01 Dec 1987

Multilist and inverted file system performance measurements

Ashok Chandramouli

George Winston Zobrist
Missouri University of Science and Technology

MULTILIST AND INVERTED FILE SYSTEM PERFORMANCE MEASUREMENTS

Ashok Chandramouli*and George Zobrist

CSc-87-16

*This report is substantially the M.S. thesis of the first author, completed December,]987.

ABSTRACT

This study evaluates the multilist and inverted file systems. It describes the structure of the two file system and then proceeds to investigate the performance. The performance is based on quantitative estimates of space requirements for file system, time to retrieve records, time to insert a record, time to delete a record, time to update a record and time to exhaustively read and reorganize the file system. The study then investigates specific situations in which one file system seems to perform better than the other.

TABLE OF CONTENTS

Page

ABSTRACT ii

ACKNOWLEDGEMENT iii

LIST OF ILLUSTRATIONS vii

LIST OF TABLES viii

I. INTRODUCTION 1

II. HARDWARE PARAMETERS 3

A. BLOCKS AND BLOCKING FACTOR 3

B. SEEK TIME 4

C. ROTATIONAL LATENCY 4

D. TRANSFER RATE 6

E. BULK TRANSFER RATE 6

F. BUFFERS 7

G. BLOCK UPDATE 7

H. QUANTITATIVE MEASURES 8

1. FETCH RECORDS 8

2. INSERT RECORD 8

3. DELETE RECORD 8

4. UPDATE RECORD 8

5. READ ENTIRE FILE 9

6. REORGANIZATION OF FILE 9

7. FILE SIZE 9

III.MULTILIST FILE SYSTEM 10

A. STRUCTURE 10

1. FILE 10

2. DIRECTORY 10

3. SEQUENTIAL STRUCTURE 11

4. B-TREE STRUCTURE 13

B. RECORD SIZE 16

C. DIRECTORY SIZE 16

a. SEQUENTIAL STRUCTURED DIRECTORY . . . 16

b. B-TREE STRUCTURED DIRECTORY 17

D. TIME TO FETCH 18

E. TIME TO INSERT 20

F. TIME TO DELETE 22

1. DELETION OF AN ATTRIBUTE NAME-VALUE PAIR . 22

2. DELETION OF RECORDS 23

G. TIME TO UPDATE 23

1. IN-PLACE UPDATE 23

2. UPDATE WITH NEW ATTRIBUTES 24

H. TIME FOR EXHAUSTIVE READ 24

I. TIME TO REORGANIZE 26

1. DIRECTORY REORGANIZATION 26

2. FILE REORGANIZATION 27

IV. INVERTED FILE SYSTEM 28

A. STRUCTURE 28

1. FILE 28

2. DIRECTORY 28

B. FILE SIZE 32

C. TIME TO FETCH 34

D. TIME TO INSERT 35

E. TIME TO DELETE 36

1. DELETION OF AN ATTRIBUTE NAME-VALUE PAIR . 36

2. DELETION OF A RECORD 37

F. TIME TO UPDATE 37

1. IN PLACE UPDATE 37

2. UPDATE WITH NEW ATTRIBUTES 37

G. TIME FOR EXHAUSTIVE READ 38

H. TIME FOR REORGANIZATION 38

V. RESULTS 40

A. CONCLUSION 41

BIBLIOGRAPHY 42

VITA 43

APPENDICES

A. NOMENCLATURE 44

B. PROGRAM TO CALCULATE THE PERFORMANCE PARAMETERS . . 46

C. INPUT PARAMETERS AND RESULTS 59

LIST OF ILLUSTRATIONS

Figure		Page
1	BLOCK WITH VARIABLE LENGTH RECORDS	5
2	FILE AND SEQUENTIAL INDEX	12
3	B-TREE STRUCTURED INDEX	14
4	INVERTED LIST RECORDS	29
5	INVERTED LIST DIRECTORY AND ADDRESS LISTS	30

LIST OF TABLES

Table		Page
I	INPUT VALUE SET 1	59
II	PERFORMANCE VALUES FOR INPUT SET 1	60
III	PERFORMANCE VALUES FOR INPUT SET 2	61
IV	PERFORMANCE VALUES FOR INPUT SET 3	62
V	PERFORMANCE VALUES FOR INPUT SET 4	63

I. INTRODUCTION

Information storage and retrieval systems organize and store information in a space efficient manner and provide for the quick retrieval of necessary information. The information may be structured data as in database systems or it may be loosely tied textual data of documents. The information items must be collected and organized in a coherent manner into a file structure so that operations such as retrieval and update can be achieved efficiently. File structures vary in complexity and organization and should be chosen in order to obtain optimum performance. Inverted and multilist structures are two such file structures. Many commercial information storage and retrieval systems are based on inverted and multilist file structures.

The multilist structure (also known as threaded lists in literature) permits records having the same characteristics to be linked together in a list [1]. A record may contain many links and may be a member of one or more lists. The inverted structure, on the other hand, removes the links from the records and places all the links as a separate record. Both systems require a directory which is a structure to assist in locating the desired record or set of records.

Inverted and multilist file structures have been analysed for a track oriented tree structured directory [2]. However, the following analysis is carried out for both a sequential structured directory and a hardware independent B-Tree structured directory. The hardware parameters as derived by Wiederhold [3], are used in the analysis.

Chapter 2 presents the hardware parameters necessary for analysis and introduces measures of performance. Chapter 3 and chapter 4 provide a detailed analysis of the file systems and chapter 5 discusses the situations under which one file system seems to perform better than the other.

II. HARDWARE PARAMETERS

This chapter investigates the basic hardware parameters necessary to evaluate the multilist and inverted list file organizations. The parameters are applicable to direct access devices such as disks and drums.

A. BLOCKS AND BLOCKING FACTOR

A block is the unit of information or data transferred between external storage devices and the buffer in the core memory. Between any two blocks there is a gap (G) to permit the read/write head to prepare for the next operation. The gap size depends on the device characteristics and is specified by the manufacturer. An optimal block size should be selected for good performance. Small block sizes increase the number of interblock gaps (G) and thus waste secondary storage. Large blocks, on the other hand, require larger buffers and greater transmission times. A logical record is the smallest unit which a program manipulates. The fitting of one or more records into a block is referred to as blocking. The number of records per block is referred to as the blocking factor (Bfr). In fixed blocking, each block contains an integral number of records. If B denotes the block size and R, the average record size, then the blocking factor (Bfr) is

$$Bfr = FLOOR(B/R) \quad \text{Records / block} \tag{2.1}$$

The wasted space per record (W) is the sum of waste due to gap (G) and waste due to record fitting. With fixed blocking, the waste due to record fitting can be neglected. In this case,

$$W = G / Bfr \quad \text{Characters / Record} \tag{2.2}$$

With variable length records, an average of (1/2 * R) of the block is unused. Within each block record delimiters are necessary to separate the records. If the size of record delimiter is denoted by P, then

$$Bfr = (B - 1/2 * R) / (R + P) \quad \text{Records / Block} \tag{2.3}$$

The wasted space per record W, with variable length records, as shown in figure 1, is

$$W = P + (0.5 * R + G) / Bfr \quad \text{Characters / Record} \tag{2.4}$$

B. SEEK TIME

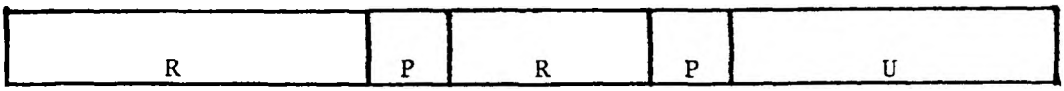
The seek time is the time required to position the access mechanism over the right track. Since it is not feasible to compute the instantaneous seek time for each access, the average seek time is used for the performance evaluation. The average seek time (s) is generally provided by the manufacturer and is based on uniform access over all the cylinders. However, by placing files on adjacent cylinders, a lower value of seek can be obtained.

C. ROTATIONAL LATENCY

Rotational latency (r) includes the delay between the completion of seek and actual transfer of data. In devices which can recognize blocks at their own begin point, the average latency is given by

$$r = 60 * 1000 / (2 * rpm) \quad \text{ms} \tag{2.5}$$

where rpm is revolutions per minute and r is the average latency in milliseconds.



R: Record
U: Unused
P: Pointer

Fig. 1. BLOCK WITH VARIABLE LENGTH RECORDS

D. TRANSFER RATE

The transfer rate (t) is the instantaneous rate at which the data is transferred from or to the direct access device. The transfer rate is dependent on the device characteristics and is provided by the manufacturer. The time to transfer a block of size B (block transfer time), Btt is

$$Btt = B / t \quad ms \tag{2.6}$$

E. BULK TRANSFER RATE

When transferring large quantities of data sequentially, the transfer rate has to be scaled by a factor to take into account the interblock gaps and minimal seeks which occur at cylinder boundaries. The adjusted transfer rate is referred to as bulk transfer rate (t').

The effect of gaps on transfer rate can be evaluated by considering the time to transfer a block of data. In time (R + W) / t, R bytes of data are transferred. Thus, the bulk transfer rate is

$$t' = R / ((R + W) / t) \quad \text{Characters/ms} \tag{2.7}$$

If s' denotes the effective seek time per block, then the bulk transfer rate adjusted for gaps and seeks is given by

$$t' = R / (((R + W) / t) + s') \quad \text{Characters/ms} \tag{2.8}$$

To evaluate s', the operating environment has to be considered. In a multiprogrammed environment, a seek may occur after transfer of every block. In this case s' = s / Bfr. If there is no contention for the seek mechanism, a minimal seek occurs once every cylinder and in most

direct access devices this seek to an adjacent cylinder is less than 2r. The value of s' lies between 2r and s / Bfr. It must be chosen to reflect the operating environment.

F. BUFFERS

A block from a direct access device is read into an area of core memory called the buffer. The required record is selected from the buffer for processing. For sequential reading of large quantities of data using bulk transfer rate, it is assumed that two buffers are available. This permits the loading of one buffer while reading from the other buffer. The computation time (c_{block}) for processing one buffer, should be less than the time to load the buffer. If this condition is violated, but the condition r > c_{block} > Btt holds, then t' can be modified to

$$t' = 2 * B / ((2 * r + 2 * Btt + s') \quad \text{characters/ms} \qquad (2.9)$$

G. BLOCK UPDATE

Updating a record in a block requires a read and a subsequent write operation. In many instances, it is necessary to rewrite the block in the same location on the direct access device. If the insertion of the record in the block in core buffer can be done under the condition that the computation time c << 2 * r, then the time to rewrite the record (T_{rw}) is given by

$$T_{rw} = 2 * r \quad \text{ms} \qquad (2.10)$$

H. QUANTITATIVE MEASURES

The quantitative measures used to study the performance of file systems are described in this section.

1. FETCH RECORDS. Fetching is the operation of retrieving a record or a subset of records to satisfy a query. Fetching of records is a two step process. The necessary blocks have to be located followed by the transfer of blocks to core memory. The time required for this operation is quantified as time to fetch (Tf).

2. INSERT RECORD. Insertion is the operation of adding a new record. The inserted record may have to be fitted into a specific location, which may necessitate the shifting of certain records. These costs are measured as time to insert (Ti).

3. DELETE RECORD. Deletion is the operation of removing a record from the file. This is accomplished by physically removing the record and shifting subsequent records or by logically deleting the record with a special mark. This cost is denoted by time to delete (Td).

4. UPDATE RECORD. Certain situations may require that data within a record be modified. This operation is termed as record update. The modified data is merged with the unmodified data, and the record is rewritten either in the same location or at a new location. This cost is quantified as time to update (Tu).

5. READ ENTIRE FILE. Some application functions require the reading of the entire file. This time is quantified as time for exhaustive read (Tx).

6. REORGANIZATION OF FILE. Periodically, it may be necessary to rearrange the records within the file to improve performance and to reclaim space occupied by deleted records. The periodicity of reorganization is dependent on the type of file and the frequency of insert, delete and update operation. This time is quantified as time for reorganization (Ty).

7. FILE SIZE. File size indicates the storage required by the file system including the directory and other access structures.

The operations of record fetches, record insertions, record updates record deletions, exhaustive reading and reorganization of the file are executed by combination of seeks, latency, reads and writes. Generalized hardware parameters are used in the performance measurements to provide independence from the actual physical specifics of the underlying hardware.

Most file systems have access structures, besides data, to facilitate the access of data in the file. This improved access to data is reflected in the reduced fetch times but is obtained at the expense of increased space requirements to store the data and the access structures. An optimal file system should be designed considering the performance requirements for a given application in relation to the space requirements of the file system.

III. MULTILIST FILE SYSTEM

A. STRUCTURE

The multilist file system is composed of two parts - a file and a directory.

1. FILE. The file is a collection of logical records. Each record consists of one or more data values called the attribute value and an identification of that value, called the attribute name. Each record may be considered as a collection of one or more attribute name-value tuple (also referred to as attribute) and non indexed data. All records, possessing a particular attribute name-value tuple are linked together to form a list. The head of the list is obtained by decoding the directory. There are no restrictions on placement of records within the file. One of the distinguished attribute name-value pairs is the primary key which is found in all records. It uniquely identifies the record in question. The structure of the records is shown in Figure 2.

2. DIRECTORY. The directory decodes the key into an address of a head of a list. It is a collection of indexes, one for each attribute in the file. The attribute name-value pair of the records to be matched with a term in a query is known as a key. A query is a disjunction or conjunction of one or more terms or negated terms. The output of the directory for a given key specifies the length of the list, in addition to the address of the head of the list.

