms -        33660   23.8%
         Entries with  2 synonyms -        17979   12.7%
         Entries with  3 synonyms -        12708    9.0%
         Entries with  4 synonyms -         9800    6.9%
         Entries with  5 synonyms -         7404    5.2%
         Entries with  6 synonyms -         4347    3.1%
         Entries with  7 synonyms -         2448    1.7%
         Entries with  8 synonyms -         1197    0.8%
         Entries with  9 synonyms -          640    0.5%
         Entries with 10 synonyms -          209    0.1%
         Entries with 11 synonyms -           96    0.1%
         Entries with 12 synonyms -            0    0.0%
         Entries with 13 synonyms -           14    0.0%
             Overflow block count = 729
                Total block count = 6821

17780+ *
     | *
     |**
     |**
13335+**
     |**
     |****
     |****
 8890+****
     |****
     |****
     |*****
 4445+*****                  *
     |******      **         **
     |*************** ***********    * ****************
     |**************************************************
     +----+----+----+----+----+----+----+----+----+----+
         10%  20%  30%  40%  50%  60%  70%  80%  90%  100%
```

When IMAGE hashes to a slot occupied by a synonym, it moves the existing synonym entry into another empty slot, adjusts the synonym chain pointers, and places the new entry into the vacated slot. This is called migrating a secondary. It also can be very disk intensive.

The IMAGE hashing algorithm is a function of two variables: the key type and the master dataset capacity. We modified our analysis tool so that we could analyze up to fifty different capacities in one computer run and summarize the result into a single table. The results of this series of tests showed that prime numbers are not necessarily the best capacities to choose. Now we could see the best capacities for the data in the databases. However, the occurrence of secondaries in the best selection was still larger than we could accept. Figure 2 shows that, for these tests, the number of entries with zero synonyms was not even equal to forty percent.

Figure 2



Our next step was to analyze several alternate key structures. To do this, we created another program that would accept input from a flat file and perform the same functions as the first program. This allowed us to analyze potential key structures without first building and loading a database.

The keys of concern are the pointers to the file and a logical record within the file. In accordance with IMAGE recommendations, we originally specified this to be an alpha key. To put it another way, we purposely avoided using an integer key.

We chose an integer key for our next series of analysis runs. Figures 3 and 4 show a performance comparison of the use of an integer key versus the use of a text key over a broad range of capacities. We observed that the integer key resulted in significantly improved distribution, and smaller clusters, with a significant reduction in the number of synonyms and the length of synonym chains.

