XIV Conferencia Latinoamericana de Informática
17avas. Jornadas Argentinas de Informática
e Investigación Operativa
Buenos Aires, Setiembre 1988.

Un Modelo de Administración de Buffers para la Predicción del Desempeño de Sistemas de Bases de Datos *

Ignacio R. Casas
Departamento de Ciencia de la Computación
Escuela de Ingeniería
Universidad Católica de Chile
Casilla 6177, Santiago, Chile
Fax: 56-2-5510046 Telex: 240395 PUCVA CL
e-mail: icasas@puc.cl o ...uunet!uchdcc!juncal!icasas

Kenneth C. Sevcik
Computer Systems Research Institute
University of Toronto
Toronto, Ontario M5S 1A4
Canada

Resumen

El diseño de sistemas de bases de datos es muy complejo debido al gran número de soluciones de compromiso que afectan la futura eficiencia y desempeño del sistema. Es una necesidad, entonces, el poder contar con herramientas de modelación de desempeño que ayuden en las distintas fases del desarrollo de un sistema de bases de datos.

El desempeño de un sistema de bases de datos se mide, principalmente, en base a tiempos de respuesta de transacciones y cantidad de accesos a memoria secundaria. En este contexto, uno de los aspectos importantes en la eficiencia de un sistema de bases de datos es el comportamiento del administrador de buffers el cual debe reducir los accesos a memoria secundaria que generan las distintas transacciones. En este artículo se presenta un modelo de desempeño de la administración de buffers, como parte de un model generalizado para la predicción del desempeño de sistemas de bases de datos. El modelo generalizado permite comparar decisiones de diseño, y sus correspondientes medidas de desempeño, en distintos niveles de abstracción del sistema computacional.

Se presenta la hipótesis de que las referencias a una base de datos se comportan de acuerdo a la distribución empírica Bradford-Zipf. Para fundamentar esta hipótesis, se presenta evidencia empírica obtenida de la observación y medición, durante cinco días, de un sistema de bases de datos en un ambiente de producción. Se presentan también los resultados de validación del modelo, con los datos obtenidos del monitoreo del sistema de bases de datos de producción.

* Esta investigación ha sido financiada por el Consejo Candiense de Investigación en Ciencias Naturales e Ingeniería (NSERC-Canada) y por la Dirección de Investigación de la Pontificia Universidad Católica de Chile (DIUC-Chile).

1. Introduction

Database (DB) systems are difficult to design and evaluate. The efficiency and performance of these systems is substantially affected by many decisions required throughout the development stages. Unfortunately, the impact on performance of many design alternatives is not yet well understood.

A number of comprehensive database performance models and methodologies have been proposed to help in the selection among design alternatives ([Miya75], [Naka75], [Sevc81], [Teor76], [Yeh77]). A weak point of these models has been the evaluation of the effect of buffer management on the performance of the database systems. Usually, it is assumed that block (page) references are uniformly distributed over all the database blocks. This assumption causes the use of buffering to appear relatively ineffective. Consequently, the performance predictions produced by models that use this assumption are unrealistically insensitive to the selection of buffer sizes and allocation strategies.

It is generally accepted that the buffer manager of a DB system can significantly affect the overall performance of the database system and computer system on which it runs. For this reason, in this paper we propose an improved submodel of buffer management for use within a comprehensive database performance prediction methodology. Our results will show that it is necessary to make a more realistic asssumption about the distribution of block accesses, but that the specific replacement policy used for buffer management need not be modelled in detail. We replace the assumption of uniform distribution of block accesses with the assumption that block references are distributed across database blocks according to a Bradford-Zipf (B-Z) distribution ([Broo69]).

Our analytic model will be based on the assumption of a random replacement policy for buffer management. Measurements of an operational database management system will be presented to validate our model, showing both that empirical block access distributions have a B-Z form and that conclusions based on a random replacement policy are indicative of those based on a variety of replacement policies used in real systems. We conclude that the resulting simple analytic model represents a suitable tradeoff between simplicity and accuracy in choosing a buffer management submodel for use in the context of a comprehensive database system performance prediction package.

In the next section we briefly describe a multi-level database performance model into which the buffer management model proposed in this paper can be embedded. While the terminology and notation used are consistent with that of the database literature, analogies with the virtual memory problem are indicated when appropriate. In section 3 we discuss the main differences between DB buffer management and OS virtual memory management. In section 4 we provide some empirical evidence for the operational hypothesis that database block references are Bradford-Zipf (B-Z) distributed. In section 5 we describe the database buffer management submodel and report the results of validating it against data from a real-life environment.

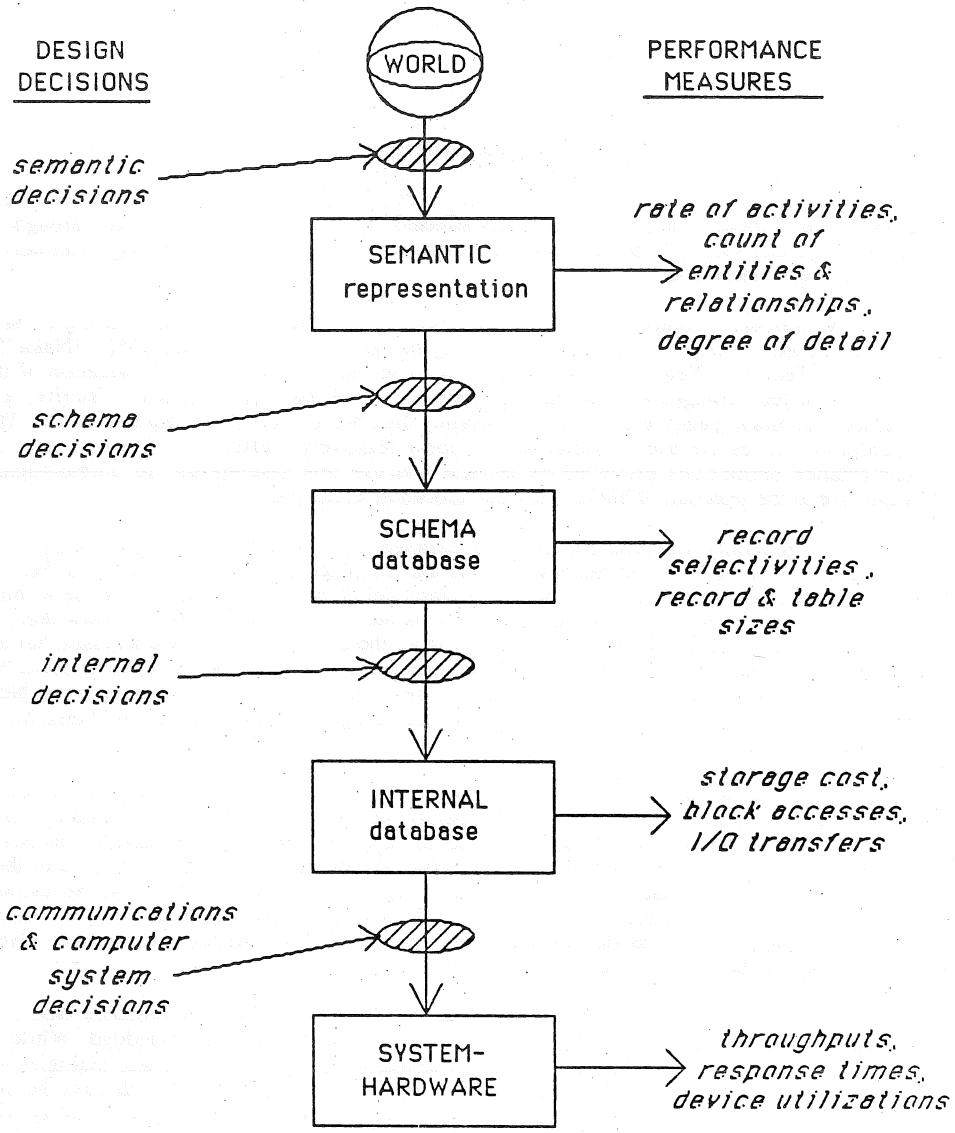

Figure 1: Layered Database Performance Model

2. A Layered DB Performance Model

In this section we briefly describe a layered performance model that provides a means of formal comparison of database design alternatives. A detailed description of the model and its application to actual systems is given elsewhere ([Casa86]).

The database design process can be viewed as the gradual transformation of an abstract data model into successively more concrete representations. Several levels of abstraction can be distinguished, where each successive level uses the decisions demarcated at this and higher layers, to represent the environment in more detail. As shown in Figure 1, we identify four levels of database description: semantic, DB schema, DB internal, and system-hardware. The bottom level, the most concrete one, includes the hardware and system software that runs the database application.

A set of design decisions is identified at each layer of the model, such that the representation at the previous level can be transformed into a more concrete representation at this level. Central to this hierarchical representation are models of computation (defined at each level) that provide performance measures to evaluate design choices.

A four-layer structure was chosen for several reasons. Firstly, levels are distinguished by the nature and longevity of the design decisions. For instance, decisions at the system-hardware level refer to the computer system where the database would eventually reside (e.g., file placement on devices, and processor scheduling strategies). The DB internal level refers to decisions relevant to the choice or design of a DBMS (e.g., file structures, access paths, and buffer management), while decisions at the DB schema level refer to the way the abstract data from the semantic level (e.g., entities and relationships) is represented in a formal logical data model (e.g, a relational schema). Secondly, this four-level structure can be naturally transformed into a software model for DBMS-specific performance predictors.

The first transformation layer of the system (see figure 2) involves decisions that lead to a formal specification of the entities to be represented in the database, the attributes of each of these entities to be recorded, and the types of operations (queries and updates) to be carried out. Criteria that can be used to evaluate the choices made at this level, with respect to their ultimate impact on system performance, include counts of the entities to be represented and rates of enquiry and update activities.

