Hashing in Practice, Analysis of Hashing and Universal Hashing

M.V. Ramakrishna
Computer Science Department
Michigan State University
East Lansing, Michigan 48824 - 1027

ABSTRACT

Much of the literature on hashing deals with overflow handling (collision resolution) techniques and its analysis. What does all the analytical results mean in practice and how can they be achieved with practical files? This paper considers the problem of achieving analytical performance of hashing techniques in practice with reference to successful search lengths, unsuccessful search lengths and the expected worst case performance (expected length of the longest probe sequence). There has been no previous attempt to explicitly link the analytical results to performance of real life files. Also, the previously reported experimental results deal mostly with successful search lengths. We show why the well known division method performs "well" under a specific model of selecting the test file. We formulate and justify an hypothesis that by choosing functions from a particular class of hashing functions, the analytical performance can be obtained in practice on real life files. Experimental results presented strongly support our hypothesis. Several interesting problems arising are mentioned in conclusion.

1. Introduction

Hashing is an efficient, simple and popular technique for organizing both internal tables and external files. The main issues in hashing is summarized by the following quotation from Knuth, who also gives an excellent introduction to hashing [KN74].

> "In order to use a hash table, a programmer *must* make two almost independent decisions:
> a) he *must* choose a hash function $h(k)$, and
> b) he *must* select a method for collision resolution".

There are scores of papers which have addressed efficient handling of collisions (overflow is a more appropriate word, especially while dealing with multirecord bucket hash files). More recently there has been some work in the area of perfect hashing (hashing with no overflows, i.e., (b) above is not true) [RL88]. There has been little work in addressing (a), how to choose hashing functions. Moreover, there has been no attempt to explicitly address the issue, what does all the analytical results mean in practice. This paper addresses the problem and shows how the analytical performance results can be achieved in practice.

Hashing has been an active area of research since early sixties. However, there are only four main papers [DS75, KN75, LY71, LM73], all in the period 1971-75, which address the performance of hashing functions. It is rather surprising that none of them address the problem of directly linking the analytical results to performance of different hashing functions, although papers continue to appear analyzing hashing schemes. All the above four papers, to state briefly, show which particular functions perform "well" on the average on several real life files. They mostly consider the average length of successful searches. The unsuccessful search lengths and the expected worst case performance has not been considered (of course, the analysis of the expected worst case performance is of recent origin [GN81, LR82]). Our paper addresses the basic question, how to achieve the theoretical analysis results with practical files with reference to successful and unsuccessful search lengths and the length of the longest probe sequence. That is, what kind of hashing functions can give us the predicted performance. We show why the well known division method performs "well" using a specific model suggested by Lum for selecting the test file [LM73]. After indicating how trivial it is to construct worst case key sets for the methods such as the division method, we formulate and justify an hypothesis that by choosing functions from a particular class of hashing functions, the analytical performance can

be obtained in practice on real life files. Experimental results presented strongly support our hypothesis.

2. Previous work

Lum, Yuen and Dodd were the first to address the problem of practical performance of hashing functions in 1971 [LY71]. Earlier work in this regard by Buckholz [BC63], "presents an excellent discussion of various aspects of key-to-address transformation, but his experimental results are minimal"[LY71, p228]. Lum, Yuen and Dodd consider six different kinds of hashing functions such as division-remainder, mid-square, digit analysis (a complicated method involving analyzing the distribution of digits of keys) etc. Each "type" of hashing yields a fixed hashing function for a given table size. Considering minor variations of the six types, they report eight sets of experimental results. For any given file in practice, there are only eight possibilities for the hashing functions. They report results of average successful search length for different bucket sizes and load factors (they consider separate chaining and linear probing overflow handling methods).

In a follow up paper, Lum developed a framework for analyzing nonuniform key distributions [LM73]. For a given hashing function, the idea was to consider its performance over all possible key sets of size n from the given universe. One alternative is to assign equal probabilities for each of the n element key sets. Further under the new frame work, he tried to explain the experimental behavior of each of the six different hashing functions considered in [LY71]. In practice however, the designer has no control over the key set.

Knott provided an excellent survey of the various hashing functions and overflow handling strategies [KN75]. He classified hashing functions into (key) distribution dependent and distribution independent functions. If the cumulative probability distribution, F_x of the key set is known, $F_x(t) = Pr(X \leq t)$ where X is the random variable defined as $X(x) = x$(x being the key), then it can be readily seen that the hashing function $h(x) = \lceil mF_x(x) \rceil - 1$ would yield very few overflows, if any at all. He also discussed polynomial hashing functions, adapted from algebraic coding theory for key sets with clusters. These two methods assume that the distribution of the key set is known (distribution dependent). Properties of a number of distribution dependent hashing functions were discussed. The basic idea behind all discussion is to minimize the number of overflows.