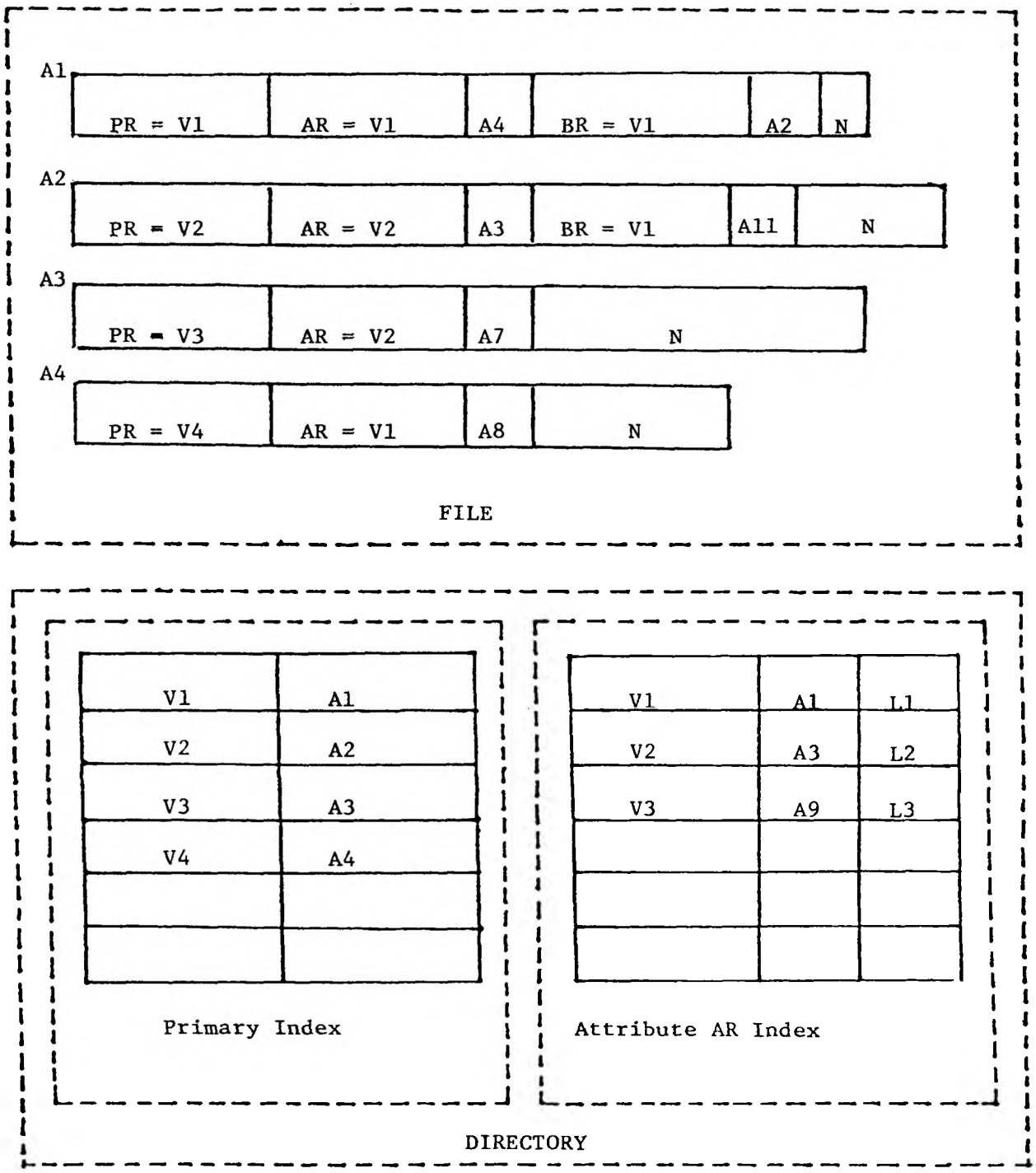
To satisfy a query, the directory is decoded for each term in the query to obtain the head of the list. If the query is a conjunction of terms, then the records of the shortest list are retrieved, and checked for the presence of other terms of the query. If the query is a disjunction of terms, then all records associated with all the terms of the query are retrieved. Negated terms require the searching of all the records in the file which is accomplished by the use of the primary index since this index has pointers to all the records in the file.

The length of the list obtained by decoding the directory is used to provide presearch statistics such as the anticipated number of records which would satisfy the query.

There are a number of possible structural representations of the directory. The sequential and B-Tree representation are considered in the following analysis.

3. SEQUENTIAL STRUCTURE. In this type of structure, each attribute is associated with an index. The index entry consists of an attribute value, head address and the length of the list, as shown in Figure 2. The primary index has one entry for each record in the file.

The index entries are not maintained in any specific order. A new index entry for an attribute is created by appending the entry to the index at the end. Deletion of an index entry is carried out by turning on an associated bit with the entry.



A1, A2.....: Address
PR.....: Primary Attribute
L1, L2.....: List Length

V1, V2.....: Value
AR, BR.....: Attribute

Fig. 2. FILE AND SEQUENTIAL INDEX

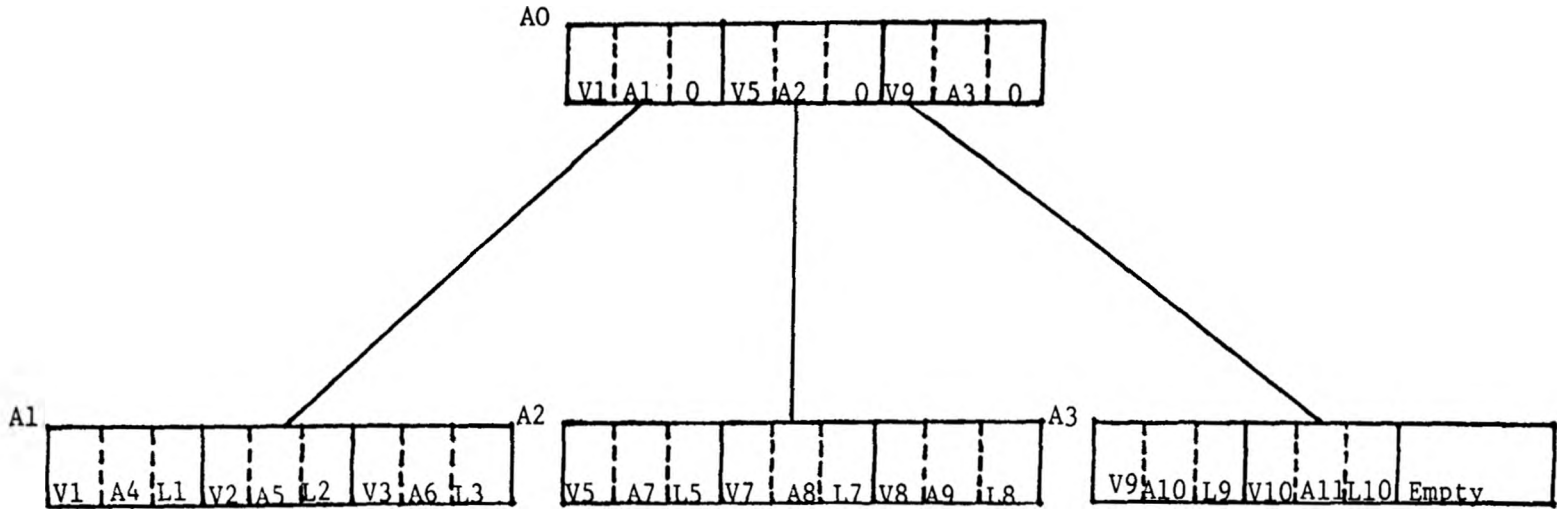
This type of arrangement is suited for files in which the attribute index is small and can be kept in core memory for the duration of the search.

4. B-TREE STRUCTURE. The B-Tree structured directory is suited for files which have a large number of values for an attribute. The directory maintains an index for each attribute. The index for a particular attribute is tree structured as shown in the Figure 3. More information on B-Tree structure can be obtained from references [4], [5] and [6].

Each entry in an index block is a value-address-length tuple and each block has some integral number of tuples. The lowest level of the tree has one entry for every value of the attribute. The blocks at the lowest level are successively indexed, until there remains only one index block at the root.

The number of entries per index block y is known as fanout. To facilitate insertions and deletions in the index, each index block is allocated some reserve space. The insertion and deletion algorithms maintain the effective fanout y_{eff} between y and y/2 and all leaves at the same level.

The insertion process is as follows. The appropriate location of the new value in the lowest level block is found. If there is space in the index block, the entry is inserted and the insertion process is complete.



A1, A2.....: Addresses
0 : Null Addresses

V1, V2.....: Values

L1, L2.....: List Length

Fig. 3 B-TREE STRUCTURED INDEX

If the block is already full, then a new block is fetched, half the entries are transferred to the new block and a new index entry is created at the next higher level block. This higher level block may become full, necessitating a split. The splitting process may propagate all the way up to the root, in which case, a new root block is created with two entries, one for the old root block and one for the new block at that level.

During the deletion process, an entry from the block may be removed, leaving less than y/2 entries. If the total number of entries in the block and its neighbor is less than y, then the blocks should be combined and the entry corresponding to one of the blocks in the higher level is deleted. This deletion process may propagate and lead to the deletion of an entry in the root block. The root block itself may be deleted, if it has only one entry, thus reducing the height of the tree by one.

The height of the B-Tree for a given index depends on the average number of attribute values n_a for an attribute name and the effective fanout y_{eff} . It is given by

$$x = \text{CEIL}(\log_{y_{eff}} n_a) \quad \text{Levels} \tag{3.1}$$

B. RECORD SIZE

Each logical record is a collection of attribute name-value tuples. Associated with each tuple is a link field which indicates the next record possessing the same tuple. Since the name and the value are of variable length, two separator characters are necessary to mark them. If A denotes the average attribute name length, V, the average attribute value length, a', the average number of attribute name-value tuples per record, P, the link field length, N, the average non indexed data length per record, then the Record Size (R) is

$$R = a' * (A + V + P + 2) + N \quad \text{Characters} \tag{3.2}$$

If all the attribute name-value pairs occur in the same order and position within a record, then the attribute name need not be explicitly stored within each record. In this case,

$$R = a' * (V + P) + N \quad \text{Characters} \tag{3.3}$$

The number of records that can be stored per block Bfr, can be computed using equations 2.1 or 2.3.

C. DIRECTORY SIZE

The index entry length (R_i) is the sum of the value field, head address field and the list length field. The length of the head address field and the list length field can be assumed to be of the same length (P). Then, R_i is given by

$$R_i = V + 2 * P \quad \text{Characters} \tag{3.4}$$

a. SEQUENTIAL STRUCTURED DIRECTORY. The fanout or the number of index entries per block, y_{eff} is given by

$$y = \text{FLOOR}(B \ / \ R_i \ ) \tag{3.5}$$

The directory size is the product of the number of attributes in the file and the average index size of an attribute. If a denotes the total attribute name-value tuples in the file and n_a denotes the average number of entries per attribute in each index then the directory size (S_d) is

$$S_d = \text{CEIL}(n_a \ / \ y) * B * a \quad \text{Characters} \tag{3.6}$$

b. B-TREE STRUCTURED DIRECTORY. The number of blocks necessary for an index with x levels is calculated as follows. If i_1 denotes the number of blocks in level 1 , then

$$i_1 = \text{CEIL}(n_a \ / \ y_{eff} \ ) \quad \text{Blocks} \tag{3.7}$$

and the number of blocks necessary at any level p, is given by

$$i_p \ = \text{CEIL}(i_{p-1} \ / \ y_{eff} \ ) \quad \text{Blocks} \tag{3.8}$$

The size of the directory S_d , is sum of the size of index for each attribute.

$$S_d = a * B * \sum_{p=1}^{p=x} i_p \quad \text{Characters} \tag{3.9}$$

The number of blocks necessary for n records is CEIL(n / Bfr). The total space (S) for the file and the directory, if there are n records in the file is

$$S = S_d + B * CEIL(n / Bfr) \quad \text{Characters} \tag{3.10}$$

D. TIME TO FETCH

In order to satisfy a query, the directory is decoded and the records associated with a particular list are retrieved.

For a sequential structured directory, half the number of blocks in the attribute index have to be searched to get the desired entry. With an average of n_a entries in each index, the number of blocks per index is $CEIL(n_a / y)$. The index blocks may not be contiguous on the storage device and therefore each access of the index block requires a seek, a latency and a block transfer operation or (s + r + Btt) ms. If T_{fd} denotes the time to decode the directory then,

$$T_{fd} = (1 / 2) * CEIL(n_a / y) * (s + r + Btt) \quad ms \tag{3.11}$$

For a tree structured directory, the root block has to be fetched and the appropriate entry in the index blocks has to be followed until the lowest level block is obtained which provides the head address and the list length. The required blocks may not be contiguous and therefore a seek, a latency, and a block transfer operation is required to access each block. The time required to decode the directory (T_{fd}) is

$$T_{fd} = x * (s + r + Btt) \quad ms \tag{3.12}$$

Every query can be considered as a disjunction of subqueries. Each subquery consists of

1. A term or a conjunction of terms with at least one non-negated term (type 1 subquery).
2. A negated term or a conjunction of negated terms (type 2 subquery).

