Comput. & Ops. Res. Vol. 13, No. 4, pp. 489–505, 1986
Printed in Great Britain

0305–0548/86 $3.00 + 0.00
Pergamon Journals Ltd

PHYSICAL DESIGN FOR A RANDOM-ACCESS FILE WITH RANDOM INSERTIONS AND DELETIONS*

HAIM MENDELSON†
Graduate School of Management, University of Rochester, Rochester, NY 14627, U.S.A.

and

URI YECHIALI‡
Department of Statistics, Tel Aviv University, Tel Aviv 69978, Israel

(*Received November* 1984; *revised August* 1985)

Scope and Purpose—This paper addresses issues of physical design for a direct-access file which is subject to random record additions and deletions. We derive the performance measures of interest in explicit closed form, and apply them to solve a number of physical design problems. Such problems frequently arise in the design of various database and file systems.

Abstract—This paper studies in detail the performance of a database that is subject to random additions and deletions. The relevant performance measures are derived in explicit closed-form. These performance measures are applied to study the problem of optimal physical design.

1. INTRODUCTION

Most (if not all) commercial database systems possess the capability of dynamic record additions and deletions. These capabilities enhance the flexibility of use which is required by the database approach, and are necessary for transaction-processing systems. In contrast, most analyses of the performance of database systems consider only a growing database system and ignore the possibility of dynamic record deletions. Such analyses implicitly assume that deletions would be performed only at reorganization epochs (cf. Shneiderman [1], Mendelson and Yechiali [2] and Heyman [3]). Clearly, when records are added and deleted continuously over time, this gives rise to a misspecification of database behavior that could severely hamper the physical design process. The dearth of results on the behavior of data structures when deletions are dynamically mixed with insertions has been noted by Knuth [4]; some exceptions to this lack of results are due to Van der Pool [5], Jonassen and Knuth [6] and Mendelson [7].

Our objective is to provide an exact analysis of the performance of a direct-access file which is subject to random dynamic insertions and deletions. We consider a random-access file with an independent overflow area. When a record is added to the file, a hash-function assigns it to a bucket in the primary area. If the bucket so selected is full, the record is placed in the overflow area. In that event, a pointer in the prime area starts a list of all overflow records initially assigned to this bucket by the hash function. Each record resides in the system a random amount of time during which it generates retrieval requests. At the end of this residence time, the record is deleted.

*Partial financial support by the IBM Program of Support for Education in the Management of Information Systems is gratefully acknowledged.

†Haim Mendelson is Associate Professor of Computers and Information Systems and Business Administration in the Graduate School of Management, University of Rochester. His interests include the economics of information systems management, database management systems, applied economics and finance, computer-based trading systems, and stochastic modelling. He has published extensively in the leading journals in these areas. Mr Mendelson holds a B.Sc. from the Hebrew University of Jerusalem, and an M.Sc. and Ph.D. from Tel Aviv University. He serves as Computers and Information Systems Area Coordinator at the University of Rochester Graduate School of Management and is coordinating the School's activities under the IBM Program of Support for Education in the Management of Information Systems.

‡Uri Yechiali is a Professor of Operations Research and Statistics at the School of Mathematical Sciences of Tel Aviv University, where he served as Chairman of the Statistics Department from 1980 to 1983. Professor Yechiali received his B.S. (I.E.) and M.S. (O.R.) from the Technion, Israel, and a Dr. Eng. Sc. from Columbia University. He was a visiting professor at New York University and at Columbia University, and he is a consultant to the Israeli Ministry of Telecommunications. Dr Yechiali has published in numerous O.R. journals and his research interests are in queueing systems, teletraffic, reliability and stochastic models.

The pioneering work of Van der Pool [5] demonstrates the possible difficulties in analyzing this kind of system. Assuming exponential residence times, Van der Pool [5, p. 29] derives a set of recursive equations on his state probabilities which are based on a constant number of records on file. As Van der Pool recognizes, a constant number of records on file can hardly be justified in a model with additions and deletions. He proceeds from these equations to compute the stationary state distribution; since this cannot be done in closed form, he suggests a delicate (and quite complex) approximation procedure. These values are then used to compute the costs of storage and successful searches (his second performance measure is again heuristic and ignores the fluctuations in the number of records in the system).

In this paper we study a different model which allows us to obtain explicit closed-form results on the behavior of all the relevant performance measures without sacrificing generality. We demonstrate how maintaining sorted overflow chains (by key value) leads to reduced operating costs, and examine the magnitude of this reduction. We allow our system to immediately reuse the space released by a deleted prime record so as to conserve on subsequent operating costs. Our model admits general record residence times (as the assumption of exponentiality will not be valid in most systems) as well as transient time-dependent behavior. We analyze in detail and provide closed-form results for the behavior of storage costs, successful and unsuccessful search costs, record addition and deletion costs. These performance measures are studied both for the case of sorted overflow chains and for the case of unsorted overflow chains. Then we apply these results to study the problem of optimal physical design. We study, through examples, the case where the primary storage area is fixed (the "constrained" problem) and then the case where we can vary the primary storage area, at a cost. We find optimal values for the bucket size and number of buckets, and examine the sensitivity of our results to the design parameters.

Our model of the database system is presented in the following section. In Section 3 we study in detail the performance measures of the system. These measures are applied in Section 4 to optimize the physical design parameters of a specific system. Our concluding remarks are offered in Section 5.

2. THE SYSTEM

Consider a random-access file consisting of a fixed prime area and a variable overflow area. The prime area consists of N buckets numbered $1, 2, \ldots, N$. The capacity of each bucket is b (fixed-length) records. In addition, each bucket contains a pointer to the beginning of a list structure in the overflow area (initially, all list structures are null). Record arrivals to the system follow a Poisson process with rate Λ. Each record is identified by a kew w; keys of arriving records are independent, identically distributed (i.i.d.) random variables sampled from an arbitrary continuous distribution $H(\cdot)$.

When a record possessing key w is to be added to the system, it is assigned a hash address $h(w)$. The hash addresses of arriving records are i.i.d. random variables possessing a uniform distribution over the bucket numbers $\{1, 2, 3, \ldots, N\}$. Note that this implies that record arrivals to individual primary buckets follow independent Poisson processes with rates $\lambda \equiv \Lambda/N$. The bucket arrival rate, $\lambda = \Lambda/N$, is a function of the number of buckets in the system, N.

If bucket $h(w) = j$ is not full, the record is added to bucket j in the prime area. If bucket j is full, the new record is added to a *chain* of all overflow records whose initial hash address was j. The beginning of the overflow chain is pointed to by a pointer in bucket j. Chains are organized as either sorted or unsorted lists. If chains are unsorted, the new record will be added at the end of the chain. If chains are sorted, the records in the chain of bucket j are inspected, starting from the first record in the prime bucket, to find the first key w_i such that $w_i < w < w_{i+1}$ (if w is greater than all the keys in the chain, set $w_{i+1} = \infty$). Now, if $i < b$, the new record will be written in the prime bucket, and the last record in the bucket will be displaced to the overflow area; if $i \geqslant b$, the new record will be written in the overflow area. In the former case, the overflow pointer of the primary bucket will be updated to point to the displaced record; in the latter case, the pointer field of the record with key w_i (or the overflow pointer of the primary bucket if $i = b$) will point to the added record.

A record remains on file a random amount of time. Let V_k be the amount of time the kth record added to the system resides in the database until it is deleted; $V_1, V_2, \ldots, V_k, \ldots$ are i.i.d. random variables distributed as a random variable V, possessing distribution function $G(\cdot)$ and a finite mean $E[V] = 1/\mu$.

We assume that the overflow area is effectively infinite (i.e. the probability of overflow from the overflow area is negligible). There is a list of empty overflow slots; a pointer to the beginning of the list is maintained in main storage. Overflow empty slot management is LIFO. Deletion of a record is accomplished by reversing the steps taken when the record was added to the system. To delete a record residing in the overflow area, its slot is simply returned to the list of empty overflow slots and the pointer pointing to it is replaced by the value of the pointer of the deleted record. When a record is deleted from a primary bucket which possesses an overflow chain, the first record of the overflow chain is moved into the prime bucket, and its pointer becomes the pointer of the prime bucket. This is a low-cost procedure which prevents degradation of system performance over time.

Interestingly, our model can be directly related to the $M/G/\infty$ queueing model. Let $X_i(t)$ ($i = 1, 2, 3, \ldots, N; t \geqslant 0$) denote the number of records belonging to bucket i at time t, and let $\mathbf{X}(t) = (X_1(t), X_2(t), \ldots, X_N(t))$ ($\mathbf{X}(0) \equiv \mathbf{0}$). The above assumptions imply that the components of $\mathbf{X}(t)$ are independent. Furthermore, for each i, $\{X_i(t), t \geqslant 0\}$ is an $M/G/\infty$ queueing process with arrival rate λ and service time distribution $G(\cdot)$. It follows (cf. Ross [8, Section 2.3]) that

$$P\{X_i(t) = k\} \equiv q(k; t) = \exp[-\rho(t)]\,[\rho(t)]^k/k! \tag{1}$$

where

$$\rho(t) \equiv \lambda \int_0^t [1 - G(x)]\,\mathrm{d}x,$$

and

$$P\{X_i(\infty) = k\} \equiv q(k) = \exp(-\rho)\frac{\rho^k}{k!}, \tag{2}$$

where $\rho \equiv \rho(\infty) = \lambda/\mu$.