Deutscher, Sorenson, and Trembley further investigated the idea of distribution dependent hashing functions [DS75]. For comparison purposes, they chose division-remainder method of hashing, reported to be the "best" in [LY71]. Their distribution dependent hashing functions are based on Knott's discussion [KN75], and uses the notion of the cumulative probability distribution of the keys, F_x. Four methods of approximating F_x are described: 1) Piecewise linear function (PL) 2) Fourier hashing function (FH) 3) Multiple frequency distribution function (MF) 4) Piecewise linear function with interval split (IS).

Hashing simulations were performed using the above four methods and division-remainder method. Three files were chosen for experiments: a file of Library of Congress catalog card numbers, a uniform key set and a clustered key set. For each of the files and hashing functions the average successful search length was measured for chaining and open addressing overflow handling methods. They drew several conclusions based on the results such as with open addressing MF performed better than division-remainder; MF is especially good for small bucket sizes; with chaining, none of the distribution dependent functions performed better that division-remainder method; with dynamic key sets, the advantage of MF for uniform key sets decreased as insertions and deletions were performed; for clustered key sets, MF outperformed division-remainder method, etc.. Their final conclusion is that the distribution dependent hashing functions can be advantageous when used with open addressing and static key sets and that MF is the best overall distribution dependent hashing function of those studied in the paper.

Here we should mention about the recent theoretical results on hashing. Mairson has established that hashing functions of size (rather the "program size") $O(n)$ are required to achieve guaranteed bounded retrieval performance (in particular for perfect hashing) [MR83, MR84]. Fredman, Komlos and Szemeredi have given a data structure based on hashing and show how $O(1)$ retrieval performance can be achieved using no more that $O(n)$ space [FK84]. Carter and Wegman have theoretically shown that using *universal*$_2$ classes of hashing functions $O(1)$ retrieval can be obtained on the average [CW79]. All of these results are based on number theory and combinatorics. They do not deal with probabilistic analysis on which the performance results are based and practical performances of hashing, which we are concerned with in this paper.

Critical comments

All the previous work, briefly described above, discuss how good or bad is each hashing function. Moreover, in each "type" of functions, there is (almost) a single function possible for a given table size and key set. All these work was in the period of 1971-75. There is very little mention of the connection between the theoretical probabilistic analysis (on which many papers continue to appear) and hashing functions which can perform accordingly in practice with real life files. We address the problem of how/what functions achieve the theoretical performance predictions in practice.

3. A class of hashing functions

Usually hashing techniques are analyzed under the assumption that the hash addresses are uniformly distributed. Let $I = \{x_1, x_2, \cdots, x_n\}$ be the given set of keys, and without loss of generality let $I \subset A$, $A = \{1, 2, \cdots, a\}$, be the universe and $B = \{0, 1, \cdots, m-1\}$ be the address range. Given a hashing function $h: I \rightarrow [0, 1, \cdots, m-1]$ and a key $x_i \in I$, the probability that the key hashes to a particular bucket is $1/m$ and independent of the out come of hashing other keys. There are m^n ways in which n keys can be distributed among the m buckets, i.e., there are m^n functions which map the given set of keys into the table. We assume that each of these distributions are equally likely when we hash the n keys into m buckets. The analytically expected performance then corresponds to the expected performance of a randomly chosen function from this set of m^n functions.

Let H be a class of hashing functions, $H = \{h_1, h_2, \cdots\}$, where each h_i is a function $A \rightarrow \{0, 1, \cdots, m-1\}$. H is said to be a *universal*$_2$ class of hashing functions if for every pair $x, y \in A$, they collide under no more than $|H|/m$ of the functions [CW79]. If two elements of A are distributed at random over m addresses, the probability that they collide is $1/m$. This implies that the collection of all functions from A to B, whose cardinality is m^a, is *universal*$_2$ [SD80]. H_1 is a particular class of *universal*$_2$ class of hashing functions defined by Carter and Wegman as follows,

> Let Z_p denote the finite field $\{0, 1, 2, \ldots, p-1\}$ where p is a prime number. Without loss of generality, assume $a = p-1$.
>
> $f_{c,d} : A \rightarrow Z_p$, $f_{c,d}(x) = (cx + d) \bmod p$ where $c > 0$, $d \geq 0$ and $c, d < p$.
>
> $g : Z_p \rightarrow B$, $g(x) = x \bmod m$.
>
> H_1 is the set of functions $\{h_{c,d} \mid c, d \in Z_p \text{ and } c \neq 0\}$, where $h_{c,d} : A \rightarrow B$, a composition of the functions g and f, and is given by,
>
> $$h_{c,d}(x) = g(f_{c,d}(x)) = ((cx + d) \bmod p) \bmod m.$$