Let N_t denote the average number of terms per subquery and N_n denote the average number of non-negated terms per subquery. For the first type of subquery, the directory is decoded for all the N_n non-negated terms of the subquery and the shortest list is determined. The records in the shortest list are retrieved and checked for the presence or absence of the required attribute name-value pairs. It is possible to group records according to a particular list to reduce block accesses. This grouping of records is possible according to only one list. Since there are a number of lists, one for each attribute name-value pair, it is assumed that retrieval of each record of the list requires a block access. Since the blocks may not be contiguous on the storage device, a seek, a latency and a block transfer operation is required for each block transfer. The time required to satisfy this type of query is

$$T_{fn} = N_n * T_{fd} + L_s * (s + r + Btt) \text{ ms} \qquad (3.13)$$

For the second type of subquery, all the records in the file have to be retrieved to try to satisfy the subquery. This is accomplished by retrieving the records through the primary index, and checking for the absence and presence of the necessary attribute name-value pairs. Retrieval of all the records through primary index requires $n * T_{fd}$ ms. Retrieval of all the records can be accomplished more efficiently by exhaustively reading the file which requires T_{xd} ms as calculated

by equation 3.25 and 3.26. The time required (T_{fg}) to satisfy this type of subquery is

$$T_{fg} = n * T_{fd} + n * (s + r + Btt) \text{ ms} \quad (3.14)$$

The time to retrieve all the records in the query which is a disjunction of subqueries is the sum of the fetch times of individual subqueries. If n_1 and n_2 denote the the number of subqueries of type 1 and type 2 respectively, then

$$T_f = n_1 * T_{fn} + n_2 * T_{fg} \text{ ms} \quad (3.15)$$

E. TIME TO INSERT

With every insertion of a record, the index for each attribute has to be updated. The inserted record may contain attributes for which

1. Entries exist in the index. Let a'_o denote the average number of attributes per inserted record for which values exist in the index.
2. There are no entries in the index. Let a'_n denote the average number of attributes per inserted record for which new values have to be created in the attribute index.

In the case of a sequential structured directory, for a'_o attributes, the index entry is located and fetched, and the current head address of the index entry is inserted in the link field of the new record. The head address field of the index entry is written with the address of the inserted record. The time for this process can be quantified as

$$T_{io} = a'_o * (T_{fd} + T_{rw}) \text{ ms} \quad (3.16)$$

For a'_n attributes, the insertion procedures involves fetching the last block of the appropriate index and creating a new entry. It can be assumed that the address of the last block of the index is available in core memory. If T_{in} denotes the time for this process, then

$$T_{in} = a'_n * (s + r + Btt + T_{rw}) \text{ ms} \quad (3.17)$$

For a B-Tree structured directory, the insertion process causes one of the two conditions listed above to occur. For a'_o attributes, the insertion process requires $T_{fd} + T_{rw}$ as before.

For a_n attributes, a new entry has to be inserted in the appropriate index at the leaf. This would involve a search from the root block, until the appropriate leaf block is found. With a probability of (1 / (y / 2)) an index block split will be necessary. The entries have to be distributed in the new block followed by rewriting of the new block and its parent block with a new entry. The time for the this process T_{in} is

$$T_{in} = a'_n * (T_{fd} + T_{rw} + 2 / y * (s + r + Btt + 2 * T_{rw})) \text{ ms} \quad (3.18)$$

The time to update the directory for an insertion of a record is

$$T_{id} = T_{io} + T_{in} \text{ ms} \quad (3.19)$$

After updating the directory for the record insertion, the record has to be inserted in the desired block, which has to be fetched and rewritten with the record requiring $s + r + Btt + T_{rw}$ ms. The total time to update the index and insert the new record is

$$T_i = T_{id} + (s + r + Btt + T_{rw}) \text{ ms} \quad (3.20)$$

F. TIME TO DELETE

1. DELETION OF AN ATTRIBUTE NAME-VALUE PAIR. The link field of each record contains a delete bit which indicates the logical presence of that attribute name-value pair. Deletion of an attribute name-value pair can be accomplished by logically turning on the associated delete bit. It is assumed that the deletion procedure specifies the primary key value of the record in which the attribute name-value pair is to be deleted in order to make an unambiguous deletion.

In both types of directory structures, the appropriate primary index entry has to be located and then the record has to be fetched. This requires $T_{fd} + (s + r + Btt)$ ms. The record is rewritten in the same location with a delete bit turned on. This requires T_{rw} ms. With the probability 1/L, the attribute name-value pair may be the only record in the list. Deletion of this attribute name-value pair would require that index entry be logically deleted which would take T_{rw} ms. The time required to delete an attribute from a record is

$$T_d = T_{fd} + (s + r + Btt) + T_{rw} + (1 / L) * T_{rw} \text{ ms} \quad (3.21)$$

2. DELETION OF RECORDS. Deletion of an entire record is accomplished by turning on a delete bit which is associated with the primary attribute name-value pair. The time necessary for this operation is T_{fd} + (s + r + Btt) + T_{rw}. The primary index entry has to be logically deleted, which requires rewriting the primary index entry within T_{rw} ms. If T_{dr} denotes the time to delete the entire record, then

$$T_{dr} = T_{fd} + (s + r + Btt) + 2 * T_{rw} \quad ms \tag{3.22}$$

G. TIME TO UPDATE

1. IN-PLACE UPDATE. When the attribute value is modified to another existing value, the process is referred to as in-place update. For each attribute to be updated, the record has to be physically deleted from the list and inserted as the head of a new list. In order to physically delete an an attribute name-value pair, the index entry has to be located and on the average, half the list has to be searched to locate the desired attribute name-value pair. This requires T_{fd} + L / 2 * (s + r + Btt) ms. The link field of the predecessor of the attribute name-value pair to be deleted, has to be rewritten with the address of the successor attribute name-value pair, requiring T_{rw} ms. The record to be updated has to be inserted as the head of the new list which requires locating the appropriate entry for the new value and changing the head address to point the updated record. This requires T_{fd} + T_{rw} ms. Finally, the updated record is written in the same location with the link field of the modified attribute containing the head address of the entry prior to insertion. The rewriting of

the record would require s + r + Btt ms because the list update operation may have moved the seek mechanism to a different track. If a'_u denotes the number of in-place update attributes, then

$$T_u = a'_u * (T_{fd} + (L / 2) * (s + r + Btt) + T_{rw} + T_{fd} + T_{rw}) + s + r + Btt \quad ms \tag{3.23}$$

2. UPDATE WITH NEW ATTRIBUTES. A record updated with the addition of an attribute name-value pair has to be rewritten in a new location, since it cannot fit in the old location. The record has to be logically deleted from the current position which requires T_{dr} ms and then inserted in the new location which requires T_i ms.

$$T_u = T_{dr} + T_i \quad ms \tag{3.24}$$

H. TIME FOR EXHAUSTIVE READ

Exhaustive reading requires the fetching of all records in the file. Since each record is uniquely identified by the primary key value, exhaustive reading can be done through primary index.

In the case of a sequential structured directory, the entries in the primary index have to be fetched and the only record associated with each entry in the index has to be retrieved. The primary index has n entries, one for each record, and therefore CEIL(n / y_{eff}) blocks have to be read. If T_{xd} denotes the time to read the directory exhaustively, then

$$T_{xd} = CEIL (n / y_{eff}) * (s + r + Btt) \text{ ms} \tag{3.25}$$

In the case of B-Tree structured directory, it can be assumed that the index blocks which are in the path from the root to the lowest level of the index are available in core memory. Most practical file systems do not have index depth exceeding three to four levels and therefore only a few index blocks have to be made available in the core.

From each second level index block, with y_{eff} accesses, it is possible to access all the child level 1 index blocks. Similarly, with $y_{eff}^{2} + y_{eff}$ accesses, it is possible to access all level 1 descendants of a level 3 block. With an x level tree, the number of accesses to exhaustively read the directory is

$$T_{xd} = \sum_{p=1}^{p=x-1} y_{eff}^{p} \quad ms \tag{3.26}$$

If horizontal pointers are maintained linking all level 1 blocks, then exhaustive read can be performed by reading all level 1 blocks sequentially. In this case, equation (3.25) applies after taking into account the reduced fanout due to pointers linking level 1 index blocks.

The total time to exhaustively read the records is the sum of the time to read the directory and n records in the file.

$$T_x = T_{xd} + n * (s + r + Btt) \quad ms \tag{3.27}$$

I. TIME TO REORGANIZE

Reorganization becomes necessary to reclaim space occupied by deleted records and attributes. When available space becomes critical, reorganization is necessary. Reorganization may also be necessary to improve performance when lists become excessively long with deleted attributes and records.

1. DIRECTORY REORGANIZATION.

Reorganization of the directory requires reading the index entries of all the attributes in the file and rewriting it after removing the deleted entries. In the case of a sequential structured directory, for each index CEIL(n_a / y_{eff}) blocks have to be read and rewritten after removing the deleted entries. The time to reorganize the directory, T_{yd} is

$$T_{yd} = a * (CEIL(n_a / y_{eff}) * (s + r + Btt) + T_{rw}) \quad ms \quad (3.28)$$

For a B-Tree structured directory, the physical deletion of an entry may require combining the block and its neighbor and thus leading to the deletion of an entry in the higher level block. This process may propagate up to the root. The probability of combining two blocks is 1 / (y / 2). The time required for directory reorganization would be

$$T_{yd} = a * (x * (s + r + Btt) + T_{rw} + 2 / y * (s + r + Btt + 2 * T_{rw})) \text{ ms} \quad (3.29)$$

2. FILE REORGANIZATION. The records have to be read one at a time and written at the new location requiring $T_x + n * (s + r + Btt)$ ms. Since the address of the record has been modified, the link fields and the head address fields pointing to this record have to be modified. This is done by stepping through the lists until the desired record is located and updating the predecessor record link field or the head address field of the index entry with the new address. This process has to be carried out for n records each with a' attribute name-value pairs, requiring $n * a' * (T_{fd} + L / 2 * (s + r + Btt) + T_{rw})$ ms. The time required for the reorganization of the directory and the file is

$$T_y = T_{yd} + T_x + n * (s + r + Btt) + n * a' * (T_{fd} + L / 2 * (s + r + Btt) + T_{rw}) \text{ ms} \quad (3.30)$$

The reorganization process can be carried out in increments, since after rewriting each record and updating the link fields, the file is in a usable state. With incremental reorganization, it is possible to service requests between reorganizations and thus the system can remain on line at all times.

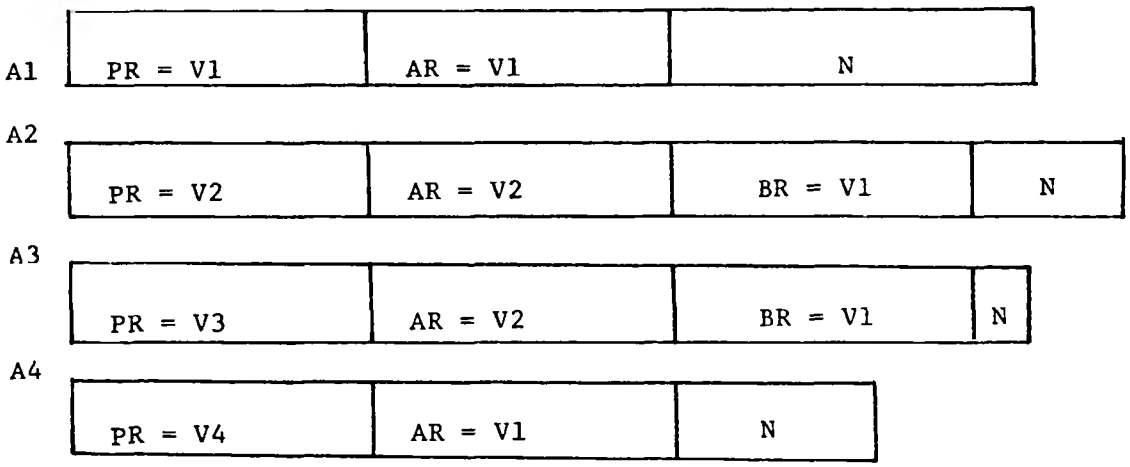
IV. INVERTED FILE SYSTEM

A. STRUCTURE

The inverted file structure, like the multilist file structure, is composed of two parts, file and directory.

1. FILE. The file is a collection of logical records. Each record consists of attribute name-value pairs and non indexed data and is distinguished by a special attribute name-value pair called the primary key. The structure of the records in the file is shown in fig 4.

