Algorithms for a Hashed File with Variable-Length Records

Y. MANOLOPOULOS and N. FISTAS

Division of Electronics and Computer Engineering, Department of Electrical Engineering, Aristotelian University of Thessaloniki, Thessaloniki, Greece, 54006

ABSTRACT

Variable-length records are very frequent in database environments. This work gives a set of algorithms for maintaining a new model of static hashed files designed to accommodate variable-length records. Two special techniques for variable-length records are used. First, records are ordered by length and not by key value, e.g., the shortest (longest) records are stored in the main (overflow) blocks. Second, at the cost of a small directory for the overflow records stored in the main file blocks, the successful and unsuccessful searches are faster. The directory consists of the triplet key value, record length, and pointer to the overflow block. In addition, this work reports a simulation of the file structure. Numerical results show the benefits of the model. These two techniques may be applied in any primary key file organization that uses overflow chaining, including the recent versions of dynamic hashing.

1. INTRODUCTION

Various file organizations utilize overflow areas. Examples of such files are the late versions of hashing [10] as well as the traditional indexed sequential files [9]. Insertions and deletions hit file blocks under different frequencies. Therefore, it is almost certain that at some time after loading a number of main blocks will demand excess overflow space. Overflowing means that the performance will deteriorate owing to additional required accesses. Reorganization will take place either locally and dynamically or globally and periodically as in the cases of hashed and indexed sequential files, respectively.

A basic assumption of this work is that the file records length is not constant. Although variable-length records are very common in database environments owing to a number of reasons such as variable-length fields, missing attribute values, multiple values of an attribute, and compression, little has been reported in the literature about their effect on the file performance. We note the works of Diehr and Faaland [5], Larmore and Hirchberg [8], McCreight [12], and Szwarcfiter [13] that concern the design of algorithms for constructing an optimal B-tree with variable size keys, not records. On the other hand, Hakola and Heiskanen [6], Hubbard [7], Manolopoulos and Faloutsos [11], Teorey and Fry [14], and Wiederhold [16] provide analytic results on the wasted block space at the end of block due to the variability of the records. To our knowledge, only the work of Christodoulakis [2, 3] contains some formal analysis of the effects of variable-length records in file search performance. Recently, in [4], a new technique especially designed for files with variable-length records is proposed. This technique may be applied in any primary key file scheme using an overflow area.

This work is based on the model presented and analyzed in [4]. The results of a simulation of a static hashed file designed to accommodate variable length records are presented. Records are ordered by length and not by key value as in [1]. The shortest (longest) records are stored in the main (overflow) blocks. In addition, at the cost of a small directory for the overflow records stored in the main blocks of the file, the successful and unsuccessful search is faster. The directory consists of the triplet key value, record length, and pointer to the overflow block. Overflow blocks may be shared by many main blocks as proposed in [17].

The remainder of this paper is organized as follows. In Section 2 the file structure is explained and the searching, insertion, and deletion algorithms are presented. In Section 3 the simulation is described, and in Section 4 the numerical results are given. In the final section conclusive discussion shows the benefits of the model and some directions for future research are given.

2. TECHNIQUES FOR VARIABLE-LENGTH RECORDS

Suppose that a static hashed file consists of *NB* main blocks. By *static* we mean that this number of *NB* main blocks may not change with time because of successive insertions or deletions. In these cases overflow chaining will be used or the space utilization factor will be very low. In the beginning the file is loaded up to a certain load factor. According to the usual scheme the records are placed in blocks ordered in ascending (or descending) key value order [1] or on a first-come, first-stored basis. Record lengths are assumed to be independent of the primary key values.

The usual statistical distributions describing the way that records are directed to the blocks are the binomial or the Poisson ones. If the load factor is low (high) then the probability that there are overflow records is negligible (considerable). Here, two assumptions are made. First, there is an equilibrium between insertions and deletions and, second, the space of the deleted records is not actually freed for later use but is flagged instead. Under these assumptions it follows that the probability of overflowing is considerable. It is reasonable, also, to accept that the deletion probability is independent of the record length. If these conditions are met then the statistical distribution of the record lengths will be the same for the main or the overflow area.

