Analysis of two hash-coded file reorganization policies under steady-state conditions

Michel Scholl

IRIA
laboria

Institut de Recherche
d'Informatique
et d'Automatique

Domaine de Voluceau
Rocquencourt
B. P. 105 78150 - Le Chesnay
France
Tél.: 954 90 20

laboratoire de recherche
en informatique
et automatique

ANALYSIS OF TWO HASH-CODED FILE REORGANIZATION POLICIES UNDER STEADY-STATE CONDITIONS

Michel SCHOLL

Rapport de Recherche N° 354

Juin 1979

ANALYSIS OF TWO HASH-CODED FILE REORGANIZATION POLICIES UNDER STEADY-STATE CONDITIONS

Michel Scholl
IRIA/LABORIA

Résumé :

Un modèle simple, prenant en compte les opérations élémentaires d'adjonction, de suppression et de consultation d'articles dans un fichier hash-codé est présenté dans cet article. Il permet l'analyse de deux politiques de réorganisation d'un fichier hash-codé dont le nombre d'articles varie peu dans le temps. A l'aide d'une modélisation par files d'attente, on montre qu'une politique de compactage périodique de l'espace mémoire ne dégrade pas sensiblement les performances d'accès au fichier, comparée à une politique qui réalloue immédiatement l'espace libéré lors d'une suppression. Si le taux de mises à jour (adjonctions/suppressions) est élevé, la première politique procure même de meilleures performances d'accès que la seconde, si le facteur de charge est élevé.

Abstract :

A simple model which takes into account elementary operations on a hash-coded file, such as records insertion, deletion and searching is presented. It is used to analyze two reorganization policies for hash-coded files under steady-state conditions, i.e. when the number of records does not substantially change with time. By using a queueing modelling approach, it is shown that a policy which delays the relocation of records does not significantly affect the access performance, compared to a policy which relocates records upon each deletion. If the frequency of updating (insertions/deletions) is high, the former strategy performs even better than the latter, under heavy load conditions.

1 - INTRODUCTION

Hash-coding used as a file access technique has been thoroughly analyzed, from a static viewpoint (see for example [1, Section 6.4]). By static we mean that the extent of the data remains unchanged during processing : only values are updated.

Initially the file is divided into M primary buckets. Each record provided with a unique key, hashing to an integer in [1, M], falls into one among the M buckets. If more records are assigned to a bucket than it can hold, a link to an "overflow" record can be inserted at the end of the primary bucket. Thus, after a while, the file includes M separate chains of records. Usually, a new record is inserted at the end of the chain, after it has been checked that the record has not been previously stored in the file (the whole records chain is scanned).

Then, given the global size of the file, a combinatorial analysis leads to performance measures such as the expected number of accesses to secondary storage for searching or inserting a record.

However if the file is subject to frequent insertions/deletions of records, the file performance may strongly depend on the rate of updating (insertion/deletion) compared to the rate of searching records and on the file reorganization policy. In particular, one has to decide whether records are relocated or not (a deletion frees some storage space), and when this relocation must occur (upon each deletion or every now and then). If the records are not relocated, the number of overflow records may significantly increase [2], although the number of records actually stored in the file does not substantially increase with time.

In [2, 3] such a situation is analyzed. Both studies consider a hash-coded file to which records are added and from which records are deleted such that the rates of additions to and deletions from the file are equal.

This approach is generalized below by using a queueing modelling of hash-coded files. This allows the comparison of the dynamic behavior of the file under two reorganization policies.

The first strategy referred to as "Dynamic" Reorganization and analyzed in Section 3 assumes that any "hole" created in the file by a deletion is filled right away. For the sake of simplicity records are assumed throughout the paper, to be equal in length. Then, each time a record is deleted, the most recently inserted record (located at the end of the chain in which occurs the deletion) is moved to occupy the space freed by the record just deleted.

The second strategy, called "Periodical" Reorganization does not relocate records upon a deletion (the space is not freed). The deleted record is unvalidated (marked). If the frequency of updatings is not negligeable then the file size substantially increases, which leads to a waste of storage space, as well as to a degradation in the access performance. Therefore after a while a compression of the file is necessary which frees the storage space occupied by unvalidated (deleted) records. Otherwise, the file will grow without any bound.

The following alternative is considered and analyzed in Section 4 : instead of being inserted at the end of the chain, a new record is inserted into the first "hole" (space occupied by a record previously deleted) encountered when scanning the chain of records whose keys hash to the same bucket. This does not save the time-consuming operation of scanning the whole chain upon each insertion[1] in order to check whether the record does exist in the file, or not. However this alternative keeps the chains of growing without any bound. On the contrary the file size reaches an equilibrium.