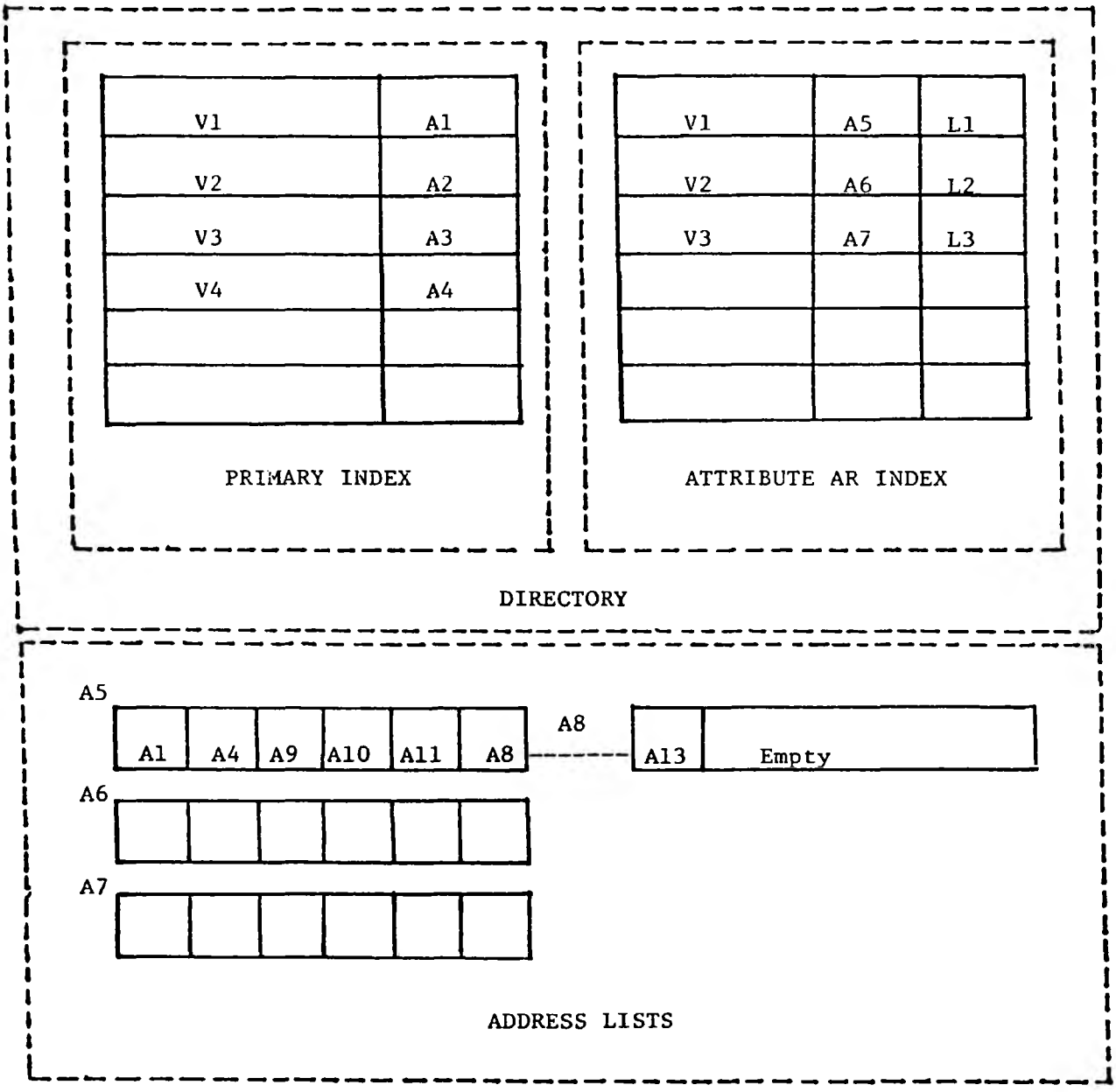
2. DIRECTORY. The directory is a collection of indexes, one for each attribute in the file. Each index entry consists of an attribute value, an address and a list length field. The address field points to a list of addresses of all records possessing the attribute name-value pair in question (figure 5). The list length field indicates the number of records possessing the attribute name-value pair. The addresses in the address lists are maintained in a monotonic sequence, which improves fetch time of records. The primary index is a special index in which the address field of an index entry directly points to a record rather than a list of addresses since each primary value is associated with only one record. The primary index is identical to the primary index of a multilist file system.



PR: Primary Attribute
A1, A2: Address

AR, BR....: Attributes
V1, V2....: Values

Fig. 4. INVERTED LIST RECORDS



V1, V2.....: Values
L1, L2.....: List Length

A1, A2,.....: Addresses

Fig. 5. INVERTED LIST DIRECTORY AND ADDRESS LISTS.

To satisfy a query, the directory is decoded for each term in the query to obtain the address lists. If the query is a conjunction of terms, an intersection of the address lists gives the addresses of the records satisfying the query. If the query is a disjunction of terms, then an union of address lists is performed. To satisfy a query consisting of a conjunction of negated and non-negated terms the addresses found in the negated term's list are deleted from the non-negated term's list and the records corresponding to the remaining addresses are retrieved. An isolated negated term or a conjunction of negated terms require the search of the entire file, which is best accomplished by accessing all the records through the primary index. The length field in the index entry is used to provide presearch statistics. The directory is assumed to have a sequential or a B-Tree structure which have been discussed in previous sections.

The address lists are stored in contiguous blocks on the storage devices and in most practical file systems only a few blocks are necessary to accommodate each address list because of the large number of addresses that can be stored in a block. Each address list can be visualized as a collection of L small records, each of size P, arranged in contiguous blocks. The time requirements to manipulate the address list are analysed and used later to evaluate the performance of an inverted file system.

In order to locate an address in the address list after decoding the directory, the address list blocks have to be read sequentially until the desired block is obtained. This requires (L * P / t') ms. The use of bulk transfer rate is appropriate here since the address

blocks are contiguous and are read sequentially. If T_{fa} denotes the time to fetch an address from the address list, then

$$T_{fa} = 1 / 2 * (L * P / t') \quad ms \tag{4.1}$$

Insertion of an address in the address list requires locating the position of insertion and rewriting all addresses beyond the insertion point. Locating the position of insertion requires T_{fa} ms. On the average, half the blocks of the address list are rewritten after locating the insertion point requiring (1 / 2) * CEIL(L / y_a) * (Btt + T_{rw}) ms. The seek and the latency can be neglected here since it is assumed that all the address blocks of a given list are contiguous. The time required to complete the insertion of an address in the address list is

$$T_{ia} = T_{fa} + 1 / 2 * (CEIL(L / y_a) * (Btt + T_{rw})) \text{ ms} \tag{4.2}$$

B. FILE SIZE

Each record is a collection of attribute name-value pairs and non indexed data. Since the name and value fields are of variable length, two separator characters are necessary to mark them. Unlike the multilist structure, no link fields are necessary. If A denotes the average attribute name length, V, the average value length, and N, the average non indexed data length per record, then the record size R, is estimated as

$$R = a' * (A + V + 2) + N \quad \text{Characters} \tag{4.3}$$

If the attribute values can be identified without explicitly storing the attribute name within the record, then the record size R is estimated as

$$R = a' * V + N \quad \text{Characters} \tag{4.4}$$

The space requirements for the address lists is estimated as follows. Each address block contains a fixed number of addresses and a pointer to another block to accommodate additional addresses. If P denotes the size of the address and y_a denotes the number of addresses per block, then

$$y_a = \text{FLOOR}((B - P) / P) \quad \text{Addresses/Block} \tag{4.5}$$

The total space for the address lists, assuming that L denotes the average number of addresses per attribute name-value pair, a, the total number of attribute name-value pairs in the file and n_a , the average number of entries per index , is is

$$S_a = a * n_a * B * \text{CEIL}(L / y_a) \quad \text{Characters} \tag{4.6}$$

The total space requirement for the file, the directory and the address lists assuming that there are n records in the file, is

$$S = S_d + S_a + B * \text{CEIL}(n / Bfr) \quad \text{Characters} \tag{4.7}$$

where S_d , the space requirement for the directory, can be estimated using equations 3.6 through 3.9.

C. TIME TO FETCH

To satisfy a query, the directory has to be decoded to get the index entry of each term in the query. The addresses in the address lists have to be retrieved. Every query can be considered as a disjunction of subqueries as discussed in the previous chapter.

If the subquery is a type 1 subquery, then the directory has to be decoded for each of the N_t terms of the subquery and the address list has to be retrieved. The time required to fetch all the addresses in the address list is $(L * P / t')$ or $2 * T_{fa}$ ms. Finally, the records associated with the addresses which satisfy the query have to be retrieved. If L_s denotes the shortest address list, a fraction of records f, of this list, satisfying the query is retrieved and the time required for this process can be estimated as $f * L_s * (s + r +$ Btt) ms. The time required to satisfy this type of subquery is

$$T_{fn} = N_t * (T_{fd} + 2 * T_{fa}) + f * L_s * (s + r + Btt) \text{ ms} \qquad (4.5)$$

For the subquery with a negated term or a conjunction of negated terms, the address of all the records have to be retrieved through the primary index, in addition to the address lists corresponding to each term of the query. The addresses corresponding to the negated terms have to be deleted from the primary address list and the remaining records have to be retrieved. The addresses of all the records in the file can be retrieved by exhaustively reading the primary index, the time for which can be computed using equations 3.25 and 3.26. A fraction q of all the n records in the file have to be retrieved. The time requirements to process a negated subquery is

$$T_{fg} = T_{xd} + N_t * (T_{fd} + 2 * T_{fa}) + q * n * (s + r + Btt) \text{ ms} \quad (4.9)$$

If the query contains n_1 subqueries of the first type and n_2 queries of the second type, then

$$T_f = n_1 * T_{fn} + n_2 * T_{fg} \text{ ms} \quad (4.10)$$

D. TIME TO INSERT

The inserted record has a'_o attribute name-value pairs for which entries exist in the directory and a'_n attribute name-value pairs for which new entries have to be created in the directory. The directory has to be decoded for the attribute name-value pairs existing in the index and the list length field of the appropriate entry has to be incremented by one. This requires $T_{fd} + T_{rw}$ ms. For new attribute name-value pairs, new indexes have to be created requiring s + r + Btt ms. The total time to update the directory T_{id} for an insert operation is

$$T_{id} = a'_o * (T_{fd} + T_{rw}) + a'_n * (s + r + Btt) \text{ ms} \quad (4.11)$$

For the a'_o attributes with entries existing in the index, the address of the record being inserted has to placed in position in the address lists, requiring T_{ia} ms as shown in equation 4.2. For a'_n attributes, new address lists have to be created requiring s + r + Btt + T_{rw} ms. The time required to update the directory and the address list for an insertion operation is

$$T_{ua} = T_{id} + (a'_o * T_{ia}) + a'_n * (s + r + Btt) \text{ ms} \quad (4.12)$$

Finally, the record has to be written, which requires s + r + Btt ms. The total time to complete the insertion of a record is

$$T_i = T_{ua} + (s + r + Btt) \text{ ms} \quad (4.13)$$

E. TIME TO DELETE

1. DELETION OF AN ATTRIBUTE NAME-VALUE PAIR. Deletion of an attribute is accomplished by turning on the delete bit which is associated with the address in the address list. Since the delete bit is located with the address and not with the record, it is not possible to ascertain the deletion of an attribute name-value pair when the record is read through some other index. Therefore, another bit associated with the attribute name-value pair is also turned on. The deletion procedure requires that the primary key value be specified to make an unambiguous deletion. The primary index has to be decoded to obtain the address of the record in which the attribute name-value pair has to be deleted. This requires T_{fd} ms. The index entry for attribute name-value pair to be deleted has to be located and retrieved, which requires T_{fd} ms. The address list has to be searched and the the appropriate address located for the delete bit to be turned on. This requires $T_{fa} + T_{rw}$ ms. The record has to be fetched and the delete bit associated with the attribute name-value pair has to be turned on, requiring s + r + Btt + T_{rw} ms. With a probability (1 / L), the index entry may have to deleted since the attribute being deleted may have been the only one in the address list. This requires (1 / L) * T_{rw} ms. The time required to

selectively delete an attribute is the sum of all these components and is given by

$$T_d = 2 * T_{fd} + T_{fa} + T_{rw} + (1 / L) * T_{rw} + s + r + Btt + T_{rw} \text{ ms} \quad (4.14)$$

2. DELETION OF A RECORD. Deletion of a record is accomplished by writing a special marker in the list length field of the primary index entry and by turning on the delete bit associated with the primary attribute value. The time required is

$$T_{dr} = T_{fd} + T_{rw} + s + r + Btt + T_{rw} \text{ ms} \quad (4.15)$$

F. TIME TO UPDATE

1. IN PLACE UPDATE. In place update involves the deletion of the record address from the current list and inserting the address in a new list. Deletion of the address requires T_d ms and insertion of the address in the new list requires $T_{fd} + T_{ia}$ ms. The record has to be fetched and rewritten with updated attribute name-value pairs requiring $s + r + Btt + T_{rw}$ ms. The time required for an in-place update can be estimated as

$$T_u = a_u * (T_d + T_{fd} + T_{ia}) + (s + r + Btt + t_{rw}) \text{ ms} \quad (4.16)$$

2. UPDATE WITH NEW ATTRIBUTES. Since the updated record cannot be fitted in the old location, the record has to be rewritten in the new location after deleting it from the old location. The deletion of

record requires T_{dr} ms and the insertion of the record requires T_i ms.

$$T_u = T_{dr} + T_i \quad \text{ms} \tag{4.17}$$

G. TIME FOR EXHAUSTIVE READ

Exhaustive read can be performed by accessing all records through primary index. The time to exhaustively read the directory can be calculated using equations 3.25 and 3.26. For each of the n entries in the primary index, the records have to be read, which requires n * (s + r + Btt) ms. The time required for exhaustive read is

$$T_x = T_{xd} + n * (s + r + Btt) \text{ ms} \tag{4.18}$$

H. TIME FOR REORGANIZATION

The reorganization procedure for the file system is as follows. First, the directory is reorganized, the time for which can be computed using equations 3.28 and 3.29.

The address lists may have to be moved to new locations, which requires reading and rewriting the address lists. If the time for rewriting an address list is the same as reading it, then the time to reorganize the address lists T_{ya} , for all the a * n_a address lists is

$$T_{ya} = a * n_a * 2 * (L * P / t') \quad \text{ms} \tag{4.19}$$

The records have to be read exhaustively and rewritten, which requires T_x + n * (s + r + Btt) ms. For each record, the old address of the record has to be deleted in all the a' address lists of which this

record is a member and the new address has to be inserted in the right location to maintain the sequence. Deletion of an address from the address list requires T_d ms and insertion of the address in the appropriate location in the address list requires T_{ia} ms as computed by equations 4.14 and 4.4. The time required for the reorganization of the entire file is

$$T_y = T_{yd} + T_{ya} + T_x + n * (s + r + Btt) + n * a' * (T_d + T_{ia}) \text{ ms} \quad (4.20)$$

The reading and rewriting of records can be done in increments. Thus, it is possible to service requests between reorganizations and the system can remain on line at all times.