According to the scheme proposed in [4] all the records directed to a specific block are ordered by length. The shortest (longest) records are stored in the main (overflow) blocks. Therefore, the distribution of the record lengths differs from the main to the overflow area. It is evident that, under the assumption of great block size with respect to the record lengths, the main (overflow) area will contain a larger (smaller) number of records. Under the same assumption, variable-length records may be packed better within a block resulting in better space utilization and reduced search cost.

In addition, a small directory is stored in the main file blocks. This directory is an index to the overflow records and consists of a triplet for every overflow record: key value, record length, and pointer to the overflow block in which the specific record is accommodated. The length of the triplet may be less than one order of magnitude smaller than the length of the record. Another scheme by Torn [15] proposes the use of an index for the overflow records residing in main memory. This technique is suitable for small- to medium-sized files but not for large files because in the last case this index becomes too large to fit in main memory. In Figure 1 our main block directory technique is illustrated.

Overflow records of a specific main block are neither connected in a chain nor clustered in a specific overflow block. According to our scheme several records from different main blocks or the same main block may exist in the same overflow block. In other words, overflow blocks are shared by a varying number of main blocks. The number of consecutive main blocks, forming a

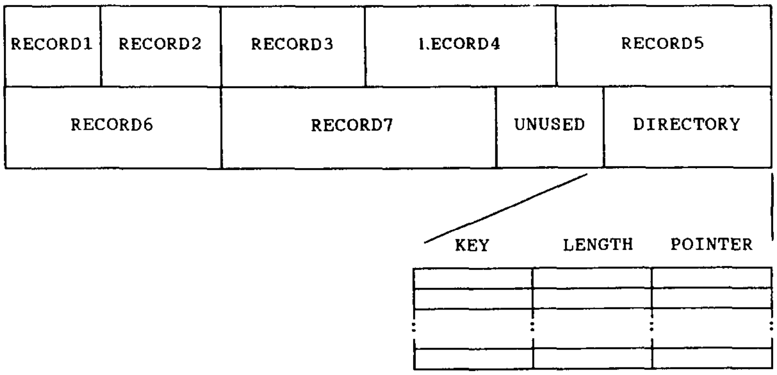

Fig. 1. Main block records ordered by length and main block directory techniques.

main block cluster, which overflow to a specific overflow block is a parameter of the file. A main memory structure is needed for keeping the addresses of the current available overflow blocks of all the main block clusters. A similar scheme, with shared overflow blocks, is employed by Yuen and Du [17] in variations of linear hashing for partial match retrieval. In the past, the opposite assumption has been adopted too [9], e.g., it has been accepted that the overflow blocks have record capacity one. In other words it has been accepted that the probability that a varying number of main blocks shares the same overflow block is negligible. As a consequence of the last scheme a chain of records is identical to a chain of blocks. Figure 2 depicts our overflow scheme.

In the following the algorithms performing the operations of searching, insertion, and deletion are given. It is clear that at most one (no) additional overflow access is paid for the successful (unsuccessful) search at the cost of slightly more complex insertion and deletion algorithms as well as the directory space cost. However, it will be shown that this space is negligible.

SEARCHING ALGORITHM

```
The key record is hashed and the main block address is derived;
IF the record is stored in the main block THEN process the record
ELSE IF the key record resides in the main block directory THEN
        BEGIN
                read the overflow block;
                process the record
        END
ELSE the search is unsuccessful;
```

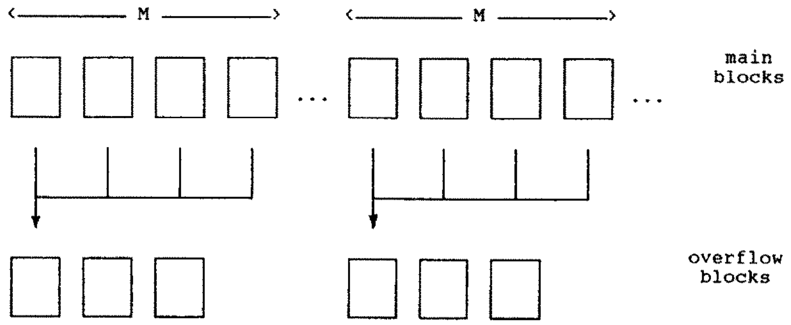

Fig. 2. Shared overflow buckets per M = 4 main blocks.

