Access Path Selection in Databases with Intelligent Disc Subsystems

W. KIESSLING
*Technische Universität, Institut für Informatik, Arcisstr. 21, D-8000 München 2, West Germany**

As the performance demands imposed on relational database systems are steadily increasing, it becomes more difficult to master them with conventional database architectures. New architectural proposals suggesting the use of intelligent disc subsystems in combination with appropriate set-oriented interfaces look promising, because they can be realised by standard, high-performance hardware now. In particular, it makes sense to exploit the fast sequential read-out capabilities of modern discs. In such a changed environment the access-path selection problem, especially the question of whether index usage can speed up retrieval query execution, requires new solution approaches. Based on easily acquired performance data, an analytical model for restriction queries is developed that allows us to determine threshold hit ratios separating profitable from non-profitable index usage. As this model is founded on an extent-based file organisation, it is likewise applicable to conventional architectures permitting chained I/O. Simulation results for two common disc-pack drives are reported, showing that the bottleneck for random disc accesses sharpens for modern discs, and thus makes exhaustive relation scans preferable in more cases. However, indexes will still be mandatory on highly selective attributes to achieve satisfactory performance. The presented optimisation criteria are directly amenable to incorporation into a query optimiser.

Received May 1986, revised November 1986

1. INTRODUCTION

In relational database (DB) systems the optimisation of query execution is mandatory to achieve acceptable performance. Typically, the query optimiser selects a 'cheapest' access path among several feasible processing strategies. As an example consider the following query in SQL-form.[1]

SELECT r1 FROM R WHERE r2 $\geqslant$ 17

As an alltime available access path a complete scan over the relation R is one feasible processing strategy. Alternatively, if an index – e.g. B-tree or ISAM – exists on the restriction attribute r2, a second processing strategy would be to first pick up the qualifying tuple references by examining the index and secondly to materialise only those qualified tuples. Clearly there is a trade-off between the potential savings in tuples to be fetched and the extra expenditure required to traverse the index. This issue was also addressed by some DB machine designs that tried to avoid the index overhead costs entirely by heavy use of parallelism supported by specialised hardware (see e.g. Refs 15 and 9). However, recently the opinion of what should be the architecture of an efficient general-purpose relational DB system favours those systems that rely on indexes to speed up retrieval query execution (see e.g. Refs 3–5). But since conventional DB architectures, being characterised by disc subsystems with a fetch/store–block interface, can hardly master the steadily increasing performance demands, different architectures must be considered. A recent proposal, which uses only off-the-shelf hardware and which aims in this direction, can be found in Ref. 12. That design proposes a set-oriented interface to an intelligent disc subsystem, much in the style of a processor-per-disc architecture.[4] Because an intelligent disc has its read/write arm dedicated at its own disposal, it can optimise set-oriented disc access requests better than is possible in conventional DB systems. There, in general, the DB system has little or no influence on the underlying operating system that independently schedules the discs and manages the file system. Thus, for such new architectures, the known standard optimisation criteria that decide when index usage is preferable over relation scans (see e.g. Refs 10, 17 and 14), must be revised to suit the changed environment. However, it must be emphasised that the results developed in this paper should also be applicable to conventional DB architectures, which have an extent-based file organisation and sufficient control over the disc arm movements for optimised chained I/O.

The rest of this paper is organized as follows. In Section 2 those features from Ref. 12 are explained that are relevant for this paper. Section 3 presents our optimisation approach. In Subsection 3.1, disc access times for elementary and more complex disc operations are estimated from a couple of easy-to-get basic performance quantities. In Subsection 3.2, optimisation criteria are developed that allow us to recognise the thresholds separating profitable from non-profitable index usage. Subsection 3.3 establishes the connection between estimated hit ratios on the tuple level to those on the block level for disc access. Next, Section 4 reports results of our threshold model for two common discpack drives, the IBM/3330 and 3380. Finally, Section 5 states some conclusions about the usefulness of indexes and about the integration of our optimisation model in a query optimiser.

2. ACCESS PATH TYPES IN DATABASES WITH INTELLIGENT DISC SUBSYSTEMS

The tremendous technological progress on the hardware sector nowadays makes computer installations with

* Present address: MAD Intelligent Systems, Prinzregentenplatz 10, D-8000 München 80, West Germany.

intelligent subsystems not only available, but also economical. The following areas are of special interest for this paper.

(i) Fast magnetic discs with large capacity. Most certainly, conventional moving-head discs will remain the dominant medium for long-term storage of the physical DB.

(ii) Powerful micros with an excellent price/performance ratio: such micros can be employed for a functional upgrading of discs ('intelligent discs').

This trend of a strong technological advancement of conventional hardware formed the starting point for the gross design of a hardware architecture of future high-performance DB systems, as depicted in Fig. A1 (compare also Ref. 12). The **Intelligent Disc Subsystem** is responsible for accesses to data currently not stored in the DB cache. It comprises a set of conventional moving-head disc-pack drives, each of them being upgraded by a microprocessor μP.

Concerning the mapping of relations to files on disc, we assume an **extent-based file system**, which maps a relation on to physically adjacent cylinders on disc. The advantages of such an organisation are well known (see e.g. Ref. 19); it aims to reduce the number and distance of expensive disc arm moves. Because the storage density per track steadily increases, the new generation of modern discs shows two performance features: the access bottleneck for random disc accesses will become even more acute, while consecutively reading physically adjacent blocks, tracks and cylinders will get faster and faster. Therefore the implementation of an extent-based file system is highly desirable.

The **query class** Q_{restr} we want to optimise is the class of restriction queries on a single relation R and is specified by the following query schema in SQL-form (* denotes any attribute(s) of R):

$$Q_{\text{restr}} \equiv \text{SELECT } * \text{ FROM R WHERE } F^+ \text{ AND } F^-$$

F^+ denotes a restriction predicate on R, that can be processed by using existing indexes. In general, F^+ may involve several attributes of R and therefore several indexes can be engaged in the evaluation process. On the other hand, the residue F^- is a restriction on R that is evaluated by directly examining the respective tuples without index usage.

Now let us introduce the retrieval interface operations to our intelligent disc subsystem, which can be used to evaluate $F \equiv F^+ \wedge F^-$. We consider the following two **access path types**.

(AP1) *Exhaustive relation scan* of R against F.

(AP2) *Index processing of* F^+, producing a list of block pointers for tuples satisfying F^+. Subsequently, the extent where R is stored is sequentially scanned, but only those blocks are read that are mentioned in the pointer list. This is called a *selective relation scan*. Finally, tuples picked up by this selective scan are checked for F^-.

Index processing is supposed to be implemented as follows. An index page currently not available in the DB cache is fetched from disc with a random disc access. The two remaining interface operations, exhaustive and selective scan, deal with sets of tuples, stored in an extent. All relevant blocks of the extent are supposed to be accessed by requiring only a single, one-directional sweep of the disc arm over the extent.

The goal of this paper is to develop optimisation criteria that choose the cheaper of the above access-path types. Also criteria will be given that allow us to decide whether all applicable indexes for F^+ or only a subset should be used.