The hashing function $g: A \rightarrow B$ as defined above is the same as the division method referred to as the "best" in [LY71] and used as a reference in [DS75]. Consider the model proposed by Lum for analyzing the performance of hashing functions [LM73]. Accordingly, the performance of a hashing function should be studied considering all possible subsets of the universe A, having size n. There are a total of $\binom{a}{n}$ possible subsets of size n from the universe A. One way of choosing a subset of A suggested by Lum, is to assign probabilities to each of the elements (probability whether that element is included in the chosen set). An assignment is to give equal weight to each of the keys. That is, the n element set I is formed by choosing elements randomly from the set A such that each element is equally likely to be selected. If all the keys in A are hashed by the function g, there will be precisely (a/m) keys in each bucket (since we assume $m \ll a$, we ignore using *floor* and *ceiling* functions). If the key set I is selected as described above, then each key is equally likely to be in any one of the m buckets, independent of the keys already selected (assuming $n \ll a$ as usually is the case). This is exactly same as the model of hashing analyzed (as described at the beginning of this section). Thus we have shown that,

Proposition:

> The hashing function g (also known as division method in the literature) gives the theoretically predicted performance under the model, proposed by Lum [LM73], of evaluating the hashing functions (the expected performance of a randomly chosen file from the $\binom{a}{n}$ possible subsets is as predicted by the theoretical model). Note that this proposition implicitly assumes that overflow handling is by separate chaining.

The programmer does not have any control over the given key set. Although, we assert that the function g above gives theoretically predicted performance, it is rather straightforward to construct "worst case" key sets for this hashing function (viz. the set $\{x \mid g(x) = constant\}$). For a given table size, there is only one function possible, i.e., $g(x) = x \bmod m$. So, one has to simply accept the resulting performance which is fixed. What functions perform well for any given key set? (or for a given key set what can be said about functions which give theoretical performance?). The above proposition can be extended as follows.

Knuth extensively discusses the following pseudo-random number generator, called the linear congruential method (which is "by far the most popular random number generators is use today" [KN81]),

$$y_{i+1} = (cy_i + d) \bmod p$$

where c and d are constants. The above recurrence gives successive values of y_i, starting from $y_0 = SEED$, which are uniformly distributed in the interval 0 to $p-1$. Each pair c, d gives a different pseudo-random number generator. Many of the restrictions on the constants, as discussed in [KN81], are automatically satisfied if p is chosen a prime. The function $f_{c,d}$ defined above is exactly the same as the pseudo-random number generator and the hashing function $h_{c,d} \in H_1$ is a composition of $f_{c,d}$ and g. Mapping a given set of keys by the function $f_{c,d}$ can be regarded as generating a sequence of pseudo-random numbers uniformly distributed in the interval 0 to $p-1$, using n different *SEED*s. That is, each key is equally likely to be mapped into any one of the numbers $0, 1, \cdots, p-1$. These arguments about f and the above proposition about g leads us to the following hypothesis about their composition, gf.

Hypothesis:

> For a given set of keys, by choosing functions at random from the class H_1, the theoretically predicted performance of the hashing functions can be achieved in practice, independent of the key distribution.

It is only an hypothesis shown true experimentally in the following section. The preceding discussion and arguments do not constitute a proof since there is no proof that the linear congruential method indeed generates uniformly distributed random numbers. Most of the theorems in [KN81] deal with which constants constitute bad choice for the random numbers generated (in regard to the distribution and period of the generated random numbers). However, our application puts a less stringent requirement, since we deal with a number of *SEEDs* and do not feed back the numbers generated.

4. Experimental Results

In order to test our hypothesis we performed several hashing simulations on the following real life ASCII files. The figures in parenthesis indicate the approximate size of the key sets.

A. Words from UNIX dictionary (24,000)

B. Userids from a large IBM installation (12,000)

C. Last names from a medical database of the Michigan State University (4,200)

D. Part of a file containing titles from University of Waterloo Library (10,000)

E. Library call numbers from Washington University Physics Library (10,500)

All of the keys are alphanumeric strings, the number of characters in each key varying from 1 to 55 characters. Usually the keys encountered in practice are alphanumeric strings. Most hashing functions operate on integers and hence the key strings need be converted into integers. The binary representation, in internal format, of the string itself can be treated as an integer. But, the resulting numbers will be too large to be handled by ordinary machines. There are algorithms, such as XORing method often used to convert alphanumeric strings into integers [LK84]. The problem with this algorithm and an improved algorithm for the conversion can be found in [RM86]. In the rest of the discussion, a key set refers to the (converted) integer key set and not the actual alphanumeric key set.