We also denote the right tails of the above distributions by

$$Q(k; t) = \sum_{j=k}^{\infty} q(j; t); \qquad Q(k) = \sum_{j=k}^{\infty} q(j). \tag{3}$$

In the next section we apply this model to analyze the performance of the system.

3. PERFORMANCE MEASURES

The costs of operating our file system include *access* costs (i.e. costs of input/output operations performed on the file system) and *storage* costs, both for the prime area and for the overflow area. The essence of the physical design problem is trading off these cost components. We let c_b denote the cost of reading or writing a primary record; c_0—the cost of reading or writing an overflow record; s_b—the storage cost per record in the prime area (per unit of time), and s_0—the storage cost per record in the overflow area. Since input/output operations involve only whole buckets, these cost parameters will depend on the bucket size. Thus, the above cost parameters will depend on the physical design parameters chosen for the system. For example, since the data-transfer rate is finite, c_b will be an increasing function of the bucket size b. The specific dependence of the cost parameters on the physical design parameters varies from installation to installation; we present the details for a typical installation in the next section.

In order to be able to analyze performance measures for the database system, we have to first specify the usage pattern of the database. Retrieval of records from the database can be classified into *successful* and *unsuccessful* searches: A successful search is a retrieval request for a record currently stored in the database; an unsuccessful search occurs when it is discovered that the requested record is not present in the database*. We associate each record present in the system with a random stream of

*Unsuccessful searches are frequent when the system is used to verify the nonexistence of a record in the database (e.g. checking the validity of a credit card number). Also, whenever a record is added to the database, an unsuccessful search has to be performed to verify that the added key is not already present in the system. See, e.g. Knuth [9], Mendelson and Yechiali [10] and Mendelson [11]).

retrieval requests to that record (clearly, all these retrieval requests will become successful searches). We assume that the streams of retrieval requests for records present in the database follow independent, homogeneous Poisson processes with rate R_s (per record). We also assume that unsuccessful search requests follow an independent Poisson stream with rate R_u. The remainder of the usage pattern involves additions and deletions which have already been specified in Section 2. Thus, we are now in a position to analyze the various performance measures of the database system.

3.1. Storage cost

The storage cost for the prime area is, obviously, $s_b \cdot Nb$. The size of the overflow varies over time; let $\theta_j(t)$ denote the number of overflow records in the chain of bucket j at time t. We have

$$\theta_j(t) = \begin{cases} 0 & \text{if } X_j(t) \leqslant b, \\ X_j(t) - b & \text{if } X_j(t) > b. \end{cases}$$

Thus, the expected length of the chain-overflow for any bucket j is given by

$$\begin{aligned} M(t) \equiv E[\theta_j(t)] &= \sum_{k=b+1}^{\infty} (k-b) P\{X_j(t) = k\}, \\ &= \rho(t) \cdot Q(b; t) - bQ(b+1; t). \end{aligned} \tag{4}$$

The expected overflow size is $N \cdot M(t)$, and the corresponding cost rate per unit of time is $Ns_0M(t)$.

3.2. Successful search cost

To evaluate the cost of a successful search, consider a retrieval request from a chain j when it contains n records. Each of the n records generates an expected number of R_s retrieval requests per unit of time. If $n \leqslant b$, each of these requests results in a single input/output operation (reading the corresponding bucket), hence the expected retrieval cost rate per unit of time is R_snc_b. If $n > b$, the expected cost rate per unit of time is given by

$$R_snc_b + R_snc_0\left[\frac{1}{n}(1+2+\ldots+(n-b))\right] = R_snc_b + \tfrac{1}{2}R_sc_0(n-b)(n-b+1).$$

Thus, the expected retrieval cost rate per unit of time is given by

$$NR_sc_b \cdot \rho(t) + \tfrac{1}{2}NR_sc_0E[\theta_j(t)(\theta_j(t)+1)].$$

Now,

$$\begin{aligned} E[\theta_j(t)\cdot(\theta_j(t)+1)] &= \sum_{j=b}^{\infty} (j-b)(j-b+1)q(j;t) \\ &= \left[\sum_{j=b}^{\infty} (j-b)^2 q(j;t)\right] + \left[\sum_{j=b}^{\infty} (j-b)q(j;t)\right] \\ &= [b^2Q(b;t) + \rho(t)(1-2b)Q(b-1;t) + \rho^2(t)Q(b-2;t)] + \\ &\qquad [\rho(t)Q(b-1;t) - bQ(b;t)] \\ &= b(b-1)Q(b;t) - 2\rho(t)(b-1)Q(b-1;t) + \rho^2(t)Q(b-2;t), \end{aligned} \tag{5}$$

where the sums $Q(k;t)$ are defined by equation (3). Finally, the expected retrieval cost rate is

$$NR_sc_b\rho(t) + \tfrac{1}{2}NR_sc_0[b(b-1)Q(b;t) - 2\rho(t)(b-1)Q(b-1;t) + \rho^2(t)Q(b-2;t)], \tag{6}$$

where we define $Q(k;t) = 1$ for all $k \leqslant 0$.

The first term of equation (6) represents the expected cost of reading the primary bucket, an operation that is performed once for each retrieval. The second term represents the expected added cost of search in the overflow chain. This term is $NR_s c_0$ times $\frac{1}{2}E[\theta_j(t)\cdot(\theta_j(t)+1)]$; the fact that the cost is *quadratic* in $\theta_j(t)$ represents a *clustering effect*: the expected cost of retrieval when overflow chains are long goes up for two reasons. First, the expected search along the chain is obviously longer; and second, the probability that the required record is stored in the overflow area (i.e. the probability that this search will be required) is also higher. The result is that when the expected number of records per bucket is significantly greater than b, we should expect an increasingly significant deterioration in the retrieval performance of the system. The implications of this clustering effect will be evident in the design problems studied in Section 4.

3.3. Unsuccessful search cost

The cost of unsuccessful searches depends on the organization of the overflow chains: when overflow chains are not sorted, it is necessary to search throughout the chain to discover that the requested key is not present in the database; in the case of sorted chains, it is sufficient to search up to the first key which is greater than the requested key. Thus, we have to distinguish between these two cases.

3.3.a. Unsorted chains. An unsuccessful search at time t involves reading the primary bucket (say bucket j), and then reading $\theta_j(t)$ overflow records. The expected cost per request is then $c_b + c_0 M(t)$. Since the arrival rate of unsuccessful search requests is R_u, the expected cost rate per unit of time is $R_u[c_b + c_0 M(t)]$.

3.3.b. Sorted chains. Let w be the key requested in the unsuccessful search request, and assume that its hash-address is $h(w) = j$ and $X_j(t) = n$. If $n \leqslant b$, the cost of search is only c_b. If $n > b$, the cost will be $c_b + i\cdot c_0$ if $w_{b+i-1} < w < w_{b+i}$ $(i = 1, 2, \ldots, n-b)$, where the keys in chain j are $w_1 < w_2 < \ldots < w_n$. If $w > w_n$, the cost is $c_b + (n-b)\cdot c_0$. Thus, given n $(n > b)$ and $h(w)$, the (conditional) expected cost is

$$c_b\cdot P\{w < w_b\} + \sum_{i=1}^{n-b} (c_b + i\cdot c_0)P\{w_{b+i-1} < w < w_{b+i}\} + [c_b + (n-b)\cdot c_0]\cdot P\{w > w_n\}$$

$$= c_b + c_0\cdot\left[\sum_{i=1}^{n-b+1} i\binom{n}{b+i-1}H(w)^{b+i-1}(1-H(w))^{n-(b+i-1)} - (H(w))^n\right].$$

But since the probability-integral transformed variate $H(w)$ is uniformly distributed over $[0, 1]$, we have

$$E_w\left[\binom{n}{b+i-1}H(w)^{b+i-1}(1-H(w))^{n-(b+i-1)}\right] = \binom{n}{b+i-1}\int_{u=0}^{1} u^{b+i-1}(1-u)^{n-(b+i-1)}\,du,$$

$$= \binom{n}{b+i-1}B(b+i, n-b-i+2) = \frac{1}{n+1},$$

where $B(\cdot,\cdot)$ is the beta function. It follows that the expected conditional cost per request (*given* n; $n > b$) is equal to

$$c_b + c_0\cdot\left[\left(\sum_{i=1}^{n-b+1} i\right) - 1\right]\Big/(n+1)$$

$$= c_b + c_0\cdot\frac{(n-b)(n-b+3)}{2(n+1)},$$

$$= c_b + \frac{c_0}{2}\cdot[n + 2(1-b) + (b^2 - b - 2)/(n+1)],$$

and the expected cost rate per unit of time is

$$R_u c_b + \frac{R_u c_0}{2}[\rho(t)Q(b; t) - 2(b-1)\cdot Q(b+1; t) + (b^2 - b - 2)\cdot Q(b+2; t)/\rho(t)],$$

since

$$\sum_{n=b+1}^{\infty} (n+1)^{-1} \exp[-\rho(t)]\,[\rho(t)]^n/n! = Q(b+2;t)/\rho(t).$$

3.4. Addition cost

In analyzing the cost of record addition, we apply the well-known fact that "Poisson arrivals see time averages" (cf. Heyman and Sobel [12, Section 11-2]), namely: the state of the system encountered by an arriving record at time t is distributed as $\mathbf{X}(t)$. Since the cost of record addition clearly depends on whether overflow chains are sorted or not, we study each of these cases separately.