3. THE PERFORMANCE MODEL

Known optimisation techniques used in query optimisers only count the estimated number of disc accesses, pretending all are random.[17, 14] This is reasonable, if the disc arm is not dedicated to a specific DB task and therefore can be manoeuvred to unfavourable positions by concurrent I/O service requests. On the other hand, our intelligent discs are supposed to schedule their arms more efficiently to execute the exhaustive and selective scanning operations.

3.1. Modelling of DB-disc Accesses

Our analytical performance model of an intelligent DB disc will rely only on some few basic performance parameters of the standard disc-pack drives in use. These few basic data are often available from the vendors' advertisements or manuals of their disc products. The quantities of interest are as follows.

Basic performance parameters

Storage-capacity-related numbers
cap_tr: track capacity, [bytes]
tr_cyl: number of tracks per cylinder
cyl_dp: number of cylinders per disc-pack drive
blo_tr: number of blocks per track

Mechanical/electronic numbers
rot: rotation time, [msec]
start_h: start time to move the arm, [msec]
next_cyl: time to cross a cylinder, [msec]

Really, *blo_tr* is not a predefined number, instead it can be fixed for each installation by properly formatting the disc. Further, we use the following derived quantities:

cap_cyl = *cap_tr* · *tr_cyl*: cylinder capacity, [bytes]
blo_cyl = *blo_tr* · *tr_cyl*: number of blocks per cylinder

Our performance model will be constructed in a modular way. We shall start with modelling some elementary disc operations, based on the listed basic performance data. Then we shall develop cost equations for the more complex scanning operations on top of these performance estimates for elementary disc operations.

3.1.1. *Access Time Behaviour of Elementary Disc Operations*

As already mentioned, instead of counting disc accesses we must estimate disc access times. The access time behaviour of standard disc subsystems has been investigated in detail in past years, see e.g. Ref. 6 or Ref. 18. However, the given solutions are mostly not directly related to database processing or are too expensive to be used by an optimiser in a relational database system. Also, most existing disc performance models are tied to some special disc architectures, and an extrapolation of the performance trade-offs for future disc models is not

easy. Starting from our few basic performance parameters, we shall develop a disc performance model that assumes some idealisations, but it will model the incurred costs accurately enough for the intended purpose.

Arm positioning time **seek_cyl**, if c cylinders have to be crossed (as a slight idealisation, we assume that *seek_cyl*(c) is linear in c).

$$seek\_cyl(c) = \begin{cases} start\_h + c \cdot next\_cyl \ [msec], & \text{if } 0 < c \leqslant cyl\_dp \\ 0 \ [msec], & \text{if } c = 0 \end{cases} \qquad \text{(M1)}$$

Average seek time **avg_seek**. Let X_i be a random variable, characterising the number of cylinders to be crossed for the next seek, if the disc arm is currently positioned on cylinder i. Let j be the cylinder to be located by the next seek. Then the required arm movements are: j cylinders for $j \in \{1,\ldots,i-1\}$, $j-i$ cylinders for $j \in \{i+1,\ldots,c\}$. Under the assumption that the choice of j is equally likely from $\{1,\ldots,cyl\_dp\}$, the expectation $E(X_i)$ of X_i is as follows: ($c := cyl\_dp$)

$$E(X_i) = \frac{1}{c}\left(\sum_{j=1}^{i-1} j + \sum_{j=1}^{c-i} j\right) = \frac{1}{2c}((i-1)i + (c-i)(c-i+1))$$

Thus the average number of cylinders to be crossed amounts to:

$$\begin{aligned} \frac{1}{c}\sum_{i=1}^{c} E(X_i) &= \frac{1}{2c^2}\left(\sum_{i=1}^{c}(i-1)i + \sum_{i=1}^{c}(c-i)(c-i+1)\right) \\ &= \frac{1}{2c^2}\left(\sum_{i=1}^{c} i^2 - \sum_{i=1}^{c} i + \sum_{i=1}^{c-1} i^2 + \sum_{i=1}^{c-1} i\right) \\ &= \frac{1}{2c^2}(2\tfrac{1}{6}c(c+1)(2c+1) - c^2 - c) \\ &= \tfrac{1}{3}(c - \tfrac{1}{3}) \end{aligned}$$

This result yields: $avg\_seek = seek\_cyl(\frac{1}{3}(c-\frac{1}{3}))$.

Because in practice we always have $c > 100$, we give an approximation for *avg_seek*:

$$avg\_seek = seek\_cyl(\tfrac{1}{3} \cdot cyl\_dp) \ [msec] \qquad \text{(M2)}$$

Elapsed time **read_bl(i)**, required to read i arbitrarily dispersed blocks from one track (after arm has already been positioned on respective cylinder).

For the subsequent derivations we assume an interleaving factor of 1, i.e. several blocks on one track can be read out within a single revolution. Also it is assumed that any of those i relevant blocks can be accessed first. The model in Fig. A2 will be used, where the desired blocks are marked by 1 to i.

It can be assumed that the search starts with the read head positioned in the middle of a block. Thus, initially half a block must be skipped on the average. Now let $\bar{x}_i$ denote the average number of irrelevant blocks that must be skipped until the first relevant block is encountered. This number is identical to the average number of relevant blocks between block i and the start position (compare model in Fig. A2). Therefore we get:

$$read\_bl(i) = \left(\frac{1}{2} + (blo\_tr - \bar{x}_i)\right) \cdot \frac{rot}{blo\_tr}$$

Denoting the binomial coefficient by C_s^r and abbreviating b for *blo_cyl*, there are C_i^b possibilities to place the i relevant blocks on to the b available blocks. Therefore $\bar{x}_i$ computes as:

$$\bar{x}_i = \frac{1}{C_i^b}(0C_{i-1}^{b-1} + 1C_{i-1}^{b-2} + \ldots + xC_{i-1}^{b-1-x} + \ldots + (b-i)\,C_{i-1}^{i-1})$$

Further, it holds that:

$$\begin{aligned} \bar{x}_i &= \frac{1}{C_i^b}\sum_{x=0}^{b-i} xC_{i-1}^{b-1-x} \quad /* \ terms\ for\ x \geqslant b-i+1\ are\ 0\ */ \\ &= \frac{1}{C_i^b}\sum_{x=0}^{b-1} C_1^x C_{i-1}^{b-1-x} \quad /*,^{13}\ \text{pp. 58, eq. 24}\ */ \\ &= \frac{1}{C_i^b}C_{i+1}^b = \frac{b-i}{i+1} \quad (***) \end{aligned}$$

Thus we get: ($1 \leqslant i \leqslant blo\_tr$)

$$read\_bl(i) = \left(\frac{1}{2} + blo\_tr - \frac{blo\_tr - i}{i+1}\right) \cdot \frac{rot}{blo\_tr} \ [msec] \qquad \text{(M3)}$$

Elapsed time **read_tr** to read an entire track (after seek has been done).