The second transformation layer (see figure 3) involves decisions that lead to a logical database design in the context of a chosen logical data model. Also, classes of transactions that perform database updates or pose queries are characterized as parameterized templates expressed in the data manipulation language associated with the logical data model. At this level, the number of records of each type and their sizes are determined. Also, for each transaction class, the rate of transactions is known, and the average number of records accessed can be estimated .

The third transformation layer (see figure 4) includes decisions that lead to a physical database design. It is at this level that the buffering strategy is represented. After determining the file structures and supporting access structures for the database, along with the encoding of database attributes, blocking factors for various portions of the database can be chosen. This provides enough information to estimate the average number of block accesses to each file structure component for each class of transactions. If buffers are used to hold some blocks in main memory for one use to the next, then the average number of physical block transfers (I/O operations as opposed to block accesses or references) per transaction can be reduced. The

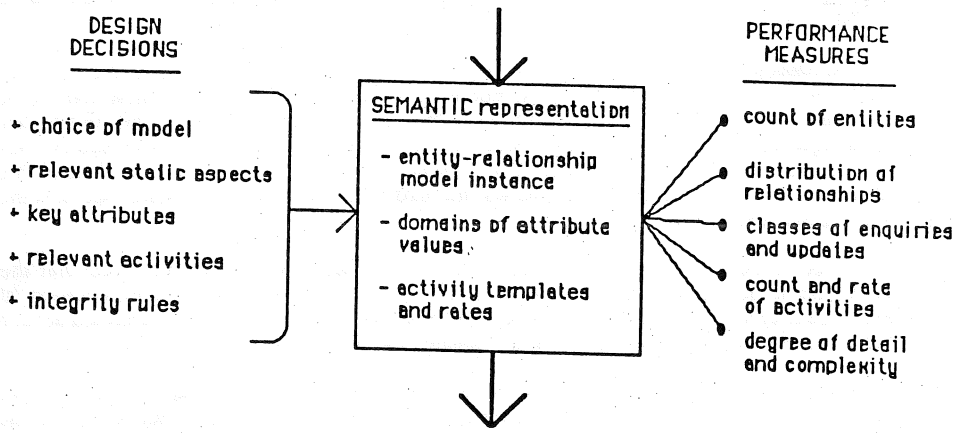

Figure 2 : Design Decisions, Representation, and Performance Measures at the Semantic Layer

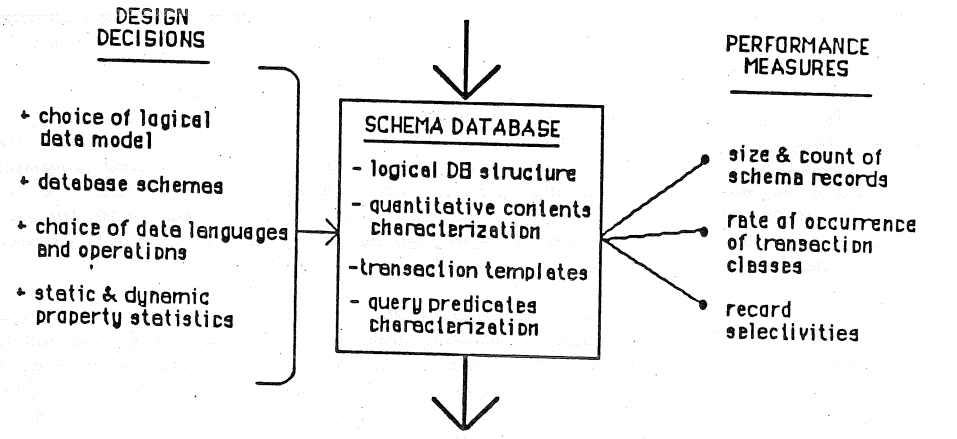

Figure 3 : Design Decisions, Representation, and Performance Measures at the Scheme Layer

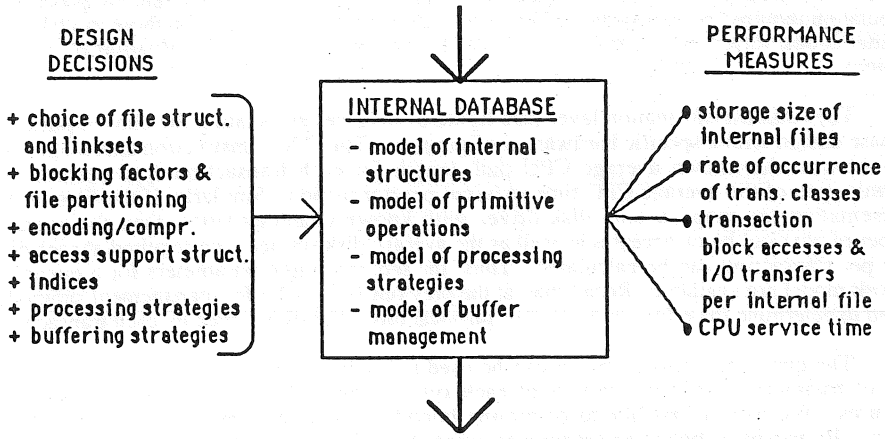

Figure 4 : Design Decisions, Representation, and Performance Measures at the Internal Layer

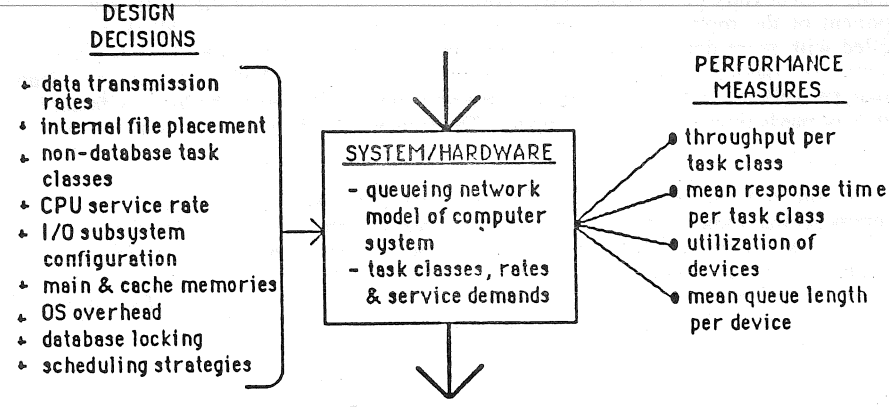

Figure 5 : Design Decisions, Representation, and Performance Measures at the System-Hardware Layer

larger the buffer size chosen to hold blocks of one part of the file structure, the greater the reduction in number of physical block accesses per transaction. At this layer, the evaluation of all design decisions made to this point is based on the expected number of physical block transfers required per transaction (and, to a lesser extent, on the amount of processor computation required per transaction). This is an improvement with respect to those models that consider block references instead of physical block transfers for the evaluation of design decisions.

The final transformation layer (see figure 5) involves associating the chosen physical database design with a specific hardware configuration. The CPU's instruction execution rate, together with the known average CPU path length for each transaction type permits the determination of the average CPU time required per transaction. Similarly, if the files of the representation are mapped onto disk drives with known characterisitics, then the average number of physical block accesses as well as the average disk service time required at each disk drive per transaction can be calculated. Thus, the required input parameters for a queueing network model are available. Recall that, at the previous layer, a buffer management submodel is used to determine the effective number of physical block transfers issued by each transaction.

The queueing network model can be used to predict average response times for each class of transactions, and utilizations of each system device, as well as other performance measures. We would first like to determine bounds on the performance measures of the system. By means of bounding techniques, upper bounds on throughput and lower bounds on mean response times can be obtained. These bounds are easy to calculate, even by hand, and they give a rough idea of the behaviour of the system. Thereafter, if these bounds show that performance would satisfy requirements and expectations, the model can be solved to estimate throughputs and mean response times for each task class, and the utilization and mean queue length by task class for each device.

At each level of the multi-layered model, simplifying assumptions are required to make the transformation tractable. Because of this, the design of various components of the multi-layered model must be balanced. Each component must achieve sufficient accuracy to keep it from limiting the overall accuracy desired from the model, and it should do so without requiring unnecessary detail or excessive computation. In the case of the buffer management component of the model, this balance requires that the distribution of block accesses be modelled with more detail than simply assuming it to be uniform. On the other hand, the assumption of "random" as the buffer replacement model simplifies both parameter specification and analysis while not reducing accuracy to the point where it becomes the limiting factor in the accuracy of predictions of the overall model. The facts will be demonstrated in the sections that follow.

In the next sections we describe the buffer management submodel and present measurements that indicate that actual DB block access distributions have a Bradford-Zipf form. We first discuss the main differences between DB buffer management and OS virtual memory management. The application of the multi-layered model to several commercial DBMSs, including the design, implementation and validation of a performance predictor for System 2000 databases is reported elsewhere ([Casa86], [Casa87]).

3. DB Buffering and OS Paging

Even though there are many similarities in the activity of DB transactions using a buffer pool and the activity of concurrent programs using virtual memory, important differences have motivated the particular study of buffer allocation and replacement algorithms within the context of data base management systems. In this section we discuss some of these differences.

As a first issue we note the importance of active and passive elements in both the virtual memory and DB context. That is, while a virtual memory operating system (OS) assigns physical memory partitions to software programs (active elements), some traditional DB applications allocate buffer partitions to internal files. In the latter case, buffers are assigned to the passive elements (internal files) rather than to the active elements (transactions) of the DB system. The System 2000 DBMS, for instance, establishes a maximum number of buffer frames for each internal file. If the buffer limit for a given file is reached, when performing some I/O transfers, then the least recently accessed buffer from the set of buffers currently associated with that file is chosen for replacement.

