A SIMULATION MODEL TO AID IN THE DESIGN AND TUNING OF HIERARCHICAL DATABASES

Wayne A. Hall

Honeywell Information Systems, Inc.

ABSTRACT

Bell Laboratories has developed a data management system for hierarchical databases. The system is designed in such a way that its performance can be significantly affected by a number of parameters which can be manipulated to "tune" an installation for minimum response time and lowest cost. In order to study the relationships among these parameters, a simulation model was developed which has enabled the preparation of published guidelines to be used by database designers and administrators. In addition, the model itself is available for direct use on an interactive basis as a standard part of a utility package provided as a component of the system. The work on which this paper is based was done while the author was with Bell Telephone Laboratories in Holmdel, New Jersey.

1. Introduction

Bell Telephone Laboratories has developed a data management system for hierarchical databases which is expected to receive widespread usage. Since the data base structure and usage patterns will probably be unique for each application, a considerable amount of effort must be directed toward "tuning" each system to handle its job most economically. The tuning process begins when the database is designed and continues throughout its life. Certain parameters have been built into the system to facilitate the tuning process; the intelligent use of these parameters is the responsibility of the database designer initially and of the technical administrator after the system is in use.

Many factors enter into the description of the way in which a given application uses the system. The relative importance of these factors and their relationship with the tuning parameters is a rather complex subject. Thus, in order to study the problem thoroughly, it was decided to develop a discrete-event simulation model which would serve three purposes:

1. To identify the most significant factors describing the usage and structure of a particular database.

2. To study the relationships among these factors and the tuning parameters and develop a set of general guidelines for the technical administrator and designer.

3. To provide in itself an interactive tool through which particular cases could be studied directly.

This paper discusses the modeling approach and the general design of the model. Representative results from the experiments performed will be shown graphically in an Appendix. The guidelines for technical administrators and designers are not of general interest to non-users, so they are not included herein.

2. Description of the Database System

The system is designed in such a way that the pointers needed to traverse the hierarchy are maintained physically apart from the data values. In fact, each group (collection of nodes at the same level which are of like "type"), has its own "file" for pointers and another for its data items. Let us introduce the following terminology:

1. data block: The data for each group is logically structured as a matrix, where each column represents a node and each row a particular data field.

2. subblock: one or more complete columns in a data block; the number of columns is called CPSUB (columns per subblock). Within a subblock, data is physically stored rowwise, with all of one subblock being stored contiguously.

3. family: Each group consists of one or more families, where all nodes having the same parent comprise a family.

4. reserved node: It is possible, for each family, to reserve pointer space and columns in the data block to accomodate nodes added in the future. The number of nodes to be reserved for each family in a group is called the bubble factor.

5. bubble-out: If a new node is to be added to a family which has no remaining reserved space, the pointer set for the entire family must be copied out to the end of the pointer file so that pointers associated with the new node may be appended to it. This process is called a "bubble-out", and it results in the reservation of space for (bubble factor - 1) subsequent additions. The space which contains the original pointer set becomes wasted "gas", and serves only to point to the family pointer set's new location in the pointer file.

3. Optimizing Performance

"Optimization" can be thought of as the minimization of two components - response time and cost to the user. Since the system has been designed to be "portable" (that is, it can be installed on many time-sharing systems with very little recoding), cost and performance will be heavily influenced by characteristics of each "host" system such as charging algorithms, type of direct-access devices, etc. Hence the cost function to be described below is general in that the unit costs for such items as I/O operations are indicated as coefficients whose values will differ among various installations. Specific values for these coefficients for a given host system can be derived, provided that one has a thorough understanding of the internal operation of the data management system and of the host system.

Let us define the following over some arbitrary period in the active life of a database system application:

RD: number of data-block records read

RM: number of pointer records read

WD: number of data-block records written

WM: number of pointer records written

BD: quantity of core allocated for buffering data records

BM: quantity of core allocated for buffering pointer records

NB: number of "bubble-outs" performed

GM: maximum "gas" (wasted pointer space) accumulated

NC: number of "compresses" (pointer file garbage collections) performed

In describing the configuration and usage patterns of a particular application, three basic classes of parameters are involved; they will hereafter be referred to as "usage" parameters, "data base" parameters and "media" parameters. All parameters are defined at the group level. The mnemonics shown will be used in later sections.