Basic urn model experiment

Consider the traditional urn model with m urns and n balls. Each ball is tossed randomly into an urn so that the probability of each ball falling in an urn is $1/m$ independent of other tossings. This is analogous to hashing n keys into m buckets. The distribution of the balls after tossing all the balls is the same as choosing one of the m^n possible distributions at random. The probability that there are precisely k balls in an urn is given by the binomial,

$$Pr.(n,m,k) = \binom{n}{k} (1/m)^k (1-1/m)^{n-k}$$

When m is large and n/m is bounded, the above distribution can be simplified using the poisson approximation to the binomial distribution [FL68],

$$Pr.(n,m,k) = \frac{e^{-n/m}(n/m)^k}{k!}$$

and is a function of n/m which we shall denote by λ, and hence

$$Pr.(\lambda,k) = \frac{e^{-\lambda}(\lambda)^k}{k!}$$

Most of the analysis on hashing is based on the above probability distribution. In hashing with separate chaining, the expected length of successful search follows straightaway from the above distribution. (for other examples refer [KN74, LR83]). In view of this, the first set of experiments performed were aimed at testing the hypothesis to see if the above theoretical distribution can be obtained in practice. Five hundred hashing functions were chosen "at random" from H_1, i.e., 500 c, d pairs were generated using pseudo-random number generator. The value of the prime p being fixed, close to the *MAXINT* of the machine ($p = 2100000011$). Each of the key sets was hashed separately by each one of the 500 hashing functions. In each case the relative frequency of the number of buckets in which k records were hashed was computed and were averaged over all the 500 functions. These averages are plotted in Figure 1. The solid line represents the theoretical probabilities computed using the above equation. There is one symbol (x,o,+,c,.) used to plot the results for each of the five different key sets. The results for all the files coincide very closely with the theoretical values and hence different symbols are not seen distinctly in the figure. The dotted lines enclose a critical region which represents the central 95% of the sampling distribution, for a sample size of 500, for the theoretical probability distribution [FR71]. For a particular value of k, the point on the solid line represents the probability $Pr.(\lambda,k)$ which we denote by θ for convenience. Let $\bar{\theta}$ and $\underline{\theta}$ denote the corresponding points on the dotted lines above and below the solid line. This means that, if $Pr(\lambda,k)$ is computed from choosing 500 distributions at random from m^n distributions, the $Pr(\lambda,k)$ will fall in the range $\underline{\theta}$ to $\bar{\theta}$ with 95% probability. For a given value of θ, the approximate value of $\bar{\theta}$ and $\underline{\theta}$ may be obtained using the relation $\bar{\theta} = \theta + 1.96\sigma$ and $\underline{\theta} = \theta - 1.96\sigma$ where $\sigma = \sqrt{\theta(1-\theta)/500}$. The exact values of $\bar{\theta}$ and $\underline{\theta}$ were obtained using the statistical tables in [BR71]. We should point out that showing the critical region in the figure appears superfluous, since the experimental results are almost coinciding with the theoretical probabilities, strongly supporting the hypothesis.

Uniform hashing

Uniform hashing or random probing is a theoretical model of open addressing schemes of cluster free overflow handling methods such as double hashing. In this model, a key to be inserted probes the buckets repeatedly until a non-full bucket is found where the key is stored. The probe sequence is modeled either as

a) a permutation of the addresses $0, 1, \cdots, m-1$ or

b) each probe position is a random number uniformly distributed in the interval $0..m-1$.

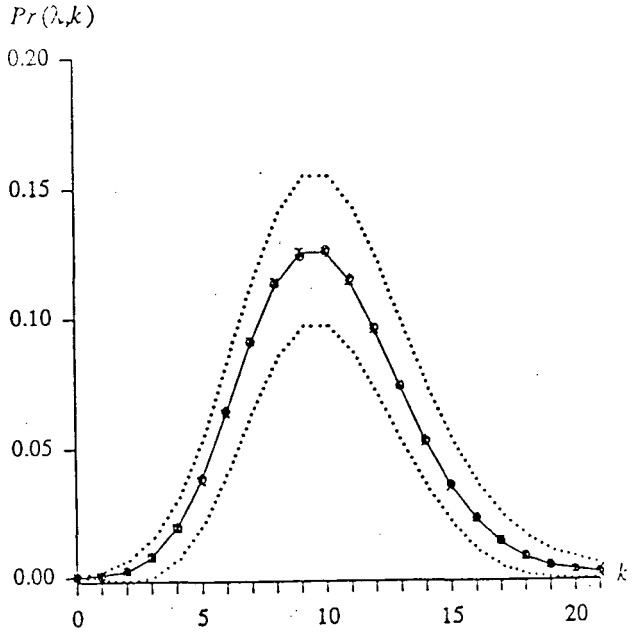

Figure 1. Comparison of the experimental and theoretical values of $Pr(\lambda,k)$, $\lambda = 10$.

The two are asymptotically equivalent and in [LR83] Larson provided a comprehensive analysis using model (b) (the model we use here). Our hypothesis can be readily tuned to accommodate uniform hashing: use a sequence of hashing functions to determine the probe sequence.