INSERTION ALGORITHM

```
The key record is hashed and the main block address is derived;
The main block records are sorted in ascending length order;
IF the longest record fits in this specific main block
        THEN write the main block on disk
ELSE
BEGIN
    IF there is not an overflow directory THEN
    BEGIN
        fetch an overflow block from the system;
        store the longest record in this overflow block;
        initialize the main block directory;
        accommodate the triplet in the directory;
        IF the directory fits in the main block THEN
        BEGIN
            accommodate the directory in the main block;
            write the main and overflow blocks on disk
        END
        ELSE
        BEGIN
            move the next longest record of the main block
                in the overflow block;
            accommodate the new triplet in the directory;
            accommodate the directory in the main block;
            write the main and overflow blocks on disk
        END
    END
    ELSE {there is an overflow directory}
    BEGIN
        read the current overflow block of the cluster;
        IF the overflow block has enough space THEN
        BEGIN
            store the longest record in this overflow block;
            accommodate the triplet in the directory;
            IF the directory fits in the main block THEN
            BEGIN
                accommodate the directory in the main block;
                write the main and overflow blocks on disk
            END
            ELSE {the directory does not fit}
```

```
                BEGIN
                    move the next longest record of the main block
                        in the overflow block;
                    accommodate the new triplet in the directory;
                    accommodate the directory in the main block;
                    write the main and overflow blocks on disk
                END
            END
        END
    END;
```

DELETION ALGORITHM

```
    The key record is hashed and the main block address is derived;
    IF the record is not stored in the main block
        AND its key is not included in the directory
        THEN the search is unsuccessful
    ELSE IF there is not an overflow directory THEN
        BEGIN
            delete the record;
            write the main block on disk
        END
    ELSE
        BEGIN
            IF the record to be deleted is an overflow one THEN
            BEGIN
                read the overflow block;
                delete the record;
                update the main block directory;
                IF the size of the shortest overflow record minus
                    a triplet size does not fit in the main block
                    THEN write the main and overflow blocks on disk
                ELSE
                BEGIN
                    move the shortest overflow record in the main block;
                    update the main block directory;
                    write the main and overflow blocks on disk
                END
            END
            ELSE {the record to be deleted resides in the main block}
            BEGIN
                delete the record;
                IF the size of the shortest overflow record minus
                    a triplet size does not fit in the main block
                    THEN write the main and overflow blocks on disk
                ELSE
                BEGIN
                    move the shortest overflow record in the main block;
                    update the main block directory;
                    write the main and overflow blocks on disk
                END
            END
        END
    END;
```

3. THE SIMULATION

Current file systems use only preformatted disks (disks with fixed block size). Let BS be the size of either a main or an overflow block. Let, also, KS be the size of the key, LS be the size of the record length, and PS be the size of the pointer to the overflow block. All sizes are measured in bytes. In total, the size of the triplet is $KS + LS + PS$ is in the area of 12 bytes, which is negligible when compared with the block size, which may range from 512 bytes to several Kbytes.

In the general case it may be assumed that the population of records is divided into t classes: $C_1, C_2, \ldots, C_t$. Class C_1 contains the records with length RL_1, class C_2 contains the records with length $RL_2, \ldots,$class C_t contains the records with length RL_t. The probability that a record is in class C_i is PR_i, $i = 1, 2, \ldots, t$ and it is known. The assumption of a fixed number of classes is justified from the fact that frequently variable-length records are the result of missing attribute values for certain attributes, or repetitions of a certain set of attribute values several times, or mixing of more than one record type with a common key. In environments where variable-length records are the result of some other compression scheme and a continuous probability distribution of record lengths is observed, the range of lengths can be subdivided into subranges and a class can be identified with a subrange. Table 1 summarizes the parameters of the file.