V. RESULTS

The performance measurement formulas derived in the preceding chapters have been incorporated into a program to compute the results. The results for different sets of input values were studied and the following observations can be made.

For the input values of table I, the performance measurement results for sequential and tree structured directory are shown in table II. It can be observed that for multilist and inverted file systems with a large number of entries per index, the B-Tree structured directory exhibits better performance over the sequential structured directory. This is due to the fact that with a B-Tree structured directory, fewer index blocks have to be retrieved to obtain a desired index entry as compared to a sequential structured directory.

The input values in table III and table IV are identical except that the input values of table III represents a single non-negated term query (type 1 subquery) while the input values of table IV represent a single negated term query (type 2 subquery). It can be observed that the fetch time values of records satisfying a subquery consisting of a negated term or a conjunction of negated terms (type 2 subquery) is very large. This is because negated subquery processing amounts to searching the entire file. This type of subquery processing may have to be relegated to background (off line) or restricted during peak hours to avoid performance deterioration.

The fetch time of an inverted file system, in most cases, is better than the fetch time of a similar multilist file system. However, The fetch time of a multilist file system is better than that of a similar inverted file system when the query consists of a single non-negated term or a disjunction of non-negated single terms as illustrated by table V.

A. CONCLUSION

The preceding analysis makes no attempt to determine how to measure many of the input parameters such as ratio of query response to total number of records, average number of entries per index, etc. A user trying to select a file system is encumbered to understand the structure of the file systems and carefully estimate the input parameters to get meaningful results. It is suggested that a layer of software be designed which uses record templates, graphics, etc so that a user can actually enter a model file as he sees it, and from which the input parameters necessary for this software package can be calculated. This would make the performance measurement techniques transparent to the user.

BIBLIOGRAPHY

1. Weizenbaum, J., "Knotted List Structures", Communications of the ACM, Vol III, No 4, 1960, 161-165.

2. Lefkovitz, David, File Structures For On-Line Systems, Hayden Book Company,Inc, New Jersey, 1969.

3. Wiederhold, Gio, Database Design, McGraw-Hill, 1983.

4. Yao,A., "Random 3-2 Trees", Acta Informatica, vol 2, No 9, 1978.

5. Bayer, R., "Symmetric Binary B-trees: Data Structures And Maintenance Algorithms", Acta Informatica, Vol 1, 1972, 290-306.

6. Landauer, W.I., "The Balanced Tree And Its Utilization In Information Retrieval", Transactions On Electronic Computer of IEEE, Vol EC-XII, No 5, 1963.

VITA

Ashok Chandramouli was born in Attur, India. He received his primary and secondary education in Madras, India. He received a Bachelor of Engineering degree in Mechanical Engineering from University of Madras, India in July 1984.

He has been enrolled in the Graduate School of The University of Missouri-Rolla since August 1985 in the Department of Computer Science.

APPENDIX A

NOMENCLATURE

A	Attribute name length
a'	Average number of attributes per record
a'_u	Average number of update attributes per update
a'_o	Average number of insert attributes with values in the index
a'_n	Average number of attributes for an insert with new values
Bfr	Blocking factor
Btt	Block transfer time
c	Compuation time
f	Ratio of query response to L_s
G	Gap size
i_p	Number of blocks at level p for a B-Tree index
L	Average number of records per attribute name-value pair.
L_s	Shortest list length for a subquery
N_n	Avg Number of non-negated terms in a subquery
N_t	Avg number of terms in a negated and a non-negated subquery
n	Number of records in the file
n_a	Average number of entries in a index.
P	Pointer length or address length
q	Ratio of query response to n
r	Rotational Latency
R	Record length
R_i	Index entry length
s	Seek Time
S	Size of the file system

S_d	Directory size
t'	Bulk Transfer rate
T_d	Time to delete an attribute of a record
T_{dr}	Time to delete the entire record
T_f	Time to satisfy a query
T_{fd}	Time to fetch an entry in the index
T_{fn}	Time to process non-negated query
T_{fg}	Time to process negated query
T_i	Time to insert a record
T_{ia}	Time to insert an address in an address list
T_{id}	Time to update the directory for inserting a record
T_{rw}	Time to rewrite
T_u	Time to update a record
T_{ua}	time to update the directory and the address lists
T_y	Time to reorganize the file and the directory
T_{ya}	Time to reorganize the address lists
T_{yd}	Time to reorganize the directory
T_x	Time to exhaustively read the records
T_{xd}	Time to exhaustively read the directory
W	Wasted space per record
x	Number of levels in a B-Tree structured index
y	Fanout
y_a	Number of addresses per block
y_{eff}	Effective fanout