Assuming again an interleaving factor of 1, *read_tr* is composed of:

time until first block begin rotates under the head:

$$\frac{rot}{2blo\_tr}$$

1 rotation to read the entire track

$$read\_tr = \left(1 + \frac{1}{2blo\_tr}\right) \cdot rot \ [msec] \qquad \text{(M4)}$$

Elapsed time **read_tr(j, i)** to read from j tracks on the same cylinder i blocks each (after seek completed).

This cost is composed as follows:

(a) Time *read_bl*(i) to read the i relevant blocks from the first of the j relevant tracks.

(b) Assuming that the time to switch from one active head to another can be neglected, the remaining $j-1$ tracks do not require the initial skipping of one half-block. Therefore for each of these $j-1$ tracks we get:

$$read\_tr(j,i) = read\_bl(i) + (j-1) \cdot (blo\_tr - \frac{blo\_tr - 1}{i+1}) \cdot \frac{rot}{blo\_tr}$$

Rewriting this equation, we get: ($1 \leqslant j \leqslant tr\_cyl$, $1 \leqslant i \leqslant blo\_tr$)

$$read\_tr(j,i) = j \cdot \left(blo\_tr - \frac{blo\_tr - i}{i+1}\right) \cdot \frac{rot}{blo\_tr} + \frac{rot}{2blo\_tr} \ [msec] \qquad \text{(M5)}$$

Elapsed time **read_cyl** to read an entire cylinder.
This cost consists of:
time until first block begin of track rotates under head:

$$\frac{rot}{2blo\_tr},$$

tr_cyl rotations to read *tr_cyl* tracks.

$$read\_cyl = \left(\frac{1}{2blo\_tr} + tr\_cyl\right) \cdot rot \quad [msec] \quad \text{(M6)}$$

Now we have all the elementary cost factors at our disposal that we need to model the more complex DB-disc operations. But let us shortly summarise some neat properties of the model developed so far.

Lemma 3.1. (special cases for elementary operations)
(a) $read\_bl(blo\_tr) = read\_tr$
(b) $read\_tr(1, i) = read\_bl(i)$
(c) $read\_tr(tr\_cyl, blo\_tr) = read\_cyl$
Proof: Obvious.

3.1.2. *Access Time Behaviour of Complex Disc Operations*

Based on the cost estimates for elementary disc operations, time estimates for random block access, exhaustive and selective extent scanning will be provided.

Random block access time **b_at** to read an arbitrary block from the DB-disc:

$$b\_at = avg\_seek + read\_bl(1) \quad [msec] \quad \text{(M7)}$$

Time expenditure **sel_at(n, m)** to selectively read *m* spreading over *n* consecutive cylinders. Assuming that all $n \cdot blo\_cyl$ blocks of the considered extent must be read out, the single cost constituents amount to:
Seek for first cylinder of this extent.
Read out this first cylinder.
For the remaining $n-1$ cylinders:
Position arm on next cylinder.
Read out this cylinder.
Because the disc arm can cross the whole extent without being dislocated by a concurrent request, we arrive at: $(1 \leqslant n \leqslant cyl\_dp)$

$$exh\_at(n) = avg\_seek + (n-1) \cdot seek\_cyl(1) + n \cdot read\_cyl \quad [msec] \quad \text{(M8)}$$

Time expenditure **sel_at(n, m)** to selectively read *m* blocks from an extent covering *n* cylinders. To compute this quantity $sel\_at(n, m)$, we must know how those *m* blocks are distributed on to the *n* cylinders in question. Again we assume that this distribution is random and uniformly distributed. Now let *k* denote the average number of cylinders (out of those *n* cylinders), that contain at least one from those *m* blocks. Then the wanted time $sel\_at(n, m)$ can be gained by summing up the following cost portions:
Seek for first of these *k* cylinders.
Read all relevant blocks from this cylinder.
For the remaining $k-1$ cylinders:
Position arm on next out of these cylinders.
Read all relevant blocks from the cylinder under consideration.
The exact solution for *k* uses the hypergeometrical distribution and is discussed in Ref. 20. (Yao considered the problem of random distribution of tuples on to blocks.) As it is expensive to evaluate that formula numerically, simpler approximate solutions are of interest. Such a formula, applying the binomial distribution, is mentioned in Ref. 7:

$$k = n\left(1 - \left(1 - \frac{1}{n}\right)^m\right)$$

In Ref. 20 the deviation of this approximation from its exact solution was analysed, reporting that, when those *m* pick-ups are made out of a collection of more than ten, the error can be neglected in practice. Note that this prerequisite is satisfied for our case, because a typical disc-pack drive comprises 10 to 20 disc surfaces and therefore $blo\_cyl > 10$ holds. Thus, for our application the Cardenas formula gives nearly exact results, which is essential because *k* contributes by far the largest cost fraction to $sel\_at(n, m)$. So we define:

$$k := \begin{cases} n\left(1 - \left(1 - \frac{1}{n}\right)^m\right), & \text{if } 1 \leqslant m < n \cdot blo\_cyl \\ n, & \text{if } m = n \cdot blo\_cyl \end{cases} \quad \text{(M9)}$$

In each of these *k* cylinders there is at least one out of our *m* blocks. To approximate the distribution of the remaining $m-k$ blocks on to the *k* cylinders, again (in the absence of any specific distribution knowledge) we choose the binomial distribution. This results in an average of $1+(m-k)/k = (m/k)$ blocks to be read from each of the *k* cylinders. Now, to determine the time required for selectively reading (m/k) blocks from a cylinder, we need to know how the (m/k) blocks are distributed on to the *tr_cyl* cylinders. This is achieved by applying the same model as we used for determining *k*.

Let *l* be the average number of tracks (in one of those *k* cylinders), that contain at least one of those (m/k) blocks already considered. Recalling that $blo\_cyl = blo\_tr \cdot tr\_cyl$, we get:

$$l := \begin{cases} tr\_cyl \cdot \left(1 - \left(1 - \frac{1}{tr\_cyl}\right)^{m/k}\right), & \text{if } \frac{m}{k} < blo\_cyl \\ tr\_cyl, & \text{if } \frac{m}{k} = blo\_cyl \end{cases} \quad \text{(M10)}$$

In analogy to the way the quantity (m/k) was determined, it is concluded that in each of these *l* tracks there are $[1+(1/l)(\{m/k\}-l) = (m/kl)]$ relevant blocks to be read. What finally remains to be figured out is the average time required to position the arm from one out of the *k* relevant cylinders to the next one. For this purpose we can apply the same model that was used to compute $read\_bl(i)$ and $read\_tr(j, i)$. Tailored to the new problem to be solved, this model is illustrated in Fig. A3, where we view a disc's cylinders in a circular way. According to the formula marked (***) when deriving $read\_bl(i)$, for $\bar{x}_i$ in Fig. A3 we have $\bar{x}_i = (n-k/k+1)$.