An extensive simulation has been written in Turbo Pascal and runs on IBM

TABLE 1

List of parameters

BS	Block Size in bytes
LS	Length Size in bytes
KS	Key Size in bytes
PS	Pointer Size in bytes
NB	Number of main Blocks
M	Number of Main blocks forming a cluster
t	Number of record classes
C_i	Class i ($1 \leqslant 1 \leqslant t$)
RL_i	Record Length of class i ($1 \leqslant i \leqslant t$) in bytes
PR_i	PRobability of arriving records of class i

compatibles. Key record values range from 1 to the constant maxlongint of the programming language (2.147.483.647). The number of main blocks is $NB = 1000$. The main and overflow block size is 1 or 2 Kbytes. The file parameter M is set equal to 5, 10, or 20 main blocks.

For simplicity, only two record classes are considered. The record size of the first class is always 400 bytes; the other class contains records of length 400, 200, or 100 bytes. The arriving probability corresponding to the first (second) class is equal to 0.9, 0.5 or 0.1 (0.1, 0.5 or 0.9).

In the beginning the file is empty. Arriving records are directed in the appropriate main block according to the remainder by division transformation. Sooner or later the main blocks become full. Overflow blocks are fetched from the system and the main block directory is used. Another structure used is an array containing the addresses of the current overflow blocks of the clusters. The length of this array is NB/M. The loading of the file continues up to the point that 15,000 records will have been stored. No deletions or updates are considered.

What we are interested in is the performance for successful and unsuccessful search as well as the space occupied by the file. Measurements are taken as follows. We stop the file loading process eight times (the end of the simulation is included in this number) and calculate the desired estimates. Therefore, the curves showing the search and space cost as a function of the time constist of eight points. For comparison, another structure similar to the proposed structure is considered, e.g., the main block directory technique is used to index the overflow records but no attention is paid in ordering the records ascendingly according to the record length. Intuitively it seems that this structure will result in worse space utilization and as a consequence worse search performance. The values depicted in the following figures are produced by running the simulation 10 times and taking the mean values.

4. NUMERICAL RESULTS

In this section we present and discuss some numerical results produced by the simulation. The main points of interest are the space utilization and the search performance. In this respect the main block directory and order-by-length techniques are examined.

Figures 3 and 4 show the distribution of records in the main or overflow blocks as a function of time for some parameter values. In particular, $BS = 1$ Kbyte, $RL_1 = 400$ bytes, $RL_2 = 100$ bytes, $PR_1 = PR_2 = 0.5$. Figure 3 (4) corresponds to the case that the records are (not) ascendingly ordered according to their length. Evidently the linear curve represents the total number of records in the main and overflow blocks. The curve, which after some time stabilizes (grows linearly), represents the records of the main block (overflow blocks). It is remarked that in Figure 3 (4) the number of records stored in the main block is equal to the number of records in the overflow blocks at the fourth (sixth) time instant. This remark means that the use of the order-by-length technique achieves better space utilization. From extensive experimentation it is concluded that ordering by ascending record lengths gives better results when the size difference between the short and the long record is considerable, as well as when the arriving probability of the short record is greater than the