Usage parameters:

1. Average number of requests per user session (AVGRPS)
2. Relative frequencies of 3 types of requests:
   a. Retrievals plus changes to existing data values (PRETRV)
   b. Addition of a node to an existing family (PNEWN)
   c. Creation of a new family (PNEWF)
3. For a retrieval request, on the average (these 3 values will be referred to collectively as "retrieval density"):
   a. Fraction of data words per node (words per column) to be retrieved (PCTV)
   b. Fraction of families to be retrieved from (PCTF)
   c. For each family selected, the average fraction of nodes in the family to be retrieved from (PCTE)

Data base parameters:

1. Words per node in pointer record (WNPS)
2. Words of data per node (words per column) (WPC)
3. Average number of nodes per family (MEANF)
4. Number of families in the group (NFING)

Media parameters (the first 6 comprise the "tuning" parameters):

1. Words per pointer record (MWPR)
2. Words per data record (DWPR)
3. Number of buffers (1 record each) allocated for pointer records (MBEA)
4. Number of buffers allocated for data (DBEA)
5. Columns per subblock (CPSUB)
6. Bubble factor (BUBBLE)
7. "Gas" level for running "COMPRESS" utility (CPLEVL), defined as a number r, $0 < r \leq 1$, such that a pointer record compress is run by the administrator whenever the condition arises that (total "gas")/(active nodes + total "gas") $\geq$ r

Given the foregoing, it is now possible to define a cost/performance function:

$$f(U,D,M) = c1RD + c2RM + c3WD + c4WM + c5BD + c6BM + c7NB + c8GM + c9NC$$

where U is the set of usage parameters
D is the set of database parameters
M is the set of media parameters
c1, c2...,c9 are the unit costs for a given host system

It is important to note that RD, RM,...,NC are actually themselves functions of U, D, and M.

The above function does not appear to contain any mention of CPU time, which represents a substantial part of the total cost of running the database system. The model, as will be shown later, does not attempt to represent CPU activity explicitly. Other studies have attacked this quantity directly. It is possible, however, to include a CPU cost component in certain of the cost coefficients, namely c1, c2, c3, c4, c7, and c9, if it is deemed worthwhile to expend the considerable amount of effort required to determine accurate values for these coefficients. The administrator or designer may well be content to compare various sets of parameter values to observe their effect on RD, RM,...,NC rather than to predict costs explicitly.

The process of optimizing a system application, then, consists of choosing those values of the "media" parameters which, for given values of the "usage" and "database" parameters, will minimize the value of the cost function. Considering the complexity of the interactions involved, the development of a simulation model through which these interactions could be studied seemed to be the best way to aid the administrator and designer in this process.

4. The Model

4.1 Choosing a Language

Before starting the development of the model, the selection of a programming language for its implementation had to be made. Three major requirements seemed paramount:

1. Easy use of independent streams of uniformly distributed random numbers.

2. Parameterization, or the ability to supply data for each experiment as input.

3. Economy, since a large number of experiments (at least 300) were envisioned.

Because there was no need to consider such time-dependent problems as multiple concurrent updates for the purposes of the study, it was not necessary to include the element of time in the model. This eliminated the need to consider the use of a general-purpose simulation language like GPSS with its inherent inefficiencies in favor of a more economical language like FORTRAN or PL/1; FORTRAN was ultimately chosen.

4.2 Restrictions and Assumptions

The "tuning" parameters (CPSUB, BUBBLE, DBEA, MBEA, MWPR, DWPR) may be manipulated independently for each group in the hierarchical structure. Each group normally has its own buffer space for pointer records and data. The data block and pointer set for each group are distinct from those of all other groups. Therefore, a model need only consider a single group at a time, with the commonly made assumption that buffers are not shared among groups. Each user interacting with a database at a given time has his own set of buffers, so input to the model in the form of user requests can be represented as a stream of strictly sequential user sessions. As mentioned previously, details of the host system are not considered; this means, among other things, that paging, I/O buffering other than that done by the database system itself, and all aspects of hardware behavior are ignored. Thus not only can the model be conceptually fairly simple but it can also be generally adaptable to any system application.