A combination of buffer partitions to passive and active DB elements can be used as well. For instance, the buffer management strategy for relational DB systems proposed by Chou and DeWitt ([Chou85]) subdivides the buffer pool and allocates it on a per file (relation) basis. Each query is associated with a distinct active instance of a file. Each active file instance is then given a distinct buffer pool partition.

A second issue refers to the fact that concurrent DB transactions frequently share a set of pages in the buffer pool. Indeed, the concurrent execution of DB transactions may increase the probability of pages being re-referenced in the buffer pool ([Effe84]). This effect is referred to as *inter-transaction locality* ([Maga81]). Consequently, the locality of database references depends not only on individual transactions but it is also dependent on the degree of data sharing of the database workload. (*Intra-transaction* locality is the locality exhibited in the page references issued by a single database transaction.) Although a similar situation may occur in a virtual memory context where programs can share code pages, the potential for data sharing is greater in the database context.

A third issue is the existence of well-defined internal access paths in a DBMS which permits the prediction of some aspects of the reference behaviour of transactions ([Chou85], [Sacc82]). It is often the case that DB pages containing specific indices, special directories, etc., are more often referenced than other data pages. In relational systems, for instance, a query optimizer uses pre-defined access plans such that the patterns of generated page references are regular and can be predicted.

Finally, a fourth issue deals with the fact that all references to records within a database page are treated as a single page (block) reference, while each machine instruction that specifies a program page accounts for a page reference ([Effe84]). Indeed, immediate re-references to a database page within the execution of a transaction are usually not considered as part of the database locality behaviour. This is so because records within that page can be addressed without a new block access or I/O transfer.

In retrospect, all the described differences between reference bahaviour of DB systems and programs under virtual memory have motivated the study of buffering strategies for database systems in particular. In spite of these differences, many of the replacement algorithms developed for virtual memory paging may be adapted for DB buffer management, due to the locality observed in database reference strings ([Effe84], [Sacc82]).

4. Locality of Reference

As in program reference behaviour, locality of database (DB) references represents the tendency of DB workloads to concentrate their block (page) references heavily into subsets of the DB stored blocks. Several empirical studies of operational DB systems indicate the presence of locality patterns in workload block references ([Effe84], [Hawt79], [Kear83], [Maga81]). The buffer manager of a DBMS, which attempts to exploit this locality behaviour, can significantly affect the overall performance of the computer system.

Within the context of the multi-layered DB performance model of the previous section, a model of DB buffer management is necessary for predicting the miss ratio at the buffer pool of a DB system. Once this ratio is known, the I/O transfer count (number of physical block accesses) for distinct transactions can be estimated. Prior to discussing a buffer management performance model in more detail, we discuss in this section the type of locality that empirical DB block references present.

Bunt and other researchers ([Bunt84]) have proposed a measure of program reference locality based on the Bradford-Zipf (B-Z) distribution ([Broo69]). This measure is based on the following expression:

$$L = B \times \bar{n}$$

B is intended as a measure of the *concentration* or *clustering tendency* of locality, while $\bar{n}$ reflects the *persistence tendency* of locality. B is a function of the proportion of highest productivity items that contribute to half the total productivity. $\bar{n}$ is defined as the mean number of consecutive references to a set of pages.

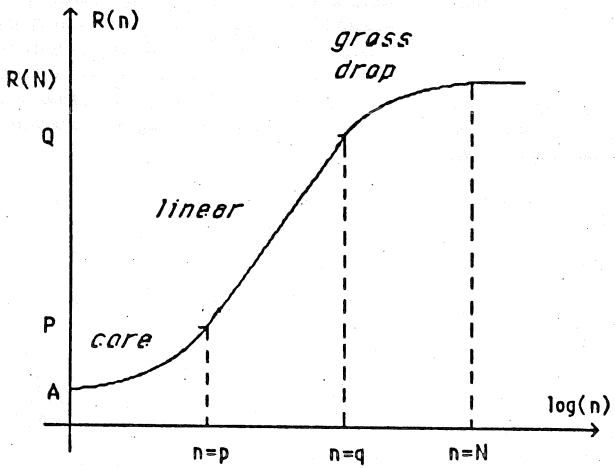

Figure 6: A Typical Bradford-Zipf Distribution ([Bunt84])

A typical B-Z distribution is depicted in figure 6 ([Bunt84]) where the natural logarithm of the rank of the items *(log(n))* has been plotted against the cumulative

productivity (relevance) of the items *(R(n))*. Items in the sample are ranked in order of decreasing productivity. Phenomena exhibiting behavioral patterns that fit the B-Z distribution include the yearly circulation of books or journals in a library, the number of articles on a given field in a given journal, the number of references to the article in subsequent journals, and the distribution of wealth, among others.

In the realm of computer systems, the Bradford-Zipf distribution has been used, for instance, to model information retrieval ([Benn75]), and to model the distribution of search keys in hashing tables ([Sams78]). More recently, Peachey ([Peac82]) has applied the B-Z distribution to examine the page reference patterns of executing programs. By plotting the natural logarithm of the rank of the pages against the cumulative productivity (frequency of references) for a series of program traces, Peachey observed a B-Z type of distribution. The assumption that program references are Bradford-Zipf distributed has been used as an operational hypothesis to analyze program restructuring methods, for the generation of synthetic program reference strings, and for the identification and analysis of program phase behaviour ([Peac82], [Bunt84]).

Motivated by Peachey's findings and the similarities between a database buffering setting and a virtual memory paging setting ([Effe84]), we have investigated the reference pattern in an operational database environment. We used the block reference strings generated during a 5-day-long observation period of five System 2000 databases (with a combined size of about 200 Megabytes) in a production setting. Daily average block reference frequencies were obtained for accesses to each of the DB internal files and for the databases as a whole ([Maga81]).

In this paper we describe only the results obtained with one of the databases. Similar results were obtained for the other four databases. The reported database is 1 Megabyte in size and activities of a single day include, on average, 361 commands executed, 41 seconds of CPU, and 27,556 block (page) references (block size is 2492 bytes).

A System 2000 database is composed of six internal files ([Casa86]). Given the measured block reference strings and frequencies, a graph of cumulative block reference productivity was obtaind for each file. Thereafter, for each of the measured cumulative productivities we have produced a corresponding hypothetical (fitted) B-Z distribution. To do this, we have used the B-Z formulation proposed by Brookes ([Broo69]):

$$R(n) = A \times n^{b} \qquad (A \leq R(n) \leq P)$$

$$R(n) = k \times \log(n / s) \qquad (P \leq R(n) \leq Q)$$

Parameter b determines the curvature of the "core" portion of the graph (Figure 6). P is the cumulative productivity at which the graph becomes linear. For the "gross drop" portion of the graph (Figure 6) we use:

$$R(n) = Q + (1 - e^{(q - n)}) \times m \qquad (Q \leq R(n) \leq R(N))$$

Q is the point at which the drop of the graph begins, and N represents the number of productive blocks.

Given these formulae, four points are needed to produce a fitted B-Z distribution (see figure 6): A, P, Q and N. These four points where determined for each of the measured block reference distributions of the System 2000 experiments, and corresponding fitted distributions

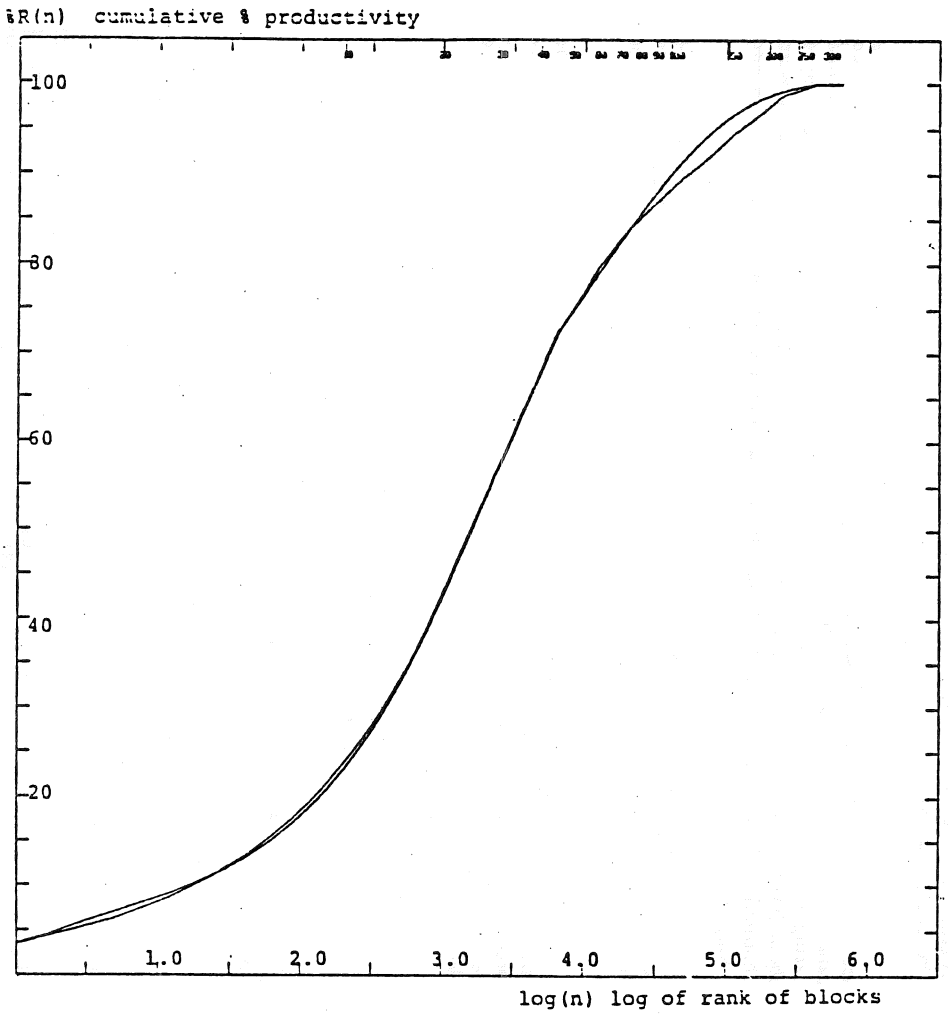

Figure 7(a)

Database B, All Files
Measured and
Hypothetical Distrib.