3.4.a. Unsorted chains. Suppose that the record to be added to the database has key w with hash address $h(w)=j$. The addition then consists of

(1) an unsuccessful search along chain j;
(2) the actual addition, depending on
  (2a) if $X_j(t) < b$, the primary bucket is updated and rewritten (cost c_b);
  (2b) if $X_j(t) = b$, the new record is stored in the overflow area (cost c_0) and a pointer is added to the primary bucket, which has to be updated (cost c_b);
  (2c) if $X_j(t) > b$, the new record is stored in the overflow (cost c_0) and the previously last record is updated with a pointer (cost c_0).

Thus, the expected cost per request is the expected unsuccessful search cost, plus a term which can be written as

$$\begin{aligned} c_b\cdot P\{X_j(t)<b\} &+ (c_b+c_0)\cdot P\{X_j(t)=b\} + 2c_0\cdot P\{X_j(t)>b\} \\ &= c_b\cdot[1-Q(b+1;t)] + c_0\cdot q(b;t) + 2c_0\cdot Q(b+1;t), \\ &= c_b + (2c_0 - c_b)Q(b+1;t) + c_0 q(b;t). \end{aligned}$$

The expected (total) addition cost rate per unit of time is thus equal to

$$N\lambda[2c_b + c_0 M(t) + (2c_0 - c_b)Q(b+1;t) + c_0 q(b;t)].$$

3.4.b. Sorted chains. If the arriving record has key w with hash address $h(w)=j$, then the addition consists of

(1) An unsuccessful search along chain j;
(2) the actual addition, depending on
  (2a) if $X_j(t) < b$, the primary bucket is updated (cost c_b);
  (2b) otherwise, if $h(w)$ is to be inserted between the ith and the $(i+1)$st record of chain j, and $X_j(t) \geq b$, we must distinguish the following cases:
    (i) $i \leq b$: the primary bucket is updated (cost c_b) and an overflow record is written (cost c_0);
    (ii) $i > b$: the pointer of the ith record (an overflow record) is updated, and the new record is written in the overflow area (cost $2c_0$).

It follows that the expected conditional addition cost, given w and n, is the cost of an unsuccessful search, plus

$$L \equiv c_b\cdot I_{\{n<b\}} + I_{\{n=b\}}\cdot(c_0+c_b) + I_{\{n>b\}}\cdot P\{w<w_{b+1}\}\cdot(c_0+c_b) + I_{\{n>b\}}\cdot P\{w>w_{b+1}\}\cdot 2c_0,$$

where $I_{\{E\}}$ denotes the indicator of event E. Now,

$$P\{w_{b+1}<w\} = \sum_{i=b+1}^{n} \binom{n}{i} (H(w))^i (1-H(w))^{n-i},$$

hence

$$L = I_{\{n<b\}}\cdot c_b + I_{\{n=b\}}\cdot(c_0+c_b) + I_{\{n>b\}}\cdot(c_0+c_b)\cdot\sum_{i=0}^{b}\binom{n}{i}(H(w))^i(1-H(w))^{n-i}$$
$$+ I_{\{n>b\}}\cdot 2c_0\cdot\sum_{i=b+1}^{n}\binom{n}{i}(H(w))^i(1-H(w))^{n-i}.$$

Integrating over w and noting, as before, that $H(w) \sim U(0,1)$, we obtain

$$\begin{aligned} E_w[L] &= I_{\{n<b\}}\cdot c_b + I_{\{n=b\}}\cdot(c_0+c_b) + I_{\{n>b\}}\cdot(c_0+c_b)\\ &\quad\cdot\sum_{i=0}^{b}\binom{n}{i}\cdot B(i+1, n-i+1) + I_{\{n>b\}}\cdot 2c_0\\ &\quad\cdot\sum_{i=b+1}^{n}\binom{n}{i}B(i+1, n-i+1)\\ &= I_{\{n<b\}}\cdot c_b + I_{\{n=b\}}\cdot(c_0+c_b) + I_{\{n>b\}}\cdot(c_0+c_b)\\ &\quad\cdot(b+1)/(n+1) + I_{\{n>b\}}\cdot 2c_0\cdot(n-b)/(n+1). \end{aligned}$$

Summing over the possible values of n, we obtain

$$E[L] = [1-Q(b;t)]\cdot c_b + q(b;t)\cdot(c_0+c_b) + (c_0+c_b)\cdot(b+1)\cdot\sum_{n=b+1}^{\infty} q(n,t)/(n+1)$$
$$+ 2c_0\cdot\sum_{n=b+1}^{\infty} q(n;t)\cdot(n-b)/(n+1).$$

Noting that

$$\begin{aligned}\sum_{j=r}^{\infty}(j+1)^{-1}\exp(-\rho)\,\rho^j/j! &= \rho^{-1}\sum_{j=r}^{\infty}\exp(-\rho)\,\rho^{j+1}/(j+1)!\\ &= \rho^{-1}\sum_{j=r+1}^{\infty}\exp(-\rho)\,\rho^j/j!\end{aligned}$$

and that

$$(n-b)/(n+1) = 1-(b+1)/(n+1),$$

we finally obtain

$$\begin{aligned} E[L] &= [1-Q(b;t)]\cdot c_b + q(b;t)\cdot(c_0+c_b)\\ &\quad+ (c_0+c_b)\cdot(b+1)\cdot\rho(t)^{-1}\cdot Q(b+2;t) + 2c_0\cdot Q(b+1;t)\\ &\quad- 2c_0\cdot(b+1)\cdot\rho(t)^{-1}\cdot Q(b+2;t)\\ &= c_b\cdot[1-Q(b+1;t)] + c_0[q(b;t)+2Q(b+1;t)]\\ &\quad+ (b+1)\rho(t)^{-1}Q(b+2;t)(c_b-c_0). \end{aligned}$$

Finally, the expected (total) addition cost rate per unit of time when the chains are sorted is $E[L] + E[\text{cost of an unsuccessful search}] =$

$$= N\lambda c_b[2-Q(b+1;t)] + N\lambda c_0[(3-b)Q(b+1;t) + q(b;t)$$
$$+ \tfrac{1}{2}\rho(t)Q(b;t)] + N\lambda(b+1)\rho(t)^{-1}Q(b+2;t)\cdot[c_b + \tfrac{1}{2}c_0(b-4)].$$

3.5. *Deletion cost*

Deletion of a record consists of a successful search for the record to be deleted, followed by the actual deletion. Although it will turn out that we can use the results in Subsection 3.2 in the evaluation of the expected cost of the required successful search, this is not automatically true since the distribution of the system state upon deletion may be different from that of the system state at an arbitrary instant. In fact, we have to carefully evaluate a probability of the form

$$P\{X_j(t-0)=n \mid t \text{ is a deletion epoch}\}.$$

To evaluate this probability, consider the situation where $X_j(t)=n$, and number the n records in bucket j by a random permutation of the numbers $(1,2,3,\ldots,n)$. Then, let $\boldsymbol{\eta}(t)=(\eta_1(t),\eta_2(t),\ldots,\eta_n(t))$, where $\eta_i(t)$ is the residual time the ith record (using this numbering) has left to stay in the system (that is, $t+\eta_i(t)$ is the deletion epoch of the record numbered by i). Following the usual conditioning procedure of the $M/G/\infty$ queueing system, we obtain

$$P\{X_j(t)=n,\ \boldsymbol{\eta}(t)\leqslant\boldsymbol{\eta}\}=q(n;t)\cdot\prod_{i=1}^{n}\frac{\int_0^t[G(z+\eta_i)-G(z)]\,\mathrm{d}z}{\int_0^t[1-G(z)]\,\mathrm{d}z}.$$