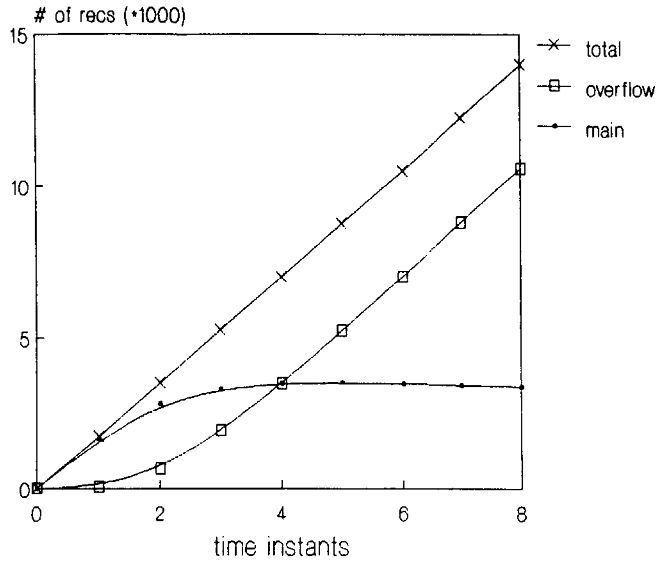

Fig. 3. Distribution of records in the main and overflow blocks according to the order-by-key technique as a function of time. Parameters: $BS = 1$ Kb, $RL_1 = 400$ b, $RL_2 = 100$ b, $PR_1 = PR_2 = 0.5$.

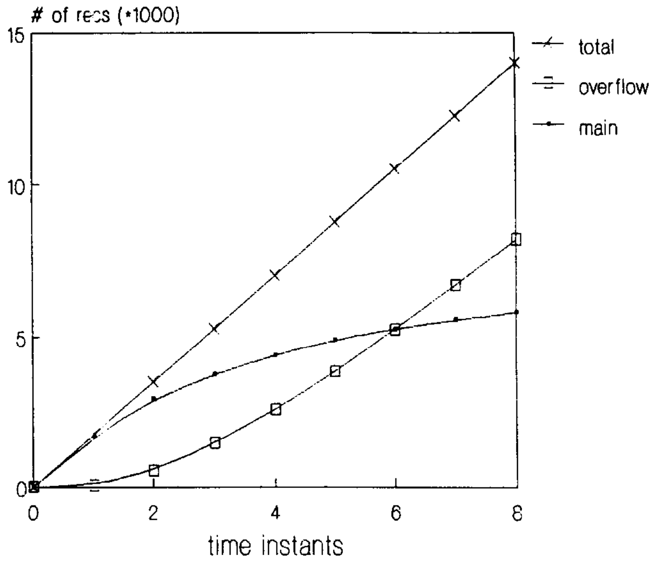

Fig. 4. Distribution of records in the main and overflow blocks according to the order-by-length technique as a function of time. Parameters: $BS = 1$ Kb, $RL_1 = 400$ b, $RL_2 = 100$ b, $PR_1 = PR_2 = 0.5$.

probability of the long record. This is because the smaller the average record length, the more records are needed to fill the main block and therefore the less expected number of overflow blocks are fetched from the system. In addition, it is remarked that the greater the block size, the better the record packing.

Better space utilization has as a consequence better searching performance. Figure 5 shows the overflow access cost as a function of time in the case of successful search for the same parameter values of Figures 3 and 4. This cost is equal to the number of overflow records divided by the total number of records. As expected, the cost is always smaller than one overflow access. It is anticipated that asymptotically the curves will tend to the unity. The upper (lower) curve represents the case in which the order-by-key (length) technique is used. Considerable gain is achieved under the same circumstances as explained previously. No additional overflow accesses are needed for unsuccessful search.

Another parameter taken in consideration in Figures 6 and 7 is the number of main blocks M pointing to the same overflow blocks. Intuitively, it is clear that changing the value of M affects the required space both in the main and secondary memories. First, it influences the main memory required because of the array for keeping the addresses of the currently available overflow blocks. Doubling the value of M results in halving the length of the array and vice

