Information Systems Vol. 18, No. 1, pp. 63–74, 1993
Printed in Great Britain

0306-4379/93 $6.00 + 0.00
Pergamon Press Ltd

MULTI-DIRECTORY HASHING

SAKTI PRAMANIK, ANASTASIA ANALYTI, HENRY DAVIES and HSIAO-YU CHOU
Computer Science Department, Michigan State University, East Lansing, MI 48824, U.S.A.

(Received 10 September 1990; in revised form 10 August 1992)

Abstract—We present a new dynamic hashing scheme for disk-based databases, called Multi-Directory Hashing (MDH). MDH uses multiple hash directories to access a file. The size of each hash directory grows dynamically with the file size. The advantages of MDH are enhanced concurrency, improved bucket utilization and smaller total directory size than single-directory hashing. The expected utilization of MDH increases monotonically and approaches 100% as the number of hash directories increases. A variation of MDH, called Main Memory Multi-Directory Hashing (MM-MDH), is also described. MM-MDH achieves optimal search time in main memory databases. The performance of both methods is analyzed through theoretical and experimental results.

Key words: Multi-Directory Hashing, extendible hashing, parallel processing, main memory databases, performance analysis

1. INTRODUCTION

Hashing is a common technique to speed up search in database applications. Several hashing methods for dynamic files have been proposed since the mid 1970's including extendible hashing [1], linear hashing [2, 3] and dynamic hashing [4]. An overview of dynamic hashing schemes is presented in [5, 6]. Attractive features of extendible hashing are the absence of overflow buckets, the simple structure of its hash directory and the ability to grow and contract the hash directory gracefully as records are inserted into and deleted from the file. Extendible hashing requires at most one disk access for a record search. The problems of extendible hashing are that the directory size grows by doubling itself and that the expected bucket utilization is as low as 69% [1]. In linear hashing, on the other hand, the directory grows gradually, one entry at a time. Overflow buckets are allowed and they are linked in a chain. The bucket utilization is determined by the user. The main drawback of the method is that high bucket utilization results in long overflow bucket chains and consequently in increased search time [7].

In Multi-Directory Hashing (MDH), the data file is accessed through several hash directories. The records of the file are distributed among the different hash directories which can be accessed in parallel. Therefore, MDH is suitable for parallel processing. Each hash directory of MDH expands in the same way as the directory of extendible hashing. We have chosen extendible hashing for constructing each hash directory because extendible hashing requires at most one disk access for a record search (no overflow buckets). MDH achieves improved expected bucket utilization and smaller expected directory size than extendible hashing. The expected bucket utilization of MDH approaches 100% and the oscillation of the bucket utilization reduces as the number of hash directories increases. Parallel access of the hash directories reduces the search time significantly.

Main Memory Multi-Directory Hashing (MM-MDH) is a variation of MDH which obtains optimal search time (at most one key comparison for a record search) in main memory databases. The expected total directory size of this method is smaller than that of extendible hashing. Assuming a uniform distribution of the records over the hash directories, the expected total directory size of this method decreases as the number of hash directories increases.

The organization of the rest of the paper is as follows: Section 2 describes MDH. Section 3 analyzes the bucket utilization of MDH. Section 4 analyzes the directory utilization of MDH. Section 5 describes an application of MDH to range queries. Section 6 describes MM-MDH and analyzes its performance. Section 7 contains the concluding remarks.

2. MULTI-DIRECTORY HASHING

In Multi-Directory Hashing (MDH), a set of hash directories is used to access a file. The number of hash directories is fixed and each directory grows and shrinks dynamically in the same way as the directory of extendible hashing. When MDH uses a unique hash directory, it coincides with extendible hashing. We present the insertion algorithm of MDH which is simple and requires no inter-directory restructuring. We assume that the hash function produces binary numbers and can generate more bits as the file expands. One such hash function is given in [6]. All hash directories have the same structure and cover the whole address space. A record can be inserted in any of the hash directories based on the availability of space in the buckets corresponding to its hash address. We say that a bucket corresponds to a hash address if the bit sequence that is used for addressing the bucket is prefix of the hash address.

Insertion algorithm 1 *(MDH):*

Step 1: Obtain the hash address h of the record to be inserted.
Step 2: If there is space in a bucket corresponding to hash address h in any of the hash directories, insert the record into the bucket and stop.
Step 3: If there is one fully occupied bucket corresponding to hash address h which is pointed to by more than one directory entries, split the bucket into two as in extendible hashing and go to Step 2.
Step 4: Find the smallest directory of which could be more than one, double it as in extendible hashing and go to Step 2.

The following example illustrates insertion algorithm 1. Assume that after a series of insertions, the directory looks as in Fig. 1(a) with bucket 0 fully occupied and bucket 1 having space for one more record. Insert another record:

Step 1: Get its hash address which is the bit sequence 0... or 1....
Step 2: Since there is space in bucket 1 of directory 1, insert the record into the bucket [Fig. 1 (b)] and stop.

Insert another record:

Step 1: Get its hash address. The hash address is 1....
Step 2: Since there is no space in any of the buckets corresponding to hash address 1..., continue to Step 3.
Step 3: Since the bucket 0 is shared by the directory entries 0 and 1, split bucket 0 into two buckets [Fig. 1(c)] and go to Step 2.
Step 2: Since there is space in bucket 2 of directory 0, insert the record in bucket 2 and stop.

