Edgar H. Sibley
Panel Editor

Bloom filter technique of hashing finds several applications, such as in efficient maintenance of differential files, space efficient storage of dictionaries, and parallel free-text searching. The performance of hash transformations with reference to the filter error rate is the focus of this article.

Practical Performance Of Bloom Filters and Parallel Free-Text Searching

M. V. Ramakrishna

The Bloom filter technique of hash coding finds several applications. Consider the problem of reducing disk accesses to a differential file, as discussed by Gremillion [3] and Mullin [5]. A differential file is essentially a separate file which contains records that are modified in the main file. The main file itself is not modified in order to preserve integrity. This implies that each reference to the main file must first access the differential file in order to check whether or not the record has been modified. If the record has not been modified, the main file is searched. Usually, the record would not be found in the differential file. Nonetheless, it has to be searched every time. The searching of the differential file can be greatly reduced using the following Bloom filter technique.

The filter consists of m bits of memory initialized to zero and k hash transformations on the primary key of the files. For a given key, each hash transformation produces an integer in the range 0 to $m - 1$. Initially, the differential file is empty. Whenever a record in the main file is to be modified, the modified record is entered into the differential file. At the same time, each of the k transformations are applied to the key and the corresponding bits of the bit vector is set to 1. Note that no changes are made to the main file. On subsequent searches, the transformations are applied to the key. If all the corresponding bits are not set in the bit vector, then the record could not have been stored in the differential file (and hence there is no need to access the differential file). If all k bits for the record in question are set, however, it does not imply that the record is stored in the differential file. The differential file has to be searched to see if the record is present. If the record is not found in the differential file, when all k bits have been set by chance collisions with other keys, a *filter error* is said to have occurred. The filter error rate and the performance of hash transformations is the topic of this article.

Let n be the number of entries in the differential file, and α the fraction of records in the differential file. The probability of a filter error occurring can be shown to be [5]

$$P_e = (1 - \alpha)\{1 - (1 - 1/m)^{nk}\}^k.$$

This analytical result assumes that each of the transformations is equally likely to set each of the m bits, i.e., ideal in some sense. The simulation results reported in the literature indicate that the filter error rate is larger than the analytical results. Table I, extracted from [3, 5], compares the filter error rate of simulations and analytical results. In Mullin's words,

> An alarming feature of the simulation is that the probability of a filter error is higher than the theory predicts. It may, indeed, be difficult to develop good transformations, but Table I shows that perfect transformations with a filter size of 49,152 bits yield the same performance as the actual transformations with 65,536 bits. If space is important, one might wish to investigate transformations a bit more [5].

This article shows that the theoretical performance can be achieved by the right choice of transformations. My investigations into a class of transformations, which give theoretically predicted filter error rates with real-

TABLE I. Filter Error Rate (from [3, 4])

Filter size	Simulation k		Theory k	
m	4	6	4	6
24,576	0.356	0.431	0.214	0.302
32,768	0.210	0.252	0.109	0.142
49,152	0.078	0.082	0.036	0.036
65,536	0.035	0.029	0.014	0.011

$n = 7000$, $\alpha = 0$

life test files, are reported.

Bloom filter techniques find other applications. Free-text search systems allow the user to search the text of the stored documents for arbitrary combinations of words, without the help of any indexing. One problem with the free-text database systems, which hinders its wide usage, is that they make computational demands beyond the practical limits of conventional computers.

Stanfill and Kahle presented an implementation of free-text searching on the Connection Machine that takes advantage of the possibilities offered by massively parallel computers [8]. The data structure they use and the search operation follow the same principle as Bloom filtering. The bit array is referred to as a surrogate table. A document is said to have been stored in the table when the transformations are applied to the words of the document and the corresponding bits are set to one. To search if a word from the given query is present in the document stored, the transformations are applied to the word and the corresponding bits are checked in the surrogate table. If all the bits are not set, then the word is not contained in the document. Otherwise, with a high probability, the word is contained in the document. The way the parameters of the surrogate table are chosen make the probability of error quite small. The interested reader can refer to [8] for further details of free-text searching.

The same data structure has been used for space efficient storage of dictionaries (see Jon Bentley's column, "Programming Pearls" [1]). Instead of storing the words as such in a dictionary for use in spelling checking, the idea is to store the words in a bit array similar to the surrogate table which has to be sufficiently large. Apart from memory-space savings with this technique, searching also becomes very fast. There is still a small probability of error though. Further details of error rate are discussed by Nix in [6]. The transformations referred to as "perfect" by Mullin are referred to as "random" by Nix. The transformations presented in this article satisfy the requirements of this application also, and, in fact, we have used the UNIX® dictionary as a test file for the experiments.

UNIX is a registered trademark of AT&T Bell Laboratories

[1] A conversion method based on a radix-R representation of alphanumeric strings was used. Each of the R characters of the alphabet is assigned an integer 1 through R. Then each alphanumeric string is treated as a base-R integer and is converted into the corresponding decimal number. If the number so obtained is larger than the prime p, its modulus to p is taken. Duplicates, if any, are deleted. The values used for the experiments are, $R = 36$ and $p = 2\ 100\ 000\ 011$. Further details about the conversion procedures can be found in [7].