APPENDIX B

PROGRAM TO CALCULATE THE PERFORMANCE PARAMETERS

```
*PROCESS OPTIONS MAR(2,72,1) NOSOURCE;
 /*************************************************************
 **    THIS PROGRAM COMPUTES THE INVERTED AND MULTILIST      **
 **    PERFORMANCE PARAMETERS                                **
 *************************************************************/
 (SIZE): CALC:
   PROCEDURE OPTIONS(MAIN);
0 DECLARE
    (A_PRIME,
    A,
    A_UPD,
    ATTR_LEN,
    VALUE_LEN)          FIXED DEC(4),
    (REC_SZ,
    BLOCK_SZ,
    BFR)                FIXED DEC(9,2),
    PTR_SZ              FIXED DEC(3),
    N                   FIXED DEC(8),
    DISK_TYPE           FIXED DEC(2),
    TRW                 FIXED DEC(8,2),
    BTT                 FIXED DEC(9,2),
    T_PRIME             FIXED DEC(7,2),
    S                   FIXED DEC(10,2),
    R                   FIXED DEC(8,2),
    GAP                 FIXED DEC(8,2),
    TRANSFR_RTE         FIXED DEC(8,2),
    /* DENSITY IS FREE SPACE IN AN INDEX BLOCK  */
    DENS                FIXED DEC(3,2) INIT(0.69),
    (F_RAT,
    Q_RAT)              FIXED DEC(4),
    (N_N,
    N_T)                FIXED DEC(3),
    (LIST_LEN,
    LIST_SHR)           FIXED DEC(6),
    (NON_DATA,
    A_NEW)              FIXED DEC(4),
    (N_ONE,
    N_TWO)              FIXED DEC(3),
    VAL_TOT             FIXED DEC(5),
    (SDS,
    SDB)                FIXED DEC(15,2),
    (TFDS,
    TFDB)               FIXED DEC(10,2),
    (TYDS,
    TYDB,
    TXDS,
    TXDB)               FIXED DEC(15,2),
```

```
   REC_FRMT            CHAR(1);
DECLARE INFILE STREAM FILE INPUT,
        SYSIN  STREAM FILE INPUT,
        SYSPRINT PRINT FILE OUTPUT,
        (DEC,ABS,FLOOR,CEIL,ONLOC) BUILTIN;
CALL ENTER_DATA;
CALL DISP_DATA;
CALL DISK_PARM;
CALL DISK_CALC;
CALL MULT_FILE(SDS,SDB,TFDS,TFDB,TXDS,TXDB,TYDS,TYDB);
CALL INVT_FILE(SDS,SDB,TFDS,TFDB,TXDS,TXDB,TYDS,TYDB);

/*****************************************
**  READ THE NECESSARY DATA             **
*****************************************/

ENTER_DATA: PROCEDURE;
 GET FILE (SYSIN) EDIT (BLOCK_SZ) (COL(1),F(9));
 CALL VALID(1,9999,BLOCK_SZ);
 GET FILE(SYSIN) EDIT(N) (COL(1),F(8));
 CALL VALID(1,9999999,N);
 GET FILE(SYSIN) EDIT(N_N) (COL(1),F(3));
 CALL VALID(0,9,N_N);
 GET FILE(SYSIN) EDIT(N_ONE) (COL(1),F(3));
 CALL VALID(0,9,N_ONE);
 GET FILE(SYSIN) EDIT(N_T) (COL(1),F(3));
 CALL VALID(1,99,N_T);
 GET FILE(SYSIN) EDIT(N_TWO) (COL(1),F(3));
 CALL VALID(0,99,N_TWO);
 GET FILE(SYSIN) EDIT(VAL_TOT) (COL(1),F(6));
 CALL VALID(1,9999,VAL_TOT);
 GET FILE(SYSIN) EDIT(LIST_LEN) (COL(1),F(6));
 CALL VALID(1,99999,LIST_LEN);
 GET FILE(SYSIN) EDIT(LIST_SHR) (COL(1),F(6));
 CALL VALID(1,99999,LIST_SHR);
 GET FILE(SYSIN) EDIT(F_RAT) (COL(1),F(4));
 CALL VALID(1,100,F_RAT);
 GET FILE(SYSIN) EDIT(Q_RAT) (COL(1),F(4));
 CALL VALID(1,100,Q_RAT);
 GET FILE(SYSIN) EDIT(PTR_SZ) (COL(1),F(3));
 CALL VALID(1,9,PTR_SZ);
 GET FILE(SYSIN) EDIT(REC_FRMT)(COL(1),A(1));
 IF (REC_FRMT = 'F') | (REC_FRMT = 'V') THEN;
 ELSE PUT SKIP LIST ('INVALID RECORD FORMAT');
 GET FILE(SYSIN) EDIT(A_PRIME) (COL(1),F(4));
 CALL VALID (1,99,A_PRIME);
 GET FILE(SYSIN) EDIT(A) (COL(1),F(4));
 CALL VALID (1,99,A);
 GET FILE(SYSIN) EDIT(ATTR_LEN) (COL(1),F(4));
 CALL VALID (0,25,ATTR_LEN);
 GET FILE(SYSIN) EDIT (VALUE_LEN) (COL(1),F(4));
 CALL VALID (1,25,VALUE_LEN);
 GET FILE(SYSIN) EDIT (NON_DATA) (COL(1),F(4));
 CALL VALID (1,200,NON_DATA);
 GET FILE(SYSIN) EDIT (A_UPD) (COL(1),F(4));
```

```
 CALL VALID (1,99,A_UPD);
 GET FILE(SYSIN) EDIT (A_NEW) (COL(1),F(4));
 CALL VALID (1,99,A_NEW);
 GET FILE (SYSIN) EDIT(DISK_TYPE) (COL(1),F(2));
 CALL VALID(1,2,DISK_TYPE);
END ENTER_DATA;

/************************************
** DISPLAY INPUT DATA              **
************************************/

DISP_DATA: PROCEDURE;
 PUT FILE(SYSPRINT) PAGE;
 PUT SKIP(2) EDIT ('INPUT PARAMETERS') (COL(20),A);
 PUT SKIP(0) EDIT ('_________________') (COL(20),A);
 PUT SKIP(2) FILE(SYSPRINT) EDIT ('BLOCK SIZE   (BYTES)',BLOCK_SZ)
   (COL(1),A,COL(75),F(10));
 PUT SKIP(2) FILE(SYSPRINT) EDIT ('TOTAL '||
 'NUMBER OF RECORDS IN THE FILE (N)',N) (COL(1),A,COL(75),F(10));
 PUT SKIP(2) FILE(SYSPRINT) EDIT ('NUMBER OF NON NEGATED TERMS '||
 'IN TYPE 1 SUBQUERY (NN)',N_N) (COL(1),A,COL(75),F(10));
 PUT SKIP(2) FILE(SYSPRINT) EDIT('NUMBER OF TYPE 1 SUBQUERIES (N1)',
  N_ONE) (COL(1),A,COL(75),F(10));
 PUT SKIP(2) FILE(SYSPRINT) EDIT ('NUMBER OF TERMS '||
 'IN TYPE 1 OR TYPE 2 SUBQUERY (NT)',N_T) (COL(1),A,COL(75),F(10));
 PUT SKIP(2) FILE(SYSPRINT) EDIT ('NUMBER OF TYPE 2 SUBQUERIES (N2)',
  N_TWO) (COL(1),A,COL(75),F(10));
 PUT SKIP(2) FILE(SYSPRINT) EDIT ('AVERAGE NUMBER OF ENTRIES '||
 'PER INDEX (NA)',VAL_TOT) (COL(1),A,COL(75),F(10));
 PUT SKIP(2) FILE(SYSPRINT) EDIT ('AVERAGE LIST LENGTH (L)'
      ,LIST_LEN) (COL(1),A,COL(75),F(10));
 PUT SKIP(2) FILE(SYSPRINT) EDIT ('AVERAGE NUMBER OF RECORDS '||
 'IN THE SHORTEST LIST (LS)',LIST_SHR) (COL(1),A,COL(75),F(10));
 PUT SKIP(2) FILE(SYSPRINT) EDIT ('THE RATIO OF QUERY RESPONSES '||
 'TO THE SHORTEST LIST (F)',F_RAT,'%') (COL(1),A,COL(75),
     F(10),X(1),A);
 PUT SKIP(2) FILE(SYSPRINT) EDIT ('THE RATIO OF QUERY RESPONSES TO '||
'TOTAL RECORDS IN THE FILE (Q)',Q_RAT,'%') (COL(1),A,COL(75),
 F(10),X(1),A);
 PUT SKIP(2) FILE(SYSPRINT) EDIT ('POINTER SIZE (P) '
  ,PTR_SZ,'BYTES') (COL(1),A,COL(75),F(10),X(1),A);
 IF (REC_FRMT = 'V') THEN
  PUT SKIP(2) FILE(SYSPRINT) EDIT ('RECORD FORMAT ','VARYING')
  (COL(1),A,COL(77),A);
 ELSE
 PUT SKIP(2) FILE(SYSPRINT) EDIT ('RECORD FORMAT ','FIXED')
  (COL(1),A,COL(79),A);
 PUT SKIP(2) EDIT ('AVERAGE # OF ATTRIBUTES PER RECORD (A'')',A_PRIME)
  (COL(1),A,COL(75),F(10));
 PUT SKIP(2) EDIT('TOTAL # OF ATTRIBUTES IN THE FILE (A) ',A)
 (COL(1),A,COL(75),F(10));
 PUT SKIP(2) EDIT ('AVERAGE ATTRIBUTE_NAME LENGTH ',ATTR_LEN)
 (COL(1),A, COL(75),F(10));
 PUT SKIP(2) EDIT( 'AVERAGE ATTRIBUTE_VALUE LENGTH (V)',VALUE_LEN)
 (COL(1),A,COL(75),F(10));
```

```
 PUT SKIP(2) EDIT( 'AVERAGE NON INDEXED DATA PER RECORD (N)',
  NON_DATA,'BYTES') (COL(1),A,COL(75),F(10),X(1),A);
 PUT SKIP(2) EDIT( 'AVERAGE NUMBER OF UPDATE ATTRIBUTES '||
  'PER UPDATE OPERATION (AU)',A_UPD) (COL(1),A,COL(75),F(10));
 PUT SKIP(2) EDIT( 'AVERAGE NUMBER OF NEW ATTRIBUTES '||
 'PER INSERT OPERATION (AN)', A_NEW) (COL(1),A,COL(75),F(10));
 PUT SKIP(2) EDIT('TYPE OF STORAGE DEVICE USED',DISK_TYPE) (COL(1),A,
   COL(75),F(10));
END DISP_DATA;

/*****************************************
**  VALIDATE INPUT DATA                 **
*****************************************/

VALID: PROCEDURE (LBOUND,HBOUND,VALUE);
  DCL  (HBOUND,
        LBOUND,
        VALUE )     FIXED DECIMAL (11,2);
  IF (VALUE < LBOUND) | (VALUE > HBOUND) THEN
    DO;
     PUT FILE (SYSPRINT) LIST('ERRROR: '||
        'INPUT VALUES NOT IN PROPER RANGE');
     SIGNAL ERROR;
    END;
 END VALID;

/*****************************************
**  READ DISK PARAMETERS FROM FILE     **
*****************************************/

DISK_PARM: PROCEDURE;
 DCL  TEMP FIXED DEC(2,0);
 GET FILE (INFILE) EDIT (TEMP) (COL(1),F(1,0));
 DO WHILE(DISK_TYPE
  GET FILE(INFILE) EDIT (TEMP) (COL(1),F(1,0));
 END;
 GET FILE(INFILE) EDIT(S,R,GAP,TRANSFR_RTE) (COL(3),F(10,2),
  COL(15),F(8,2),COL(25),F(8,2),COL(35),F(8,2));
 END DISK_PARM;

/*********************************************
**  CALCULATE BTT, BFR, AND T_PRIME        **
*********************************************/

DISK_CALC: PROCEDURE;
 DCL  YA           FIXED DEC(14),
      WASTE       FIXED DECIMAL(9,6);
 BTT = BLOCK_SZ / TRANSFR_RTE;
 TRW = 2 * R;
 YA = (BLOCK_SZ - PTR_SZ) / PTR_SZ;
 WASTE = GAP / YA,
 T_PRIME = 0.5 * TRANSFR_RTE * (PTR_SZ / (PTR_SZ + WASTE));
 END;

/*****************************************
```

```
**  MULTILIST FILE CALCULATIONS        **
*****************************************/

MULT_FILE: PROCEDURE(SDS,SDB,TFDS,TFDB,TXDS,TXDB,TYDS,TYDB);
DCL YEFFS            FIXED DEC(9,2),
    YEFFB            FIXED DEC(9,2),
    X                FIXED DEC(3),
    FILE_SZS         FIXED DEC(15,2),
    FILE_SZB         FIXED DEC(15,2),
    (SDS,
    SDB)             FIXED DEC(15,2),
    TFDS             FIXED DEC(10,2),
    TFDB             FIXED DEC(10,2),
    (TFS, TFB,
    TIS, TIB,
    TDS, TDB,
    TDRS, TDRB,
    TUS, TUB,
    TUNS, TUNB,
    TXS, TXB,
    TXDS, TXDB,
    TYDS, TYDB,
    TYS, TYB)            FIXED DEC(15,2);

 CALL MULT_SIZE(FILE_SZS,FILE_SZB,YEFFS,YEFFB,X,SDS,SDB);
 CALL MULT_FETCH(TFS,TFB,TFDS,TFDB,X);
 CALL MULT_INST(TIS,TIB);
 CALL MULT_DLET(TDS,TDB,TDRS,TDRB,TFDS,TFDB);
 CALL MULT_UPDT(TDS,TFDS,TDB,TFDB,TDRS,TDRB,TUS,TUB,
    TUNS,TUNB,TIS,TIB);
 CALL MULT_EX(TXS,TXB,X,TXDS,TXDB);
 CALL MULT_REORG (TYS,TYB,TXS,TXB,TYDS,TYDB);
 CALL MULT_DISP;

 /*****************************************
 **  MULTILIST FILE SIZE CALCULATIONS    **
 *****************************************/

 MULT_SIZE: PROCEDURE(FILE_SZS,FILE_SZB,YEFFS,YEFFB,X,SDS,SDB);
 DCL FILE_SZS         FIXED DEC(15,2),
     FILE_SZB         FIXED DEC(15,2),
     INDX_BLK         FIXED DEC(8),
     TOT_BLK          FIXED DEC(8),
     BFR              FIXED DEC(8),
     X                FIXED DEC(3),
     YEFFS            FIXED DEC(9,2),
     YEFFB            FIXED DEC(9,2),
     REC_SZ           FIXED DEC(9,2),
     REC_IND          FIXED DEC(9),
     SDS              FIXED DEC(15,2),
     SDB              FIXED DEC(15,2);
 IF REC_FRMT = 'V' THEN
  DO;
  REC_SZ = A_PRIME * (ATTR_LEN + VALUE_LEN + PTR_SZ + 2) + NON_DATA;
  BFR = FLOOR ( BLOCK_SZ - 0.5 * REC_SZ) / (REC_SZ + PTR_SZ);
```

```
 END;
 ELSE DO;
  REC_SZ = A_PRIME * (VALUE_LEN + PTR_SZ) + NON_DATA;
  BFR = FLOOR(BLOCK_SZ / REC_SZ);
 END;
 REC_IND = (VALUE_LEN + 2 * PTR_SZ);
 YEFFS = FLOOR(BLOCK_SZ / REC_IND);
 YEFFB = FLOOR ( DENS * BLOCK_SZ / REC_IND);
 SDS = CEIL(VAL_TOT / YEFFS) * BLOCK_SZ * A;
 FILE_SZS = SDS + DEC(CEIL(N / BFR),10,0) * BLOCK_SZ;
 INDX_BLK = CEIL(VAL_TOT /  YEFFB);
 TOT_BLK = INDX_BLK;
 X = 1;
 DO WHILE(INDX_BLK
   INDX_BLK = CEIL(INDX_BLK / YEFFB);
   TOT_BLK = TOT_BLK + INDX_BLK;
   X = X + 1;
 END;
 SDB = A * BLOCK_SZ * TOT_BLK;
 FILE_SZB = SDB + DEC(CEIL(N / BFR),10,0) * BLOCK_SZ;
 END MULT_SIZE;

 /*********************************************
 **  MULTILIST FETCH TIME CALCULATIONS     **
 *********************************************/

MULT_FETCH: PROCEDURE(TFS,TFB,TFDS,TFDB,X);
 DCL TFS          FIXED DEC(15,2),
     TFB          FIXED DEC(15,2),
     X            FIXED DEC(3),
    (TFNS,
    TFNB,
    TFGS,
    TFGB)         FIXED DEC(15,2),
    (TFDS,
     TFDB)        FIXED DEC(10,2);
TFDS = 0.5 * CEIL(VAL_TOT / YEFFS) * (S + R + BTT);
TFDB = X * (S + R+ BTT);
TFNS = N_N * TFDS + LIST_LEN * (S + R+ BTT);
TFNB = N_N * TFDB + LIST_LEN * (S + R+ BTT);
TFGS = N * TFDS + N * (S + R+ BTT);
TFGB = N * TFDB + N * (S + R+ BTT);
TFS = N_ONE * TFNS + N_TWO * TFGS;
TFB = N_ONE * TFNB + N_TWO * TFGB;
END MULT_FETCH;

/*********************************************
**  MULTILIST INSERT TIME CALCULATIONS     **
*********************************************/

MULT_INST: PROCEDURE(TIS,TIB);
 DCL TIOS    FIXED DEC(14,2),
     TIOB    FIXED DEC(14,2),
     TINS    FIXED DEC(14,2),
     TINB    FIXED DEC(14,2),
```

```
     TIS    FIXED DEC(15,2),
     TIB    FIXED DEC(15,2);
 TIOS = (A_PRIME - A_NEW)* (TFDS+ TRW);
 TIOB = (A_PRIME - A_NEW) * (TFDB + TRW);
 TINS = A_NEW * (S + R+ BTT + TRW);
 TINB = A_NEW * (TFDB + TRW + DEC((2 / YEFFS),10,2) *
 (S + R+ BTT + 2 * TRW));
 TIS = TINS + TIOS + (S +R + BTT+ TRW);
 TIB = TINB + TIOB + (S + R+ BTT+ TRW);
END MULT_INST;

/*********************************************
**  MULTILIST DELETE TIME CALCULATIONS     **
*********************************************/

MULT_DLET: PROCEDURE(TDS,TDB,TDRS,TDRB,TFDS,TFDB);
  DCL  (TDS,
        TDB,
        TDRS,
        TDRB)       FIXED DEC(15,2),
       (TFDS,
        TFDB)      FIXED DEC(10,2);
 TDS = TFDS + (S + R+ BTT) + TRW +
    DEC((1 / LIST_LEN),6,2) * TRW;
 TDB = TFDB + (S + R+ BTT) + TRW +
    DEC((1 / LIST_LEN),6,2) * TRW;
 TDRS = TFDS + TRW + S + R + BTT;
 TDRB = TFDB + TRW + S+ R + BTT;
END MULT_DLET;

/*********************************************
**  MULTILIST UPDATE TIME CALCULATIONS     **
*********************************************/

MULT_UPDT: PROCEDURE(TDS,TFDS,TDB,TFDB,TDRS,TDRB,TUS,TUB,