After another series of insertions the directories look as in Fig. 1(d). All the allocated buckets are fully occupied. Insert another record:

Step 1: Get its has address. The hash address is 01....
Step 2: Since there is no space in any of the buckets corresponding to hash address 0..., continue to Step 3.
Step 3: Since there are no directory entries sharing a bucket that corresponds to the hash address 0..., continue to Step 4.
Step 4: Double directory 0. Split bucket 0 into two buckets [Fig. 1(e)] and go to Step 2.

MDH is suitable for parallel processing because each hash directory covers the whole address space and no relocation of records among the different hash directories is needed.

3. BUCKET UTILIZATION OF MDH

This section analyzes the bucket utilization of MDH through theoretical and experimental results. Bucket utilization is the ratio of inserted records to the space reserved for the records. A lower bound on the expected bucket utilization of MDH is given by Proposition 2. This lower

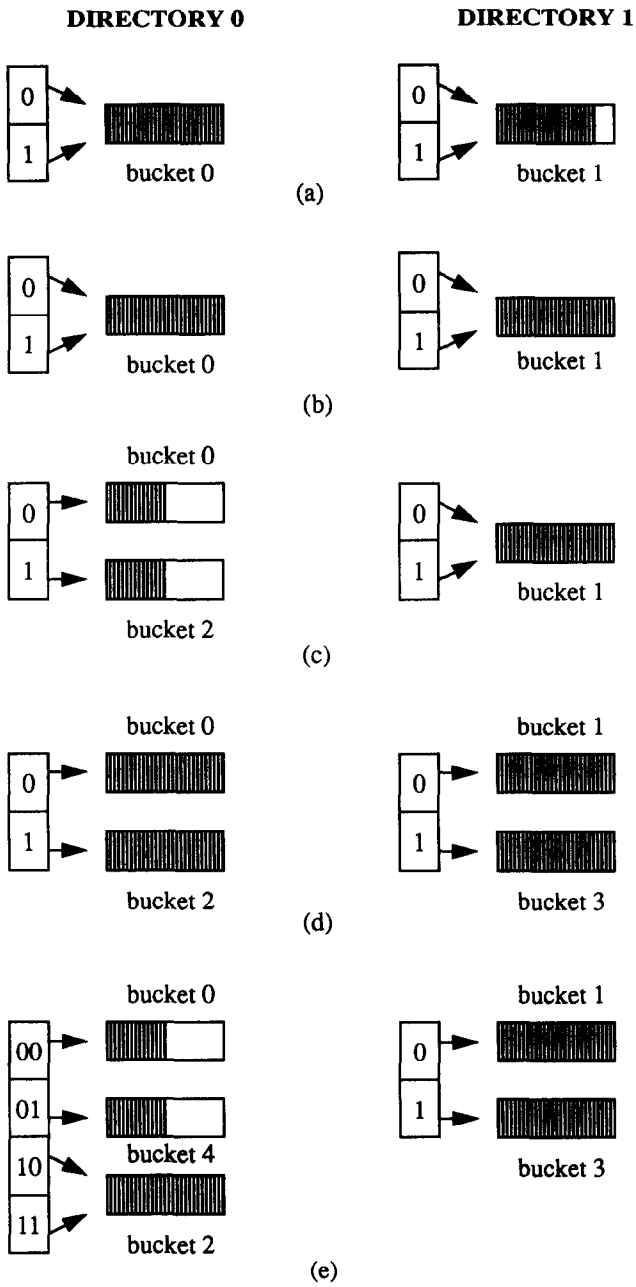

Fig. 1. Insertion algorithm of MDH.

bound equals to the expected bucket utilization of a variation of extendible hashing called extendible hashing with overflow chaining (EHOC). A brief description of EHOC follows. We will prove that the expected bucket utilization of MDH is better than that of EHOC.

EHOC is a variation of extendible hashing. A bucket in extendible hashing corresponds to a chain of buckets in EHOC. In EHOC, when a new record is to be inserted and all buckets in the chain corresponding to its hash address are full, a new bucket is created and linked to the last bucket in the chain. When a chain of buckets exceeds a certain length, all records in the bucket are rehashed using one more bit from their hashed address. The directory of EHOC grows similarly to the directory of extendible hashing. When the maximum chain length is 1, EHOC coincides with extendible hashing. We compare MDH with n hash directories with EHOC with maximum chain length n. First, we compute the expected bucket utilization of EHOC with maximum chain length n, denoted as $U_{\text{EHOC}}(N, n, b)$, where N is the number of inserted records and b is the bucket size.

Lemma 1:

$$U_{\mathrm{EHOC}}(N,n,b) = N\left[b\sum_{r=1}^{\infty}2^{r}\sum_{i=1}^{n}\sum_{k=nb+1}^{N}\sum_{x=(i-1)b+1}^{nb}\binom{N}{k}(2^{-r})^{k}(1-2^{-r+1})^{N-k}\binom{k}{x}\right]^{-1}$$

$$\approx \ln 2\left[b\sum_{i=1}^{n}\sum_{k=nb+1}^{N}\sum_{x=(i-1)b+1}^{nb}\frac{2^{-k+1}}{(k-1)k}\binom{k}{x}\right]^{-1}$$

where N, n, b are positive integers and $N > nb$.

Proof:

The proof is given in the Appendix. □

The following proposition compares the expected bucket utilization of MDH with n hash directories and bucket size b, denoted as $U_{\mathrm{MDH}}(N,n,b)$, with that of EHOC with maximum chain length n and the same bucket size. N denotes the number of inserted records.