Certain assumptions were made in the design of the model; we shall mention only the most important ones here.

1. The number of requests per user session is calculated for each session from an exponential distribution whose mean is supplied as an input parameter.

2. Three types of requests are considered - retrievals, adding a node to an existing family, and creating a new family. The model does not differentiate between simple retrievals and retrievals in which the value of one or more retrieved data fields is modified.

3. Each experiment starts with an existing data base configuration defined by inputs to the model. An average family size is supplied, from which the actual size of each family is determined from a geometric distribution.

4. When simulating the creation of a new family, its size is determined as above, and the insertion of data values for all fields for each new node is simulated.

5. When adding a node to an existing family, the family to be used is selected randomly and all data values are stored.

6. Retrievals involving multiple fields and/or multiple nodes are always taken in the sequence of all selected fields for each selected node in each selected family in ascending order of field index within ascending node index within ascending family index. This corresponds logically to a top-to-bottom ordering for fields in the data block, and a left-to-right ordering for nodes and families as they appeared in the original pointer set at the beginning of the experiment.

4.3 Program Structure

At this writing, several implementations of the model exist, all having identical logic. Currently implemented as batch programs are a standard version accommodating up to 1000 families, 1000 nodes per family, and 1000 data words per node which requires about 74K bytes of IBM 370 memory. A larger version permits up to 5000 of each of the above items and requires about 224K. Both versions permit up to 500 pointer record buffers and 2000 data buffers. The large version of the model is also available through the standard utility package supplied as a part of the database system. In this form it can be easily used interactively by the designer or administrator for the study of specific problems, drawing some of its input parameter values, if desired, from values actually stored in control tables for the real database itself. Execution times are highly data-dependent.

4.4 Some Comments on Level of Detail

The model is not a large program. It consists of a driving module and fifteen subprograms with a total of approximately 600 FORTRAN statements and uses a multiple-stream random number generator to eliminate correlation between stochastic processes which are in reality independent. The input required for each experiment consists of the eighteen items defined in section 3 plus the number of user sessions to be simulated.

When developing a model of this type, one is often faced with the question of determining the level of detail necessary to strike an appropriate balance between accuracy of representation and efficiency of execution for various parts of the system being simulated. This problem proved particularly acute in the representation of the buffering of pointer records and data records.

The buffering algorithm employed by the system for both types of records consists of always putting a particular record in the buffer location computed by taking the remainder when its record sequence number is divided by the total number of buffers allocated. For example, if 100 data buffers were allocated for a particular group, the 279th record in the data block would be read into the 79th buffer if it were not already there. If either the 79th, 179th, 379th,...record were in the buffer and had been updated since the last time it was read, it would be rewritten before bringing in the new record. All updated records present in the buffers at the end of a session are flushed out. A considerable amount of effort was directed toward devising a purely Monte Carlo representation of this process, which would not require keeping track of each individual record. Several schemes for both pointer records and data records were tried, but none proved sufficiently accurate for the needs of the study. Consequently the buffering process has been modeled in a rather explicit manner, which involves keeping track of exactly which records are buffered at any given time, and whether or not they have been updated. This technique, while requiring more memory than a strictly Monte Carlo approach, was considerably easier to implement, much more efficient in terms of CPU time, and somewhat more accurate.

When simulating a retrieval request, the portion of the group's data block actually retrieved is computed using the three components of "retrieval density" defined in section 3. The selection process is performed by a subroutine using a uniform distribution. This routine is called once to select the fields to be retrieved, once to select families, and again, for each chosen family, to select nodes. Since one of the arguments is an index which identifies which of the three types of selections is being made, it is a simple matter to modify the routine to represent other selection rules, such as arranging certain frequently retrieved fields in adjacent rows of the data block to minimize the number of records which actually have to be read.