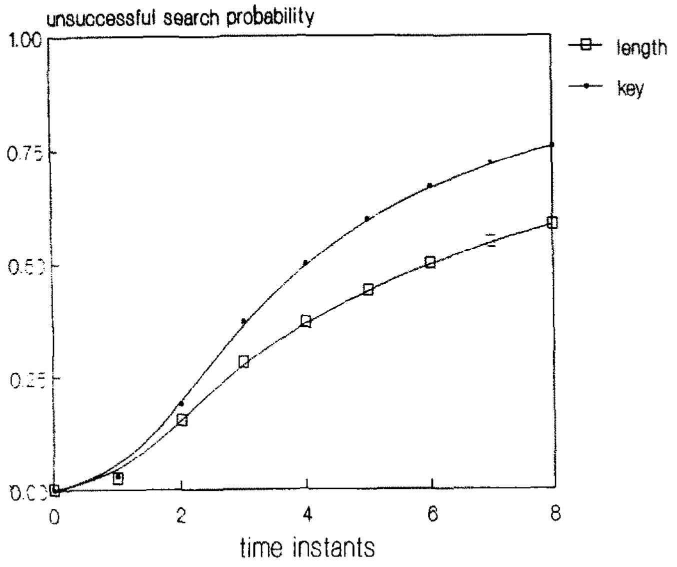

Fig. 5. Overflow access probability (cost) as a function of time according to both the order-by-length and order-by-key techniques. Parameters: $BS = 1$ Kb, $RL_1 = 400$ b, $RL_2 = 100$ b, $PR_1 = PR_2 = 0.5$.

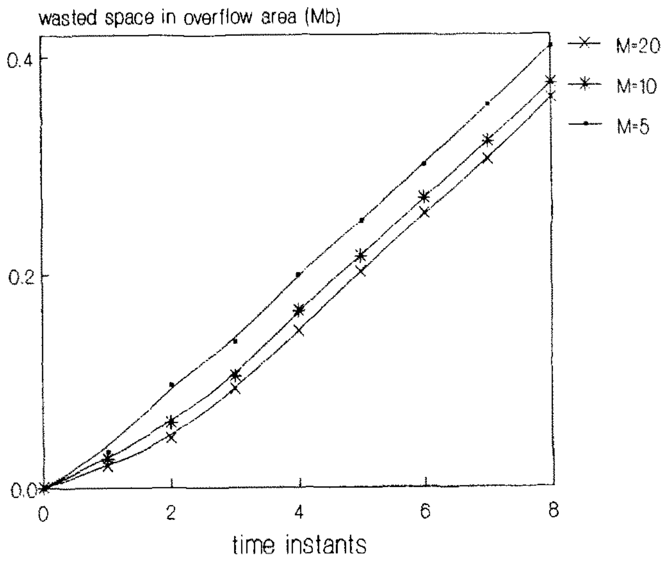

Fig. 6. Wasted space in the overflow area as a function of time according to the order-by-length technique. Parameters: $BS = 1$ Kb, $RL_1 = 400$ b, $RL_2 = 200$ b, $PR_1 = PR_2 = 0.5$, $M = 5, 10, 20$.

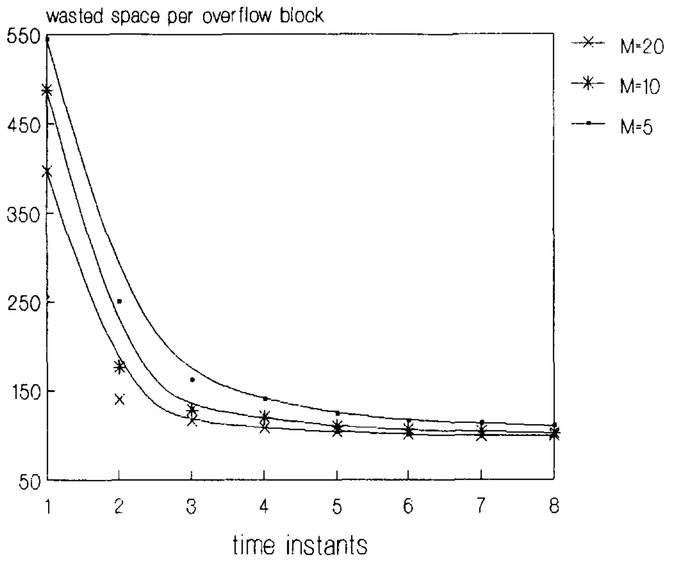

Fig. 7. Wasted space per overflow block as a function of time according to the order-by-length technique. Parameters: $BS = 1$ Kb, $RL_1 = 400$ b, $RL_2 = 200$ b, $PR_1 = PR_2 = 0.5$, $M = 5, 10, 20$.