This implies

$$P\{X_j(t)=n,\ \text{a record is deleted in } (t,t+\Delta t)\},$$

$$=q(n;t)\cdot\Delta t\cdot\frac{nG(t)}{\int_0^t[1-G(z)]\,\mathrm{d}z}+o(\Delta t),$$

where $o(\Delta t)$ is a function satisfying $\lim_{\Delta t\to 0}o(\Delta t)/\Delta t=0$. Using Bayes' rule, we obtain for $n=1,2,3,\ldots$

$$P\{X_j(t-0)=n \mid t \text{ is a deletion epoch}\}=\frac{n\cdot q(n;t)}{\sum_{k=1}^{\infty}k\cdot q(k;t)}$$

$$=n\cdot q(n;t)/\rho(t).$$

In the case of sorted overflow chains, the location of the deleted record is independent of its insertion time (and hence also of the deletion time). Then, given $X_j(t-0)=n$, the expected cost of search for the record to be deleted is

$$c_b+c_0I_{\{n>b\}}\cdot\frac{1}{n}[1+2+\ldots+(n-b)]=c_b+\tfrac{1}{2}c_0\cdot I_{\{n>b\}}(n-b)(n-b+1)/n, \qquad (7)$$

while the expected cost of the actual deletion is

$$I_{\{n\leqslant b\}}\cdot c_b+I_{\{n>b\}}\cdot\left[\frac{b}{n}\cdot(c_0+c_b)+\frac{n-b}{n}\cdot c_0\right]=c_b+I_{\{n>b\}}\cdot(c_0-c_b)+b/n\cdot I_{\{n>b\}}\cdot c_b \qquad (8)$$

(recall that when a record is deleted from a prime bucket possessing an overflow chain, the first record of the overflow chain is moved into the prime bucket). The expected value of equation (7) is given by

$$c_b+\tfrac{1}{2}c_0\rho(t)^{-1}\sum_{n=b+1}^{\infty}(n-b)(n-b+1)q(n;t)$$

$$=c_b+\tfrac{1}{2}c_0\rho(t)^{-1}E[\theta_j(t)(\theta_j(t)+1)]$$

$$=c_b+\tfrac{1}{2}c_0\rho(t)^{-1}b(b-1)Q(b;t)-c_0(b-1)Q(b-1;t)$$

$$+\tfrac{1}{2}c_0\rho(t)Q(b-2;t),$$

while the expected cost of the actual deletion is

$$c_b + (c_0 - c_b)\cdot\rho(t)^{-1}\cdot\sum_{n=b+1}^{\infty}[b + (n - b)]\cdot q(n;t) + c_b\cdot\rho(t)^{-1}\cdot\sum_{n=b+1}^{\infty} b\cdot q(n;t)$$
$$= c_b + c_0 b\cdot\rho(t)^{-1}\cdot Q(b+1;t) + (c_0 - c_b)\cdot\rho(t)^{-1}\cdot M(t).$$

It follows that as $t \to \infty$, the expected deletion cost rate per unit of time is given by

$$\begin{aligned} &N\lambda[2c_0 + c_0 b\rho^{-1}Q(b+1) + \tfrac{1}{2}c_0\rho^{-1}b(b-1)Q(b) - c_0(b-1)Q(b-1) \\ &\qquad + \tfrac{1}{2}c_0\rho Q(b-2) + (c_0 - c_b)Q(b) - (c_0 - c_b)\rho^{-1}bQ(b+1)] \\ &= N\lambda[2c_b + c_b b\rho^{-1}Q(b+1) + \tfrac{1}{2}c_0\rho^{-1}b(b-1)Q(b) \\ &\qquad - c_0(b-1)Q(b-1) + \tfrac{1}{2}c_0\rho Q(b-2) + (c_0 - c_b)Q(b)]. \end{aligned}$$

When overflow chains are unsorted, the location of a record in the chain will depend on its insertion time, and hence equation (7) may cease to hold. However, equations (7) and (8) will still hold if the distribution of V is exponential.

4. OPTIMAL DESIGN: NUMERICAL RESULTS

Consider a file system where $\Lambda = 1000$ records are added to the system per day, and a record remains on file for $1/\mu = 200$ days on average before it is deleted. The expected long-run number of records in the system is $\Lambda/\mu = 200{,}000$, a medium file size for commercial applications. Let the record size be r bytes; r is an application-dependent parameter. Consider a file residing on an IBM 3380 disk, which is widely used in IBM installations; other contemporary disks have similar performance characteristics, and hence this example is quite representative. The IBM 3380 has an average seek time of 16 ms, an average rotational delay of 8.3 ms and a transfer rate of 3 megabytes per second. This gives rise to an expected time of $24.3 + 1/3 \times 10^{-3}rb$ ms for directly reading a block of b records, where the first term represents the expected delay which is independent of block size and the second term represents the finite data-transfer rate. The actual dollar cost per input/output operation depends on installation and application-dependent parameters; using the disk input/output time charging rates for average commercial jobs at the University of Rochester's IBM 3032, this gives rise to a cost of

$$c_b = 2.4 \times 10^{-3} + 3.3 \times 10^{-8}rb \tag{9}$$

dollars per I/O operation. Clearly, for a different device or installation, the constants in equation (9) will be different, but their derivation and the general pattern of behavior will be the same. A commercial installation will contain a large number of systems with different characteristics, but the orders of magnitude selected here are quite typical.

Direct-access storage costs at the same installation are 2 cents per track per day, where a track of the 3380 can store up to 47,476 bytes. Note, however, that the cost per byte depends on the block size, due to three effects: (i) there is an unused interblock gap of 480 bytes between consecutive blocks; (ii) under most file organizations there must be an integer number of blocks per track; and (iii) block sizes are adjusted upwards to the nearest multiple of 32 bytes. The result is that if the block size is $r\cdot b$, and we denote by $\lceil\cdot\rceil$ the ceiling operator (rounding to the nearest integer upwards), the number of blocks per track is $n = [1499/\lceil rb/32\rceil + 15)]$, and the storage cost per record per day is $0.02/(nb)$. The integrality requirement on n gives rise to the existence of a sequence of "recommended" block sizes that efficiently utilize the track space; for the 3380 this sequence includes 47,476 bytes for 1 block per track, 23,476 bytes for 2 blocks per track, 15,476 bytes for 3 blocks per track, 9076 bytes for 4 blocks per track, and so on. As the block size increases between these points, storge utilization goes up since the number of blocks per track remains constant. Thus, a track will contain 3 blocks of 9076 bytes each and also 3 blocks of 15,476 bytes each, making the latter block size much more efficient. A second effect is the increase in storage utilization along this sequence, resulting from the fact that as the number of blocks n goes up, less space is wasted on interblock gaps.

Next consider the costs associated with storage and retrieval in the overflow area. Usually there are significant advantages to using the same block size in the prime area and in the overflow area, since this makes maintenance more uniform and consequently less costly. We shall assume that this is indeed the case, and that the two areas possess the same characteristics (the modifications required when area characteristics are different are straightforward). Finally, we shall assume in our numerical examples that 10% of the records on file are retrieved per day, namely: $R_s = 0.1$; we also let $R_u = 0$ (this can easily be modified without significantly affecting the results).

Given the above specifications, the physical design of the system involves two basic decision variables: the number of buckets in the primary area, N, and the number of records per bucket (or blocking factor), b. We shall consider the long-run behavior of the system, and distinguish between two problems: the *constrained* design problem, where the overall number of records in the primary area, $N \cdot b$, is predetermined, and the problem is to determine the optimal bucket size (or the number of buckets, N); and the *unconstrained* problem, where both N and b are to be determined. It is convenient to represent the unconstrained problem as the problem of finding the minimum-cost solution among all constrained problems, where we are relaxing the value of the constraint. Thus, we wish to examine the behavior of the expected overall operating and storage cost per day as a function of the blocking factor b, where $N \cdot b = K$, a constant; then we shall examine the behavior of the cost function when we allow K to vary. We shall also examine quantitatively the impact of sorting overflow chains on the solutions. This will help determine conditions under which the reduction in operating costs due to sorting overflow chains justifies the added complexity of operation.

4.1. The constrained design problem

Here we are interested in the behavior of the expected overall cost per day as a function of the bucket size b as we keep the capacity of the primary area, $N \cdot b = K$, constant. Since the expected number of records on file in the long run is $\rho = 200{,}000$, we examine three representative values of K: (i) $K = 150{,}000$, where the primary area can store up to 75% of the file, and a significant portion of the records must reside on the overflow area; (ii) $K = 200{,}000$, where the capacity of the primary area is equal to the expected number of records in the system (but, of course, the overflow area will still be significant); (iii) $K = 300{,}000$, where the capacity of the prime area is 50% above the expected number of records and the overflow size should be negligible. The results for record size $r = 200$ are shown in Fig. 1 (additional results are presented in subsequent figures).