In this paper, we take as a performance indicator, the sum of the expected numbers of accesses to secondary storage per unit of time when inserting, searching and deleting a record denoted respectively by C_{dyn} (Dynamic Reorganization) and C_{per} (Periodical Reorganization).

---

1) The checking operation can be performed upon each deletion, if we are willing to accept some "mistakes" of the file users, i.e. if several copies of the same record are tolerated. Only the most recently inserted copy will be accessed. This alternative will be less costly for applications in which the file steadily grows (deletions being less frequent than insertions).

It is shown in Section 5 that C_{per} is not significantly larger than C_{dyn}, and is even smaller than C_{dyn} under heavy load conditions, if the insertion/deletion activity is high.

The file structure as well as the modelling approach are described in the following section.

2 - MODELLING APPROACH

Initially, secondary storage is allocated for M buckets. A storage management system is assumed from which buckets can be requested and to which free buckets can be returned. All buckets are assumed to be equal in size and of capacity b records. If more than b records fall into the same primary bucket, an overflow bucket is linked to the primary bucket, "overflow" records being inserted in this overflow bucket. If this overflow bucket is in turn full (i.e. more than 2b records keys hash to the same value) then a third bucket is linked to the second one, etc... Thus each primary bucket owns a separate chain of overflow buckets. This storage management system as well as this "chaining with separate lists" collision resolution method are used in connection with several applications, among others the Database Management System INGRES [4].

We assume that new records are inserted into any of the M chains of buckets according to an homogeneous Poisson process with intensity λ (records/second), such that the insertion processes into any two chains are independent of each other. Therefore the global file insertion process is Poisson with intensity $\Lambda = M\lambda$.

Any record in the file is assumed to have an exponential life time distribution (ltd), independent of the other records ltd's, independent of the insertion process and independent of the number of records actually stored in the file. This ltd common to all records has a mean denoted by $1/\mu$, i.e. the records deletion rate in a given chain of size n records is equal to $n\mu$.

It is shown below that the expected number of records actually stored in any among the M chains is equal to λ/μ. The expected file load factor is defined by the following equation :

(1) $$\frac{\Lambda}{\mu} = \alpha M b$$

Usually, b is given (e.g. b represents the page size which is a system constraint) and M is adjusted to the expected file size (Λ/μ) so that α is close to 1. If α is too small ($\alpha < .8$) storage space is underutilized although the access performance is excellent. If α is too large ($\alpha > 1.2$) then the access performance may be significantly degraded.

The searching activity is modelled as follows : it is assumed that the times elapsing between two successive searches of a given record are independent and identically distributed with an exponential distribution whose first moment is denoted by $1/\sigma$. All records are assumed to be searched independently of each other according to this common distribution, the searching rate in a chain including n records being $n\sigma$.

While M and b are design parameters, Λ, μ and σ characterize the file activity.

As mentionned in the introduction, we will use as a performance criterium for the dynamic behaviour of file structure i, the steady-state expected number of accesses C_i to secondary storage per unit of time. The expected "inter-insertion" time $1/\Lambda$ is chosen as a unit of time.

More precisely, C_i is defined by :

(2) $$C_i = \lim_{t\to\infty} \lim_{dt\to\infty} \frac{\mathcal{C}_i(t+dt) - \mathcal{C}_i(t)}{dt}$$

where $\mathcal{C}_i(t)$ denotes the mean number of accesses to secondary storage (including insertions, deletions and searching) performed in the time interval (0, t) and normalized with respect to $1/\Lambda$.

We first analyze a dynamic reorganization and evaluate C_{dyn}.

3 - DYNAMIC REORGANIZATION

Figure 1 illustrates the insertion and deletion operations on the file under a dynamic reorganization policy. If a record is to be inserted

into chain m (its key K hashes to m, m = 1,...,M) m's primary bucket is first scanned then the overflow buckets are scanned in sequence until either the record is found (no insertion is performed) or the end of the chain is reached : say bucket t terminates the chain. In the latter case

- either bucket t contains less than b records, then the new record is inserted into t,

- or bucket t already contains b records (Fig. 1.a), then the new record is inserted into a new bucket u allocated and linked to t.

Upon a deletion, we also have to scan the whole chain. Two cases must be considered. If the record d to be deleted is located in the last bucket (t), then d is deleted from t and only one write access (write t) is needed. Otherwise (Figure 1.b), the most recently inserted record : r (or any record located in t) is deleted from t and moved to the bucket containing d (r replaces d). Then two write accesses are required.