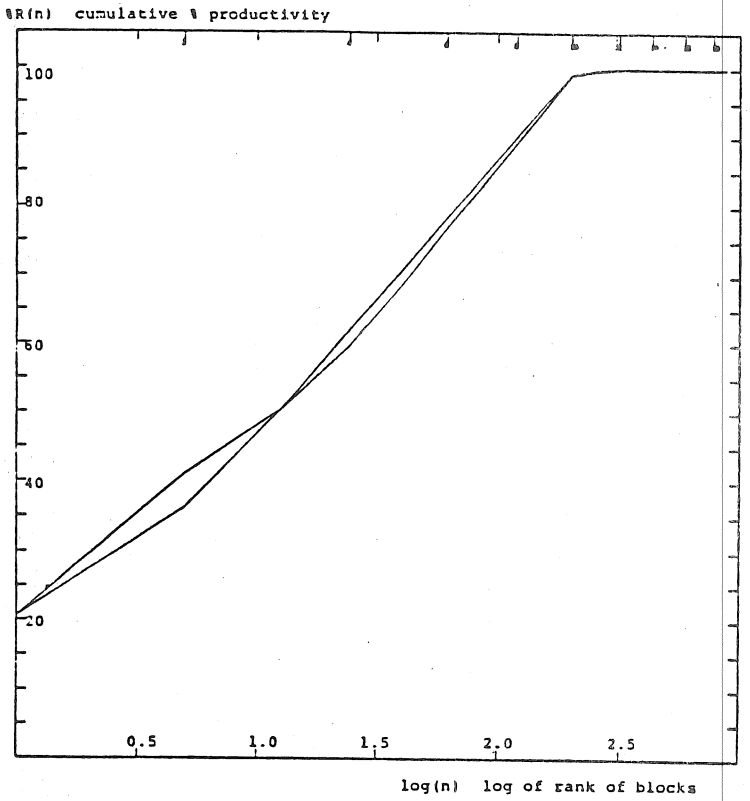

Figure 7(b)

File 2
Measured and
Hypothetical Distr.

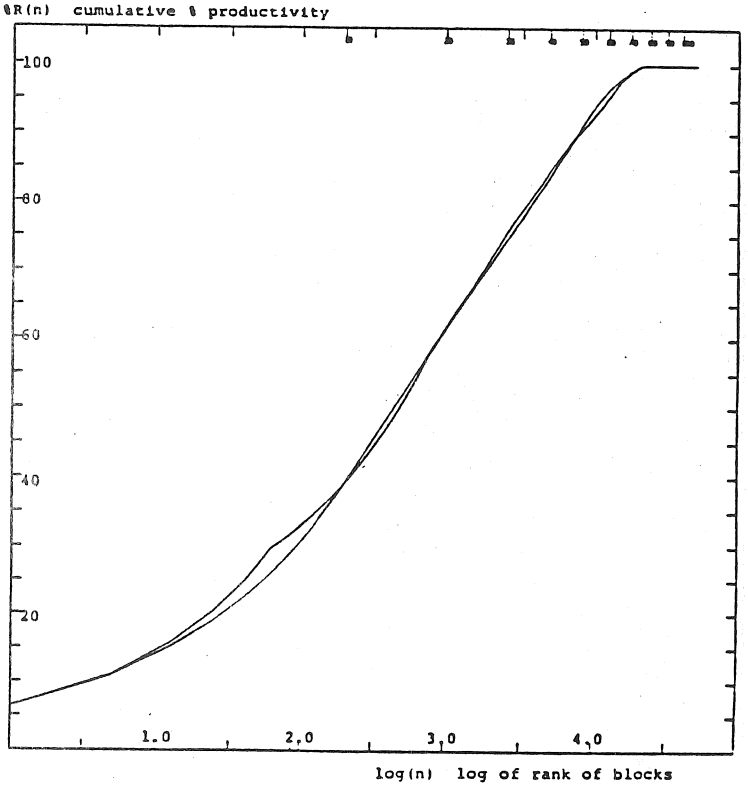

Figure 7(c)

File 4
Measured and
Hypothetical Distr.

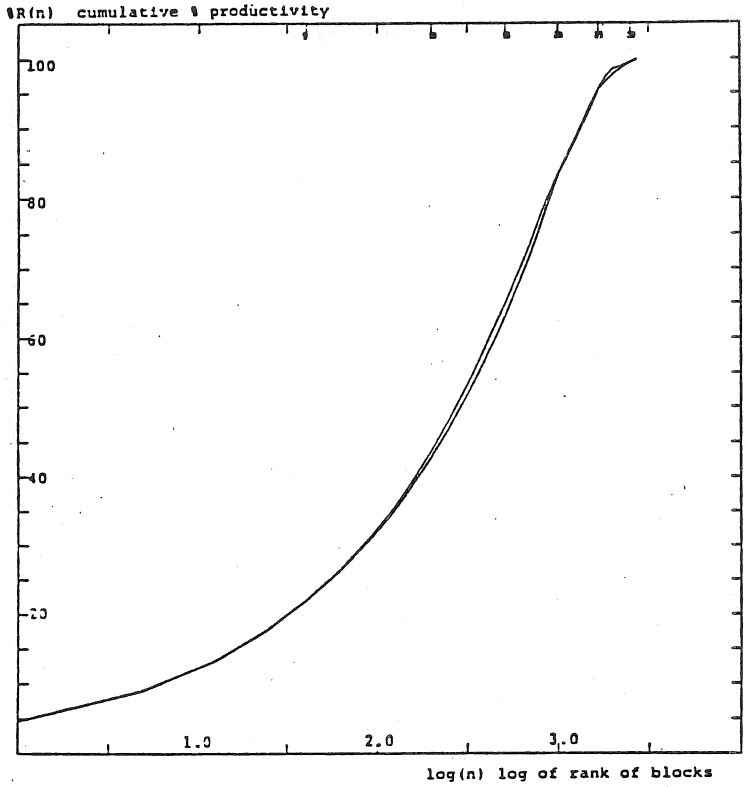

Figure 7(d)

File 5
Measured and
Hypothetical Distrib.

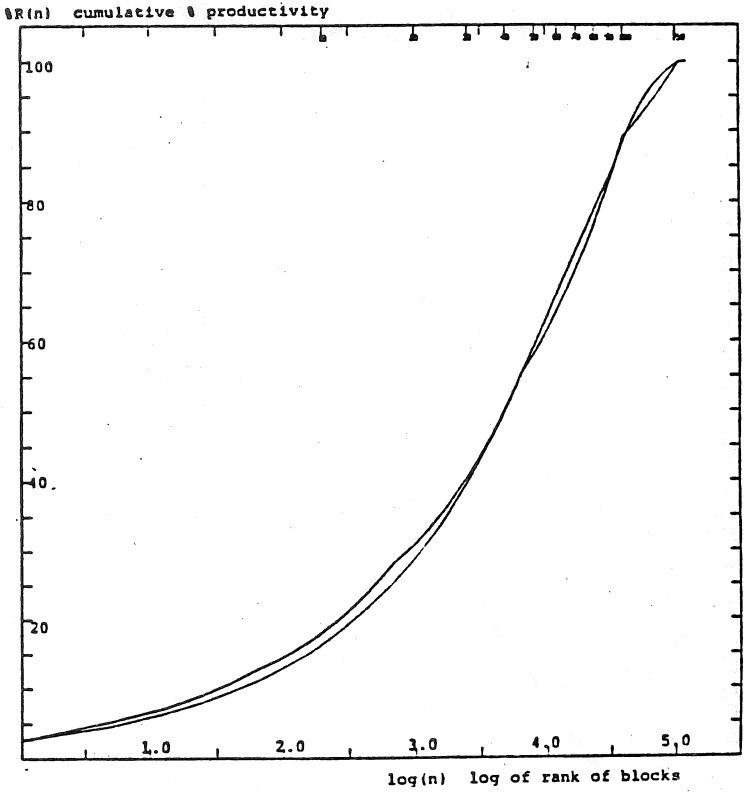

Figure 7(e)

File 6
Measured and
Hypothetical Distrib.

were generated. Figures 7(a) through 7(e) show the measured distribution (for the internal files of one of the System 2000 databases) compared to the corresponding fitted distribution. All five graphs indicate that the block reference behaviour of the database application closely approximate a B-Z type of distribution. A Kolmogorov-Smirnov test of goodness of fit was conducted for the observed distributions. The hypothesis of Bradford-Zipf distribution was accepted for all cases of the test with a level of significance of 95%. Similar results were obtained for the block reference distributions of the other four System 2000 databases.

As we mentioned earlier, most previous comprehensive performance models for database systems have used the simple (possibly simplistic) assumption of uniformly distributed accesses over database blocks. This assumption limited the ability of those models to reflect the performance improvement achievable through buffering. The B-Z distribution proposed here requires only a few parameters, but allows block access distributions to be represented with much greater accuracy.