First consider the case of a significantly constrained prime area, $K = 150{,}000$. It is evident that a low blocking factor ($b = 4$) is optimal here, a result which is contrary to the usual suggestion of a fairly high blocking factor (in particular, when—as in this case—logical records are short with respect to the track size). In fact, the expected cost per day is \$165.28 for $b = 1$, \$138.71 for $b = 2$, \$131.76 for $b = 3$, \$130.30 per day for $b = 4$ (the optimum), and \$131.37 per day for $b = 5$; then there is a fast increase in the expected cost up to \$1043.43 per day for $b = 237$, which is the maximum possible blocking factor for this record size (in Fig. 1 we show only a small portion of this range).

To explain this counterintuitive result, we decompose the total daily cost corresponding to $b = 237$ and discover that most of the cost is attributable to successful retrievals (\$919.73 per day). The reason that large values of b are associated with high retrieval cost is that when $Nb = K$, large b implies low N; thus, records are distributed among a smaller number of chains. But since the space available is $K = 150{,}000$, which is less than the expected number of records, $\rho = 200{,}000$, a significant number of retrievals will require search in the overflow area; furthermore, due to the clustering effect discussed in Section 3.2, the expected cost of overflow search increases quickly for long overflow chains. When N is low, records are distributed among less overflow chains, and hence there is a higher probability that some overflow chain(s) will turn out to be long (although the capacity of the prime area remains the same). A second effect, which is less significant here but will become important for larger values of K, is the increase in the data-transfer cost for large b: we must read all b records in the block in order to get only one of them.

Figure 1 also demonstrates the behavior of the cost as a function of b when the capacity of the prime area is equal to the expected number of records in the system ($K = 200{,}000$) and when it is 50% higher ($K = 300{,}000$). In both cases, the cost function has a minimum at a moderate value of b ($b = 24$ in both

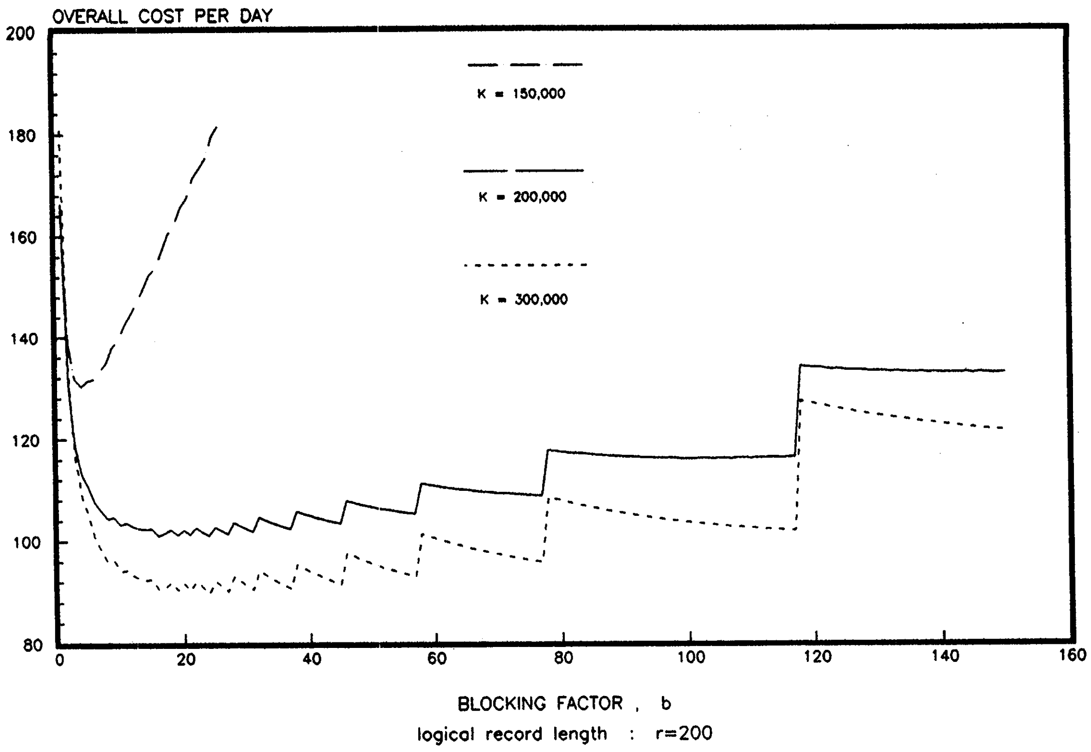

Fig. 1. Overall cost (in dollars per day) as a function of the blocking factor b for the constrained design problem. The upper curve corresponds to $K = 150{,}000$, the middle one to $K = 200{,}000$, and the lower curve to $K = 300{,}000$ records in the primary area.

cases; the expected daily cost is \$100.99 for $K = 200{,}000$ and \$89.81 for $K = 300{,}000$). As before, high values of b are undesirable. The effects of disk space utilization become prominent in these two cases, since now the storage cost becomes significant while the input/output costs are reduced. Thus, for example, the sharp local minima of both curves at $b = 117$ result from the fact that when the block size is $117 \times 200 = 23{,}400$, two blocks can fit on one track, while for $b = 118$ the block size becomes 23,600 for which only one block can fit on a track of the 3380, and the rest of the track remains unused. Consequently, the storage cost goes up steeply. Similar local minima can be found on both curves at $b = 37, 45, 57$ and 77, corresponding to $n = 6, 5, 4$ and 3 blocks per track, respectively. (Note that in the examples studied, the cost goes down as K increases. However, in Section 4.3 we shall study the cost as a function of K in some detail, showing that there is a point $K_{\min}$ where the cost starts increasing.)

4.2. Does sorting overflow chains matter?

In the following three figures we examine the constrained problem studied in the previous subsection, while assessing the improvement brought about by sorting overflow chains. These figures will also serve to study the effects of increasing the logical record length from $r = 200$ (as in Fig. 1) to $r = 500$. Figure 2 depicts the expected cost per day for $K = 150{,}000$ records as a function of the bucket size b, for both sorted and unsorted overflow chains, when the logical record size is $r = 500$. First we observe that as in the case where $r = 200$, the minimum cost blocking factor b is fairly low, at $b = 4$; the corresponding minimum daily cost is \$159.27. Since the bucket size is larger, we can readily observe the points that lead to relatively efficient disk space utilization; as in the previous figure with $K = 150{,}000$, these are not points of local minimum, and the general pattern of costs increasing with bucket size (beyond the minimum point, i.e. for $b > 4$) dominates. The explanation is similar to that of Fig. 1: due to the clustering effect, large values of b are associated with low values of N; then there is a higher probability of having some long overflow chains that bring about a significant cost increase.

Examining the savings resulting from sorting overflow chains, we note that they are significant for large b but relatively small at the optimal value of b. Thus, at $b = 4$ (the constrained optimum), the

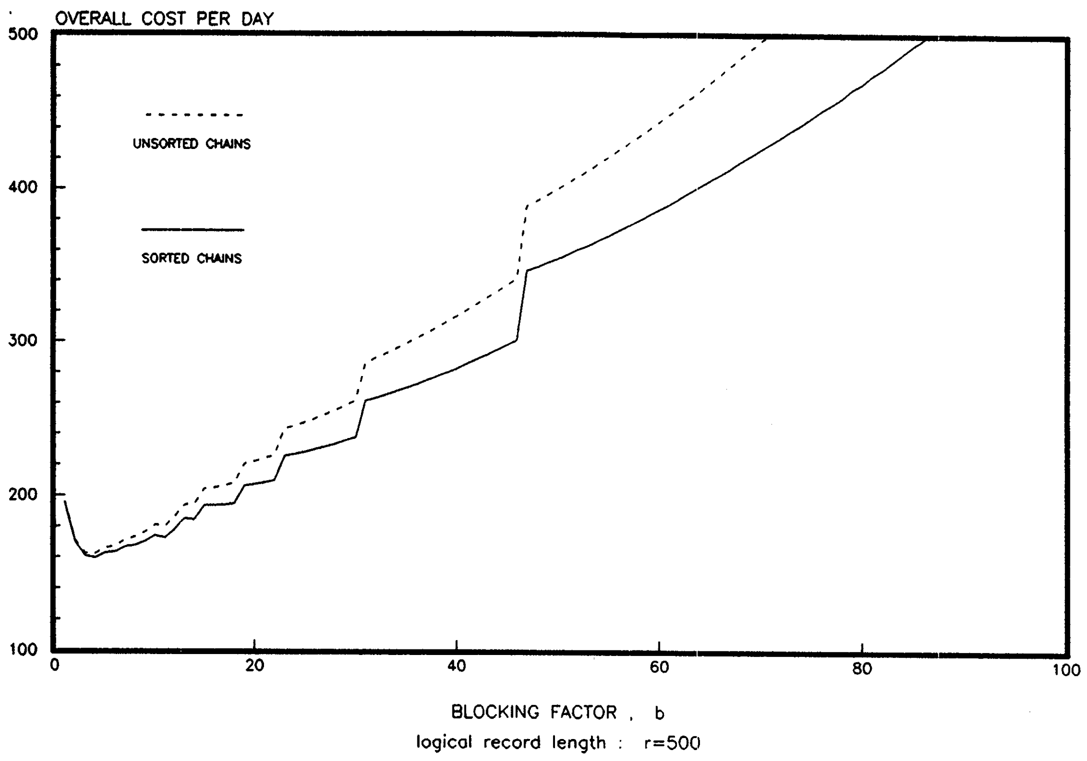

Fig. 2. Overall cost (in dollars per day) for sorted and unsorted chains as a function of the blocking factor b for the constrained design problem. The graphs correspond to $K = 150{,}000$ records in the primary area. The upper curve depicts the cost function for unsorted chains, and the lower depicts the cost function for sorted chains.