    TUNS,TUNB,TIS,TIB);
  DCL  (TDS,    TDB,
        TDRS,   TDRB,
        TIS,    TIB,
        TUS,    TUB,
        TUNS,   TUNB)     FIXED DEC(15,2),
        (TFDS,  TFDB)     FIXED DEC(10,2);
 TUS = A_UPD * ( 2 * TFDS + 2 * TRW + DEC((LIST_LEN / 2),6,2))
 + S + R + BTT;
 TUB = A_UPD * ( 2 * TFDB + 2 * TRW + DEC((LIST_LEN / 2),6,2))
 + S + R + BTT;
 TUNS = TDRS + TIS;
 TUNB = TDRB + TIB;
END MULT_UPDT;

/*********************************************
**  MULTILIST EXHAUSTIVE READ CALCULATIONS **
*********************************************/

MULT_EX: PROCEDURE(TXS,TXB,X,TXDS,TXDB);
```

```
DCL  (TXS,
      TXB)      FIXED DEC(15,2),
      X         FIXED DEC(3),
      TEMPX     FIXED DEC(3),
      TXDS      FIXED DEC(15,2),
      TXDB      FIXED DEC(15,2);
  TXDB = 0;
  TXDS = CEIL(N /YEFFS) * (S + R+ BTT);
  TXS = TXDS + N * (S+ R+ BTT);
  TEMPX = X - 1;
  DO WHILE (TEMPX >= 1);
    TXDB = YEFFB ** TEMPX + TXDB;
    TEMPX = TEMPX - 1;
   END;
  TXB = TXDB + N * (S+ R+ BTT);
END MULT_EX;

/*****************************************************
**  MULTILIST REORGANIZATION TIME CALCULATIONS   **
*****************************************************/

MULT_REORG: PROCEDURE (TYS,TYB,TXS,TXB,TYDS,TYDB);
 DCL  (TYS,
       TYB)         FIXED DEC (15,2),
       (TXS,
        TXB)        FIXED DEC(15,2),
       (TYDS,
        TYDB)       FIXED DEC(15,2);

  TYDS = A * (CEIL(VAL_TOT / YEFFS) * (S + R+ BTT + TRW));
  TYDB = A * (X * (S + R+ BTT) + TRW + DEC((2 / YEFFS),9,2) *
          (S + R+ BTT+ TRW));
  TYS =  TYDS + TXS +
         N * (S + R+ BTT) + N * A_PRIME * (TFDS + DEC((LIST_LEN / 2),
         9,2) * (S + R+ BTT) + TRW);
  TYB =  TYDB + TXB +
         N * (S + R+ BTT) + N * A_PRIME * (TFDB + DEC((LIST_LEN /2),
         9,2) * (S + R+ BTT) + TRW);
END MULT_REORG;

/*********************************************
**  MULTILIST RESULTS DISPLAY              **
*********************************************/

MULT_DISP: PROCEDURE;
RFMT: FORMAT(COL(10),A,COL(55),F(15,2),X(1),A);
PUT PAGE FILE(SYSPRINT);
PUT SKIP(2) EDIT ('MULTILIST FILE PERFORMANCE PARAMETERS ' ||
  '(SEQUENTIAL DIRECTORY)')  (COL(20),A);
PUT SKIP(0) EDIT ('_____________________________________ ' ||
  '_____________________')  (COL(20),A);
PUT SKIP(2) EDIT ('FILE SIZE: ',FILE_SZS,'BYTES') (R(RFMT));
PUT SKIP(2) EDIT ('DIRECTORY SIZE: ', SDS,'BYTES') (R(RFMT));
PUT SKIP(2) EDIT ('TF: ',TFS,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('TI: ',TIS,'MS') (R(RFMT));
```

```
PUT SKIP(2) EDIT ('TD: ',TDS,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('TDR: ',TDRS,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('TU: ',TUS,'MS') (R(RFMT));
PUT SKIP(2) EDIT('TU(WITH NEW ATTRIBUTES): ',TUNS,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('TX:',TXS,'MS') (R(RFMT));
PUT SKIP(2) EDIT('TY:',TYS,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('MULTILIST FILE PERFORMANCE PARAMETERS ' ||
  '(B-TREE DIRECTORY)')  (COL(20),A);
PUT SKIP(0) EDIT ('_____________________________________ ' ||
  '__________________')  (COL(20),A);
PUT SKIP(2) EDIT ('FILE SIZE: ',FILE_SZB,'BYTES') (R(RFMT));
PUT SKIP(2) EDIT ('DIRECTORY SIZE: ', SDB,'BYTES') (R(RFMT));
PUT SKIP(2) EDIT ('TF: ',TFB,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('TI: ',TIB,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('TD: ',TDB,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('TDR: ',TDRB,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('TU: ',TUB,'MS') (R(RFMT));
PUT SKIP(2) EDIT('TU(WITH NEW ATTRIBUTES): ',TUNB,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('TX:',TXB,'MS') (R(RFMT));
PUT SKIP(2) EDIT('TY:',TYB,'MS') (R(RFMT));
END MULT_DISP;
END MULT_FILE;

/*********************************************
**  INVERTED FILE CALCULATIONS              **
*********************************************/

INVT_FILE: PROCEDURE(SDS,SDB,TFDS,TFDB,TXDS,TXDB,TYDS,TYDB);
DCL (TFA,
     TIA)            FIXED DEC(10,2),
     YA              FIXED DEC(9,2),
     X               FIXED DEC(3),
     FILE_SZS        FIXED DEC(15,2),
     FILE_SZB        FIXED DEC(15,2),
     (SDS, SDB)      FIXED DEC(15,2),
     (TFDS,TFDB)     FIXED DEC(10,2),
     (TFS, TFB,
     TIS,  TIB,
     TDS,  TDB,
     TDRS, TDRB,
     TUS,  TUB,
     TUNS, TUNB,
     TXS,  TXB,
     TXDS, TXDB,
     TYDS, TYDB,
     TYS,  TYB)            FIXED DEC(15,2);

YA = FLOOR((BLOCK_SZ - PTR_SZ) / PTR_SZ);
TFA = 0.5 * LIST_LEN * PTR_SZ / T_PRIME;
TIA = TFA + 0.5 * (CEIL( LIST_LEN / YA) * ( BTT + TRW));
CALL FILE_SZ(FILE_SZS,FILE_SZB,SDS,SDB);
CALL INVT_FETCH(TFS,TFB,TFDS,TFDB);
CALL INVT_INST(TIS,TIB);
CALL INVT_DLET(TDS,TDB,TDRS,TDRB,TFDS,TFDB);
CALL INVT_UPDT(TUS,TUB,TUNS,TUNB,TDS,TDB);
```

```
CALL INVT_EX(TXDS,TXDB,TXS,TXB);
CALL INVT_REORG(TYS,TYB,TYDS,TYDB);
CALL INVT_DISP;

 /*****************************************
 **  INVERTED FILE SIZE CALCULATIONS      **
 *****************************************/

FILE_SZ: PROCEDURE(FILE_SZS,FILE_SZB,SDS,SDB);
 DCL FILE_SZS         FIXED DEC(15,2),
     FILE_SZB         FIXED DEC(15,2),
     REC_SZ           FIXED DEC(9,2),
     BFR              FIXED DEC(10),
     SDS              FIXED DEC(15,2),
     SDB              FIXED DEC(15,2),
     SA               FIXED DEC(15,2);
 IF (REC_FRMT = 'V') THEN
 DO;
  REC_SZ = A_PRIME * (ATTR_LEN + VALUE_LEN + 2) + NON_DATA;
  BFR = FLOOR((BLOCK_SZ - 0.5 * REC_SZ) / ( REC_SZ + PTR_SZ));
 END;
 ELSE  DO;
  REC_SZ = A_PRIME * (VALUE_LEN) + NON_DATA;
  BFR = FLOOR(BLOCK_SZ / REC_SZ);
 END;
 SA = VAL_TOT * A * BLOCK_SZ * CEIL( LIST_LEN / YA);
 FILE_SZS = SDS + SA + DEC(CEIL(N / BFR),10,0) * BLOCK_SZ;
 FILE_SZB = SDB + SA + DEC(CEIL(N / BFR),10,0) * BLOCK_SZ;
END FILE_SZ;

 /*****************************************
 **  INVERTED FETCH TIME CALCULATIONS     **
 *****************************************/

INVT_FETCH: PROCEDURE (TFS,TFB,TFDS,TFDB);
  DCL (TFDS,
       TFDB)       FIXED DEC(10,2),
       (TFGS,
       TFGB,
       TFNS,
       TFNB,
       TFS,
       TFB)        FIXED DEC(15,2);
  TFNS = N_T * (TFDS + 2 * TFA) + DEC(( F_RAT / 100),9,2) *
  LIST_LEN * (S + R+ BTT);
  TFNB = N_T * (TFDB + 2 * TFA) +  DEC((F_RAT / 100),9,2) *
  LIST_LEN * (S+ R+ BTT) ;
  TFGS = TXDS + N_T * (TFDS + 2 * TFA) +  DEC((Q_RAT / 100),9,2)
  * N * (S + R+ BTT);
  TFGB = TXDB + N_T * (TFDB + 2 * TFA) + DEC((Q_RAT / 100),9,2)
  * N * (S + R+ BTT);
  TFS = N_ONE * TFNS + N_TWO * TFGS;
  TFB = N_ONE * TFNB + N_TWO * TFGB;
END INVT_FETCH;
```

```
/*********************************************
**  INVERTED INSERT TIME CALCULATIONS      **
*********************************************/

INVT_INST: PROCEDURE(TIS,TIB);
  DCL (TIS,
       TIB,
       TIDS,
       TIDB)           FIXED DEC(15,2);
  TIDS =  (A_PRIME - A_NEW) * (TFDS + TRW) + A_NEW * (S + R+ BTT);
  TIDB =  (A_PRIME - A_NEW) * (TFDB + TRW) + A_NEW * (S + R+ BTT);
  TIS = TIDS + (A_PRIME - A_NEW) * TIA + (A_NEW + 1) * (S + R+ BTT);
  TIB = TIDB + (A_PRIME - A_NEW) * TIA + (A_NEW + 1) * (S + R+ BTT);
END INVT_INST;

/*********************************************
**  INVERTED DELETE TIME CALCULATIONS      **
*********************************************/

INVT_DLET: PROCEDURE(TDS,TDB,TDRS,TDRB,TFDS,TFDB);
  DCL  (TDS,
        TDB,
        TDRS,
        TDRB)       FIXED DEC(15,2),
       (TFDS,
        TFDB)      FIXED DEC(10,2);
 TDS = 2 * TFDS + TFA + 2 * TRW + DEC((1 / LIST_LEN),6,2) * TRW +
   S + R+ BTT;
 TDB = 2 * TFDB + TFA +2 * TRW + DEC((1 / LIST_LEN),6,2) * TRW + S +
   R + BTT;
 TDRS = TFDS + TRW;
 TDRB = TFDB + TRW;
END INVT_DLET;

/*********************************************
**  INVERTED UPDATE TIME CALCULATIONS      **
*********************************************/

INVT_UPDT: PROCEDURE(TUS,TUB,TUNS,TUNB,TDS,TDB);
  DCL  (TUS,
        TUB,
        TUNS,
        TUNB,
        TDS,
        TDB)        FIXED DEC(15,2);
  TUS = A_UPD * (TDS + TIA) + (S + R + BTT + TRW);
  TUB = A_UPD * (TDB + TIA) + (S + R+ BTT + TRW);
  TUNS = TDRS + TIS;
  TUNB = TDRB + TIB;
END INVT_UPDT;

/*********************************************
**  INVERTED EXHAUSTIVE READ CALCULATIONS **
*********************************************/
```

```
INVT_EX: PROCEDURE(TXDS,TXDB,TXS,TXB);
DCL  (TXS,
      TXB)      FIXED DEC(15,2),
      TXDS      FIXED DEC(15,2),
      TXDB      FIXED DEC(15,2);
  TXS = TXDS + N * (S+ R+ BTT);
  TXB = TXDB + N * (S+ R+ BTT);
END INVT_EX;

/*************************************************
**  INVERTED REORGANIZATION TIME CALCULATIONS  **
*************************************************/

INVT_REORG: PROCEDURE (TYS,TYB,TYDS,TYDB);
  DCL  (TYS,
        TYB,
        TYDS,
        TYDB)       FIXED DEC(15,2),
        TYA         FIXED DEC(10,2);
  TYA = 2 * VAL_TOT * A * DEC((LIST_LEN * PTR_SZ / T_PRIME),10,2);
  TYS = TYDS + TYA + TXS + N * (S + R+ BTT) + N * A_PRIME * (TDS
         + TIA);
  TYB= TYDB + TYA + TXB + N * (S + R+ BTT) + N * A_PRIME * (TDB
        + TIA);
END INVT_REORG;

/*********************************************
**  INVERTED RESULTS DISPLAY                **
*********************************************/

INVT_DISP: PROCEDURE;
RFMT: FORMAT(COL(10),A,COL(55),F(15,2),X(1),A);
PUT PAGE FILE(SYSPRINT);
PUT SKIP(2) EDIT ('INVERTED FILE PERFORMANCE PARAMETERS ' ||
  '(SEQUENTIAL DIRECTORY)')  (COL(20),A);
PUT SKIP(0) EDIT ('_____________________________________ ' ||
  '_____________________')  (COL(20),A);
PUT SKIP(2) EDIT ('FILE SIZE: ',FILE_SZS,'BYTES') (R(RFMT));
PUT SKIP(2) EDIT ('DIRECTORY SIZE: ', SDS,'BYTES') (R(RFMT));
PUT SKIP(2) EDIT ('TF: ',TFS,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('TI: ',TIS,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('TD: ',TDS,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('TDR: ',TDRS,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('TU: ',TUS,'MS') (R(RFMT));
PUT SKIP(2) EDIT('TU(WITH NEW ATTRIBUTES): ',TUNS,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('TX:',TXS,'MS') (R(RFMT));
PUT SKIP(2) EDIT('TY:',TYS,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('INVERTED FILE PERFORMANCE PARAMETERS ' ||
  '(B-TREE DIRECTORY)')  (COL(20),A);
PUT SKIP(0) EDIT ('_____________________________________ ' ||
  '___________________')  (COL(20),A);
PUT SKIP(2) EDIT ('FILE SIZE: ',FILE_SZB,'BYTES') (R(RFMT));
PUT SKIP(2) EDIT ('DIRECTORY SIZE: ', SDB,'BYTES') (R(RFMT));
PUT SKIP(2) EDIT ('TF: ',TFB,'MS') (R(RFMT));
PUT SKIP(2) EDIT ('TI: ',TIB,'MS') (R(RFMT));
```

```
 PUT SKIP(2) EDIT ('TD: ',TDB,'MS') (R(RFMT));
 PUT SKIP(2) EDIT ('TDR: ',TDRB,'MS') (R(RFMT));
 PUT SKIP(2) EDIT ('TU: ',TUB,'MS') (R(RFMT));
 PUT SKIP(2) EDIT('TU(WITH NEW ATTRIBUTES): ',TUNB,'MS') (R(RFMT));
 PUT SKIP(2) EDIT ('TX:',TXB,'MS') (R(RFMT));
 PUT SKIP(2) EDIT('TY:',TYB,'MS') (R(RFMT));
 END INVT_DISP;
 END INVT_FILE;

 END CALC;
//LKED.SYSPRINT DD DUMMY
//GO.SYSPRINT DD SYSOUT=W
//GO.SYSIN DD *
1024              BLOCK SIZE (BYTES) (1..99999)
3000              # OF RECORDS IN THE MAIN FILE (1..9999999)
0                  AVG # OF NON NEGATED TERMS IN TYPE 1 SUBQUERY(NN)
0                 AVG # OF TYPE 1 SUBQUERIES
1                 AVG # OF TERMS IN A SUBQUERY(NT)
1                 AVG # OF TYPE 2 SUBQUERIES
20                AVG # OF VALUES PER ATTRIBUTE NAME
100               AVG # OF RECORDS PER ATTRIBUTE NAME-VALUE PAIR
20                AVG # OF RECORDS IN THE SHORTEST LIST
100               AVG RATIO OF QUERY RESPONSES TO SHORTEST LIST
100               AVG RATIO OF QUERY RESPONSES TO TOTAL RECORDS
6                 POINTER SIZE(1..9)
V                 RECORD FORMAT ( F -- FIXED; V -- VARYING)
10                AVG # OF ATTRIBTES PER RECORD  (1 ..99)
20                TOTAL # OF ATTRIBUTE NAME-VALUE PAIR PER RECORD (1 ..99
4                 ATTRIBUTE_NAME LENGTH  (BYTES) (1..99)
10                ATTRIBUTE VALUE LENGTH (BYTES) (1..999)
50                AVG NON INDEXED DATA PER RECORD  (1..200)
3                 AVG # OF UPDATE ATTRIBUTES         (1..99999)
2                 AVG # OF NEW ATTRIBUTES           (1..99999)
1                 TYPE OF DEVICE (1 .. 2)
//GO.INFILE   DD *
1  16.00      8.30      524.00    3000.00    IBM3380
2  60.00      12.50     200.00    312.00     IBM2319
/*
```