Whatever the distribution of the *k* cylinders, named $C_1, \ldots, C_k$, on to the entire *n* cylinders of the extent may be, on the average the number of cylinders from C_1 to C_k (including both) is:

$$n - 2\frac{n-k}{k+1} - 1$$

Therefore the average distance **avg_dist(k)** between any

two adjacent cylinders from the k cylinders is ($k > 1$ assumed):

$$avg\_dist(k) = \frac{1}{k-1}\left(n - 2\frac{n-k}{k+1} - 1\right) = \frac{n+1}{k+1}$$

$$avg\_dist(k) := \left\{ \begin{array}{ll} \frac{n+1}{k+1}, & \text{if } 1 < k \leqslant n \\ 0, & \text{if } k = 1 \end{array} \right\} \quad \text{(M11)}$$

Now we are prepared to give the formula for our desired $sel\_at(n,m)$ by summing up the respective cost constituents:

$$sel\_at(n,m) = avg\_seek + read\_tr\left(k, \frac{m}{kl}\right) + (k-1)\left(seek\_cyl(avg\_dist(k)) + read\_tr\left(l, \frac{m}{kl}\right)\right)$$

Thus we finally arrive at: ($1 \leqslant n \leqslant cyl\_dp$)

$$sel\_at(n,m) = \left\{ \begin{array}{ll} avg\_seek + & [msec], \text{ if } 0 < m \leqslant n \cdot blo\_cyl \\ (k-1) \cdot seek\_cyl(avg\_dist(k)) + & \\ k \cdot read\_tr\left(l, \frac{m}{k \cdot l}\right) & \\ 0 & [msec], \text{ if } m = 0 \end{array} \right\} \quad \text{(M12)}$$

Again, our model developed thus far has some neat features.

Lemma 3.2. (special cases for complex disc operations)

(a) $sel\_at(n,1) = b\_at$

(b) $sel\_at(n, n \cdot \text{blo\_cyl}) = \text{exh\_at(n)}$

Proof:

(a) $m = 1 \rightarrow k = n$

$$\left(1 - \left(1 - \frac{1}{n}\right)^1\right) = 1 \rightarrow \frac{m}{k} = 1 \rightarrow l = 1.$$

Thus

$$sel\_at(n,1) = avg\_seek + read\_tr(1,1) = avg\_seek + read\_bl(1) = b\_at.$$

(b) /* $bc := blo\_cyl$, $tc := tr\_cyl$ */

$$m = n \cdot bc \rightarrow k = n \rightarrow \frac{m}{k} = bc \rightarrow l = tc \rightarrow \frac{m}{k \cdot l} = \frac{n \cdot bc}{n \cdot tc} = blo\_tr.$$

Also it holds that: $k = n \rightarrow avg\_dist(k) = 1$.

Thus:

$$sel\_at(n, n \cdot bc) = avg\_seek + (n-1) \cdot seek\_cyl(1) + n \cdot read\_tr(tc, blo\_tr) = avg\_seek + (n-1) \cdot seek\_cyl(1) + n \cdot read\_cyl = exh\_at(n).$$

Lemma 3.3

(a) $sel\_at(n,m)$ is strictly monotonically increasing in m.

(b) $sel\_at(n,m) < exh\_at(n)$ for $0 \leqslant m < n \cdot blo\_cyl$.

While these propositions are intuitively clear, the proof is somewhat lengthy and can be found in Ref. 11.

3.2. Threshold Estimates for Index Usage

3.2.1. The Cost Model

To get a tractable cost model, the CPU-costs inherent in query processing will be neglected. This is a reasonable approach, whenever the bottleneck of query processing is the disc; i.e. if the DB-system is I/O-bound. The costs taken into account are defined as the time consumed by the complex DB-disc operations (b_at, $sel\_at(n,m)$, $exh\_at(n)$). The concrete cost functions for our two principal types of access paths to answer a restriction query will depend on the following **cost parameters**:

a: number of transported index pages

m: number of blocks containing at least one tuple satisfying F^+

n: number of cylinders occupied by relation R

Also, it is assumed that those n cylinders form an extent on an intelligent disc.

Cost of an exhaustive relation scan: ($1 \leqslant n \leqslant cyl\_dp$)

$$cost\,1(n) := exh\_at(n) \quad [msec]$$

Cost of index processing, followed by a selective relation scan: ($1 \leqslant n \leqslant cyl\_dp, 1 \leqslant m \leqslant n \cdot blo\_cyl$)

$$cost\,2(n,a,m) := a \cdot b\_at + sel\_at(n,m) \quad [msec]$$

Note: $cost\,2(n,a,m)$ is strictly increasing in a and m.

3.2.2. Selection of the Fastest Access Path Type

Now our next goal is to come up with a decision criterion that tells us which of two access path types is faster to process a given query with cost parameters a and m. Consider the case of equal costs $cost\,1(n) = cost\,2(n, a^*, m)$ for some fixed n and m. Evidently, a point a^* of equal cost is reached for

$$a^* = \frac{1}{b\_at}(exh\_at(n) - sel\_at(n,m)).$$

From lemma 3.2(b) and 3.3(b) $exh\_at(n) - sel\_at(n,m) \geqslant 0$ is known to hold. Consequently, for fixed n and m, a^* is uniquely defined and lies in the interval

$$\left[0, \frac{exh\_at(n)}{b\_at}\right].$$

Such an a^* is now called the **threshold point**. To separate profitable from unprofitable index usage, the above value for a^* guides us to define the **threshold function** $ta_n(m)$ **for index usage** as:

$$ta_n(m) := \frac{1}{b\_at}(exh\_at(n) - sel\_at(n,m)), \quad 0 \leqslant m \leqslant n \cdot blo\_cyl$$

Properties of $ta_n(m)$:

(1) For fixed n and m, $ta_n(m)$ determines the threshold value a^*, i.e. it holds that:

$$cost\,1(n) = cost\,2(n, ta_n(m), m)$$

(2) $ta_n(0) = \dfrac{exh\_at(n)}{b\_at} > 0$

(3) $ta_n(n \cdot blo\_cyl) = 0$

(4) $ta_n(m)$ is strictly decreasing in m.

Proof: straightforward.

Lemma 3.4 (optimisation criterion 1 for Q_{restr}-queries) Let relation R be stored in n cylinders, $\overline{m}$ be the estimated number of block hits containing tuples that satisfy the restriction F^+, and $\overline{a}$ be the estimated number of index pages required to evaluate F^+.

(a) $\overline{a} < ta_n(\overline{m}) \rightarrow$ index processing followed by a selective relation scan is faster;

(b) $\overline{a} > ta_n(\overline{m}) \rightarrow$ exhaustive relation scan is faster.

Proof: Let $a^* = ta_n(m)$.

(a) $\overline{a} < a^* \rightarrow cost\,2(n, \overline{a}, \overline{m}) < cost\,2(n, a^*, \overline{m})$, because $cost\,2$ strictly increases in a. Due to $cost\,2(n, a^*, \overline{m}) = cost\,1(n)$ proposition (a) is correct.

(b) Analogously.