5. The Model in Use

5.1 Validation

Validation of the model was not a particularly difficult process, since the system being simulated was already operational. Detailed comparisons were made between model output and statistics obtained from a test application of the database system, using a FORTRAN program to supply the same sequence of user requests to the real system as were generated during the corresponding run of the model. The second portion of the validation effort consisted of verifying that the model produced certain relationships already known to exist.

5.2 Experiments Performed

The primary intent of the model was to investigate the sensitivity of system performance and cost to the various usage, database, and media parameters defined previously. Accordingly, extensive experimentation was carried out to learn something about the influence of these parameters on measures such as buffering efficiency, buildup of "gas", etc. The majority of these experiments consisted of holding all but one parameter fixed in one of two "nominal" sets of values, then varying a single parameter over some range to observe its effect. The two "nominal" cases were defined as follows:

Case I:

AVGRPS = 3.3	MEANF = 16
PRETRV = 1.0	NFING = 32
PNEWN = 0	MWPR = 200
PNEWF = 0	DWPR = 200
PCTV = .5	MBEA = 100
PCTF = .5	DBEA = 500
PCTE = .5	CPSUB = 1
WNPS = 1	BUBBLE = 1
WPC = 10	CPLEVL = 1.0

Case II: Same as (I) except:

PRETRV = .25
PNEW = .75
CPLEVL = .20

A complete listing of the experiments performed is beyond the scope of this paper. However, Appendix A shows the output obtained for a particular set of parameters, and Appendix B presents graphical results obtained from three typical experiments. The first figure in Appendix B shows that buffering efficiency (in terms of the number of actual data record reads required for a given stream of user sessions) increases rapidly as the record size becomes larger. B-1 indicates the effect on buffering efficiency when columns per subblock is varied for each of several values of the bubble factor. Finally, B-2 gives an idea of the tradeoff between frequent running of the COMPRESS utility and pointer record reading activity for several values of the bubble factor. The relationship shown in this graph reflects the fact that as "gas" builds up, pointer record I/O efficiency decreases, combined with an increase in pointer record reads associated with the running of COMPRESS itself.

In addition to these experiments, the model has received a "field trial" as a tool for direct use by the data base designer/administator in a large Bell System management information system project. It has provided useful guidance in selecting an effective combination of CPSUB, DWPR, and DBEA.

5.3 Results

The guidelines for technical administrators and designers derived from the experiments with the model have been published and are a part of the standard documentation supplied with the system.

For the sake of completeness, however, a brief synopsis of the major concepts follows.

a. Records should be as large as possible, and the more buffer space available, the better. This is perhaps the most important consideration of all, since the impact of most other parameters diminishes as record sizes and buffer availability grow. Thus the penalty paid for non-optimal values of other parameters becomes of less consequence.

b. It is generally unwise to choose a combination of columns per subblock and data record size which results in a record containing other than an equal number of fields for each node in the record; i.e., the logical shape of the record if it were laid out on the data block should be strictly rectangular.

c. When building the database, if certain fields are expected to be involved in a large portion of retrieval requests, they should be grouped in contigous rows in the data block, crossing as few record boundaries as possible.

d. In installations where a high percentage of the requests are retrievals, CPSUB should be high (equal to or a multiple of the data record size) if retrievals are primarily rowwise (a single field for one or more nodes), and low (between 1 and (DWPR/WPC), for instance) if columnwise retrievals (more than one field per node) predominate.

e. If new nodes are added frequently, then a low value of CPSUB is best for this process. Consequently, if new nodes are added en masse only at certain times, it might be wise to run a utility program which changes CPSUB and restructures the data block accordingly before and after this occurs.

f. If requests are almost entirely retrievals, the bubble factor should be 1.

g. In a mixed retrieval/growth situation, the more severe the limitation of data buffer space, the higher the bubble factor should be (up to a certain point).

h. It is generally true that the value of the bubble factor which yields the most efficient use of pointer record buffers is not the same as the best value from the standpoint of data records. In most cases it is probably more important to optimize data record buffering.

i. For a given value of the COMPRESS level, the number of times the compressing utility should be run decreases much faster than linearly with increasing bubble factor.