We simulated 500 file loadings of each of the five key sets under uniform hashing. For each loading, a sequence of hashing functions was generated using pseudo-random number generator (say $h_{i1}, h_{i2}, \cdots$). Each key to be inserted is hashed successively by $h_{i1}, h_{i2}, \cdots$ until a nonfull bucket is found where the key is stored. For storing n records, if a total of N hash function evaluations are required, the average successful search length is simply N/n. If F of the m pages are full after storing n keys, the expected length of an unsuccessful search is given by $\frac{m}{m-F}$. Tables 1 and 2 present a comparison of the experimental search lengths averaged over 500 loadings, for file B, with the analytical results (analytical results drawn from [LR83]). We observe that the results agree, in almost all the cases, to within two decimal places, and many of them agree to three decimal places. In view of this, no other statistical measures (to claim that the analytical and experimental results agree) are presented. Similar results were obtained for the other test key sets, but have not been included in the paper for brevity. We feel they do not further enlighten the main ideas presented in the paper.

The worst case of hashing occurs when all the keys have the same probe sequence and the corresponding retrieval cost of the "worst key" is m probes. But the probability of this event is extremely small. What is of practical interest is the expected worst case behavior of hashing. We use *llps*(length of the longest probe sequence) to denote the number of probes required to retrieve the "worst key" in any given hash table (the one which requires the maximum number of probes of all the n keys stored in the file). The *llps* is a random variable which takes a minimum value 1.0 (corresponding to a perfect hash table [RL88]) and a maximum value of m (corresponding to the worst case of hashing). Gonnet and Larson have analyzed the expected value of *llps*, $E(llps)$, which is a measure of the expected worst case behavior of hashing [GN81, LR82]. In the above experiments the *llps* for each loading is simply the maximum number of hashing functions used while storing the keys. The average of the 500 *llps*(one from each experiment) gives the experimental $E(llps)$. Table 3 compares the theoretically computed values of $E(llps)$ (from Table 1 of [LR82]) with the experimental values, for key set B and file size $m = 1000$ (note that $E(llps)$ depends on m). It should be noted that the analytical results are presented in [LR82] for only one decimal place (probably in view of the clumsy nature of the computations involved and the resulting roundoff errors). We observe that the experimental results are in close agreement with the analytical results, supporting the hypothesis.

b	$n/mb = 0.6$		$n/mb = 0.7$		$n/mb = 0.8$		$n/mb = 0.9$	
	exptl	anal	exptl	anal	exptl	anal	exptl	anal
1	1.5296	1.5272	1.7236	1.7200	2.0169	2.0118	2.5577	2.5584
2	1.2226	1.2218	1.3311	1.3300	1.4994	1.4983	1.8214	1.8177
3	1.1262	1.1260	1.2021	1.2021	1.3241	1.3245	1.5642	1.5612
4	1.0815	1.0811	1.1388	1.1395	1.2377	1.2368	1.4275	1.4290
5	1.0561	1.0560	1.1023	1.1030	1.1842	1.1839	1.3524	1.3476
10	1.0142	1.0141	1.0349	1.0352	1.0788	1.0787	1.1753	1.1773
20	1.0019	1.0020	1.0085	1.0088	1.0290	1.0289	1.0858	1.0853

Table 1. Comparison of expected length of a successful search under uniform hashing
(Analytical values obtained from [LR83, Table I, p809])

b	$n/mb = 0.6$		$n/mb = 0.7$		$n/mb = 0.8$		$n/mb = 0.9$	
	exptl	anal	exptl	anal	exptl	anal	exptl	anal
1	2.4978	2.5000	3.3287	3.3333	5.0000	5.0000	9.9196	10.0000
2	1.7551	1.7569	2.2460	2.2490	3.2395	3.2359	6.1759	6.1705
3	1.4936	1.4938	1.8574	1.8602	2.5989	2.6017	4.7877	4.8027
4	1.3567	1.3566	1.6536	1.6540	2.2642	2.2635	4.0744	4.0761
5	1.2718	1.2737	1.5239	1.5242	2.0491	2.0473	3.6473	3.6161
10	1.0990	1.0988	1.2406	1.2427	1.5664	1.5720	2.5852	2.5911
20	1.0212	1.0221	1.0887	1.0889	1.2872	1.2843	1.9504	1.9566

Table 2. Comparison of expected length of an unsuccessful search under uniform hashing
(Analytical values obtained from [LR83, Table II, p809])

b	$n/mb = 0.6$		$n/mb = 0.7$		$n/mb = 0.8$		$n/mb = 0.9$	
	exptl	anal	exptl	anal	exptl	anal	exptl	anal
1	10.4040	10.5	14.3080	14.0	20.4860	20.5	36.1180	37.4
2	6.6460	6.6	8.8480	8.7	12.5620	12.6	22.5800	22.6
3	5.1060	5.1	6.7440	6.8	9.7620	9.7	17.1280	17.2
4	4.2940	4.3	5.6960	5.7	8.2060	8.2	14.4640	14.4
5	3.8300	3.8	5.0200	5.0	7.2100	7.2	12.6660	12.6
10	2.5160	2.5	3.3840	3.4	4.9560	4.9	8.5320	8.5
15	2.0300	2.1	2.6860	2.7	3.9140	4.0	7.0580	6.9
20	1.8520	1.9	2.3120	2.3	3.4980	3.4	6.0140	6.0