Proposition 1:

$$U_{\mathrm{MDH}}(N,n,b) > U_{\mathrm{EHOC}}(N,n,b)$$

where N, n, b are positive integers and $n > 1$.

Proof:

The address of a bucket is the greatest common prefix of the hash addresses of all records that can be inserted into the bucket. In EHOC, when a bucket chain splits two buddy chains are created. Two buckets that belong to two buddy chains are called buddy buckets.

Assume that every bit of the hash address of a record has equal probability of being 0 or 1 and that the bucket size is large. When a bucket in MDH splits, two almost half-full buckets are created. These two MDH buckets correspond to two EHOC buddy buckets with the same address as the MDH buckets. The utilization of the initially half-full MDH buckets increases from 50 to 100% as more records are inserted. By contrast, the utilization of the EHOC buckets increases from 0 to 100% as more records are inserted. The reader can easily verify that a bucket in MDH splits with about twice the rate that two additional buddy buckets are allocated in EHOC. Proposition 1 follows. □

Proposition 2:

A lower bound on the expected bucket utilization of MDH is given by the expression

$$U_{\mathrm{EHOC}}(N,n,b) > \ln 2\left[b\sum_{i=1}^{n}\sum_{k=nb+1}^{N}\sum_{x=(i-1)b+1}^{nb}\frac{2^{-k+1}}{(k-1)k}\binom{k}{x}\right]^{-1}$$

where N, n, b are positive integers and $n > 1$.

Proof:

It follows directly from Lemma 1 and Proposition 1. □

Note that the expected bucket utilization of MDH with one hash directory equals the expected bucket utilization of extendible hashing which is about 69% [1, 4]. The bucket utilization of MDH increases monotonically with the number of hash directories, n. This is because when a new record is to be inserted and all n buckets corresponding to its hash address are full, only one bucket splits. Thus, as the number of hash directories increases, the percentage of wasted space declines. The bucket utilization of MDH approaches monotonically 100% with the number of hash directories.

The above analysis is verified by our experimental results. In the experiment, 100,000 records with uniformly distributed hash addresses were inserted into a file according to insertion algorithm 1. The expected bucket utilization was computed by averaging the bucket utilizations computed at every 1000 insertions. Figure 2 shows the expected bucket utilization (over the life cycle of the file) for different bucket sizes. We see that the expected bucket utilization of MDH increases

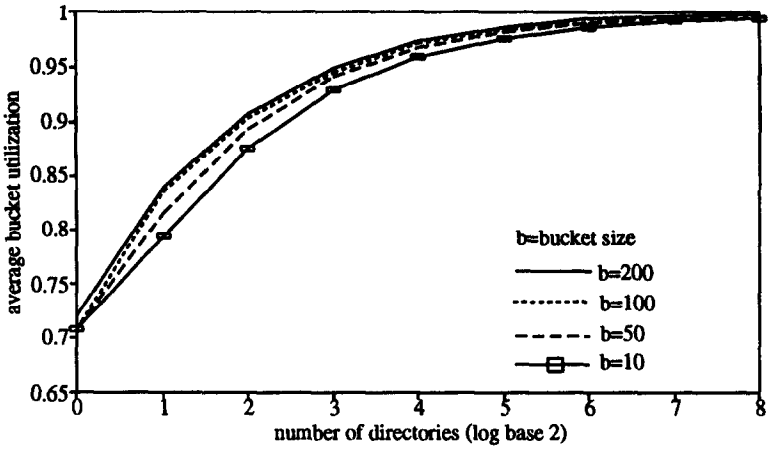

Fig. 2. Expected bucket utilization of MDH.

monotonically with the number of hash directories and it becomes close to 100% when the number of directories is 256. We have also calculated the expected bucket utilization by averaging the utilizations computed at every 100 and 500 insertions and the result was about the same. The reader might have noticed that the expected bucket utilization of MDH for 1 hash directory is about 71% instead of 69% as it is theoretically expected. This is because, even though the expected bucket utilization of extendible hashing is 69%, its instantaneous bucket utilization fluctuates between 50 and 100% [1]. Thus, the expected bucket utilization of extendible hashing can be easily overestimated or underestimated from experimental results. The oscillation of the bucket utilization of MDH is discussed at the end of this section.

Figure 3 compares the experimentally computed expected bucket utilization of MDH with n hash directories with the theoretically estimated expected bucket utilization of EHOC with maximum bucket chain length n, where n takes the values 2, 4, 8, . . ., up to 250. The bucket size is 50. The figure verifies the improved bucket utilization of MDH over EHOC. Moreover, EHOC does not allow for concurrent accesses as MDH does.

In extendible hashing, the instantaneous bucket utilization fluctuates between 50 and 100%. This is because when the hash addresses are uniformly distributed, all the buckets of extendible hashing split at about the same time. Figure 4 shows that MDH with 8 hash directories reduces the oscillation of the bucket utilization. MDH reduces the oscillation of the bucket utilization because when a new record is to be inserted and all n buckets corresponding to its hash address are full, only one bucket splits.