5. A Performance Prediction Submodel of Buffer Management

Assuming then a Bradford-Zipf-like distribution of database block references, we derive now expressions that estimate the miss ratio achieved by a buffer manager. The buffer model is then validated against data from a real-life, operational database system. Model results are compared with results obtained using the traditional assumption of uniform distribution of references over the blocks of the database.

We assume that block references are independent. Even though not realistic, the independent-reference assumption simplifies analysis and, as we shall see, seems to allow fairly good performance estimates. Let us define the following terms:

TOTN	:	size (total number of blocks) of the database or internal file under observation
N	:	number of productive blocks; i.e., the number of distinct blocks that are referenced during a conveniently large observation period *(N ≤ TOTN)*
b	:	(variable) number of buffers (frames) in the buffer pool ($b \geq 1$)
s	:	allocation/replacement strategy used by the buffer manager
P	:	an observed block reference string

The miss ratio $M(s,b,P)$, a function of the buffer pool size (b), the allocation/replacement strategy (s), and the block reference string (P), is defined as

$$M(s,b,P) = 1 - H(s,b,P)$$

where $H(s,b,P)$ represents the hit ratio; that is, the proportion of times that a requested block is found in the buffer pool.

Given the experimental evidence from program behaviour and the locality behaviour displayed by database references, we expect that the miss ratio for an hypothetical "Uniform" reference string, that references each database block with probability $1/TOTN$ at each reference, provides a pessimistic approximation to the miss ratio of a real-life reference string under some practical buffering strategy. For this "Uniform" reference string, the hit ratio for any replacement strategy s is given by

$$H(s,b,T\_Unif) = \frac{b}{TOTN}$$

T_Unif denotes a block reference string with uniform probability of reference over all *(TOTN)* blocks of the database.

Given that N represents the number of productive blocks, let *N_Unif* denote a block reference string with uniform probability of reference over the productive N blocks of the database. It is assumed that the remaining $TOTN - N$ blocks are not touched at all by the reference string. We would expect the miss ratio for *N_Unif* to be better (lower) than for *T_Unif*, but still, it would be a pessimistic approximation to the miss ratio of a real-life reference string. For all replacement strategies s, the hit ratio achieved by *N_Unif* is given by

$$H(s,b,N\_Unif) = \frac{b}{N}$$

If, given a reference string P, p_i (where $i=1, \ldots, N$) represents the relative frequency of accesses to block i (e.g., as given by a Bradford-Zipf distribution), and assuming a replacement strategy (with global allocation) *FIX* that fixes in the first *(b-1)* buffers the most frequently used blocks and uses the remaining buffer to read in other blocks, the hit ratio is given by

$$H(FIX,b,P) = \sum_{i=1}^{b-1} p_i + \frac{p_b^2 + \cdots + p_N^2}{\sum_{k=b}^{N} p_k} \qquad ;b \leq N$$

For large databases (with a large number of productive blocks) we can approximate the hit ratio achieved by P under *FIX* by

$$H(FIX,b,P) \approx \sum_{i=1}^{b-1} p_i + \frac{p_b^2}{\sum_{k=b}^{N} p_k} + \frac{\left(\sum_{k=b+1}^{N} p_k\right)^2}{(N-b) \times \sum_{k=b}^{N} p_k}$$

H(FIX,b,P) may give an optimistic approximation to the real hit ratio rendered by actual replacement algorithms, including RANDOM and LRU, for real-life database reference strings. A compromise between the hit ratio for the *N_Unif* string *(H(s,b,N_Unif))* and the hit ratio achieved by *FIX* could represent a better estimate of the real hit ratio. For this reason, we assume an hypothetical replacement strategy, *BZ*, whose hit ratio is estimated as a weighted average of *N_Unif* and *FIX*, as follows:

$$H(BZ,b,P) = \frac{1}{2} \times \left(H(s,b,N\_Unif) \times (1-Q) + H(FIX,b,P) \times Q \right)$$

$$+ \frac{1}{2} \times H(FIX,b,P)$$

That is,

$$H(BZ,b,P) = \frac{b}{N} \times \frac{(1-Q)}{2} + H(FIX,b,P) \times \frac{(Q+1)}{2}$$

where Q is given by

$$Q = \left[H(FIX,b,P) \right]^{10 \times (1 - H(FIX,b,P))}$$

Since Q is in the range (0,1), $H(BZ,b,P)$ is between $H(s,b,N\_Uniform)$ and $H(FIX,b,P)$ for all practical values of b ($b \geq 1$). For small values of b, Q provides more weight to $H(s,b,N\_Unif)$, while for bigger values of b it provides more weight to $H(FIX,b,P)$.

We compare now the estimated miss ratio curves (using the formulae for T_Unif, N_Unif, FIX, and BZ) for the System 2000 application, against the real miss ratio curves measured by Magalhaes ([Maga81]). To trace block reference strings, Magalhaes logged all the transactions issued to a System 2000 database during an observation period, and then re-issued them against the original database within an experimental environment. For the measurements, Magalhaes set the buffer pool size to one (only one frame), so that block references could be collected at read/write time ([Maga81]). (System 2000 does not provide a monitoring tool for recording all block references.)

For this reason, the block reference strings in Magalhaes' data contain all references to blocks except immediate re-references (i.e., references to the block just previously referenced). Some immediate re-references are included in Magalhaes data, however, because System 2000 allows the enforcement, on user request, of write operations whenever a record is updated, as opposed to writing the updated block at the next buffer fault (the normal case).

To take into account in our model the effect of a reduced block reference string, the hit ratio formulae were adjusted as follows:

$$H_{adj}(s,b,P) = \frac{H(s,b,P) - H(s,1,P)}{1 - H(s,1,P)}$$

Notice that with $b=1$, the hit ratio $H(s,1,P)$ represents the proportion of immediate re-references (over all references) in the buffer pool.

Figures 8(a) through 8(e) show the predicted miss ratios (based on the distribution of block reference frequencies measured by Magalhaes for one of the databases) and the

measured miss ratios (for LRU and RANDOM replacement strategies). Figure 8(a) represents the average over all the files of the database, while figures 8(b) to 8(e) each cover individual internal files.

Curves marked *N_Unif* are obtained for the "Uniform" reference string (considering only productive blocks), while curves *BZ* represent the miss ratio estimated with the formula for *H(BZ,b,P)*. Curves marked *L* and *R* represent the real miss ratios measured by Magalhaes for (global) LRU and RANDOM replacement strategies respectively ([Maga81]). Curves marked *T_Unif* represent the miss ratio for the Uniform reference string considering all blocks of the database (or internal file), as opposed to productive blocks only. Curves *FIX* indicate the estimated miss ratio for the FIX strategy.

It is interesting to notice from the measured miss ratio curves, how badly LRU performs in those cases when the buffer pool is not of an appropriate size to contain a whole locality set.

Table 1 presents an error analysis for the estimation formulae (*T_Unif, N_Unif, FIX,* and *BZ)* considering: (1) the average over all files of the database ("Whole DB"), (2) each file alone ("File x"), and (3) the average obtained for all observed cases ("Total Cases").

Table 1
Miss Ratio Curves Error Analysis
(with measured block reference distributions)

	Model	Whole DB	File 2	File 4	File 5	File 6	Total Cases
Avg.	T_Unif	32.2	26.8	33.9	6.6	31.0	27.3
Error	N_Unif	30.3	7.4	27.2	6.6	30.5	23.9
%	BZ	3.7	5.2	10.9	2.8	23.4	8.7
	FIX	10.8	9.5	1.0	5.3	16.3	8.9
Max.	T_Unif	57.4	48.5	50.0	18.5	37.9	57.4
Error	N_Unif	53.6	16.7	36.5	18.5	36.9	53.6
%	BZ	7.5	11.0	17.0	7.0	26.1	26.1
	FIX	14.7	19.0	3.3	11.1	18.4	19.0

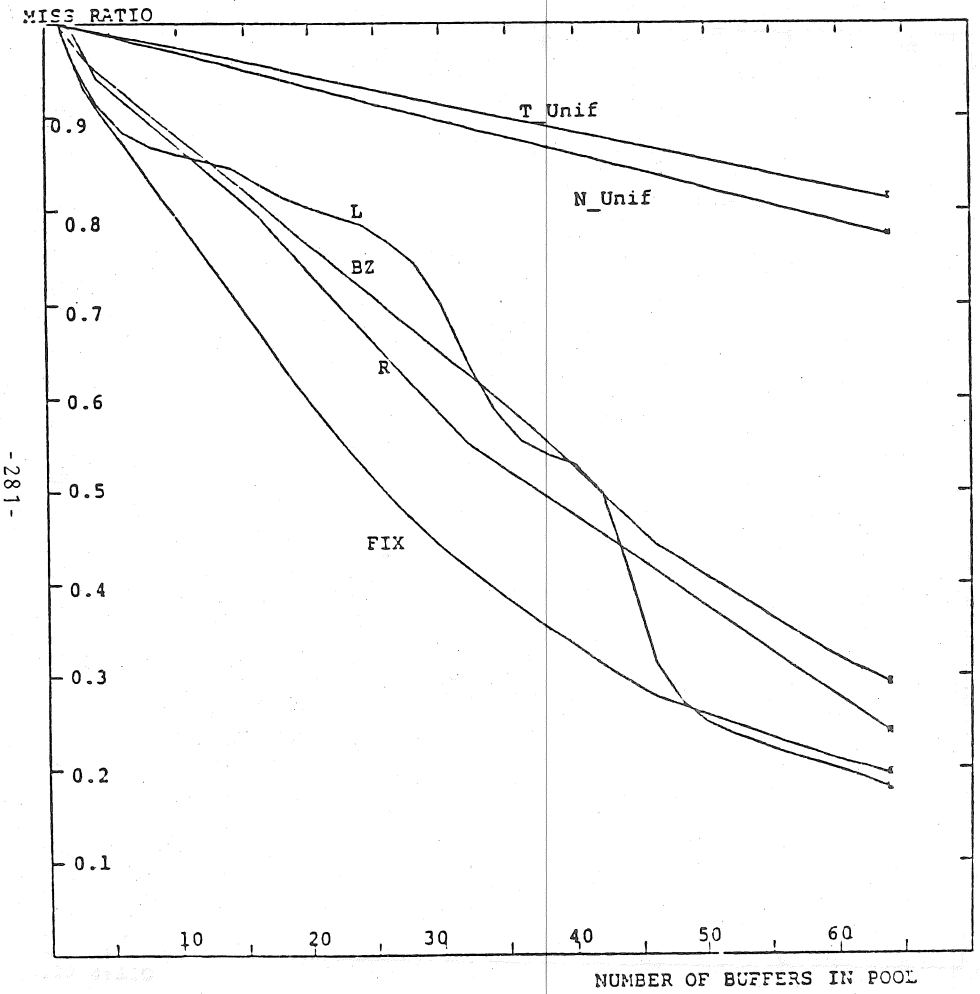

Figure 8(a)