Now an important special case of restriction queries will be investigated. We are interested in the case where the restriction predicate F is conjunctively decomposed as follows: $F \equiv F^+ \wedge F^-$, where $F^+ \equiv F_1^+ \wedge F_2^+$

SQL query scheme: $Q_{conj} \equiv$
SELECT * FROM R WHERE F_1^+ AND F_2^+ AND F^-

To evaluate this special query type, our architecture offers four basic access paths:

(AP1): exhaustive relation scan

($AP2_1$): index processing of F_1^+, followed by selective scanning for $F_2^+ \wedge F^-$

($AP2_2$): index processing of F_2^+, followed by selective scanning for $F_1^+ \wedge F^-$

($AP2_{1,2}$): index processing of $F_1^+ \wedge F_2^+$, followed by selective scanning for F^-

The costs for these access paths amount to:

$$cost\_AP1 := cost\,1(n), \quad cost\_AP2_1 := cost\,2(n, a_1, m_1), \quad cost\_AP2_2 := cost\,2(n, a_2, m_2).$$

Computation of $cost\_AP2_{1,2}$. The number of index pages to be accessed is $a_1 + a_2$. Assuming that F_1^+ and F_2^+ are statistically independent, the probability of one particular block containing a tuple satisfying $F_1^+ \wedge F_2^+$ is

$$\frac{m_1}{n \cdot blo\_cyl} \frac{m_2}{n \cdot blo\_cyl}.$$

So we get:

$$cost\_AP2_{1,2} = cost\,2\left(n, a_1 + a_2, \frac{m_1 \cdot m_2}{n \cdot blo\_cyl}\right)$$

Applying lemma 3.4, it turns out that:

(0) $a_1 < ta_n(m_1) \rightarrow AP2_1$ is faster than AP1.

(1) $a_2 < ta_n(m_2) \rightarrow AP2_2$ is faster than AP1.

(2) $a_1 + a_2 < ta_n\left(\dfrac{m_1 \cdot m_2}{n \cdot blo\_cyl}\right) \rightarrow AP2_{1,2}$ is faster than AP1.

The task of selecting the fastest access path poses new problems, if both (0) and (2) or both (1) and (2) hold. For these cases, the optimisation heuristics encountered in existing database systems mostly look like '*Use all available indexes*' or '*Use the most selective index*' (compare e.g. RDB/VI)[14]. For the following discussion, without loss of generality we assume that $m_1 \leqslant m_2$ and that $AP2_1$ is faster than $AP2_2$. Our optimisation criterion developed earlier cannot decide for the mentioned case which access path is the fastest, because $a_1 < a_1 + a_2$, but

$$m_1 \geqslant \frac{m_1 m_2}{n \cdot blo\_cyl}.$$

Thus the monotonicity of *cost 2* cannot be exploited in this situation. If (0) as well as (2) holds, we compare the respective cost differences to AP1:

$$\text{For (0)} \quad ta_n(m_1) - a_1 = \frac{1}{b\_at}(exh\_at(n) - sel\_at(n, m_1)) - a_1$$

$$\text{For (2)} \quad ta_n\left(\frac{m_1 m_2}{n \cdot blo\_cyl}\right) - (a_1 + a_2) = \frac{1}{b\_at}\left(exh\_at(n) - sel\_at\left(n, \frac{m_1 m_2}{n \cdot blo\_cyl}\right)\right) - (a_1 + a_2)$$

These two differences are identical, if:

$$a_2 = \frac{1}{b\_at}\left(sel\_at(n, m_1) - sel\_at\left(n, \frac{m_1 m_2}{n \cdot blo\_cyl}\right)\right)$$

Consequently, in case of a conjunctive query Q_{conj} we define the **threshold function** $t2a_{n,m1}(m)$ **for twofold index usage** as: $(m_1 \leqslant m \leqslant n \cdot blo\_cyl)$

$$t2a_{n,m1}(m) := \frac{1}{b\_at}\left(sel\_at(n, m_1) - sel\_at\left(n, \frac{m_1 m}{n \cdot blo\_cyl}\right)\right)$$

Properties of $t2a_{n,m1}(m)$:

(1) $cost\,2(n, a_1, m_1) = cost\,2\left(n, a_1 + t2a_{n,m1}(m), \dfrac{m_1 m}{n \cdot blo\_cyl}\right)$

(2) $t2a_{n,m1}(m_1) \geqslant 0$

(3) $t2a_{n,m1}(n \cdot blo\_cyl) = 0$

(4) $t2a_{n,m1}(m)$ is strictly decreasing in m.

Proof: straightforward.

Lemma 3.5 (optimisation criterion 2 for Q_{conj}) Let relation R be stored in an n-cylinder extent, F_1^+ and F_2^+ be restrictions on R with cost parameters a_1, m_1 and a_2, m_2, respectively, $m_1 \leqslant m_2$. Then the following holds:

(a) $a_2 < t2a_{n,m1}(m_2) \rightarrow AP2_{1,2}$ is faster than $AP2_1$

(b) $a_2 > t2a_{n,m1}(m_2) \rightarrow AP2_1$ is faster than $AP2_{1,2}$

Proof: Similar to lemma 3.4.

Note that, for Q_{conj}-queries, the general optimisation criterion 1 for Q_{restr}-queries can be applied to decide whether index usage pays at all.

3.2.3. Threshold Estimates

The threshold functions $ta_n(m)$ and $t2a_{n,m1}(m)$ can be used to optimise arbitrarily complex restriction queries Q_{restr} and Q_{conj}. The intention of this subsection is to develop an optimisation criterion that determines threshold selectivities in the following sense: if the hit ratio lies below the respective threshold, query processing by using a *single* index is not profitable over an exhaustive relation scan, and vice versa. For this purpose the following terminology is introduced. Let EP[r] be a

restriction on attribute r of R. EP[r] is called an *elementary restriction*, if EP[r] has the form $EP[r] \equiv const\,1 \leqslant r \leqslant const\,2$ or $EP[r] \equiv r = const$. (Selinger[17] calls such predicates 'SARgable'.) In the special case $F^+ \equiv EP[r]$, we define the query class $Q_{E\text{restr}}$ as follows:

$$Q_{E\text{restr}} \equiv \text{SELECT} * \text{FROM R WHERE } EP[r] \text{ AND } F^-$$

Then the following assumption can be stated, which is often satisfied in practice. Let the associated cost parameters for EP[r] be a and m, and let I_r denote an index on r. Then it is supposed to hold that:

Assumption A1

$a = a_{I_r}(m)$, where a_{I_r} denotes a monotonically increasing function in m.

Now observe that, in contrast to the general case covered by optimisation criterion 1 in lemma 3.4, for the special case $F^+ \equiv EP[r]$ we can get a threshold criterion that depends only on the cost parameter m.

We define the **threshold value** $thres_n$ **for using a single index** I_r as:

$$thres_n := \begin{cases} 0, & \text{if } a_{I_r}(0) > ta_n(0) \\ m^*\text{: } ta_n(m^*) = a_{I_r}(m^*), & \text{otherwise} \end{cases}$$

Note that the intersection point m^* is uniquely defined because of the monotonicity properties of ta_n and a_{I_r}. Moreover, it can be computed very efficiently by classical intersection algorithms known from numerical mathematics.

Lemma 3.6 (optimisation criterion 1′ for $Q_{E\text{restr}}$-queries, Let $\overline{m}$ be the number of blocks that must selectively be read after index processing of EP[r] using index I_r.