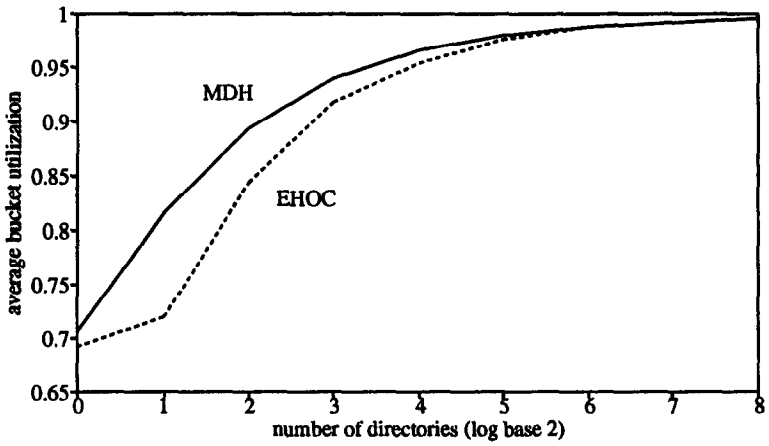

Fig. 3. Expected bucket utilization comparison of MDH and EHOC.

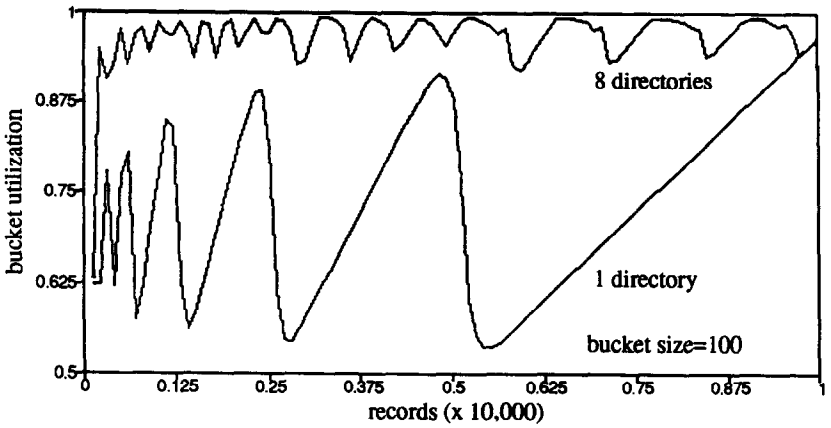

Fig. 4. Bucket utilization of extendible hashing and MDH with 8 directories.

4. TOTAL DIRECTORY SIZE OF MDH

In this section we prove that the expected total directory size of MDH is smaller than that of extendible hashing. We define as MDH optimal total directory size, the smallest total directory size of all multi-directories that can accommodate the set of inserted records. We show that the total directory size of MDH is close to MDH optimal total directory size.

Lemma 2:

Let N denote the number of records inserted. Let $D_{\mathrm{MDH}}(N, n, b)$ denote the expected total directory size of MDH with n hash directories and bucket size b. Let $D_{\mathrm{EH}}(N, b')$ denote the expected total directory size of extendible hashing with bucket size b'. Then

$$D_{\mathrm{MDH}}(N, n, b) \leqslant n^*(D_{\mathrm{EH}}(N, nb) + 1)$$

where n, N, b are positive integers.

Proof:

For a given set of records, the size of the largest hash directory of MDH with n hash directories and bucket size b equals the directory size of extendible hashing with bucket size nb. Since the total directory size of MDH consists of the size of the hash directories plus the size of an index that points to the hash directories, Lemma 2 follows. □

Proposition 3:

The expected total directory size of MDH is asymptotically smaller than that of extendible hashing when the number of hash directories is greater than 1.

Proof:

Frajolet has given a coarse approximation for the expected directory size of extendible hashing [8]: $D_{\mathrm{EH}}(N, b') \approx 3.92(N^{1+1/b'}/b')$, where N is the number of inserted records and b' is the bucket size. From Lemma 2, it follows that $D_{\mathrm{MDH}}(N, n, b) \leqslant n(D_{\mathrm{EH}}(N, nb) + 1)$, where n is the number of hash directories of MDH.

Thus,

$$D_{\mathrm{MDH}}(N, n, b) \leqslant n\left(3.92\frac{N^{1+1/(nb)}}{nb}\right) + n = 3.92\frac{N^{1+1/(nb)}}{b} + n < 3.92\frac{N^{1+1/b}}{b},$$

for $n > 1$ and large enough N. Since the expected directory size of extendible hashing with bucket size b is $3.92(N^{1+1/b}/b)$, the expected total directory size of MDH is smaller than that of extendible hashing for large enough number of inserted records. □

Proposition 4:

The MDH optimal total directory size is always less than or equal to the directory size of extendible hashing for the same set of inserted records.

Proof:

Consider the extendible hashing directory for a given set of records. A multi-directory of the same size with n hash directories can be constructed as follows: create two hash directories of half the size of the extendible hashing directory. Enter the even-numbered entries of the extendible hashing directory in one and the odd-numbered entries in the other. Do the same for these half size directories. Continue creating directories until n hash directories have been created. The multi-directory that results this way has the same directory size as the extendible hashing directory. Therefore, the MDH optimal total directory size is less than or equal to the directory size of extendible hashing for the same set of records. □

The following proposition will be used to show that the total directory size of MDH is close to optimal. Note that, for a given set of records X, the size of the largest hash directory of MDH with n hash directories and bucket size b, denoted by $D_X(n, b)$, equals the directory size of extendible hashing with bucket size nb.