Table 3. Comparison of expected *llps* under uniform hashing
(Analytical values obtained from [LR82, Table 1, p348])

Double hashing

Our hypothesis can be readily adapted to double hashing: choose two hashing functions h_{i1} and h_{i2} from the class H_1 for each loading of the file. It is well documented that the performance of double hashing is close to that uniform hashing [LR83, GB78]. The analysis of double hashing for the general case $b > 1$ is an open problem, and hence there are no analytical results to compare with experimental results. Our experimental results confirm that the performance of double hashing is close to that of the uniform hashing. Further results are not presented in view of the space limitation.

Separate chaining

The expected length of successful and unsuccessful searches are directly dependent on the probabilities $Pr.(\lambda,k)$ (for example if the secondary bucket size is chosen as one [KN74, LR82], the expected length of a successful search is given by $\sum_{k=0}^{b} 1\,(Pr.(\lambda,k)) + \sum_{k=b+1}^{\infty} 1+\frac{(k-b)(k-b+1)}{k} Pr.(\lambda,k))$.

The results presented in Figure 1 implicitly supports our hypothesis for separate chaining and hence we do not give a separate table comparing the expected search lengths under separate chaining. However, we simulated the file loadings to verify our hypothesis, for the expected *llps* under separate chaining. In each loading, the *llps* is simply the length of the longest chain (of the m chains). The experimental $E(llps)$ is simply the averaged value of the 500 *llps*s. Table 4 summarizes the analytical, and experimental values of $E(llps)$ obtained for the key set B with $m = 1000$. The analytical values were computed using the formulas provided in [LR82]. The table of $E(llps)$ values given in [LR82] was not used since it is provided as a function of the storage utilization and the total number of records (and it is too cumbersome to simulate file loadings accordingly). We observe that the two sets of values are in good agreement. Thus we conclude that under separate chaining the expected performance of the three measures, successful searches, unsuccessful searches and *llps*, can be achieved as claimed in our hypothesis.

b	$n/mb = 0.6$		$n/mb = 0.7$		$n/mb = 0.8$		$n/mb = 0.9$		$n/mb = 1.0$	
	exptl	anal	exptl	anal	exptl	anal	exptl	anal	exptl	anal
1	4.274	4.333	4.614	4.636	4.950	4.947	5.236	5.242	5.468	5.513
2	5.052	5.040	5.530	5.542	5.994	6.016	6.448	6.480	6.912	6.922
3	5.448	5.480	6.156	6.139	6.800	6.777	7.412	7.391	8.042	7.888
4	5.800	5.777	6.632	6.592	7.476	7.377	8.150	8.139	8.880	8.880
5	6.042	5.988	6.992	6.951	7.880	7.880	8.770	8.782	9.596	9.661
10	6.346	6.366	8.032	8.013	9.622	9.613	11.128	11.177	12.660	12.709

Table 4. Comparison of the expected *llps* under separate chaining hashing

Bounded Disorder file organization

Recently Litwin and Lomet proposed the Bounded Disorder (BD) file organization which uses a combination of hashing and tree indexing [LL86, LM86]. The index is organized as a regular B-tree and the multi-bucket data nodes at the leaf level are organized as small hash tables. Each data node has a single bucket designated as the overflow bucket, and stores the overflow records from the primary buckets of the hash table. A record to be inserted traverses the index as in a regular B-tree until the leaf level data node. Hashing is used to select the bucket within the data node, where the record is to be stored if the bucket is not already full. If the bucket is full, the record is stored in the overflow bucket of the node. When the overflow bucket becomes full, the node is said to have failed. The large data nodes reduce the size of the index and hence the entire index can be accommodated in the main memory. A random search can be accomplished in close to one disk access. In addition the scheme can support efficient range searching and is inherently dynamic.

Lomet proposed partial expansions for handling node failures, and provided an approximate analysis [LM86]. The preliminary results of an exact analysis of the bounded disorder file organization is provided in [RM88]. One of the measures used in the analysis is $T(n,m,b)$, defined as the probability of not having a node failure when n keys are hashed into a node with m primary pages each having a capacity b (this probability is a special case of the *table sufficiency index* first defined in [NY85] and analyzed in [RK88]). The expected storage utilization of a node etc. can be derived in terms of this probability. Our hypothesis can be tuned to the BD file organization as follows: If hashing functions are chosen at random from H_1 and n keys are hashed into m primary pages, a fraction $T(n,m,b)$ of the functions would not cause the node failure (and hence no expansion is required). To verify the hypothesis we simulated 500 loadings of data nodes for file B, and table 5 provides a comparison of the analytical and experimental results. The two results are in good agreement, and all most all experimental results are with in the 95% confidence limits of their corresponding analytical values (since the results agree very closely, we did not feel the necessity to give the confidence limits as given in figure 1). These observations and results suggest that the node expansion is not essential when a node failure occurs. Also, the space-time tradeoff techniques similar to those in [RM86, RL88] can be used to handle node failures in BD file organization. This topic is under further investigation and it is beyond the scope of this paper to go into further details in this regard.