Recently, hardware implementation of the technique has been suggested for high-speed event counting and classification [4]. Such hardware finds applications in areas such as signal processing, process monitoring and control, and computer communications network monitoring and control.

$$h_{c,d}(x) = ((cx + d) \bmod p) \bmod m, \quad \text{and}$$

$$H_1 = \{h_{c,d}(\,) \mid 0 < c < p,\ 0 \le d < p\}.$$

A Class of Hash Transformations

Consider the class of hash transformations defined as follows. The keys are assumed to be integers drawn from a universe A, $A = \{1, 2, \ldots, p - 1\}$, and let B denote the range of hash addresses, $B = \{0, 1, \ldots, m - 1\}$. We assume p is a prime, without loss of generality. Define transformations $h_{c,d}$ which map A into B, and the class H_1 as,

The results and the details of the experiments presented in the next section illustrate that transformations chosen at random from the class H_1 yield the theoretical performance of the Bloom filters. (The class H_1 is a *universal*$_2$ class of hashing functions. Definitions and theoretical investigations of *universal*$_2$ classes of hashing functions may be found in [2].)

Experimental Results and Concluding Remarks

To test the performance of the Bloom filters under the transformations just discussed, I ran several simulations of differential file loadings. In order to enable direct comparison of the results, the parameters of the Bloom filter given in Table I of [5] are used (which in turn corresponds to simulation results in [3]). Three different real-life files are used in the experiments: the dictionary on UNIX®, call numbers from a library, and the userids from a large time sharing installation. The keys are first converted into integers.[1]

Each simulation run consisted of randomly choosing k transformations from the class H_1. This was accomplished by generating $2k$ pseudo–random numbers, each pair defining a hash transformation. The m bits of the filter are cleared initially. The k transformations are applied to each of the n ($n = 7000$) keys, and the corresponding bits of the filter are set to one. After all the n keys are processed, the next 1000 keys from the file are used to determine the fault rate of the filter. Each key is transformed by the k transformations, and if all the k bits are set, it is counted as a filter failure (note that none of the 1000 keys were used before in setting the bits). The fault rate for the set of k hashing functions is determined as a fraction of the 1000 keys tried. This experiment is repeated for 100 different sets transformations, and the mean and the standard deviations of the fault rates for the parameter combination is noted. The whole procedure is repeated for different parameter sets and the results are tabulated in Table II which corresponds to the file of user ids. We observe that the experimental results are in agreement with the analyti-

TABLE II. Error Rate with Transformations from Class H_1

Filter size m	$k=4$ Theory	$k=4$ Simulation	$k=4$ Std. Dev.	$k=6$ Theory	$k=6$ Simulation	$k=6$ Std. Dev.
24,576	0.216	0.216	0.013	0.302	0.303	0.016
32,768	0.109	0.107	0.010	0.142	0.142	0.014
49,152	0.036	0.037	0.006	0.036	0.037	0.005
65,536	0.014	0.015	0.004	0.011	0.011	0.003

$n = 7000$, $\alpha = 0$

cal results, and the two are less than a standard deviation apart. Similar results are obtained with other files. We thus conclude that by choosing hash transformations randomly from the class H_1 as described, the theoretical error rate can be achieved in practice.

REFERENCES

1. Bentley, J. A spelling checker. *Commun. ACM 28*, 5 (May 1985), 456–462.
2. Carter, L. J., and Wegman, M. L. Universal classes of hash functions. *J. Comput. Syst. Sci. 18*, 2 (1979), 143–154.
3. Gremillion, L. L. Designing a Bloom filter for differential access. *Commun. ACM 25*, 7 (July 1982), 600–604.
4. McKenney, P. E. High-speed event counting and classification using a dictionary hash technique. In *Proceedings of the Intl. Conference on Parallel Processing* (Chicago, Ill., Aug. 8–12, 1989), pp. III-71–III-75.
5. Mullin, J. K. A second look at Bloom filters. *Commun. ACM 26*, 8 (Aug. 1983), 570–571.
6. Nix, R. Experience with a space efficient way to store a dictionary. *Commun. ACM 24*, 5 (May 1981), 297–298.
7. Ramakrishna, M. V. Perfect hashing for external files. Res. Rep. CS-86-25, Ph.D. Thesis, Dept. of Computer Science, Univ. of Waterloo, 1986.
8. Stanfill, C., and Kahle, B. Parallel free-text search on the Connection Machine System. *Commun. ACM 29*, 12 (Dec. 1986), 1229–1239.

CR Categories and Subject Descriptors: E.2 [**Data**]: Data Storage Representations—*hash table representations*; H.2.2 [**Database Management**]: Physical Design—*access methods*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms: Design, experimentation, performance, theory

Additional Key Words and Phrases: Bloom, differential file, free-text databases, filter, hashing

ABOUT THE AUTHOR:

M. V. RAMAKRISHNA is currently an assistant professor in the Computer Science Department at Michigan State University. His research interests include hashing techniques, database management systems, and parallel processing; and he is a member of the IEEE Computer Society, ACM, ACM SIGMOD, ACM SIGOPS, and ACM SIGACT. Author's Present Address: Computer Science Department, Michigan State University, East Lansing, MI 48824. CSnet: rama@cpswh.cps.msu.edu