Proposition 5:

Let n be the number of hash directories, b the bucket size, X the set of inserted records and $D_X(n, b)$ the size of the greatest hash directory of MDH. The minimum number of hash directories of size $D_X(n, b)$ needed is given by:

$$\max\left\{\lceil(entry(2i) + entry(2i+1))/b\rceil \mid i = 0, 1, \ldots, \frac{D_X(n,b)}{2} - 1\right\} - n$$

where $entry(i)$ is the number of records of X corresponding to entry i of a directory with size $D_X(n, b)$.

Proof:

Let

$$B = \lceil(entry(2i^*) + entry(2i^* + 1))/b\rceil$$
$$= \max\left\{\lceil(entry(2i) + entry(2i+1))/b\rceil \mid i = 0, 1, \ldots, \frac{D_X(n,b)}{2} - 1\right\}$$

be the number of buckets required to store the records corresponding to entries $2i^*$ and $2i^* + 1$ of a directory with size $D_X(n, b)$. It is easy to verify that $entry(2i^*)/b \leqslant n$, $entry(2i^* + 1)/b \leqslant n$ and $n < B \leqslant 2n$. To show that $B - n$ is the minimum number of directories with size $D_X(n, b)$ needed, we will first show that $B - n$ directories of that size are adequate. We will consider two cases:

Case 1: Assume that records of set X can fill all $2(B - n)$ buckets corresponding to entries $2i^*$ and $2i^* + 1$ of the directories with size $D_X(n, b)$. Then, the number of more buckets that are needed is $B - 2(B - n) = 2n - B$ which equals the number of the remaining directories $n - (B - n) = 2n - B$. Thus, $B - n$ directories of size $D_X(n, b)$ are adequate.

Case 2: Assume that the number of records corresponding to any one of the entries $2i^*$ and $2i^* + 1$, is less than $b(B - n)$. Since the number of records corresponding to the other entry is at most nb, the number of buckets that are needed to store the records corresponding to entries $2i^*$ and $2i^* + 1$ is less than $(B - n) + n$. Thus, $B - n$ directories of size $D_X(n, b)$ are sufficient.

We will now show that $B - n - 1$ directories of size $D_X(n, b)$ are insufficient. Let $B - n - 1$ directories hold two buckets for the pair of entries $2i^*$ and $2i^* + 1$. Then, the number of more buckets that are needed is $2n - B + 2$ whereas the number of remaining directories is $2n - B + 1$. Thus, $B - n - 1$ directories of size $D_X(n, b)$ are insufficient. □

Using Proposition 5, we have developed an insertion algorithm which achieves MDH optimal total directory size for a given file X. The total directory size of MDH will be compared with the optimal total directory size that results from this algorithm.

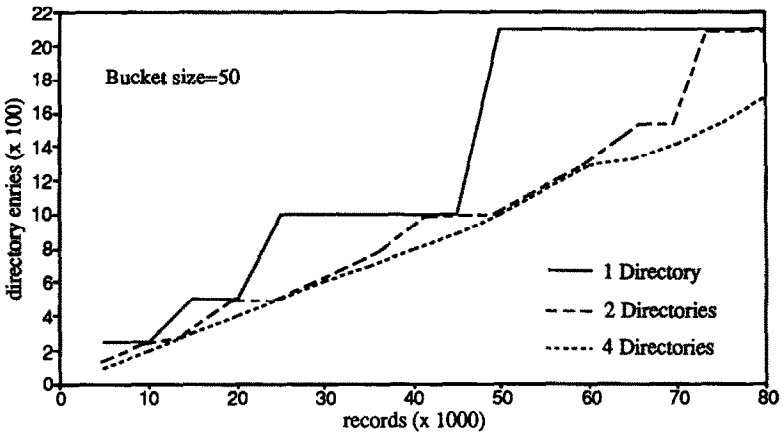

Fig. 5. Total directory size of insertion algorithm 2.

Insertion algorithm 2 *(MDH optimal total directory size):*

Let X be the set of inserted records, n be the number of hash directories and b be the bucket size.

Step 1: Set the directory count n' equal to n and the record set X' equal to X.
Step 2: Determine the size of the largest hash directory, denoted as $D_{X'}(n', b)$, needed for the record set X' and the directory count n'.
Step 3: Determine the minimum number of directories with size $D_{X'}(n', b)$ needed, by using Proposition 5 and create these directories.
Step 4: Insert the records of X' into the directories resulting from Step 3. Fill each of these directories as much as possible.
Step 5: Reduce the directory count n' by the number of directories resulting from Step 2 and remove the records already inserted in these directories from the record set X'. Then repeat from Step 2 until no directories remain.

The reader can easily verify that the above insertion algorithm achieves MDH optimal total directory size. This is because a minimum number of directories with size $D_{X'}(n', b)$ is created for the current record set X' and directory count n'. When the number of hash directories is one, the unique directory that results from insertion algorithm 2 is identical to the one produced by the extendible hashing scheme. Insertion algorithm 2 has been simulated and experimental results are given in Fig. 5. Figure 5 shows the total directory sizes resulting from insertion algorithm 2, for different file sizes and bucket size equal to 50. All record keys were generated from a random number generator that generates integers uniformly distributed in $[0, 2^{31} - 1]$. The figure shows that the MDH optimal total directory size is smaller than the directory size of extendible hashing and