$\frac{n}{mb}$	$m = 10$		$m = 20$	
	exptl	anal	exptl	anal
0.60	1.0000	1.0000	1.0000	0.9998
0.65	1.0000	1.0000	0.9980	0.9982
0.70	1.0000	0.9998	0.9800	0.9881
0.75	0.9980	0.9986	0.9280	0.9389
0.80	0.9880	0.9926	0.7580	0.7770
0.85	0.9580	0.9670	0.4620	0.4562
0.90	0.8920	0.8809	0.1320	0.1356
0.95	0.7120	0.6707	0.0160	0.0117
1.00	0.3300	0.3397	0.0000	0.0001
1.05	0.0820	0.0703	0.0000	0.0000
1.10	0.0000	0.0008	-	-

Table 5. Comparison of the experimental and analytical values of $T(n,m, 10)$

Dynamic hashing schemes

It is straightforward to adapt our hypothesis to dynamic hashing schemes such as linear hashing[LT80] and extendible hashing[FN79]. For linear hashing, once $f_{c,d}$ is chosen randomly, the sequence of hashing functions to be used for successive file expansions are $h_0 = f_{c,d} \bmod m$, $h_1 = f_{c,d} \bmod 2m$, $h_2 = f_{c,d} \bmod 4m$, $\cdots$ etc. That is, h_0 is chosen randomly from H_1 and the sequence of hashing functions is completely defined by h_0 and the initial file size m.

Extendible hashing is defined and analyzed assuming the directory size to be a power of two and that the hashing function yields a sequence of bits which map the keys into directory entries. At a given point of time when d of the least significant bits are used the directory size is 2^d. The above sequence of hashing functions, $h_0, h_1, \cdots$ can be used to map the keys into the directory entries and thus it is not necessary to require a hashing function which generates a sequence of bits. Moreover, the directory size need not be a power of two and any value of m can be used (however, the directory can not shrink below the size m, if m is not a multiple of two).

5. Conclusions and further work

Although papers continue to appear providing probabilistic analysis of hashing schemes, there has been little work in explicitly linking the analytical results with practical performance on real life files. We showed, why the division method performs well, under Lum's model of evaluating hashing functions. We formulated and justified an hypothesis, which can be readily adapted to different hashing schemes, that by choosing hashing functions at random from a class of functions the analytically predicted performance of the various hashing schemes can be achieved in practice. Experimental results for several test files under different hashing schemes were reported, all of which strongly support the hypothesis. In order to make the preliminary version of the paper concise and not clutter with too many tables, only a representative fraction of the results were presented.

One area of further experimental work is with reference to special key distributions. We experimented with an exponentially distributed key set (generated artificially) and the experimental results support the hypothesis. However, linearly dependent key sets give superior performance than analytically predicted. A natural question which arises is, is it possible to construct a key set for which the hypothesis does not hold? One way of constructing such a key set is to pass the universe A through a series of *filters*, each of which allows only those keys which hash to a particular address (i.e., each *filter* is a hashing function chosen at random from H_1). The keys coming out of such a *filter* will constitute the worst case key set for the hashing function. It is interesting to see, if the key set becomes progressively "worse" (for the general class of hashing functions) as it becomes smaller with successive filtering. It is clear that the relative values of a(the size of the universe), n and m will play a role in the answer. Can this question be answered theoretically?

We have carried out probabilistic investigation, as all the analytical performance results of hashing are, of *universal*$_2$ class of hashing functions. This is in contrast with the number theoretic investigations in [CW79], and lower bound/upper bound results is [MR83, MR84]. Gonnet showed that the fears of worst case behavior of hashing are unfounded by proving that the cumulative probability distribution of the *llps* is a negative double exponential and its expected value is only $O(\Gamma^{-1}(m))$ (a very slowly growing function, inverse factorial) [GN81]. We believe, our experimental results supporting this are the most significant of all the experimental results. In view of this, how significant probabilistically are Mairson's lower bound results for the program size of bounded retrieval schemes? Is it possible to derive probabilistic bounds, such as the "expected program size of searching a table" (similar to Gonnet's expected worst case behavior)?

Acknowledgements

Some of the programming was done by Rob Jaksa and Dave Robinson.

References

[BR71] Burstein,H. *Attribute Sampling: Tables and Explanations*. New York: McGraw-Hill, 1971.