Figure 3

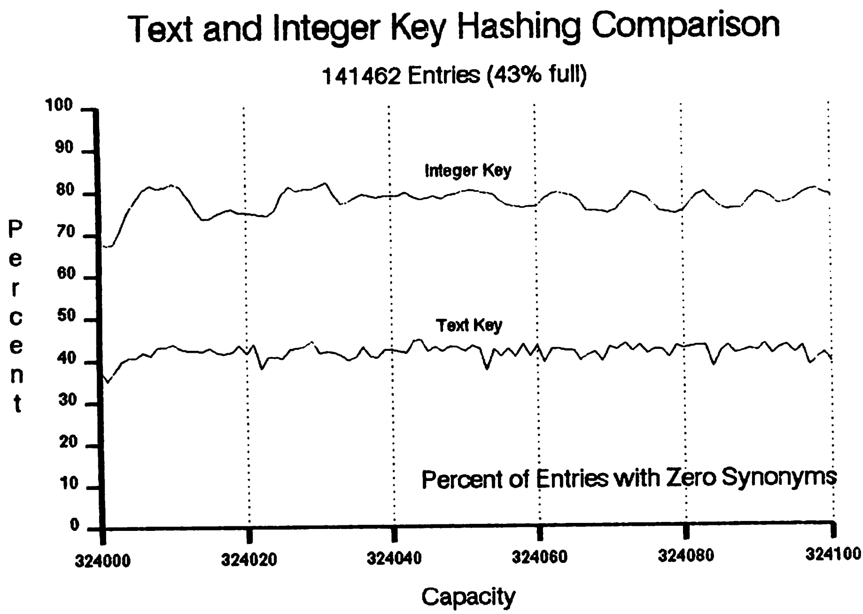
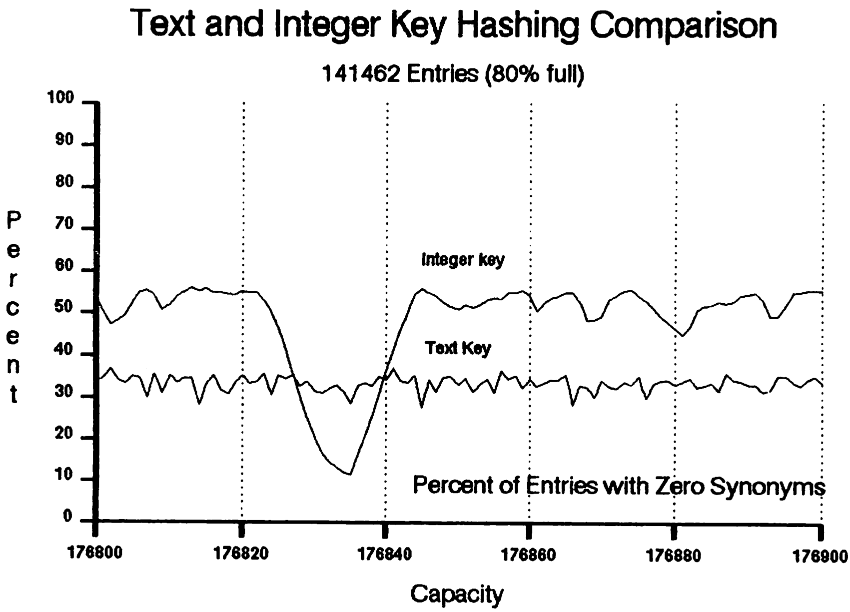


Figure 4

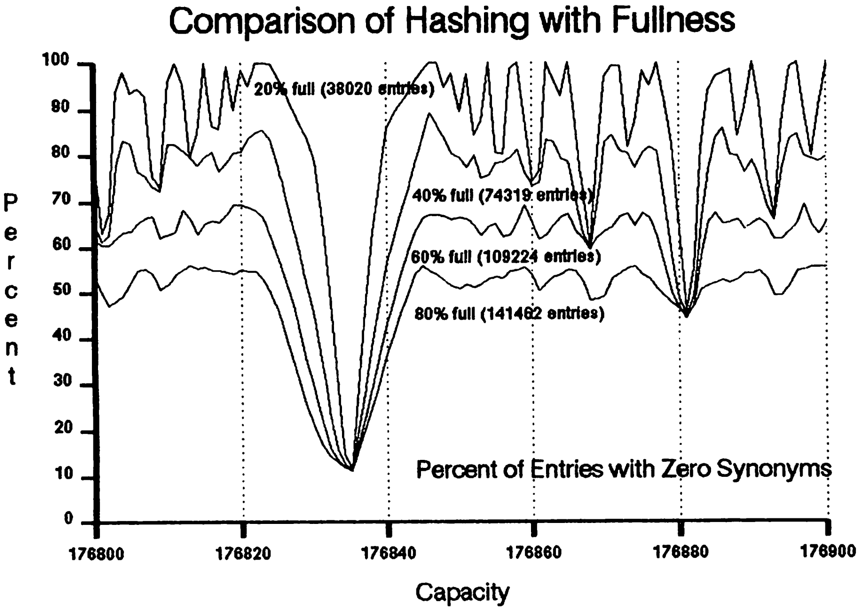


We felt that we were on the right track; however, the computer time required for the analyses, even with the ram disk, was significant on our Series 40 computer.

We wondered if we could derive any conclusions from comparing smaller chunks of the same data in the same database for a range of capacities of the master set. This caused us to make a number of runs using only enough data to fill the master to twenty percent, forty percent, sixty percent and eighty percent of its capacity.

Figure 5, on the next page, shows the analyses generated for some of these runs. We observed that goodness for a twenty percent full database never translates into badness as the database becomes full. Our conclusion was that poor capacities could be avoided by analyzing a sample as small as twenty percent of the data.

Figure 5



By now we had developed some confidence in our analysis tool. The real measure of its success, of course, could only be determined by measuring the results in the real world.

We selected the integer key and an optimum capacity and reloaded the database at our customer site. The reload that previously had not completed in two and a half weeks finished in just five hours. Encouraged by these results, we were anxious to see how these improvements would affect our database loading process.

We selected a customer with slightly more than 208,000 bibliographic records to load. The customer was using a dedicated HP3000 Series 70 with Eagle drives. Load times of eight to ten records per minute were originally anticipated. Based on a 20 hour day of productive work (time off for system backup) it would take approximately 28-29 days to load and index this database.

All 208,000 plus bibliographic records were loaded and indexed over a three and one quarter day period. It was encouraging to note no observable performance degradation as the databases filled.

We are now using the tools we developed prior to any database reloads, and any time there is a design change to an IMAGE key structure.

As graduates of the school of hard knocks, we have formed some conclusions. It is possible to tune for optimum performance and know when the optimum is reached. Prime numbers may or may not be good choices for a dataset capacity. Two identical databases can require different optimum capacities if they contain different data values. Most important of all, performance and tuning is a function of the specific data in the database, and without a tool to examine a database there is no way to know what IMAGE is really doing with your data.

1. Imperial Technology, Inc.
   831 S. Douglas Street, Suite 102
   El Segundo, CA 90245
   (213) 536-0018