Database B, All Files
Total # Blocks = 335
Prod. Blocks = 278

R =measured RANDOM
L =measured LRU
BZ =model estimated
FIX =model estimated
N_Unif =uniform over prod. blocks
T_Unif =uniform over all blocks

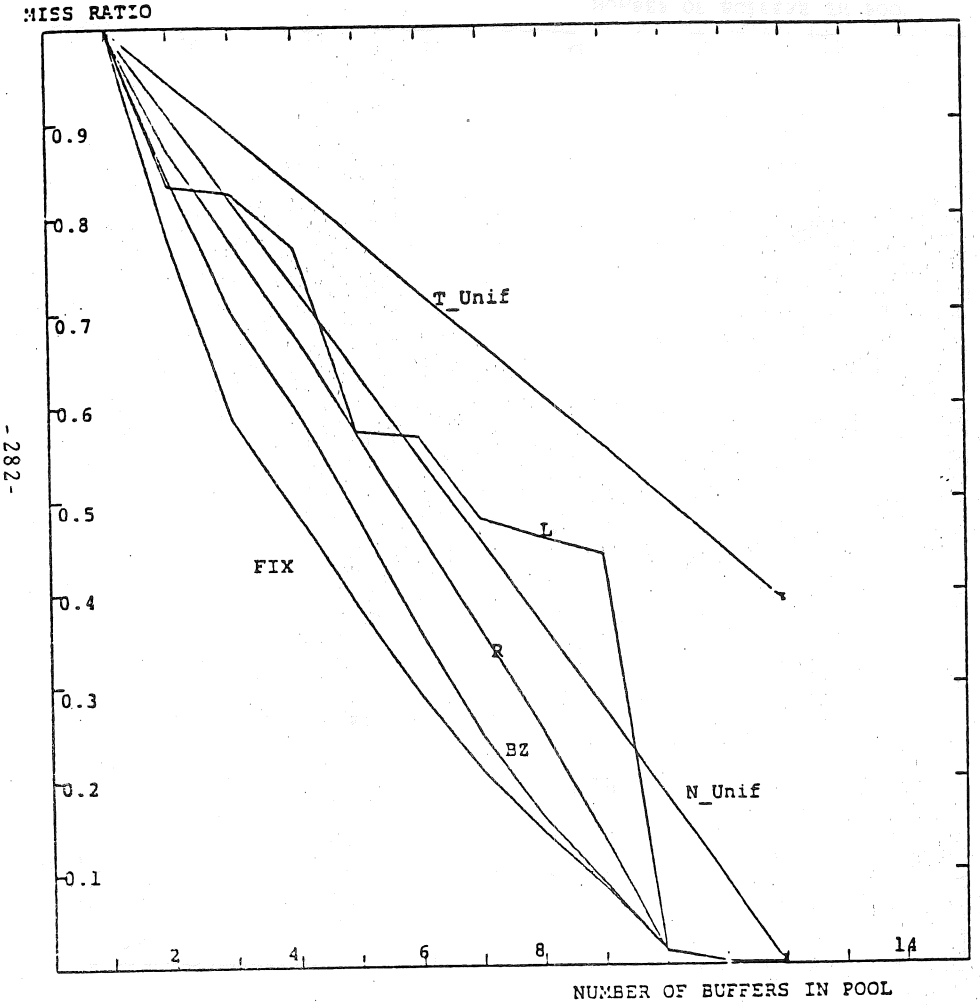

Figure 8(b)

File 2
Total # Block = 19
Prod. Blocks = 12

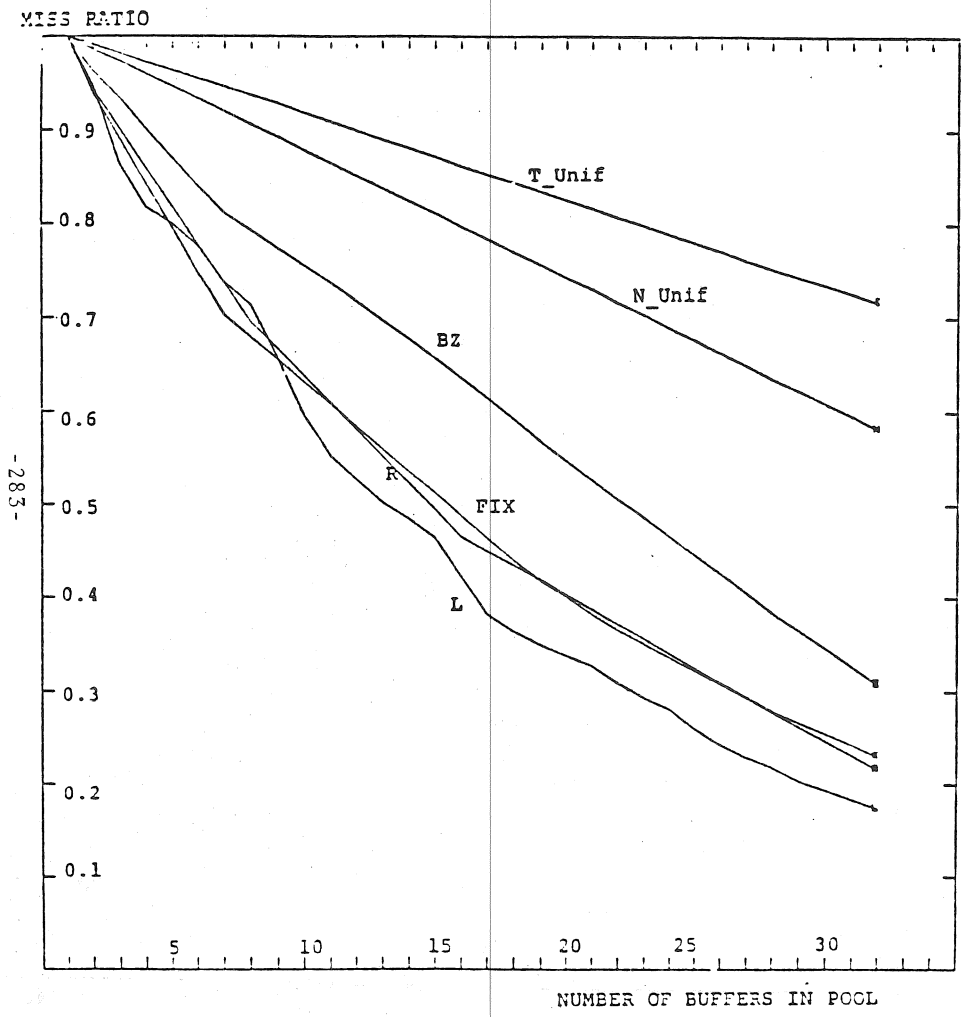

Figure 8(c)