savings due to sorting the overflow chains are \$2.58 per day ($<1.6\%$) whereas for $b = 94$, they increase to more than \$106 per day (which amount to more than 16% of the overall cost). Figure 3 depicts the corresponding results with $K = 200{,}000$ records. Here, the minimum cost is \$133.95 per day, at $b = 11$; the corresponding cost for unsorted overflow chains is \$136.59. As before, the cost differential between sorted and unsorted chains increases with b, although the difference for most values of b is not as high as in the case where $K = 150{,}000$. Finally, Fig. 4 illustrates the same comparison when $K = 300{,}000$. Note that here the cost function for sorted chains is almost identical with that of unsorted chains, and in fact it is difficult to distinguish between them in the figure. Indeed, the difference between the two is of the order of a couple of cents per day. Intuitively, this results from the fact that when $K = 300{,}000$, the expected length of overflow chains is close to zero, and consequently sorting these chains is of little impact on the expected cost function. Thus, when we can generously allocate space for the prime area, it may not be desirable to add to the complexity of the system by maintaining sorted overflow chains.

4.3. The unconstrained design problem

To solve the unconstrained design problem, we vary the storage capacity of the prime area, K, and search for the value of K which minimizes our cost function, where the optimum *given* K is obtained by searching over the possible values of b. In Fig. 5 we demonstrate the behavior of the overall cost (at optimal b) as a function of K, when $r = 200$, and in Fig. 6 we depict the optimal blocking factor b as a function of K for the same value of r. Due to the importance of this problem, we repeat the process for $r = 500$; the results are shown in Figs 7 (cost as a function of K) and 8 (b as a function of K).

The general pattern of behavior is as follows. There exists an optimal value of K, $K_{\min}$, such that for $K < K_{\min}$, the cost function decreases with K, and the optimal bucket size b increases with K; there is a fairly flat minimum of the cost function at $K = K_{\min}$, and the optimal b remains constant in the neighborhood of $K_{\min}$ (b maintains a value that corresponds to an efficient disk-space utilization point). For $K < K_{\min}$, the dominant cost factor is the cost of input/output; as K goes up, the overflow area becomes smaller and the expected input/output cost goes down. This is, of course, accompanied

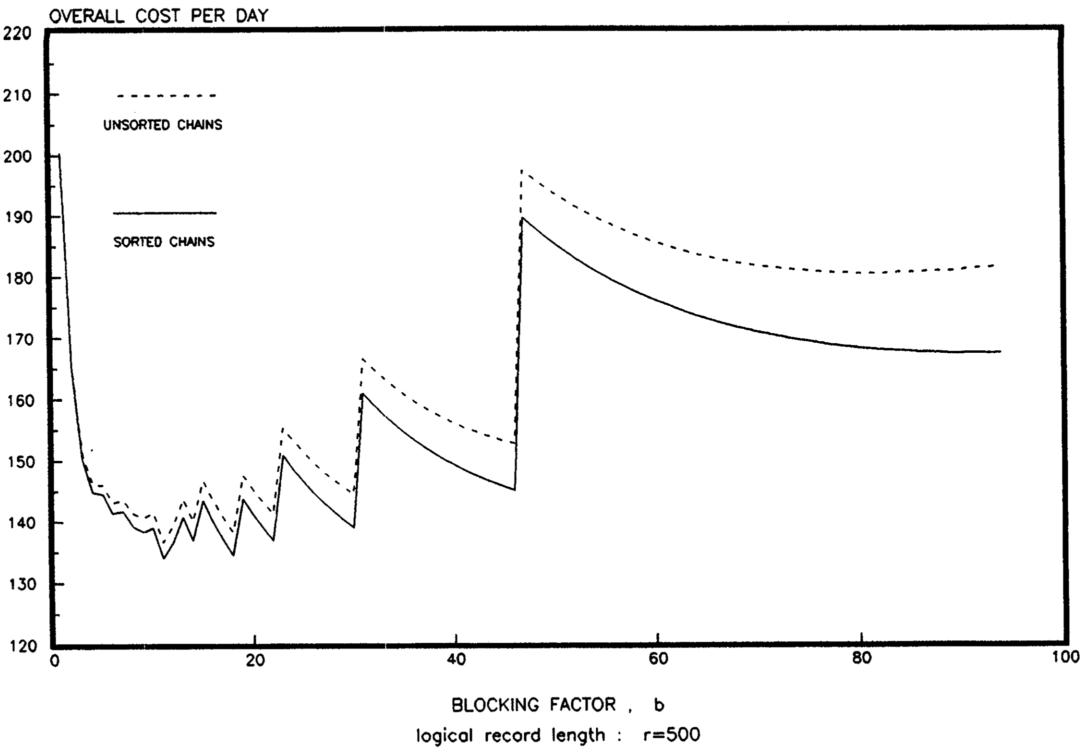

Fig. 3. Overall cost (in dollars per day) for sorted and unsorted chains as a function of the blocking factor b for the constrained design problem. The graphs correspond to $K = 200{,}000$ records in the primary area. The upper curve depicts the cost function for unsorted chains, and the lower depicts the cost function for sorted chains.

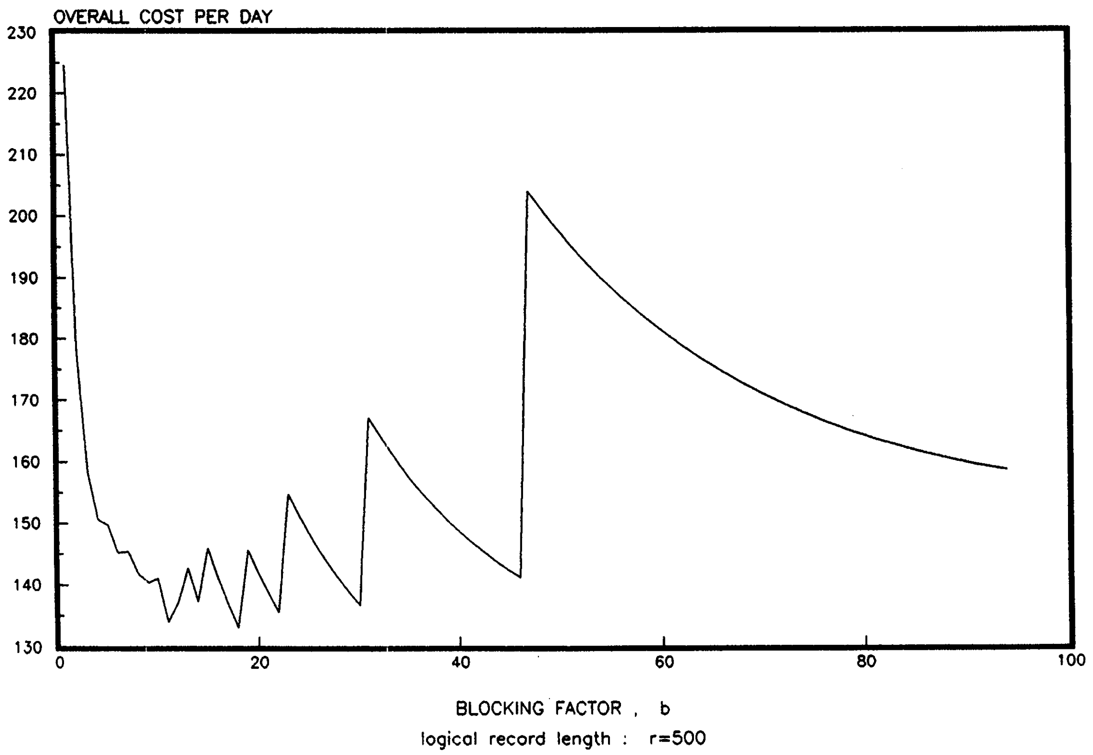

Fig. 4. Overall cost (in dollars per day) for sorted and unsorted chains as a function of the blocking factor b for the constrained design problem. The graphs correspond to $K = 300{,}000$ records in the primary area. The differences between the cases of sorted and unsorted chains are negligible, and the two graphs virtually coincide with each other.