Chain m can be viewed as an M/M/∞ queueing system, with Poisson (λ) input, exponential ($1/\mu$) "service time" (record's life time) distribution and an infinity of servers, each record location being viewed as a single server. When a new customer arrives and finds n servers busy (a new record is to be inserted in chain m which contains n records), a server is allocated to him (the n + 1st location in sequence is assigned to this new record). Upon a departure (deletion), one server becomes idle (given there were n records before the deletion, the n^{th} location is freed).

The steady-state probability P_n that chain m includes n records is Poisson [5], with mean λ/μ (= αb, see Eq. (1)) :

$$P_n \triangleq \lim_{t\to\infty} P_n(t) = \frac{(\lambda/\mu)^n}{n!} e^{-\lambda/\mu} \tag{3}$$

The expected number of records stored in the file is then :

$$E\{N\} = M\lambda/\mu = \Lambda/\mu \tag{4}$$

Note the equivalence between Eq. (4) and the methods developped in [2, 3] which assume that the insertion rate (Λ) is equal to the deletion rate ($\mu \times E\{N\}$).

Given a small enough time interval (t, t+dt) at most one of the three following operations may occur : (1) new insertion, (2) search, (3) deletion, since the three corresponding Processes are Poisson. Let n be the number of records stored at time t in chain m, and i such that $(i-1)b < n \leq ib$. Conditionning on n, we have :

$$E\{\text{nb of accesses/n records}\} = (i+1)\, P\{\text{an insertion occurs in } (t, t+dt)\}$$

$$+ \sum_{j=1}^{i} j\, P\{\text{a record stored in bucket j is searched in } (t, t+dt)\}$$

$$+ (i+2) \sum_{j=1}^{i-1} P\{\text{a record is deleted from bucket j in } (t, t+dt)\}$$

$$+ (i+1)\, P\{\text{a record is deleted from bucket i in } (t, t+dt)\}$$

(5)
$$E\{\text{nb of accesses/n records}\} = (i+1)\lambda dt + \sum_{j=1}^{i-1} jb\sigma dt + i(n - (i-1)b)\sigma dt$$
$$+ (i+2) \times \sum_{j=1}^{i-1} b\mu dt + (i+1)(n - (i-1)b)\mu dt$$

Unconditionning on n, dividing by Λ and multiplying by M, we get :

(6)
$$\mathcal{C}_{dyn}(t+td) - \mathcal{C}_{dyn}(t) = \frac{M}{\Lambda}\left[\lambda \sum_{i=1}^{\infty} (i+1) \sum_{n=(i-1)b}^{ib-1} P_n(t) + \right.$$

$$+ \sigma \sum_{i=1}^{\infty} \sum_{n=(i-b)b+1}^{ib} \left[ni - \frac{i(i-1)b}{2}\right] P_n(t) +$$

$$\left. + \sum_{i=1}^{\infty} \sum_{n=(i-1)b+1}^{ib} [n\mu(i+1) + \mu(i-1)b]\, P_n(t)\right] dt$$

Dividing Eq. (6) by dt and taking the limit as $dt \to 0$ and $t \to \infty$, we obtain C_{dyn} (see Eq. (2)) :

$$(7) \qquad C_{dyn} = \sum_{i=0}^{\infty} Q_i(\lambda/\mu) + 1 + \sigma \sum_{i=0}^{\infty} Q_i(\lambda/\mu) - \frac{Mb\sigma}{\Lambda} \sum_{i=1}^{\infty} i\, Z_i(\lambda/\mu)$$

$$+ \sum_{i=0}^{\infty} Q_i(\lambda/\mu) + 1 + \frac{Mb\mu}{\Lambda} \sum_{i=1}^{\infty} Z_i(\lambda/\mu)$$

where

$$(8) \qquad Q_i(\lambda/\mu) = \sum_{n=ib}^{\infty} e^{-\lambda/\mu} \frac{(\lambda/\mu)^n}{n!}$$

$$(9) \qquad Z_i(\lambda/\mu) = \sum_{n=ib+1}^{\infty} e^{-\lambda/\mu} \frac{(\lambda/\mu)^n}{n!}$$

Eq. (7) reduces to :

$$(10) \qquad C_{dyn} = \left[\frac{\sigma}{\mu} + 2\right] \sum_{i=0}^{\infty} Q_i(\lambda/\mu) + 2 - \frac{1}{\alpha} \sum_{i=1}^{\infty} \left(\frac{i\sigma}{\mu} - 1\right) Z_i(\lambda/\mu)$$

4 - PERIODICAL REORGANIZATION

4.1 - Updating operations

Figure 2 illustrates the processes of inserting and deleting a record. When inserting a record in chain m, the chain is scanned until either the record is found or the end of the chain is reached. In the latter case, the record is inserted into the first bucket encountered (when scanning the chain) which contains a hole, where a hole corresponds to a record previously deleted (marked)(Fig. 2.A). If the chain contains no hole, then the record is inserted at the end of the chain. When deleting a record, the chain is scanned until the record is found. The record to be deleted is marked (Fig. 2.b). In both cases, one write access is needed.