(a) $\overline{m} < thresh_n \rightarrow$ using I_r, followed by a selective relation scan is faster

(b) $\overline{m} > thres_n \rightarrow$ exhaustive relation scan is faster

Proof: Follows immediately from lemma 3.4 and the monotonicity assumption for $a = a_{I_r}(m)$.

By analogy with the previously defined Q_{conj}-queries, we are now interested in another special case, namely that F_1^+ and F_2^+ are elementary restrictions. This subclass of restriction queries is referred to as $Q_{E\text{conj}}$-queries.

$$Q_{E\text{conj}} \equiv \text{SELECT} * \text{FROM R WHERE } EP_1[r_1] \text{ AND } EP_2[r_2] \text{ AND } F^-$$

In general, an index I_{r1} on r_1 and an index I_{r2} on r_2 might incur different processing costs in terms of the number of index pages to be accessed. To get a similar simple decision criterion for $Q_{E\text{conj}}$ as for $Q_{E\text{restr}}$, we make an assumption that also often holds in practical applications (compare later formula (S4) for this):

Assumption A2

Index processing of I_{r1} and of I_{r2} is equally costly, if the same number of qualifying tuples are encountered, i.e. $a_{I_{r1}}(m) = a_{I_{r2}}(m)$.

Thus, we can omit the subscript and we will simply write $a(m)$ from here on.

Under assumptions A1 and A2, the following simplifications can be observed. Let m_1 and m_2 be the block hits for qualifying tuples, gathered by traversing I_{r1} for EP[r_1] and I_{r2} for EP[r_2], respectively. Then

$$m_1 \leqslant m_2 \rightarrow a_1 \leqslant a_2 \rightarrow sel\_at(n, a_1, m_1) \leqslant sel\_at(n, a_2, m_2)$$

This however means that $AP2_1$ has to be preferred over $AP2_2$. In this case we also get: $a_1 > t2a_{n,m1}(m_1) \rightarrow a_2 > t2a_{n,m1}(m_1)$.

Therefore, we define the **threshold value** $thresh\,2_{n,m1}$ **for twofold index usage** as:

$$thres\,2_{n,m1} := \begin{cases} m_1, & \text{if } t2a_{n,m1}(m_1) < a(m_1) \\ m^*\text{: } t2a_{n,m1}(m^*) = a(m^*), & \text{otherwise} \end{cases}$$

Because $t2a_{n,m1}(m)$ strictly decreases, $a(m)$ in turn strictly increases, the intersection point m^* exists and is uniquely determined. Again, it can be computed efficiently by numerical intersection algorithms.

Lemma 3.7 (optimisation criterion 2′ for $Q_{E\text{conj}}$-queries) Let m_1 (m_2) be the cost parameter, if EP[r_1] (EP[r_2]) is processed by an index $I_{r1}(I_{r2})$; $m_1 \leqslant m_2 \leqslant n \cdot blo\_cyl$. Then it holds that:

(a) $m_2 < thres\,2_{n,m1} \rightarrow AP2_{1,2}$ is faster than $AP2_1$

(b) $m_2 > thres\,2_{n,m1} \rightarrow AP2_1$ is faster than $AP2_{1,2}$

Proof: follows directly from lemma 3.5.

3.3. Hit Ratio Estimates

While this subsection does not propose any new methods for estimating expected hit ratios, we have to be concerned with this subject here in order to establish the link between the expected tuple hit ratio of a restriction query and the performance parameters a and m on block level, required by our cost model.

Let us consider an index I_r on an attribute r of relation R. I_r is supposed to be a dense ordered index like a B-tree, with chained leaf pages to accelerate the processing of elementary range restrictions (i.e. a B$^+$-tree). The leaf pages contain pointers to the pages where the respective tuples are stored on disc. The following statistics data are commonly available for a query optimiser:

* *ndkeys_r* := number of different r-values, currently stored in R
* *height_r* := height of I_r
* *nleaf_r* := number of leaf pages of I_r
* *card_R* := number of tuples of R

The **selectivity factor** $sf(F)$ of a restriction predicate F on R is defined as the relative tuple hit ratio. Estimates for the selectivity factor of an elementary restriction EP[r] $\equiv r = const$ or EP[r] $\equiv const\,1 \leqslant r \leqslant const\,2$ can be gained by applying known straightforward methods as described in Refs 17 and 14, or by employing more advanced techniques, see e.g. Refs 8 and 16. Either way, let us assume that we have some estimates for sf(EP[r]).

We define the **tuple hit rate** *t_hits* for an elementary restriction EP[r] as:

$$t\_hits := sf(\text{EP}[r]) \cdot card\_R \quad [\textit{number of tuples}]$$

Having available hit estimates for an EP[r], the next step is to get estimates for the crucial performance parameters m and a. As m denotes the average number of blocks

containing at least one tuple that satisfies EP[r], under the assumption of random distribution we can apply the Cardenas formula (used for derivation of $sel\_at(n,m)$ previously) again, which yields:

$$m = n \cdot blo\_cyl \cdot \left(1 - \left(1 - \frac{1}{blo\_cyl}\right)^{t\_hits}\right) \qquad \text{(S1)}$$

The number a of index pages to be visited can be estimated as follows:

number of non-leaf pages to be visited: $height\_r - 1$

number of leaf pages to be visited: $sf(\text{EP}[r]) \cdot nleaf\_r$

Therefore we can set:

$$a = sf(\text{EP}[r]) \cdot nleaf\_r + height\_r - 1 \qquad \text{(S2)}$$

What remains now is to establish the previous assumption A1, stating a functional relationship $a = a_{I_r}(m)$ such that $a_{I_r}(m)$ increases monotonically in m.

For this purpose, equation (S1) is solved for t_hits, yielding:

$$t\_hits = \log_{1-(1/n \cdot blo\_cyl)}\left(1 - \frac{m}{n \cdot blo\_cyl}\right) \qquad \text{(S3)}$$

As can be observed from (S3), t_hits is a function of m and therefore we will write $t\_hits(m)$ instead of t_hits in the sequel.

Because $t\_hits(m) = sf(\text{EP}[r]) \cdot card\_R$, from (S2) and (S3) we can conclude:

$$a = \frac{nleaf\_r}{card\_R} \cdot \log_{1-(1/n \cdot blo\_cyl)}\left(1 - \frac{m}{n \cdot blo\_cyl}\right) + height\_r - 1$$

Observing that ($card\_R/nleaf\_r$) represents the average number of tuple pointers that are stored in an index leaf page, we assign an extra name to this quantity:

$$lfout\_r := \frac{card\_R}{nleaf\_r} \quad /* \; leaf fanout \; */$$

Thus, finally we have:

$$a_{I_r}(m) := \frac{1}{lfout\_r} \cdot \log_{1-(1/n \cdot blo\_cyl)}\left(1 - \frac{m}{n \cdot blo\_cyl}\right) + height\_r - 1 \qquad \text{(S4)}$$

Because the base of the logarithm is smaller than 1, our stated assumption A1 holds here. (Note: the previously stated assumption A2 holds, if $lfout\_r_1 = lfout\_r_2$, i.e. for B$^+$-trees with equal height, if they have an equal number of leaves.)