APPENDIX C

INPUT PARAMETERS AND RESULTS

Table I. INPUT VALUE SET 1

INPUT PARAMETERS	
BLOCK SIZE (BYTES)	512
TOTAL NUMBER OF RECORDS IN THE FILE (N)	30000
NUMBER OF NONNEGATED TERMS IN TYPE 1 QUERY	2
NUMBER OF TYPE 1 SUBQUERIES (N1)	1
NUMBER OF TERMS IN TYPE 1 OR TYPE 2 SUBQUERY (N1)	2
NUMBER OF TYPE 2 SUBQUERIES (N2)	0
AVERAGE NUMBER OF ENTRIES PER INDEX (NA)	1000
AVERAGE LIST LENGTH (L)	1000
AVERAGE NUMBER OF RECORDS IN THE SHORTEST LIST (LS)	20
THE RATIO OF QUERY RESPONSES TO THE SHORTEST LIST (F)	30%
THE RATIO OF QUERY RESPONSES TO TOTAL RECORDS IN THE FILE (Q)	30%
POINTER SIZE (P)	6 BYTES
RECORD FORMAT	VARYING
AVERAGE NUMBER OF ATTRIBUTES PER RECORD (A')	10
TOTAL NUMBER OF ATTRIBUTES IN THE FILE (A)	20
AVERAGE ATTRIBUTE NAME LENGTH	4
AVERAGE ATTRIBUTE VALUE LENGTH (V)	10
AVERAGE NON INDEXED DATA PER RECORD (N)	50 BYTES
AVERAGE NUMBER OF UPDATE ATTRIBUTES PER UPDATE OPERATION (AU)	3
AVERAGE NUMBER OF NEW ATTRIBUTES PER INSERT OPERATION (AN)	2
TYPE OF STORAGE DEVICE USED	1

Table II. PERFORMANCE VALUES FOR INPUT SET 1

MULTILIST FILE PERFORMANCE PARAMETERS (SEQUENTIAL DIRECTORY)	
FILE SIZE	15810560.00 BYTES
DIRECTORY SIZE	450560.00 BYTES
TIME TO FETCH	25546.68 MS
TIME TO INSERT	4562.73 MS
TIME TO DELETE AN ATTRIBUTE	579.41 MS
TIME TO DELETE A RECORD	579.41 MS
TIME TO UPDATE A RECORD (WITH EXISTING ATTRIBUTE)	4854.11 MS
TIME TO UPDATE A RECORD (WITH NEW ATTRIBUTE)	5142.14 MS
TIME TO EXHAUSTIVELY READ THE FILE	766033.35 MS
TIME TO REORGANIZE THE FILE	3838518274.95 MS

MULTILIST FILE PERFORMANCE PARAMETERS (B-TREE DIRECTORY)	
FILE SIZE	16056320.00 BYTES
DIRECTORY SIZE	696320.00 BYTES
TIME TO FETCH	24616.82 MS
TIME TO INSERT	950.39 MS
TIME TO DELETE AN ATTRIBUTE	114.48 MS
TIME TO DELETE A RECORD	114.48 MS
TIME TO UPDATE A RECORD (WITH EXISTING ATTRIBUTE)	2064.53 MS
TIME TO UPDATE A RECORD (WITH NEW ATTRIBUTE)	1064.87 MS
TIME TO EXHAUSTIVELY READ THE FILE	734372.00 MS
TIME TO REORGANIZE THE FILE	3698973370.76 MS

Table III. PERFORMANCE VALUES FOR INPUT SET 2

INPUT PARAMETERS

BLOCK SIZE (BYTES)	1024
TOTAL NUMBER OF RECORDS IN THE FILE (N)	3000
NUMBER OF NONNEGATED TERMS IN TYPE 1 QUERY	1
NUMBER OF TYPE 1 SUBQUERIES (N1)	1
NUMBER OF TERMS IN TYPE 1 OR TYPE 2 SUBQUERY (N1)	1
NUMBER OF TYPE 2 SUBQUERIES (N2)	0
AVERAGE NUMBER OF ENTRIES PER INDEX (NA)	20
AVERAGE LIST LENGTH (L)	100
AVERAGE NUMBER OF RECORDS IN THE SHORTEST LIST (LS)	20
THE RATIO OF QUERY RESPONSES TO THE SHORTEST LIST (F)	100%
THE RATIO OF QUERY RESPONSES TO TOTAL RECORDS IN THE FILE (Q)	100%
POINTER SIZE (P)	4 BYTES
RECORD FORMAT	VARYING
AVERAGE NUMBER OF ATTRIBUTES PER RECORD (A')	10
TOTAL NUMBER OF ATTRIBUTES IN THE FILE (A)	20
AVERAGE ATTRIBUTE NAME LENGTH	4
AVERAGE ATTRIBUTE VALUE LENGTH (V)	6
AVERAGE NON INDEXED DATA PER RECORD (N)	50 BYTES
AVERAGE NUMBER OF UPDATE ATTRIBUTES PER UPDATE OPERATION (AU)	3
AVERAGE NUMBER OF NEW ATTRIBUTES PER INSERT OPERATION (AN)	2
TYPE OF STORAGE DEVICE USED	1

MULTILIST FILE PERFORMANCE PARAMETERS (B-TREE DIRECTORY)

FILE SIZE	788480.00 BYTES
DIRECTORY SIZE	20480.00 BYTES
TIME TO FETCH	2488.64 MS
TIME TO INSERT	455.95 MS
TIME TO DELETE AN ATTRIBUTE	66.04 MS
TIME TO DELETE A RECORD	65.88 MS
TIME TO UPDATE A RECORD (WITH EXISTING ATTRIBUTE)	422.08 MS
TIME TO UPDATE A RECORD (WITH NEW ATTRIBUTE)	521.83 MS
TIME TO EXHAUSTIVELY READ THE FILE	73920.00 MS
TIME TO REORGANIZE THE FILE	38345881.29 MS

INVERTED FILE PERFORMANCE PARAMETERS (B-TREE DIRECTORY)

FILE SIZE	655360.00 BYTES
DIRECTORY SIZE	20480.00 BYTES
TIME TO FETCH	2489.04 MS
TIME TO INSERT	522.48 MS
TIME TO DELETE AN ATTRIBUTE	107.48 MS
TIME TO DELETE A RECORD	41.24 MS
TIME TO UPDATE A RECORD (WITH EXISTING ATTRIBUTE)	389.69 MS
TIME TO UPDATE A RECORD (WITH NEW ATTRIBUTE)	563.72 MS
TIME TO EXHAUSTIVELY READ THE FILE	73920.00 MS
TIME TO REORGANIZE THE FILE	3633501.29 MS

Table IV. PERFORMANCE VALUES FOR INPUT SET 3

INPUT PARAMETERS

BLOCK SIZE (BYTES)	1024
TOTAL NUMBER OF RECORDS IN THE FILE (N)	3000
NUMBER OF NONNEGATED TERMS IN TYPE 1 QUERY	0
NUMBER OF TYPE 1 SUBQUERIES (N1)	0
NUMBER OF TERMS IN TYPE 1 OR TYPE 2 SUBQUERY (N1)	1
NUMBER OF TYPE 2 SUBQUERIES (N2)	1
AVERAGE NUMBER OF ENTRIES PER INDEX (NA)	20
AVERAGE LIST LENGTH (L)	100
AVERAGE NUMBER OF RECORDS IN THE SHORTEST LIST (LS)	20
THE RATIO OF QUERY RESPONSES TO THE SHORTEST LIST (F)	100%
THE RATIO OF QUERY RESPONSES TO TOTAL RECORDS IN THE FILE (Q)	100%
POINTER SIZE (P)	4 BYTES
RECORD FORMAT	VARYING
AVERAGE NUMBER OF ATTRIBUTES PER RECORD (A')	10
TOTAL NUMBER OF ATTRIBUTES IN THE FILE (A)	20
AVERAGE ATTRIBUTE NAME LENGTH	4
AVERAGE ATTRIBUTE VALUE LENGTH (V)	6
AVERAGE NON INDEXED DATA PER RECORD (N)	50 BYTES
AVERAGE NUMBER OF UPDATE ATTRIBUTES PER UPDATE OPERATION (AU)	3
AVERAGE NUMBER OF NEW ATTRIBUTES PER INSERT OPERATION (AN)	2
TYPE OF STORAGE DEVICE USED	1

MULTILIST FILE PERFORMANCE PARAMETERS (B-TREE DIRECTORY)

FILE SIZE	788480.00 BYTES
DIRECTORY SIZE	20480.00 BYTES
TIME TO FETCH	147840.00 MS
TIME TO INSERT	455.95 MS
TIME TO DELETE AN ATTRIBUTE	66.04 MS
TIME TO DELETE A RECORD	65.88 MS
TIME TO UPDATE A RECORD (WITH EXISTING ATTRIBUTE)	422.08 MS
TIME TO UPDATE A RECORD (WITH NEW ATTRIBUTE)	521.83 MS
TIME TO EXHAUSTIVELY READ THE FILE	73920.00 MS
TIME TO REORGANIZE THE FILE	38345881.29 MS

INVERTED FILE PERFORMANCE PARAMETERS (B-TREE DIRECTORY)

FILE SIZE	655360.00 BYTES
DIRECTORY SIZE	20480.00 BYTES
TIME TO FETCH	73945.04 MS
TIME TO INSERT	522.48 MS
TIME TO DELETE AN ATTRIBUTE	107.48 MS
TIME TO DELETE A RECORD	41.24 MS
TIME TO UPDATE A RECORD (WITH EXISTING ATTRIBUTE)	389.69 MS
TIME TO UPDATE A RECORD (WITH NEW ATTRIBUTE)	563.72 MS
TIME TO EXHAUSTIVELY READ THE FILE	73920.00 MS
TIME TO REORGANIZE THE FILE	3633501.29 MS

Table V. PERFORMANCE VALUES FOR INPUT SET 4

INPUT PARAMETERS

BLOCK SIZE (BYTES)	512
TOTAL NUMBER OF RECORDS IN THE FILE (N)	5000
NUMBER OF NONNEGATED TERMS IN TYPE 1 QUERY	1
NUMBER OF TYPE 1 SUBQUERIES (N1)	5
NUMBER OF TERMS IN TYPE 1 OR TYPE 2 SUBQUERY (N1)	1
NUMBER OF TYPE 2 SUBQUERIES (N2)	0
AVERAGE NUMBER OF ENTRIES PER INDEX (NA)	200
AVERAGE LIST LENGTH (L)	1000
AVERAGE NUMBER OF RECORDS IN THE SHORTEST LIST (LS)	1000
THE RATIO OF QUERY RESPONSES TO THE SHORTEST LIST (F)	100%
THE RATIO OF QUERY RESPONSES TO TOTAL RECORDS IN THE FILE (Q)	30%
POINTER SIZE (P)	6 BYTES
RECORD FORMAT	FIXED
AVERAGE NUMBER OF ATTRIBUTES PER RECORD (A')	5
TOTAL NUMBER OF ATTRIBUTES IN THE FILE (A)	5
AVERAGE ATTRIBUTE NAME LENGTH	4
AVERAGE ATTRIBUTE VALUE LENGTH (V)	10
AVERAGE NON INDEXED DATA PER RECORD (N)	50 BYTES
AVERAGE NUMBER OF UPDATE ATTRIBUTES PER UPDATE OPERATION (AU)	3
AVERAGE NUMBER OF NEW ATTRIBUTES PER INSERT OPERATION (AN)	2
TYPE OF STORAGE DEVICE USED	1

MULTILIST FILE PERFORMANCE PARAMETERS (B-TREE DIRECTORY)

FILE SIZE	889344.00 BYTES
DIRECTORY SIZE	35840.00 BYTES
TIME TO FETCH	122594.70 MS
TIME TO INSERT	377.99 MS
TIME TO DELETE AN ATTRIBUTE	90.01 MS
TIME TO DELETE A RECORD	90.01 MS
TIME TO UPDATE A RECORD (WITH EXISTING ATTRIBUTE)	1917.71 MS
TIME TO UPDATE A RECORD (WITH NEW ATTRIBUTE)	468.00 MS
TIME TO EXHAUSTIVELY READ THE FILE	122366.00 MS
TIME TO REORGANIZE THE FILE	307758560.12 MS

INVERTED FILE PERFORMANCE PARAMETERS (B-TREE DIRECTORY)

FILE SIZE	578560.00 BYTES
DIRECTORY SIZE	35840.00 BYTES
TIME TO FETCH	122634.70 MS
TIME TO INSERT	632.83 MS
TIME TO DELETE AN ATTRIBUTE	159.55 MS
TIME TO DELETE A RECORD	65.54 MS
TIME TO UPDATE A RECORD (WITH EXISTING ATTRIBUTE)	833.58 MS
TIME TO UPDATE A RECORD (WITH NEW ATTRIBUTE)	698.37 MS
TIME TO EXHAUSTIVELY READ THE FILE	122366.00 MS
TIME TO REORGANIZE THE FILE	6865610.12 MS