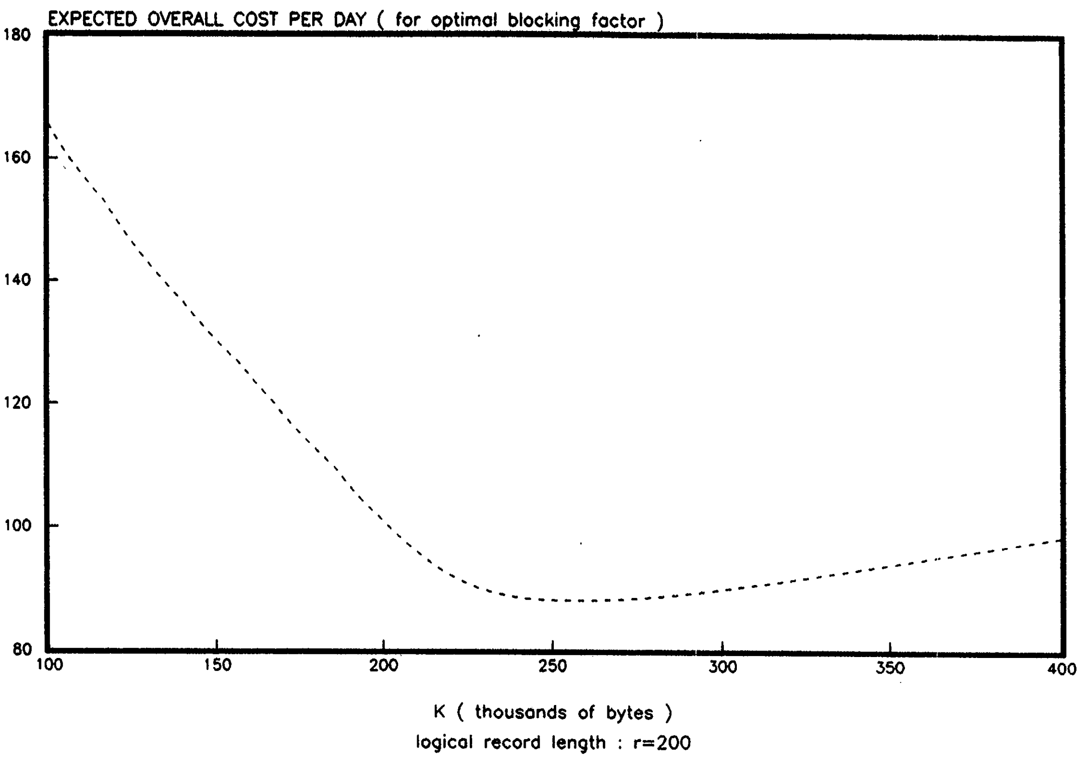

Fig. 5. The unconstrained design problem: overall cost (in dollars per day) for 200-byte records as a function of the prime area allocation, K.

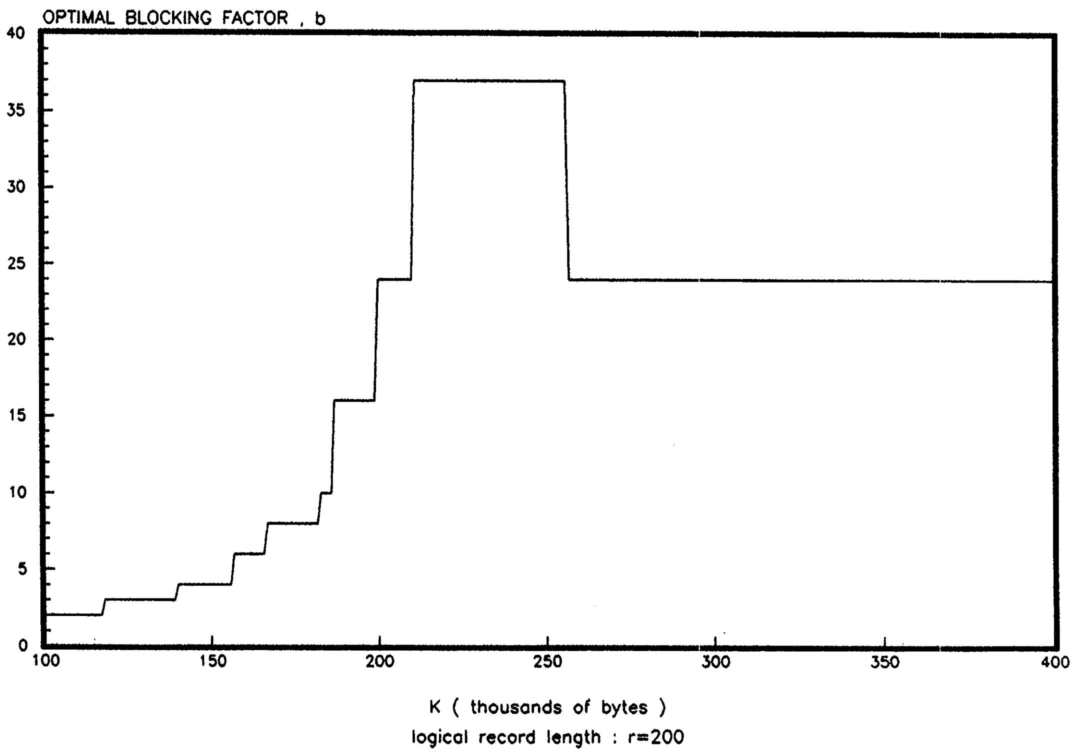

Fig. 6. The unconstrained design problem: optimal blocking factor, b, as a function of the prime area allocation K (logical record length = 200 bytes).

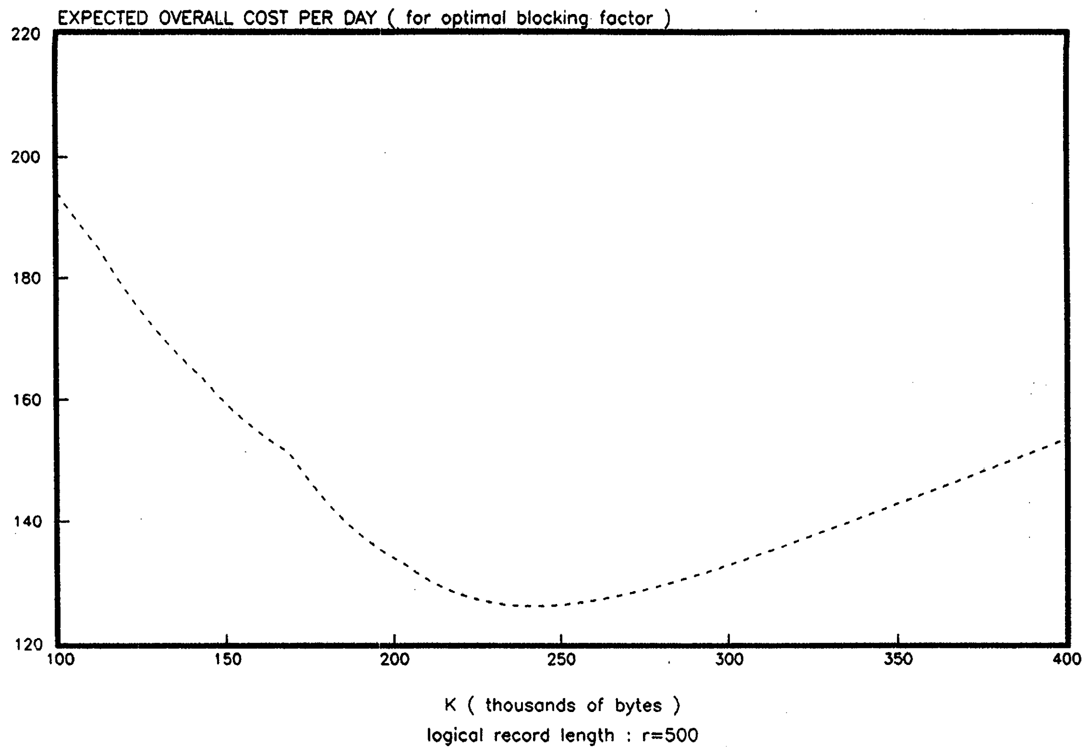

Fig. 7. The unconstrained design problem: overall cost (in dollars per day) for 500-byte records as a function of the prime area allocation, K.