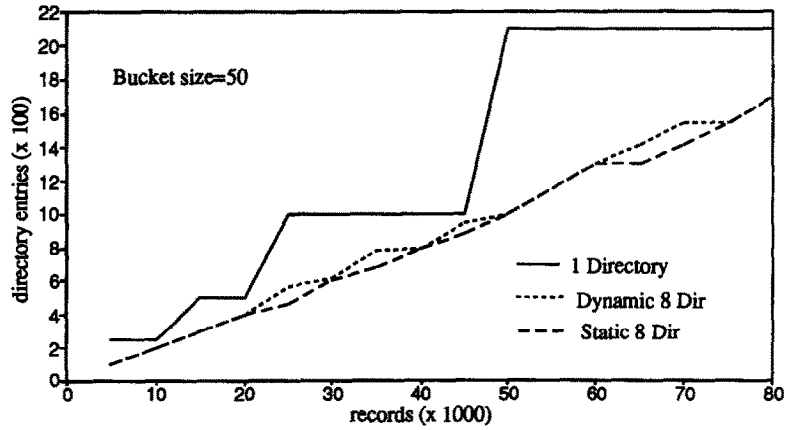

Fig. 6. Total directory sizes for insertion algorithms 1 and 2.

it becomes almost equal to the number of buckets when the number of hash directories is large.

Insertion algorithm 2 achieves optimal total directory size for static files. MDH insertion algorithm 1 works on dynamic files. Moreover, insertion algorithm 1 does not require any inter-directory restructuring. Inter-directory restructuring is particularly undesirable in parallel processing. Experimental results show that both algorithms achieve close total directory size. Figure 6 compares the total directory size of insertion algorithm 1 with that of insertion algorithm 2. We have used 8 hash directories for both the algorithms. All record keys were generated from a random number generator that generates integers uniformly distributed in $[0, 2^{31} - 1]$. The figure shows that insertion algorithm 1 yields close to optimal total directory size. The directory size for extendible hashing is also shown for the purpose of performance comparison.

Let X be the set of inserted records and n be the number of hash directories. The reader can easily verify that the number of hash directories with size $D_X(n, b)$ created by the MDH insertion algorithm 1 is the minimum needed. The rest of the MDH hash directories have size $D_X(n, b)/2$.

5. PROCESSING OF RANGE QUERIES IN MDH

Order preserving hashing can be used for range queries. High storage utilization and directly accessible buckets make MDH a good candidate for range queries in applications where order preserving high transformations are possible. This is the case when the underlying key distribution is stable [9]. For all directories, the address range of each query is computed using local directory information. Each directory can be accessed in parallel for retrieving the corresponding buckets.

We define access gain as: $[(B_{EH} - B_{MDH})/B_{EH}]$, where B_{EH} and B_{MDH} are the number of buckets accessed in a range query in extendible hashing and MDH, respectively. The access gain of MDH improves as the query range grows larger. MDH may perform worse than extendible hashing for small query ranges. This is because in MDH 1 bucket of every directory is accessed no matter how small the query range is. But as the range gets larger, extendible hashing is more likely to need more bucket accesses than MDH because the high bucket utilization of MDH reduces the total number of buckets to be accessed. Thus, on the average, the number of buckets accesses for range queries is less in MDH. We have measured the access gain by performing experiments on a MDH file of 500,000 records with uniformly distributed hash addresses. The result is shown in Fig. 7. Here, we have taken queries with ranges of $1, 2, \ldots$ up to 100% of the records in the file. Then, we computed the average of the access gains for all these ranges.

In Fig. 7, we see that the access gain obtains its maximum value at a certain number of hash directories $n_{opt}(N, b)$ which depends on the number of records inserted N and bucket size b. When the access gain is positive, the performance gain at large ranges offsets the penalties at small ranges. As the number of hash directories increases from 1 to $n_{opt}(N, b)$, this offset increases. However, it decreases as the number of hash directories increases further because the accessing penalties at

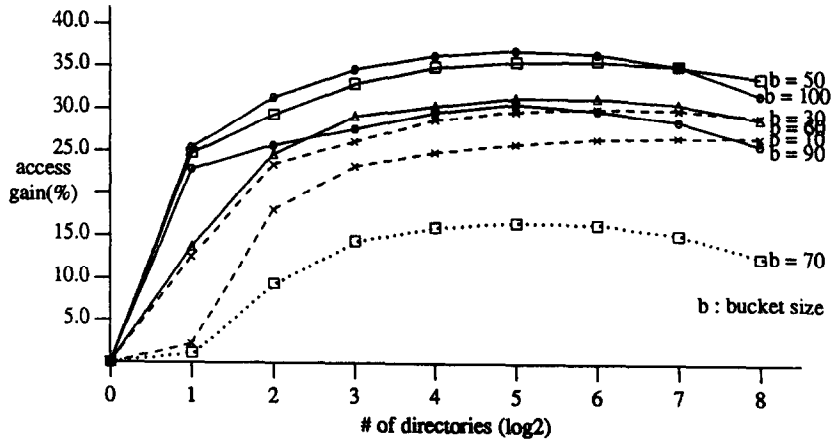

Fig. 7. Access gain of MDH for 500,000 records.

small query ranges increases. As more records are inserted, the performance penalties at small query ranges will affect the average access gain less and $n_{opt}(N, b)$ will increase.

6. MAIN MEMORY MULTI-DIRECTORY HASHING

In this section, we describe a variation of MDH, called Main Memory Multi-Directory Hashing (MM-MDH). We assume that data records reside in the main memory and that they could be scattered across all of the memory. The retrieval time in main databases consists of the time to index the directory plus the time to compare the key values. Access methods in main memory databases have been investigated in [10].