versa. Second, it influences disk space if the wasted space is considered. For example, in Figure 6 (7) the wasted space in the overflow area (per overflow block) as a function of time is depicted. The parameter values are $BS = 1$ Kbyte, $RL_1 = 400$ bytes, $RL_2 = 200$ bytes, $PR_1 = PR_2 = 0.5$. The experimentation has shown that, especially as time grows, the influence of the value of M on space is slight.

Another factor under consideration is the empty wasted space left at the end of the main block. Intuitively, the larger the size of the records (with respect to the block size), the more will be the wasted space at the end of the main block. Analytical results and simulation are reported in [11]. Figure 8 depicts the percentage of the main block size that is wasted as a function of time. The gain of the order-by-length over the order-by-key technique is noted again. The greater the block size, the smaller the wasted space owing to better variable-length record packing.

Finally, in Figure 9 the parameter values are $BS = 1$ Kbyte, $RL_1 = 400$ bytes, $RL_2 = 200$ bytes, $PR_1 = 0.9$, and $PR_2 = 0.1$. This figure illustrates the distribution of the total occupied space for the main blocks except the directories, the main block directories themselves, and the overflow blocks. With time, the main block except the directory (directory itself) space will tend to become zero (occupy the main block space). At this time some reorganiza-

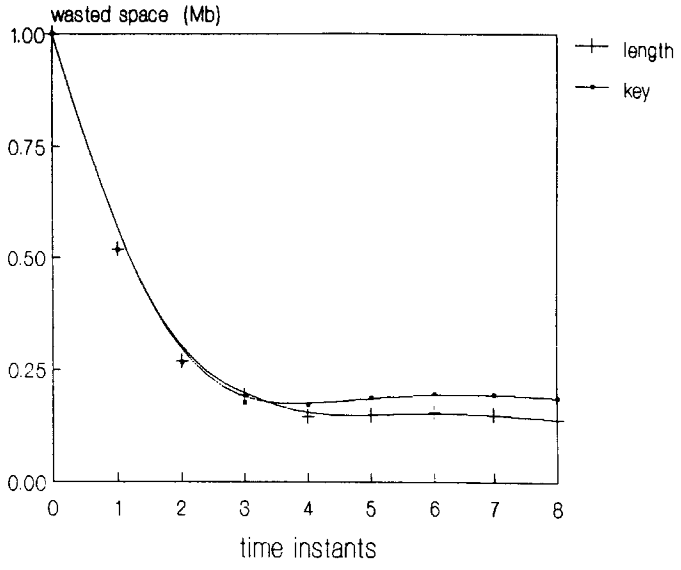

Fig. 8. Wasted space at the end of main block as a function of time according to both the order-by-key and order-by-key techniques. Parameters: $BS = 1$ Kb, $RL_1 = 400$ b, $RL_2 = 200$ b, $PR_1 = PR_2 = 0.5$.