[CW79] Carter, L.J. and Wegman, M.L. *Universal classes of hash functions*. Journal of Computer and System Sciences, 18, 2(1979), 143 - 154.

[DS75] Deutscher, R.F., Sorenson, P.G. and Tremblay, J.P. *Distribution dependent hashing functions and their characteristics,* Proc. ACM-SIGMOD Intern'l Conf. on Management of Data, (1975), 224-236.

[FK84] Fredman, M.L., Komlos, J. and Szemeredi, E. *Storing a sparse table with O(1) worst case access time,* Journal of the ACM, 31, 3(1984), 538 - 544.

[FL68] Feller, W. *An Introduction to Probability Theory and its Applications*. Vol. 1. New York: John Wiley, 1968.

[FN79] Fagin,R., Nievergelt,J., Pippenger,N. and Strong,H.R. *Extendible hashing - a fast access method for dynamic files*. ACM Trans. on Database Systems, 4, 3(1979), 315 - 344.

[FR71] Freund, J.E. *Mathematical Statistics*. Englewood Cliffs, New Jersey: Prentice-Hall, 1971.

[GB78] Guibas, L.J. *The analysis of double hashing*. Journal of Computer and System Sciences, 16, (1978), 226 - 274.

[GN81] Gonnet,G.H. *Expected length of the longest probe sequence in hash code searching*. Journal of the ACM, 28, 2(1981), 289 - 304.

[KN81] Knuth, D.E. *The Art of Computer Programming, Vol. 2*. Addison-Wesley, Reading, Massachusetts, 1981.

[KN74] Knuth, D.E. *The Art of Computer Programming, Vol. 3*. Addison-Wesley, Reading, Massachusetts, 1974.

[KN75] Knott, G.D. *Hashing functions,* The Computer Journal, 18, 3(1975), 265-278.

[LK84] Larson, P.-A. and Kajla, A. *File organization - implementation of a method guaranteeing retrieval in one access*. Comm. of the ACM, 27,7 (1984), 670 - 677.

[LL86] Litwin, W. and Lomet, D.B. *The bounded disorder access method,* Proc. 2nd Intn'l Conference on Data Engineering (Los Angeles, CA, 1988), 38-48.

[LM71] Lum, V.Y., Yuen, P.S.T., and Dodd, M. *Key-to-address transform techniques: A fundamental performance study on large existing files,* Comm. of the ACM, 14, 4(1971), 228-239.

[LM73] Lum, V.Y. *General performance analysis of key-to-address transformations methods using an abstract file concept,* Comm. of the ACM, 16, 10(1973), 603-612.

[LM86] Lomet, D.B. *A simple bounded disorder file organization with good performance*. Wang Institute of Graduate Studies, Technical Report No. TR-86-13, 1986 (under review for publication in ACM Trans. on Database Systems).

[LR82] Larson, P.-A.. *Expected worst-case performance of hash files*. The Computer Journal, 25, 3(1982), 347 - 352.

[LR83] Larson,P.-A. *Analysis of uniform hashing*. Journal of the ACM, 30, 4(1983), 805 - 819.

[LT80] Litwin,W. *Linear hashing: A new tool for files and tables addressing*. Proc. 6th Intern'l Conf. on Very Large Databases, (Montreal, 1980), 212 - 223.

[MR83] Mairson,H.G. *The program complexity of searching a table*. Proc. 24th Symposium on Foundations of Computer Science, IEEE Computer Society, 1983, 40 - 47.

[MR84] Mairson,H.G. *The program complexity of searching a table*. Ph.D. Thesis, Department of Computer Science, Stanford University, 1984.

[NY85] Norton, R.M. and Yeager, D.P. *A probability model for overflow sufficiency in small hash tables*. Comm. of the ACM, 28, 10(1985), 1068 - 1075.

[RK88] Ramakrishna, M.V. *An exact probability model for finite hash tables*. Proc. 4th Intn'l Conference on Data Engineering (Los Angeles, Feb 2-4, 1988), 362 - 368.

[RL88] Ramakrishna, M.V. and Larson, P.A. *File organization using composite perfect hashing*. (to appear) ACM Trans. on Database Systems.

[RM86] Ramakrishna,M.V. *Perfect hashing for external files*. Ph.D. Thesis, Department of Computer Science, University of Waterloo, Research Report CS-86-25, 1986.

[RM87] Ramakrishna,M.V. *Computing the probability of hash table/urn overflow*. Comm. in Statistics: Theory and Methods, A16, 11(1987), 3343-3353.

[RM88] Ramakrishna,M.V., and Mukhopadhyay, P. *Analysis of bounded disorder file organization,* (to appear) Proc. ACM Symp. on Principles of Database Systems (Austin, TX, Mar 21-23, 1988).

[SD80] Sarwathe, D.V. *A note on universal classes of hash functions* Information Processing Letters, 10, 1(1980), 41 - 45.