4. EVALUATION OF THE PERFORMANCE MODEL

In this section, numerical results from two simulation series that evaluate our threshold functions will be presented. The **average blocking factor** bf_R is introduced; bf_R is defined as the average number of tuples of R that are stored in a block of R on disc. (This definition also accounts for variable-length tuples.)

4.1. Application of the Threshold Criteria

Simulation series 1 will aim to calculate threshold percentages for tuple hits. If those tuple hits exceed the thresholds, the usage of an index will no longer be profitable to process elementary restriction queries $Q_{E\text{restr}}$. To optimise this query schema, the threshold value $thres_n$ and lemma 3.6 must appropriately be applied as follows. Choosing m in eq. (S3) to be the threshold value $thres_n$, we can conclude that $[t\_hits(thres_n)/card\_R]$ gives us the relative threshold ratio for tuple hits, above which the usage of an index is no longer profitable (as opposed to an exhaustive relation scan). Because it holds that $card\_R = n \cdot blo\_cyl \cdot bf\_R$, the subsequent threshold function $\text{ST}_{lfout\_r}(\text{n})$ represents a **threshold percentage for tuple hits**. If this threshold is exceeded, an exhaustive relation scan has to be preferred over index processing.

$$ST_{lfout\_r}(n) := \frac{100}{n \cdot blo\_cyl \cdot bf\_R} \log_{1-(1/n \cdot blo\_cyl)}\left(1 - \frac{thres_n}{n \cdot blo\_cyl}\right) [tup\%] \qquad \text{(S5)}$$

The second simulation series to be presented will provide threshold selectivities for the case of elementary conjunctive queries $Q_{E\text{conj}}$. The relevant optimisation criterion is stated in lemma 3.7. To apply this criterion, we must assume that $lfout\_r_1 = lfout\_r_2$. Denoting the expected percentage of block hits owing to $\text{EP}_1[r_1]$ by $\overline{m}_1$ (equalling $[m_1/n \cdot blo\_cyl \cdot bf\_R]\,100$), the subsequent **threshold function** $ST2_{n,\,lfout}(\overline{m}_1)$ represents the threshold percentage for tuple hits owing to $\text{EP}_2[r_2]$ as follows. If this value lies above, then the usage of two indexes for EP[r_1] and EP[r_2] is slower compared to the usage of the index for EP[r_1] alone, and vice versa.

$$ST2_{n,\,lfout}(\overline{m}_1) := \frac{100}{n \cdot blo\_cyl \cdot bf\_R} \log_{1-(1/n \cdot blo\_cyl)}\left(1 - \frac{thres\,2_{n,\,m1}}{n \cdot blo\_cyl}\right) [tup\,2\%] \qquad \text{(S6)}$$

4.2. Simulation Results

Our threshold performance model will now be evaluated for two concrete standard disc-pack drives, namely the old product IBM/3330 and the modern IBM/3380. Both discs have roughly the same mechanics, however they differ enormously with respect to their storage density.

(To reduce seek times, the IBM/3380 is equipped with two read–write arms. This capability will be neglected for the subsequent simulations.) The few basic performance quantities required to drive the threshold simulations are listed in Table 1.

The remaining required parameters are initialised as follows:

$$height\_r = 3, bf\_R = 10$$

In both cases the block size can be viewed as identical, namely 2 Kbytes ($\lfloor cap\_tr/blocksize \rfloor = blo\_tr$). Thus the tuple size is assumed to be about $(2048/bf\_R) \approx 200$ bytes.

Table 1. Basic performance parameters of two IBM computers

3330 (*c.* 100 Mbyte storage capacity)	
cap_tr = 13030 bytes	*rot* = 16.7 msec
tr_cyl = 19	*start_h* = 10 msec
cyl_cp = 404	*next_cyl* = 0.1 msec
blo_tr = 6	
3380 (*c.* 730 Mbyte storage capacity)	
cap_tr = 47500 bytes	*rot* = 16.7 msec
tr_cyl = 19	*start_h* = 10 msec
cyl_cp = 808	*next_cyl* = 0.05 msec
blo_tr = 23	

Simulation series 1

The threshold function $ST_{lfout\_r}(n)$ is plotted for two different values of *lfout_r* in Fig. A4. On the *x*-axis, *n* gives the number of cylinders that are occupied by relation R. As an example, consider the case of $n = 10$. If the estimated tuple hit percentage for EP[*r*] is, say, 4%, index usage would be preferred for 3330-type discs; however, an exhaustive scan would be faster for 3380-type discs.

Simulation series 2

Here the threshold function $ST2_{n,\,lfout}(\overline{m}_1)$ is plotted for the 3380 with $height\_r_1 = height\_r_2 = 3$, $lfout := lfout\_r_1 = lfout\_r_2 = 100$ and varying values of *n*. The result is shown in Fig. A5. The curves are drawn only for the range of values for which index usage (in some form) is faster than an exhaustive relation scan. Below each curve, only the index on r_1 should be used; above each curve, both indexes should be used. Consider, for example, the case of $n = 10$ and let the estimated percentage of block hits for EP[r_1] be 1%. If the estimated tuple hit percentage for EP[r_2] is, say, 5%, the usage of both indexes on r_1 and r_2 is preferred. On the other hand, an estimate of 6% for EP[r_2] would predict that using only the index on r_1 is faster.

The results shown should be contrasted with those for conventional DBs like System R, where index usage for much higher tuple hit percentages is selected.[2] From these two sample series the following conclusions can be derived.

The maintenance of indexes in a DB architecture with intelligent disc subsystems will bring an overall performance gain only if the following three points are satisfied.

(a) An index I_r on an attribute r must be highly selective.
(b) The retrieval frequency on I_r must be high.
(c) The update frequency for I_r must be low.

On the whole, from Fig. A4 it can clearly be observed that the increased storage density of modern discs increases the danger of bottlenecks for random disc accesses. Therefore, compared to the utilisation of indexes in standard DB architectures, in the new architecture the profitable use of indexes for restriction query processing will be more limited. However, indexes on very selective attributes – for example, on key attributes – will also be mandatory for these architectures, if excellent performance is to be achieved. This can be figured out from the given curves by noticing that the expected percentage of tuple hits for a restriction of the form $EP[r] \equiv r = const$ on a key attribute *r*, being $100/card\_R$, lies considerably below the computed threshold values.

5. CONCLUSION

In this paper a modelling technique for disc access times for an environment like a set-oriented DB architecture with intelligent disc subsystems was developed. The key assumption of this model is an extent-based file organisation on disc, where a physical relation is mapped on to a physically contiguous file. Thus our model also works for those conventional DB systems that are implemented on top of an extent-based file system and whose operating system cleverly provides chained I/O. Based on this model, several threshold criteria were presented that allow us to decide whether index usage is profitable over an exhaustive relation scan and if so, how many indexes should be used. The strength of our method is that it requires only a couple of readily available basic performance parameters. Also, forecasts for the performance trade-offs between sequential and random disc access of future discs are possible by simply choosing scaled-up parameters.