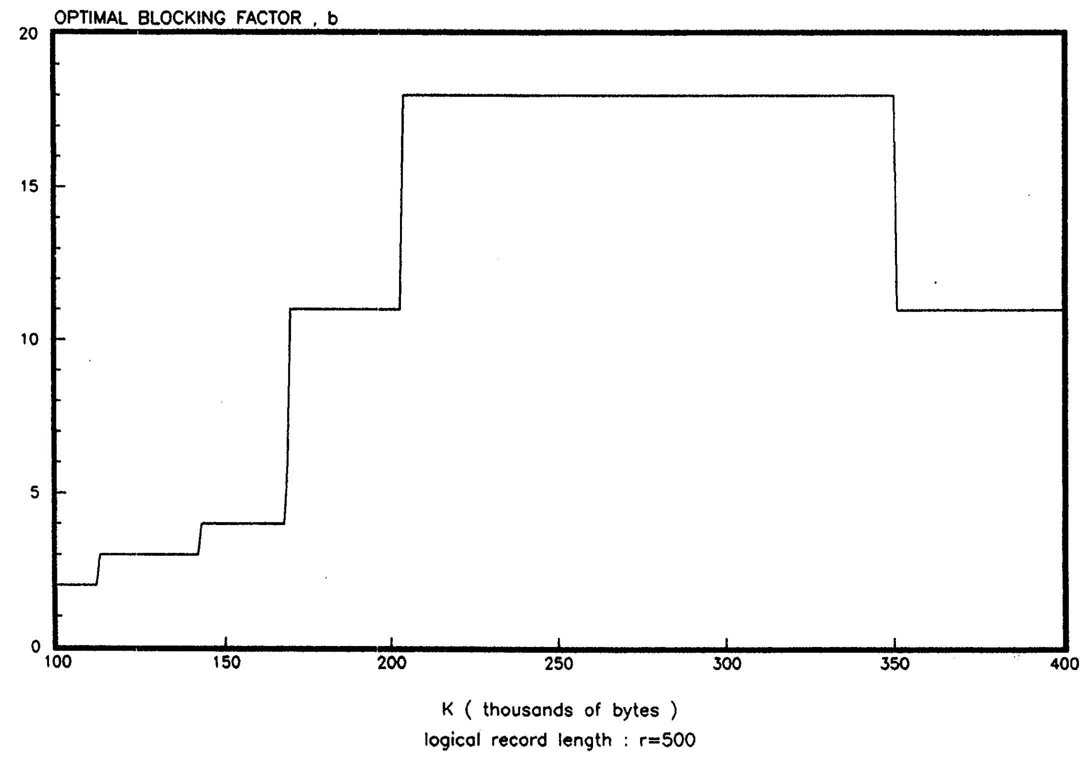

Fig. 8. The unconstrained design problem: opitmal blocking factor, b, as a function of the prime area allocation K (logical record length = 500 bytes).

by an increase in the storage cost of the prime area; but in this range the reduction in input/output cost more than compensates for the increased storage cost. For $K > K_{\min}$, the situation is reversed, and the storage cost becomes the dominant factor. In fact, when K gets large enough, the overflow chains become practically negligible, and increasing K only increases the storage cost without any significant offsetting benefits.

Figure 5 demonstrates the behavior of the expected overall cost per day as a function of the capacity K of the prime area, using the optimal blocking factor $b = b^*(K)$, when the logical record length is $r = 200$ bytes. The global optimum is at $K = K_{\min} = 253{,}000$ bytes, some 25% above the expected number of records $\rho = 200{,}000$; the corresponding overall daily cost is \$88.12. The overall daily cost remains below \$90 through the range 229–302K (at optimum b); thus, the minimum is quite flat. Note that for this value of r, the shape of the expected cost curve is quite asymmetric and the marginal cost of overallocation is low with respect to the marginal cost of underallocation.

Figure 6 demonstrates the behavior of the optimal blocking factor, b, as a function of the available prime area capacity K. At the global optimum, the optimal blocking factor is $b = 37$; the same blocking factor is optimal in the range $211{,}000 \leqslant K \leqslant 256{,}000$. Also note that the graph of $b^*(K)$ is unimodal, with its maximum at the global optimum point $K = K_{\min}$. Intuitively, when K is significantly below its optimal level, the clustering effect discussed before implies that large values of b are undesirable. As K increases towards $K_{\min}$, the clustering effect becomes less severe, thus allowing the more efficient storage utilization afforded by larger values of b. As we have also seen before, large values of b are undesirable even for large values of K, and $b^*(K)$ starts declining at $K = 257{,}000$. Note that the optimal values of b follow the sequence of blocking factors that allow for efficient track utilization, emphasizing once more the importance of this effect.

Figures 7 and 8 demonstrate the behavior of the cost and the optimal value of the blocking factor $b^*(K)$ for the case of somewhat larger records with logical record length $r = 500$ bytes. The global minimum is attained at $K = 242{,}000$ and $b = 18$ records per block; clearly, the minimum cost is somewhat higher than before (\$126.22 per day). The general pattern of behavior is similar to the case $r = 200$. However, the larger record size reduces the asymmetry in the shape of the cost function; this should be expected since storage considerations gain importance as the record size increases. The unimodality of the function $b^*(K)$, with maximum b associated with $K = K_{\min}$, occurs as before. Here, the range where $b = 18$ is optimal is wide ($204{,}000 \leqslant K \leqslant 350{,}000$); as before, the optimal values of b coincide with those values that give rise to efficient track utilization.

5. CONCLUDING REMARKS

This study provides a comprehensive analysis of the performance of a database system that is subject to random additions and deletions. Fortunately, we have been able to obtain closed-form results on the behavior of the relevant performance measures; programming these results is a trivial matter and the computational costs involved are quite insignificant†. These results are useful in the context of performance prediction, cost evaluation and system design.

We have shown a number of examples where we derived the optimal system design parameters (bucket size and/or prime area allocation). These examples also provide insights to the general patterns of system behavior, as well as to the relative importance of various considerations. In particular, we have shown that, in contrast to the standard recommendation for large bucket sizes, small to moderate blocking factors are often optimal. We have emphasized the importance of using those blocking factors that lead to efficient track utilization; although additional factors also affect the overall costs, we have seen that this consideration is, locally, a determining factor. We have also examined how the availability of storage space affects the behavior of the system and the values of blocking factors to be utilized. Also, the degree of asymmetry in the shape of the cost function (as a function of available storage capacity) is such that for small record lengths, we may be more generous in the allocation of the prime area without incurring significant excess costs, whereas for large record sizes this may be too costly, and the designer may have to be more careful in allocating space for the prime area.

Finally, it is instructive to examine the order of magnitude of the costs associated with poor design. Our results indicate that for a single medium-sized file, sub-optimal design can increase the daily

†The source programs employed in the production of our results are available upon request.

operating costs from some \$100 to an order of magnitude of \$200; poor designs could cost even \$1000 per day for our examples (many of these numbers do not appear in the graphs, which are themselves the results of optimization procedures). This translates into a difference of some \$30,000–\$300,000 per year per file. Multiplying this by the number of files and comparing to the trivial computational costs, we note that physical design is a worthwhile investment, even if we accept only the low end of the estimate.

REFERENCES

1. B. Shneiderman, Optimum data base reorganization points. *CACM* **16**, 362–365 (1973).
2. H. Mendelson and U. Yechiali, Optimal policies for database reorganization. *Opns Res.* **29**, 23–36 (1981).
3. D. P. Heyman, Mathematical models of database degradation. *ACM Trans. Database Syst.* **7**, 615–631 (1982).
4. D. E. Knuth, Deletions that preserve randomness. *IEEE Trans. Software Engng* **SE-3**, No. 5 (1977).
5. J. A. Van der Pool, Optimum storage allocation for a file in steady state. *IBM Jl Res. Dev.* **17**, 27–38 (1973).
6. A. T. Jonassen and D. E. Knuth, A trivial algorithm whose analysis isn't. *J. Comput. Syst. Sci.* **16**, 301–322 (1978).
7. H. Mendelson, Analysis of extendible hashing. *IEEE Trans. Software Engng* **SE-8**, 611–619 (1982).
8. S. M. Ross, *Applied Probability Models with Optimization Applications.* Holden-Day, San Francisco (1970).
9. D. E. Knuth, *The Art of Computer Programming*, Vol. 3, Sec. 6.4, pp. 506–518, Addison-Wesley, Reading, Mass. (1973).
10. H. Mendelson and U. Yechiali, Performance measures for ordered lists in random-access files. *J. Ass. Comput. Mach.* **26**, 654–667 (1979).
11. H. Mendelson, Analysis of linear probing with buckets. *Inform. Syst.* **8**, 207–216 (1983).
12. D. P. Heyman and M. J. Sobel, *Stochastic Models in Operations Research, Vol. I: Stochastic Processes and Operating Characteristics.* McGraw-Hill, New York (1982).
13. I. Meilijson, M. R. Newborn, A. Tenenbein and U. Yechiali, Number of matches and matched people in the birthday problem. *Commun. Statist.-Simulat. Comput.* **11**, 361–370 (1982).
14. H. Mendelson, Lattice path combinatorics and linear probing. *J. Statist. Plann. Inference* **14**, 79–94 (1986).