6. Conclusion

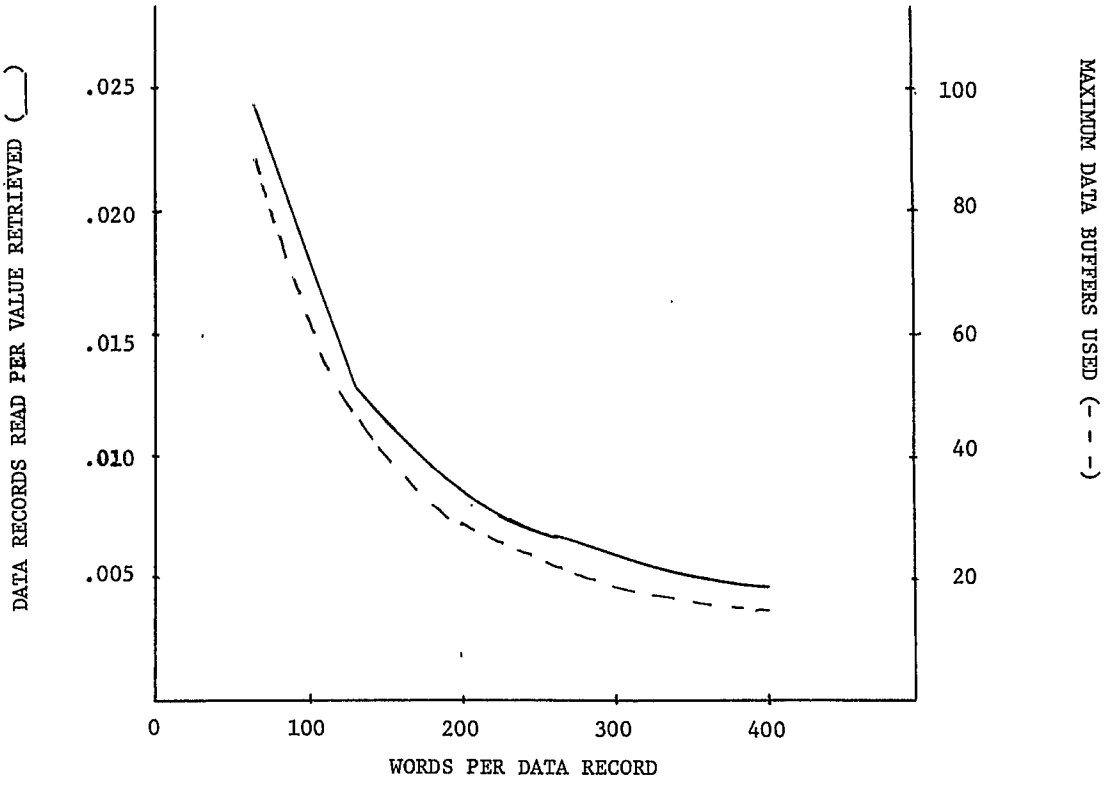
A discrete simulation model has been described which has been used to develop a set of guidelines for the designer and the technical administrator of a hierarchical database system application to aid in the "tuning" of the system for optimal performance and lowest cost. The model is also designed to be used directly as an interactive tool for the study of specific problems. This dual function together with the general adaptability of the model to any system application represents a potentially useful tool for the design and maintenance of present and future databases.