File 4
Total # Blocks = 110
Prod. Blocks = 75

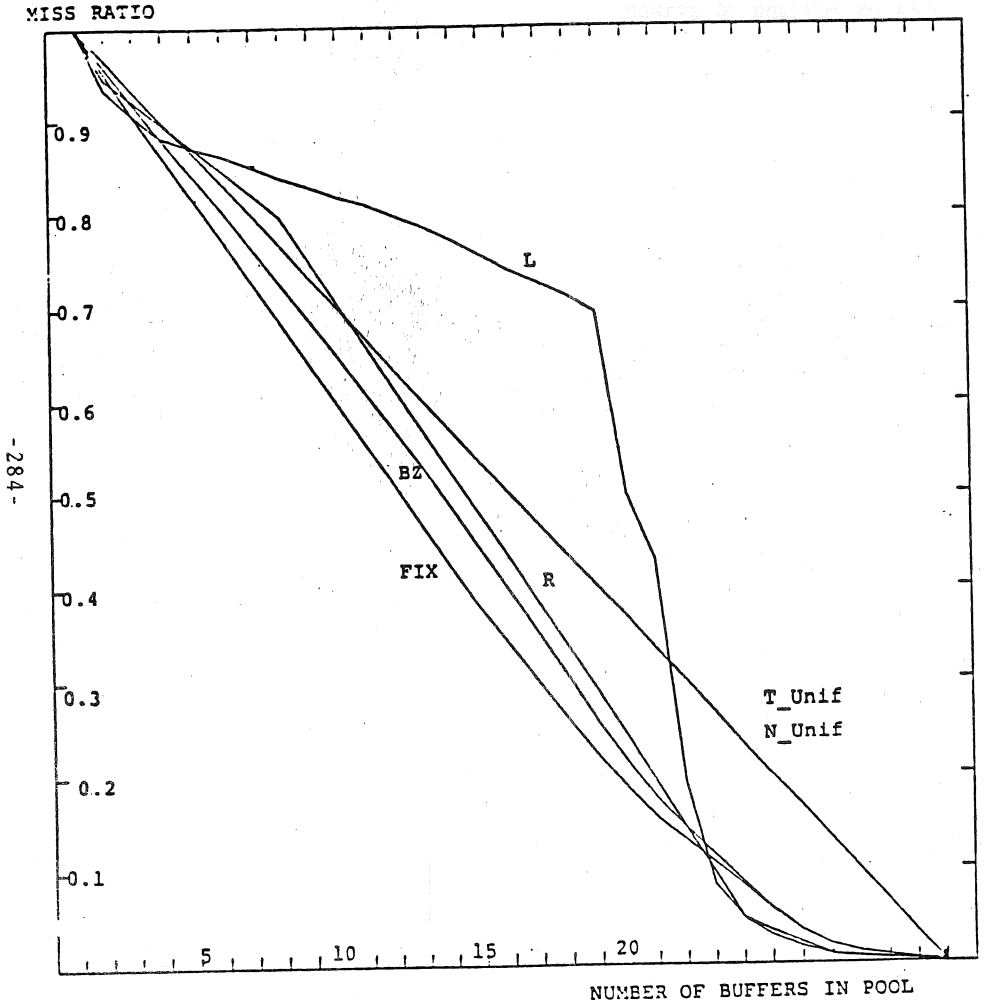

Figure 8(d)

File 5
Total # Blocks = 31
Prod. Blocks = 31

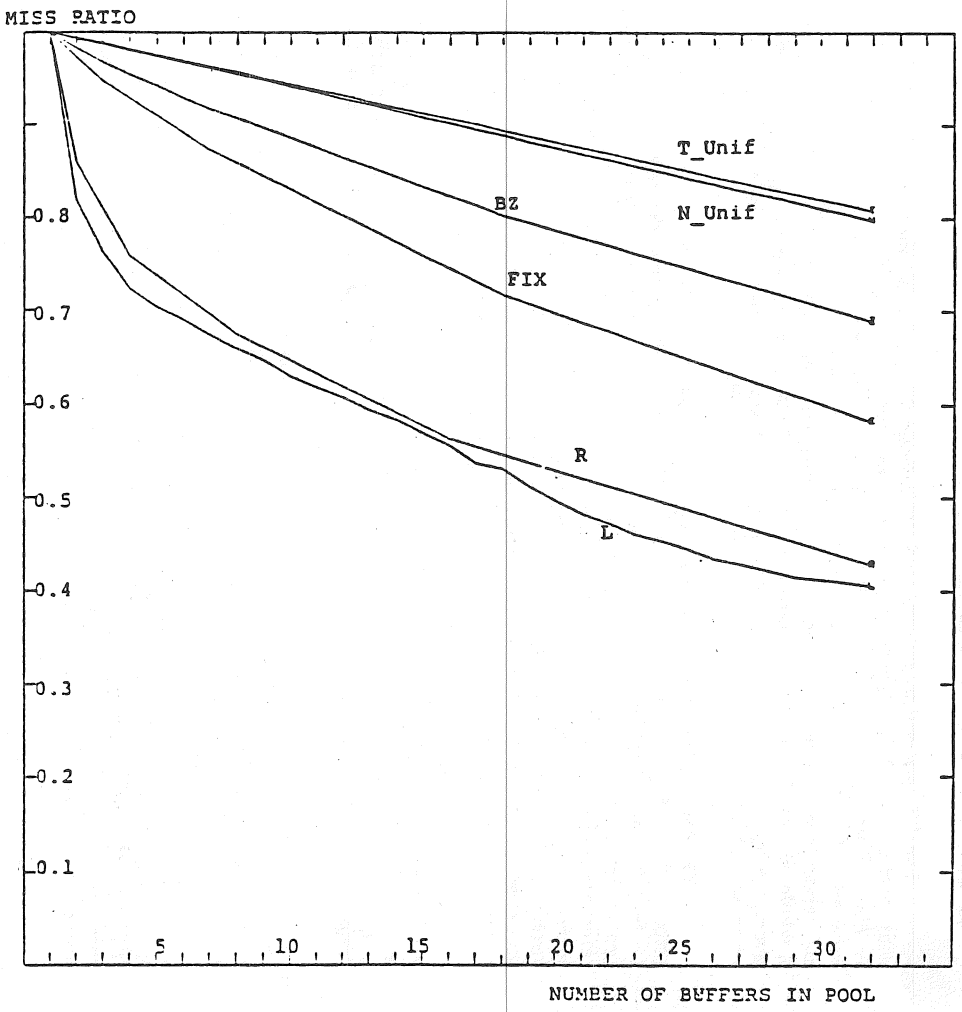

Figure 8(e)

File 6
Total # Blocks= 162
Prod. Blocks = 154

Using as a reference the miss ratio measured for the RANDOM replacement strategy, error estimates are computed as:

$$ERROR\ \% = \left| predicted\ M(s,b,P) - measured\ M(RANDOM,b,P) \right| \times 100$$

Maximum error percentages ("Max. Error %" in Table 1) represent the maximum errors rendered by the cost formulae in the observed cases. As expected, curve *N_Unif* (uniform references over productive blocks only) yields a pessimistic approximation to the real miss ratio (miss ratio *N_Unif* is higher than measured RANDOM miss ratio in 92.4% of all the observed cases). Still, *N_Unif* with a 23.9% average error and a 53.6% maximum error for all observed cases achieved a better approximation to the real miss ratio than *T_Unif* (uniform distribution of references over all blocks of the file or database) which rendered a 27.3% average error and 57.4% maximum error.

The *BZ* estimation delivered a 8.7% average error over all measured cases, with a maximum error of 26.1%, against a 23.9% average error and a 53.6% maximum errors of *N_Unif*. With respect to *N_Unif*, *BZ* represents a 63.9% improvement in terms of average error and a 51.3% improvement in terms of maximum error. Except for Files 4 and 6, curve *BZ* renders a better approximation to the real miss ratio than *FIX* does. For the database as a whole, *BZ* renders a 3.7% average error and a 7.5% maximum error against a 10.8% average error and 14.7% maximum error of *FIX*. A similar range of average and maximum errors was obtained for the other four databases in the study.

The least accurate estimates achieved by *BZ* are those for file 6 (23.4% average error, and 26.1% maximum error). This is probably due to the fact that the reference strings for file 6 contain a large number of insertions. New data is inserted at the bottom (or at empty spaces left by deletes) of file 6. Since there are many more insertions than deletions in the reference strings, new blocks (at the bottom of file 6) are referenced from day to day, and insertions continue to be done on the same block until the block is full. As a consequence, a strong correlation of references (persistence aspect of locality) is obtained which is not accounted for by the BZ formula. (This particular condition can be incorporated into our model by an appropriate separation of read and write block reference counts.)

The same argument applies to file 4. File 4 is only referenced once the required data records (and blocks) have been previously selected by means of content addressibility and data relatability operations performed on the other files of the database. The relative proportion of write operations is therefore greater on file 4 than on the other files of the database.

Also, we have conducted another set of experiments using this time the fitted B-Z block access distributions (as opposed to the measured distributions used in the previous experiments). In these experiments, we assume that we do not have complete information about block reference distributions, so we use hypothetical B-Z distributions that model the behaviour of the real database. Figure 9 shows the predicted miss ratios for *BZ* and *FIX* with the hypothetical distributions, and the measured LRU and RANDOM miss ratios. Notice that figures 8(a) and 9 are almost identical even though, in figure 9, the predicted miss ratio curves *(BZ* and *FIX)* were obtained using a fitted B-Z distribution, while in figure 8(a) the same curves were obtained using the measured block reference distribution. In both figures, curves *L* (measured LRU miss ratio) and *R* (measured RANDOM miss ratio) are exactly the same.

Table 2 presents an error analysis for the model predictions using the fitted B-Z