The two concrete discs analysed revealed that, with modern disc technology, the bottleneck of random disc accesses gets even more acute. However, indexes on highly selective attributes will still be mandatory, if excellent retrieval performance is to be accomplished. The access-path selection criteria presented can be incorporated into a query optimiser in two versions: either the respective thresholds are computed at optimisation time, or – maybe preferably – the thresholds are pre-computed and kept in the system's catalogues.

REFERENCES

1. M. M. Astrahan *et al.*, System R: relational approach to database management. *ACM TODS* **1** (2), 97–137 (1976).
2. M. M. Astrahan and M. Schkolnik, Performance of the System R access path selection mechanism. *Proc. IFIP*, pp. 487–491 (1980).
3. J. Banerjee and D. K. Hsiao, Performance study of a database machine in supporting relational databases. *Proc. VLDB*, pp. 319–329 (1978).
4. H. Boral, D. J. DeWitt and W. K. Wilkinson, *Performance evaluation of associative disk design*. Computer-Science Department, University of Wisconsin-Madison (1981).
5. H. Boral and D. J. DeWitt, Database machines: an idea whose time has passed? A critique of the future of database machines. Third International Workshop of Database Machines, pp. 166–187 (1983).
6. J. P. Buzen, I/O subsystem architecture. *Proc. IEEE* **63** (6), 871–879 (1975).
7. A. F. Cardenas, Analysis and performance of inverted data base structures. *Comm. ACM* **18** (5), 253–263 (1975).
8. S. Christodoulakis, Implications of certain assumptions in database performance evaluation. *ACM TODS* **5** (2) (1984).
9. D. J. DeWitt, DIRECT – a multiprocessor organization for supporting relational data base management systems. *IEEE Trans. on Comp.*, pp. 395–406 (1979).
10. M. Jarke and J. Koch, Query optimization in database systems. *ACM Comp. Surveys* **16** (2), 111–152 (1984).

11. W. Kiessling, *Database systems for computers with intelligent subsystems: architecture, algorithms, optimization.* Technical Report TUM-I8307, Technical University of Munich (1983) (in German).
12. W. Kiessling, Tuneable dynamic filter algorithms for high performance database systems. *Proc. Int. Workshop on High-Level Comp. Arch.*, pp. 6.10–6.20 (1984).
13. D. E. Knuth, *The Art of Computer Programming*, vol. 1. Addison-Wesley, New York (1968).
14. A. Makinouchi *et al.* The optimization strategy for query evaluation in RDB/V1. *Proc. VLDB*, pp. 518–529 (1981).
15. E. A. Ozkarahan, S. A. Schuster and K. C. Sevcik, Performance evaluation of a relational associative processor, *ACM TODS*, 2pp. 175–195 (1977).
16. G. Piatetsky-Shapiro and C. Connell, Accurate estimation of the number of tuples satisfying a condition. *Proc. SIGMOD*, pp. 256–276 (1984).
17. P. G. Selinger *et al.*, Access path selection in a relational database management system. Proc. SIGMOD (1980).
18. A. J. Smith, Input/output optimization and disk architectures: a survey. *Performance Evaluation* **1**, 104–117 (1981).
19. M. Stonebraker, Operating system support for database management. *Comm. ACM* **24** (7), 412–418 (1981).
20. S. B. Yao, Approximating block accesses in database organizations. *Comm. ACM* **20** (4), 260–261 (1977).

APPENDIX

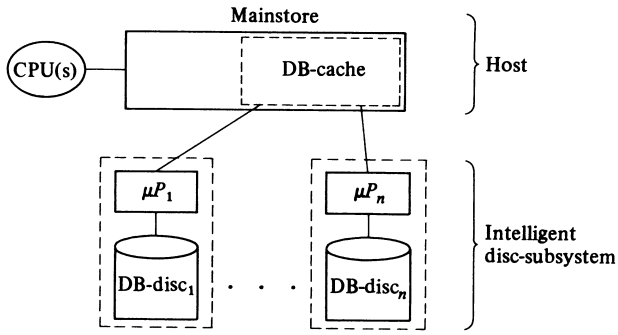

Fig. A1. Sample database architecture with intelligent disc subsystems.

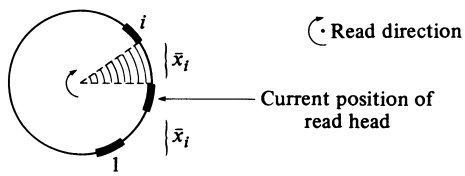

Fig. A2. Model to compute read_bl(i).

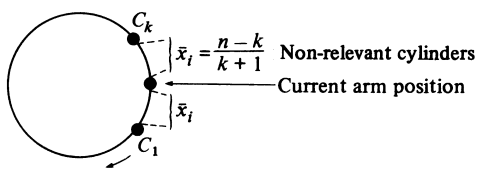

Fig. A3. Model to compute avg_dist(k).

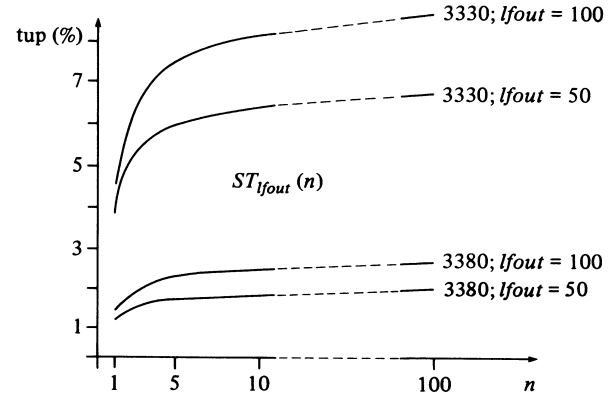

Fig. A4. Threshold values for single index usage vs. exhaustive relation scan.

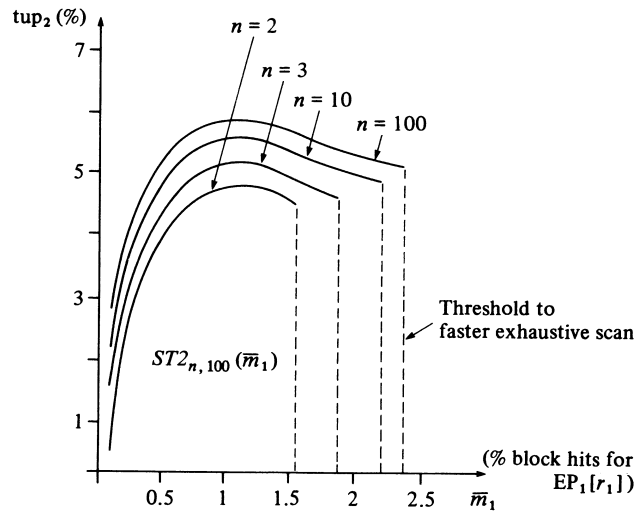

Fig. A5. Threshold values for usage of two indexes vs. one index.