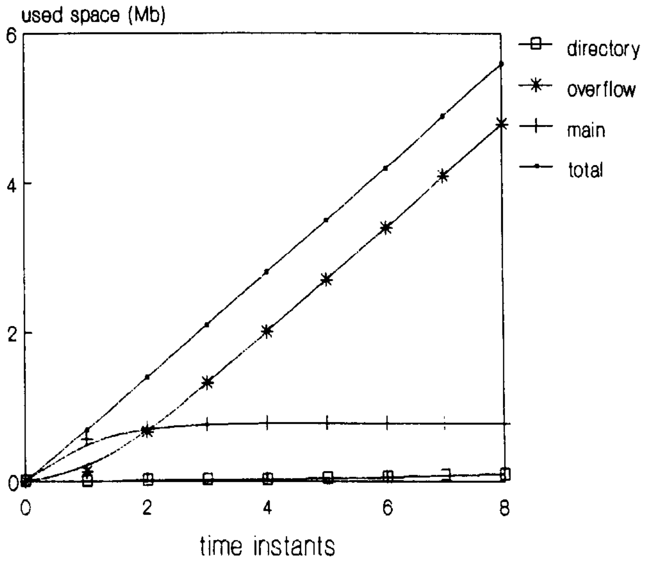

Fig. 9. Distribution of used total space as a function of time according to the order-by-key techniques. Parameters: $BS = 1$ Kb, $RL_1 = 400$ b, $RL_2 = 200$ b, $PR_1 = 0.9$, $PR_2 = 0.1$.

tion is needed. However, for this time limit it is noted that the directory space is negligible when compared with the total occupied space.

5. SUMMARY AND CONCLUSION

In this work the results of a simulation of the performance of a new file structure are reported. More specifically, under the assumption of variable-length records two techniques are examined. The first is the main block directory technique and the second is the order-by-length technique. These techniques may be used within any primary key file structure using overflow areas and will improve the search performance for very limited additional space. The successful (unsuccessful) search will always require at most one (no) overflow access.

Future research in this area involves the application of the main block directory in dynamic hashed files. Deletion of records and algorithms for better overflow space utilization is required too. Another extension would be to derive analytical performance estimates for the successful and unsuccessful search and the occupied space and compare the simulation results. Another step would be towards the development of file structures designed specially for variable-length records.

REFERENCES

1. O. Amble and D. E. Knuth, Ordered hash tables, *Comput. J.* 17:135-147 (1974).
2. S. Christodoulakis, Estimating block transfers and join sizes, in *Proceedings of the ACM SIGMOD-83 Conference*, San Jose, California, pp. 40-54, 1983.
3. S. Christodoulakis, Implications of certain assumptions in database performance evaluation, *ACM Transactions on Database Systems* 9(2):163-186 (1984).
4. S. Christodoulakis, Y. Manolopoulos, and P. A. Larson, Analysis of Overflow Handling for Variable Length Records, *Information Systems* 14(2): 151-152 (1989).
5. G. Diehr and B. Faaland, Optimal pagination of B-trees with variable length items, *Commun. ACM* 27(3):241-247 (1984).
6. J. Hakola and A. Heiskanen, On the distribution of wasted space at the end of file blocks, *BIT* 20(2):145-156 (1980).
7. G. U. Hubbard, *Computer-assisted Database Design*, Van Nostrand Reinhold, New York, 1981.
8. L. L. Larmore and D. S. Hirchberg, Efficient optimal pagination of scrolls, *Commun. ACM* 28(8):854-856 (1985).
9. P. A. Larson, Analysis of index sequential files with overflow chaining, *ACM Trans. Database Syst.* 6(4):671-680 (1981).
10. P. A. Larson, Hash Files: some recent developments, in *Proceedings of the International Conference on Supercomputing Systems*, Florida, 1983, pp. 671-679.
11. Y. Manolopoulos and C. Faloutsos, Analysis for the end of block waste space, *BIT*, 30: 620-630 (1990).

12. E. M. McCreight, Pagination of B-trees with variable-length records, *Commun. ACM* 20(9):670-674 (1977).
13. J. L. Szwarcfiter, Optimal multiway search trees for variable length size keys, *Acta Informatica* 21(1):47-60 (1984).
14. T. J. Teorey and J. P. Fry, *Design of Database Structures*, Prentice Hall, Englewood Cliffs, New Jersey 1982.
15. A. A. Torn, Hashing with overflow indexing, *BIT* 24:317-332 (1984).
16. G. Wiederhold, *File Organization for Database Design*, McGraw-Hill, 1987.
17. T. S. Yuen and D. H. C. Du, Dynamic file structure for partial match retrieval based on overflow bucket sharing, IEEE Trans. Software Eng. 12(8):801-810 (1986).

Received 18 September 1989; revised 5 June 1990