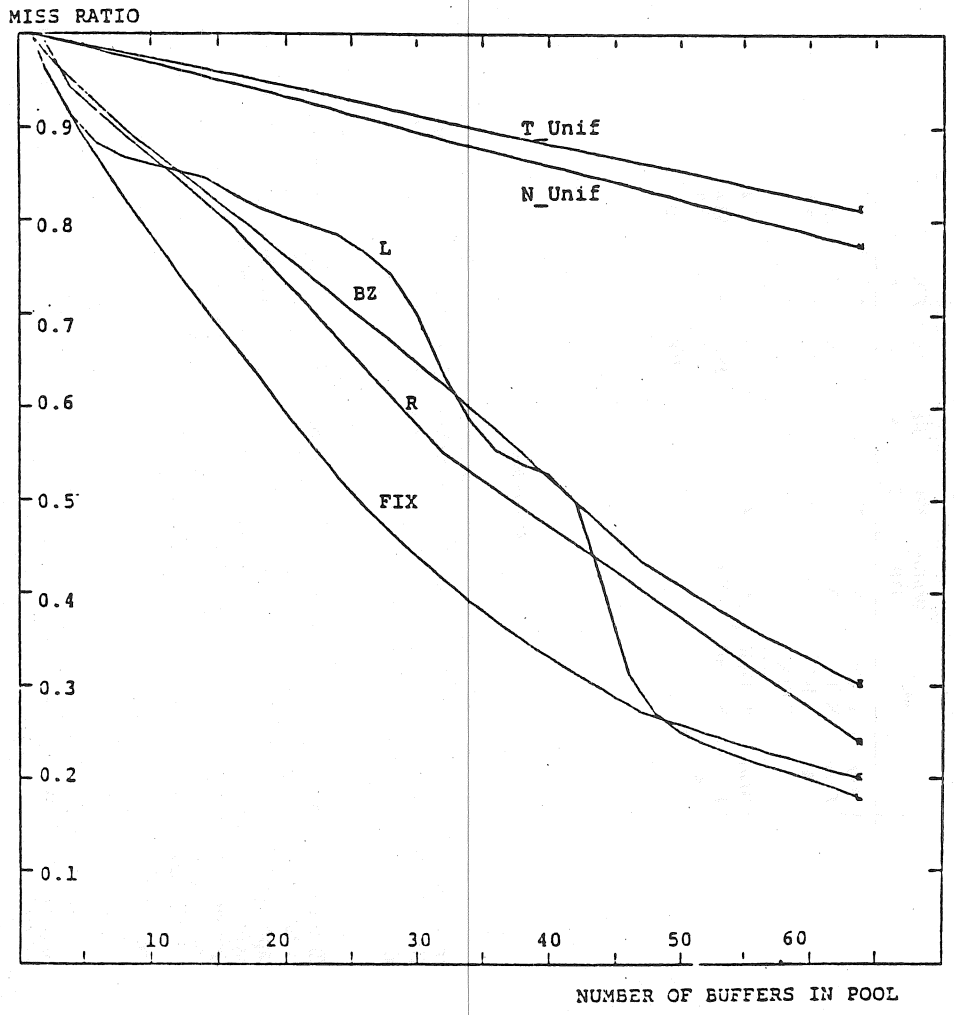

Figure 9

Database B, All Files
Miss Ratio using
Hypothetical Bradford-
Zipf Distribution

distributions against the measured RANDOM miss ratio. It can be noticed from Table 2 that average and maximum error percentages for *BZ* and *FIX* have increased slightly (with respect to Table 1) in some cases and decreased slightly in others (files 4 and 5). The average error is approximately 9% for both estimates, and the maximum error is 27.3% for *BZ* (an increment of 1.2) and 20.7% for *FIX* (an increment of 1.7). Consequently, the Bradford-Zipf hypothesis provides satisfactory predictions for all the observed cases.

Table 2
Miss Ratio Curves Error Analysis
(with hypothetical Bradford-Zipf block reference distributions)

	Model	Whole DB	File 2	File 4	File 5	File 6	Total Cases
Avg. Error %	T_Unif	32.2	26.8	33.9	6.6	31.0	27.3
	N_Unif	30.3	7.4	27.2	6.6	30.5	23.9
	BZ	3.8	5.7	11.2	2.4	24.4	8.9
	FIX	10.8	9.7	0.6	4.9	17.8	9.1
Max. Error %	T_Unif	57.4	48.5	50.0	18.5	37.9	57.4
	N_Unif	53.6	16.7	36.5	18.5	36.9	53.6
	BZ	7.5	12.7	16.3	6.9	27.3	27.3
	FIX	15.0	20.7	2.7	10.8	20.6	20.7

6. Summary

We have studied the reference behaviour of an actual production database workload and provided some evidence that block references follow a Bradford-Zipf-like distribution. We have indicated that this observation can be a useful operational hypothesis for the analysis of database systems, particularly for database performance predictions.

We have proposed a model of buffer management (the *BZ* and *FIX* formulae with B-Z block reference distributions) that estimates the buffer miss ratio of a database workload as a function of the relative frequency of block references, and the buffer pool size. These approximations can be used to determine the I/O activity generated by a particular workload, within the context of a layered database performance prediction model. This represents an intermediate step in the process of predicting database performance in terms of response times and throughputs. The model can be used to assess the effect of buffer sizing, buffer allocation (to internal files), and buffer locking strategies. In the context of selecting an appropriate buffer replacement strategy, our model predictions can be used as an indication of achievable buffering performance.

The buffer miss ratio model was validated against data from a production system, yielding an 8.7% average error and a 26.1% maximum error with the *BZ* estimate, and an 8.9% average error and a 19.0% maximum error with the *FIX* estimate. For all the observed cases, the use of an hypothetical Bradford-Zipf distribution (as opposed to the exact block reference distribution) does not increase significantly the error in the performance estimates.

It is clear that a B-Z distribution of block references conveys only the clustering aspects of locality but not its persistency tendency ([Bunt84]). For models of the type described in section 2, it is not realistic to attempt to capture the persistency aspect of DB references in a production environment, due to the high cost involved in the process. In contrast, the block access frequency distribution generated by an operational system can be easily obtained by direct measurement. For a system in the design stages, relative frequencies of block accesses can be estimated based on the B-Z distribution and the observation of similar operational systems.

The performance model of database buffer management proposed here represents a balance between simplicity and accuracy. It is more accurate than the simpler models based on the assumption that block references are uniformly distributed. To achieve this increased accuracy, the several parameters required to specify a B-Z distribution must be determined for each database file. Still our model requires for less detailed specification than do many models that concentrate on the buffer management problem in isolation (e.g., [Fern78]).

By assuming that the buffer replacement strategy in our proposed model is random, we lose the capability of comparing one buffer replacement strategy against another. This can be viewed as more of a benefit than a fault, however, in that the behaviour of the random replacement algorithm is much less sensitive to the details of specific reference strings than is LRU, for example. (Note how much more erratic the behaviour of LRU is than that of RANDOM in figures 8 and 9.) Thus, for the purpose of predicting the effects of changes in database block sizes and buffer sizes, a model based on random replacement may well yield better estimates of relative performance changes than one based on LRU replacement.

Overall, the database buffer management model proposed here has accuracy commensurate with that of the other components of multi-layer database performance models, yet does not require extensive parameterization or excessive computation.

References

[Benn75] Bennet J.M., Gelenbr E., Potier D.
"Storage Design for Information Retrieval: Scarrott's Conjecture and Zipf's Law", Proc. Int. Computer Symp. 1975, North-Holland Publ. Co., 1975, 233-237.

[Broo69] Brookes B.C.
"The Complete Bradford-Zipf 'Bibliography'", Journal of Documentation #25.1, March 1969, 58-60.

[Bunt84] Bunt R.B., Murphy J.M., Majumdar S.
"A Measure of Program Locality and Its Application", Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, ACM Performance Evaluation Review Special Issue #12.3, August 1984.

[Casa86] Casas I.R.
"Prophet: A Layered Analytical Model for Performance Prediction of Database Systems", Ph.D. Thesis, Dept. of Computer Science, Univ. of Toronto, Tech. Report CSRI-180, Toronto, Canada, May 1986.

[Casa87] Casas I.R., Sevcik K.C.
"Structure and Validation of an Analytic Performance Predictor for System 2000 Databases", Proc. 5th Canadian Information Processing Society Conf., Edmonton, Canada, Nov. 1987.

[Chou85] Chou H.T., DeWitt D.J.
"An Evaluation of Buffer Management Strategies for Relational Database Systems", Proc. 11th VLDB Int. Conf., Stokholm, Sweden, August 1985, 127-141.

[Effe84] Effelsberg W., Haerder T.
"Principles of Database Buffer Management", ACM Trans. Database Syst. #9.4, Dec. 1984, 560-595.

[Fern78] Fernandez E., Lang T., Wood C.
"Effect of Replacement Algorithms on a Paged Buffer Database System", IBM J. Research & Development #22.2, March 1978.

[Hawt79] Hawthorn P., Stonebraker M.
"Performance Analysis of a Relational Data Base Management System", Proc. ACM SIGMOD Int. Conf., Boston, 1979.

[Kear83] Kearns J.P., DeFazio S.
"Locality of Reference in Hierarchical Database Systems", IEEE Trans. Software Eng. #SE-9.2, March 1983, 128-134.

[Maga81] Magalhaes G.C.
"Improving the Performance of Data Base Systems", Ph.D. Thesis, Dept. Computer Science, Univ. of Toronto, Tech. Report CSRG-138, Toronto, Canada, Dec. 1981.

[Miya75] Miyamoto I.
"Hierarchical Performance Analysis Models for Data Base Systems", Proc. First VLDB Int. Conf., Boston, 1975, 322-352.

[Naka75] Nakamura F., Yoshida I., Kondo H.
"A Simulation Model for Data Base System Performance Evaluation", Proc. AFIPS Nat. Comp. Conf., 1975, 459-465.

[Peac82] Peachey J.B.
"The Bradford-Zipf Distribution and Program Behaviour", M.Sc. Thesis, Dept. of Computational Science, University of Saskatchewan, Saskatchewan, Canada, Tech. Report 82-1, 1982.

[Sacc82] Sacco G.M., Schkolnick M.
"A Mechanism for Managing the Buffer Pool in a Relational Database System Using the Hot Set Model", Proc. 8th VLDB Int. Conf., Mexico, 1982, 257-262.

[Sams78] Samson W.B., Davis R.H.
"Search Times Using Hash Tables for Records with Non Unique Keys", The Computer Journal #21.3, Aug. 1978, 210-214.

[Sevc81] Sevcik K.C.
"Data Base System Performance Prediction Using an Analytic Model", Proc. 7th VLDB Int. Conf., Cannes, France, Sept. 1981.

[Teor76] Teorey T.J., Das K.
"Application of an Analytical Model to Evaluate Storage Structures", Proc. ACM SIGMOD Int. Conf., 1976, 9-19.

[Yeh77] Yeh R.T., Baker J.W.
"Toward a Design Methodology for DBMS: A Software Engineering Approach", Proc. 3rd. VLDB Int. Conf. , Tokyo, Japan, 1977, 16-27.