MM-MDH uses functional mapping of keys to directories. Here, the hash address of a record is partitioned into two parts. The leftmost part is used to locate the directory and the rightmost part is used to locate the record within a directory. This requires only one directory to be accessed to find the record. In contrast, MDH does not use functional mapping of keys to directories. As a result, all the directories in MDH need to be searched to locate a record. Each hash directory of MM-MDH grows in the same way as in the directory of extendible hashing with bucket size one. Thus, each directory entry points to at most one data record. We present the insertion algorithm of MM-MDH with n hash directories.

Insertion algorithm 3 *(MM-MDH):*

Step 1: Obtain the hash address h of the record. Use the $\log_2 n$ leftmost bits of the hash address to locate the hash directory d where the record will be inserted.
Step 2: If the directory entry corresponding to hash address h is empty, have the entry point to the record and stop.
Step 3: Double directory d and rehash all records by using one more bit from their hash address.
Then, go to Step 2.

Figure 8 illustrates the directory structure of the MM-MDH. Since directory entries point directly to data records, at most one key comparison is needed to locate a record. Thus, MM-MDH yields optimal search time. When the number of hash directories is one, MM-MDH coincides with extendible hashing. However, the directory size of extendible hashing increases significantly when the bucket size is one. The following proposition shows that the expected total directory size of MM-MDH decreases with the number of hash directories. Thus, the expected total directory size of MM-MDH is smaller than that of extendible hashing.

Proposition 6:

The expected total directory size of MM-MDH decreases as the number of hash directories n increases up to a value $n_{opt}(N)$. The value $n_{opt}(N)$ increases linearly with the number of inserted records N.

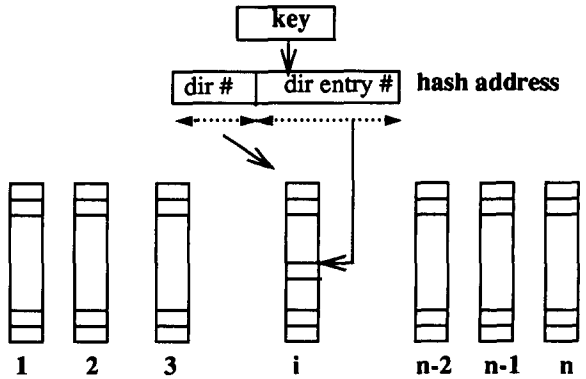

Fig. 8. Hash address mapping in MM-MDH.

Proof:

$O(f(N))$ denotes the order of $f(N)$, where f is a mapping function from the set of natural numbers to the set of real numbers. Frajolet has shown [8] that the expected directory size of extendible hashing with bucket size 1 and N number of inserted records is in $O(N^{1+x})$, where $x > 1/2$. Assuming that every bit of the hash address of a record has equal probability of being 0 or 1 and that the number of records N is much greater than n, each hash directory will get about N/n records. The expected directory size of each of the hash directories will be in $O((N/n)^{1+x})$ since each hash directories grows the same way as the directory of extendible hashing. The total directory size of MM-MDH consists of the size of the hash directories plus the size of an index that points to the hash directories. Thus, the total directory size of MM-MDH will be in $O(n(N/n)^{1+x} + n) = O((N^{1+x}/n^x) + n)$. It is easy to show that $O((N^{1+x}/n^x) + n) \subset O(N^{1+x})$ for $x > 1/2$, $n > 1$ and $n \leqslant n_{\text{opt}}(N) = x^{1/(x+1)}N$. Proposition 6 follows. □

The above analysis is verified by our experimental results. The expected total directory sizes were computed by averaging the results over 50 different runs. The keys used in each run were generated by a random number generator that produces uniformly distributed integers. Figure 9 gives the expected total directory size for different number of hash directories and records N. Note that expected total directory size decreases as the number of hash directories increases up to a value that depends on the number of inserted records.

The main memory variant of dynamic hashing is described in [11]. In dynamic hashing, records can be linked in a chain and the expected number of key comparisons for a record search is more than one. Thus, linear hashing does not yield optimal search time as MM-MDH does.

A different dynamic hashing scheme that uses multiple hash directories to access a main-memory file on a multi-processor system has been implemented and described in [12]. There, the hash directories are pointed to by another index which expands or contracts as needed by linear hashing.

7. CONCLUSIONS

We have presented a dynamic hashing scheme for disk-based databases, called MDH. MDH uses multiple hash directories to access a file. The expected bucket utilization of MDH approaches 100% with the number of hash directories. The expected total directory size of MDH is asymptotically smaller than that of extendible hashing. Parallel access of the hash directories reduces the search time significantly. MM-MDH is a variation of MDH which is best suited to main memory databases. MM-MDH obtains optimal search time, that is, at most one key comparison for a record search. The expected total directory size of MM-MDH is smaller than that of extendible hashing and it decreases as the number of hash directories increases.

Future work should be concerned with the classification of dynamic hashing schemes that use multiple hash directories to access a file. Classification characteristics should include the model of multi-directory growth and the indexing scheme of the hash directories.