APPENDIX A

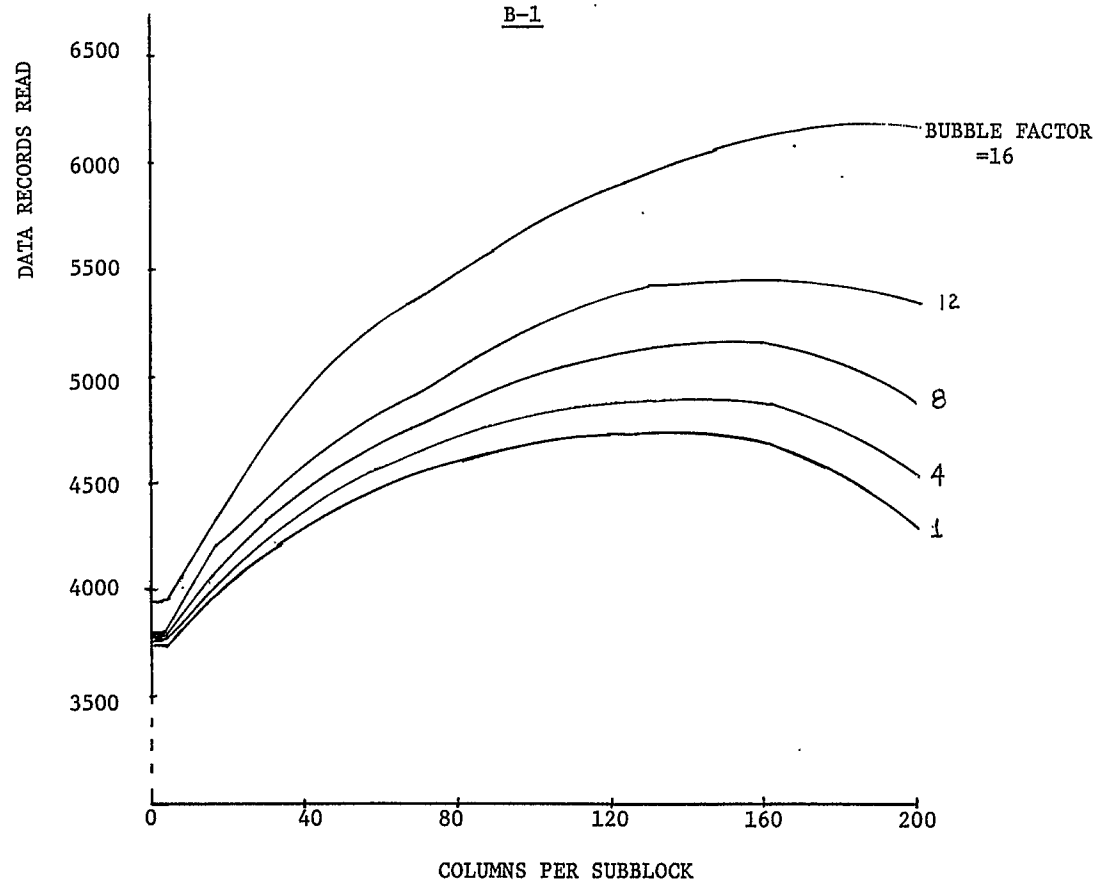
Output From a Typical Experiment

```
                    EXPERIMENT #  1


BUBBLE FACTOR=   1          COLS. PER SUBBLOCK=   1

    571 ORIGINAL NODES;          32 FAMILIES IN GROUP
  200 POINTER SET WORDS PER RECORD
  200 DATA BLOCK WORDS PER RECORD
     9 POINTER RECORD BUFFERS AND     30 DATA BUFFERS
    3 WORDS PER NODE POINTER SET
  10 FIELDS PER NODE
P(RETRIEVAL)= 0.850      P(NEW NODE IN FAMILY)= 0.130
     P(NEW FAMILY)= 0.020
"COMPRESS" RUN WHENEVER GAS/(GAS+ACTIVE NODES) EXCEEDS 0.10

   1784 REQUESTS SIMULATED;      500 SESSIONS
MEAN REQUESTS PER SESSION=   3.3
RETRIEVAL DENSITY=0.5000 OF FIELDS FOR
         0.4000 OF NODES IN 0.1000 OF FAMILIES



   1384 ACTIVE NODES
     22 NODES OF GAS REMAINING      MAX. GAS BUILDUP=      176 NODES
NUMBER OF FAMILIES CHANGED FROM   32 TO   75
   198 BUBBLE-OUTS OCCURRED
     33 COMPRESSES PERFORMED;       3914 NODES OF GAS REMOVED
    5834 POINTER RECORDS READ;      730 WRITTEN
   11884 DATA BLOCK RECORDS READ;      295 WRITTEN
MAX. POINTER BUFFER CONTENTS=     9 RECORDS
MAX. DATA BUFFER CONTENTS=    30 RECORDS

   57746 NODES RETRIEVED FROM
    291548 TOTAL VALUES RETRIEVED
```