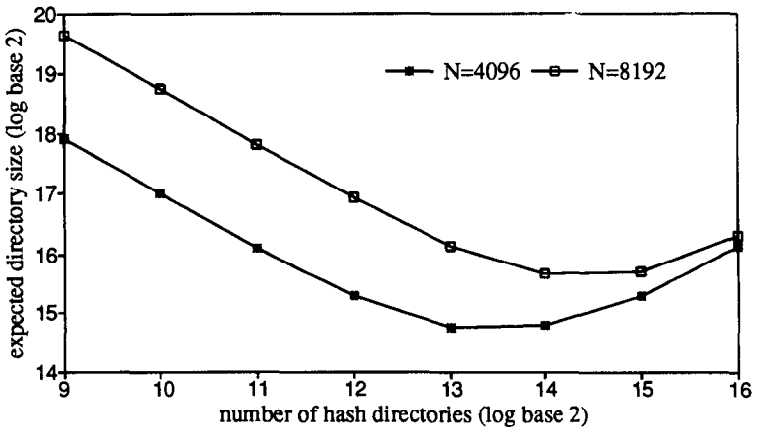

Fig. 9. Expected total directory size of MM-MDH for different number of directories.

Acknowledgements—The authors would like to thank the anonymous referees for their helpful suggestions. This work was supported in part by National Science Foundation Grant No. CCR-8706069, U.S. Army Missile Command Contract No. DAAH0188P5144 and Zonta International Amelia Earhart Fellowship Award.

REFERENCES

[1] R. Fagin, J. Nievergelt, N. Pippenger and H. R. Strong. Extendible hashing—a fast access method for dynamic files. *ACM TODS* **4**(3), 315–344 (1979).
[2] W. Litwin. Linear hashing: a new tool for file and table addressing. In *Proc. 6th VLDB Conf.*, pp. 212–223 (1980).
[3] P. A. Larson. Linear hashing with partial expansions. In *Proc. 6th VLDB Conf.*, pp. 224–232 (1980).
[4] P. A. Larson. Dynamic hashing. *BIT* **18**(2), 184–201 (1978).
[5] G. Wiederhold. *File Organization for Database Design*. McGraw-Hill, New York (1987).
[6] R. J. Enbody and H. C. Du. Dynamic hashing schemes. *ACM Comput. Surv.* **20**(2), 85–113 (1988).
[7] P. A. Larson. Performance analysis of linear hashing with partial expansions. *ACM Transact. Database Syst.* **7**(4), 566–587 (1982).
[8] Ph. Frajolet. On the performance evaluation of the extendible hashing and trie searching. *Acta inf.* **20**, 345–367 (1983).
[9] A. K. Garg and C. C. Gotlieb. Order-preserving key transformations. *ACM Trans. Database Syst.* **11**(2), 213–234 (1986).
[10] T. J. Lehman and M. Carey. A study of index structures for main memory database management systems. In *Proc. 12th VLDB Conf.*, pp. 294–303 (1986).
[11] P. A. Larson. Dynamic hash tables. *Commun. ACM*, **31**(4), 446–457 (1988).
[12] C. Severance, S. Pramanik and P. Wolberg. Distributed linear hashing and parallel projection in main memory databases. In *Proc. 16th VLDB Conf.*, pp. 674–681 (1990).

APPENDIX

Here, we prove Lemma 1. The proof follows the steps of a similar analysis for dynamic hashing in [4].

Proof of Lemma 1:

The depth of a bucket chain is the number of bits used in addressing all buckets in the chain. Assuming that all bits of the hash address of a record have equal probability of being 0 or 1, the probability that k out of N records pass through a given bucket chain with depth r is

$$P(r,k)=\binom{N}{k}(2^{-r})^k(1-2^{-r})^{N-k}$$

where $0 \leqslant k \leqslant N$, $r \geqslant 0$ and $0^0 = 1$.
A bucket is at position i of a bucket chain, if it is the ith bucket in the sequence of buckets in the chain. Since n is the maximum bucket chain length, it follows $1 \leqslant i \leqslant n$. The probability that there will exist a bucket at position i in a given bucket chain with depth r is

$$\sum_{k=nb+1}^{N}\sum_{x=(i-1)b+1}^{nb} P(r-1,k)\binom{k}{x}2^{-k},$$

because more than nb record should pass through the parent bucket chain (which was split) and from those records more than $(i-1)b$ and less than nb should pass through the given bucket chain.

Since there are 2^r bucket chains with depth r, the expected number of allocated buckets with depth r is

$$2^r\sum_{i=1}^{n}\sum_{k=nb+1}^{N}\sum_{x=(i-1)b+1}^{nb} P(r-1,k)\binom{k}{x}2^{-k}$$

where $r \geqslant 1$.
Thus, the expected number of allocated buckets when $N > nb$ is

$$\sum_{r=1}^{\infty} 2^r\left[\sum_{i=1}^{n}\sum_{k=nb+1}^{N}\sum_{x=(i-1)b+1}^{nb}\binom{N}{k}(2^{-r})^k(1-2^{-r+1})^{N-k}\binom{k}{x}\right]. \tag{1}$$

Applying Euler's summation formula†, we get the following approximation

$$\sum_{r=0}^{\infty} 2^{-r(k-1)}(1-2^{-r})^{n-k} \approx \int_0^{\infty} 2^{-r(k-1)}(1-2^{-r})^{n-k}\,dr = \frac{(n-k)!}{\ln 2(n-1)(n-2)\ldots(k-1)}.$$

Applying the above approximation to expression (1), we get that the expected number of allocated buckets is approximated by

$$\frac{N}{\ln 2}\left[\left(\sum_{i=1}^{n}\sum_{k=nb+1}^{N}\sum_{x=(i-1)b+1}^{nb}\frac{2^{-k+1}}{(k-1)k}\binom{k}{x}\right)\right].$$

Lemma 1 follows. □

†The error term computed from Euler's formula is numerically small.