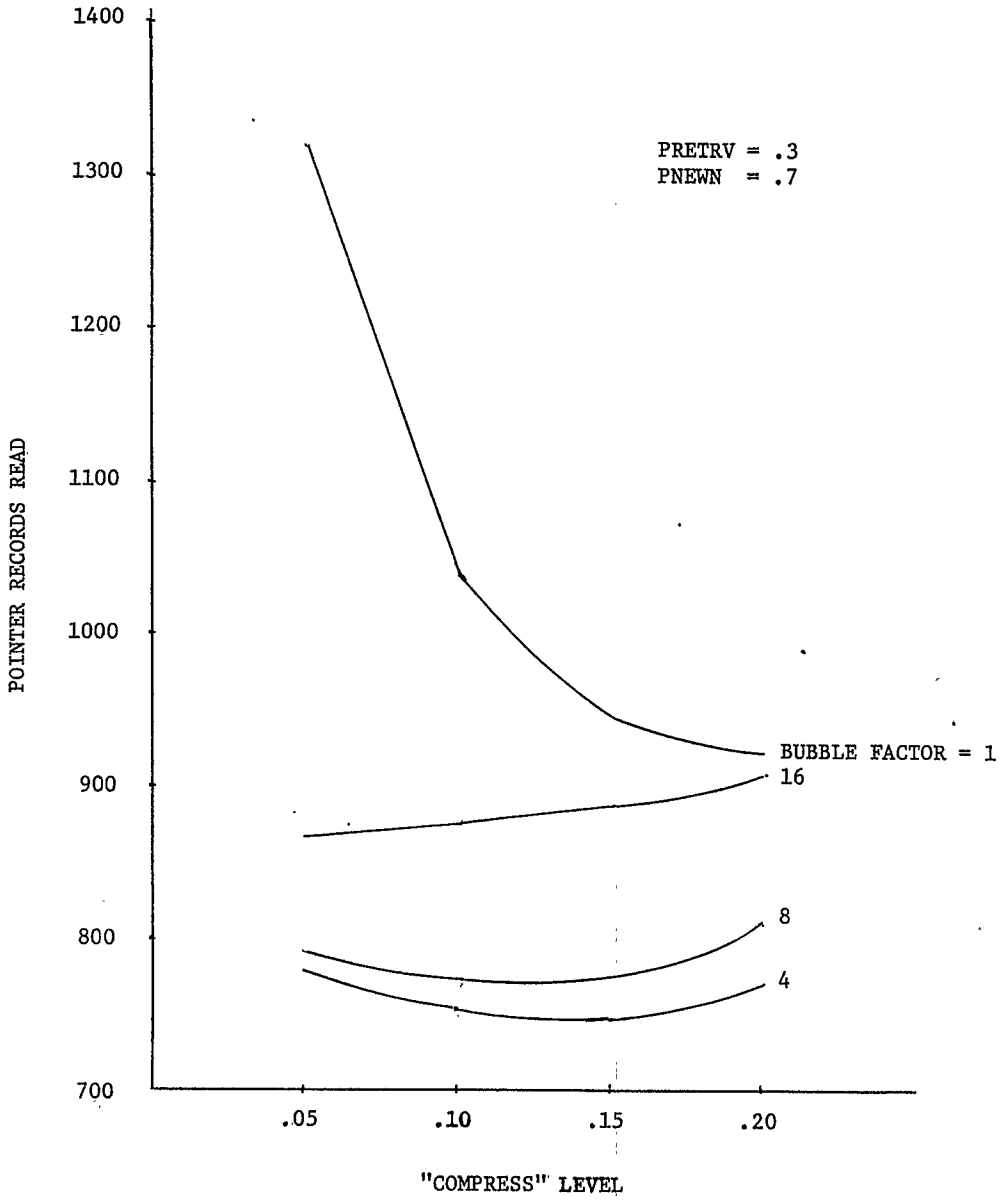
APPENDIX B



B-1



B-2


1400
1300
1200
1100
1000
900
800
700
POINTER RECORDS READ
PRETRV = .3
PNEWN = .7
BUBBLE FACTOR = 1
16
8
4
.05
.10
.15
.20
"COMPRESS" LEVEL