Intuitively, after a long time, most records actually stored in the file are eventually located at the beginning of the chain, while most of the unvalidated records are located at the end of the chain (Fig. 3). Even though, the chain size S is large and contains many marked records, it eventually

happens, because of the probabilistic nature of the updating processes, that the S first record locations are occupied. A new record to be inserted which finds the S first locations occupied will be inserted in the (S+1)st location and the size goes to S+1. Therefore, even though the actual number of records stored in any chain does not substantially change with time, the chain size may grow without any bound.

Using the same modelling approach as in section 2, it may be shown (see Appendix A) that when inserting in the first hole the expected time before the S^{th} cell is occupied for the first time given there were j cells occupied at time 0, is equal to :

$$(11) \qquad E_S^j(t) = \frac{1}{\lambda} \sum_{i=j}^{S-1} \frac{\sum_{k=0}^{i} (\lambda/\mu)^k/k!}{(\lambda/\mu)^i/i!}$$

The chain size does not reach an equilibrium. Figure 4 illustrates the chain size growth with time normalized with respect to $1/\mu$, for various values of the load factor α. The initial number of records in chain m has been taken equal to λ/μ ($=\alpha b$) which is the mean number of records actually stored in the file (see Section 3), i.e. the mean file size after a reorganization (chain compression) which occurs at time 0.

From Figure 4, it is clear that depending on the load factor, the chain size will grow more or less rapidly with time. If $\alpha=1$, the second bucket overflows for the first time on the average after 100 mean record life times, while if $\alpha=1.5$, it takes only 5 mean life times before the second bucket overflows.

4.2 - File reorganization

Eventually any chain must be compressed. One has to decide when this reorganization should be performed. This reorganization might occur $E_{2b}^{\alpha b}(t)$ seconds after the previous reorganization, where recall (Eq. (11)) $E_{2b}^{\alpha b}(t)$ is the expected time until a chain reaches for the first time 2b records given that after the previous reorganization it included αb records ($\alpha<2$). If the updating activity is

high ($1/\mu$ small) or/and if the load factor is large (e.g. $\alpha = 1.5$) this periodical reorganization would then occur too often and should be delayed until $E_{kb}^{\alpha b}(t)$ seconds where k is greater than 2. The expected file reorganization cost per unit of time $C_R(k)$ (where $1/\Lambda$ is the time unit) when delaying each chain reorganization until $E_{kb}^{\alpha b}(t)$ seconds may be evaluated by the following equation :

$$C_R(k) = \frac{M}{\Lambda} (k + \lceil \alpha \rceil)/E_{kb}^{\alpha b}(t)$$

where $\lceil X \rceil$ denotes the smallest integer $\geq X$. Indeed we assume a reorganization costs k read accesses plus on the average $\lceil \alpha \rceil$ write-accesses (since the mean chain size after the reorganization is equal to $\lambda/\mu = \alpha b$).

$C_R(k)$ is plotted versus k in Figure 5 for various values of α. Clearly $C_R(k)$ is very small and decays very rapidly as $k \geq 2$ increases.

A variant to the above reorganization policy would be the following : trigger any chain's reorganization the first time the actual number of records stored in the chain goes down to $\lceil \alpha b \rceil$ records after having reached (for the first time) kb records.

In the following analysis the latter variant is chosen with $k = 2$.

4.3 - C_{per} Evaluation

In order to evaluate the insertion cost we make the following approximation :

we assume that at all times the expected number of buckets occupied by any chain is equal to 2 for all values of $\alpha < 2$. On the one hand, this implies that a chain never occupies more than two buckets. The probability that the chain size increases over 2b records before decreasing down to $\lceil \alpha b \rceil$ records is positive, although very small if α is not too large. However, the closer to 2 α is, the more optimistic the above assumption is. On the other hand, the smaller α is, the more pessimistic this assumption is, e.g. if $\alpha = .8$, a chain may include only one primary bucket for a somewhat long period of time.

From the above assumption the number of accesses to secondary storage when inserting a record is equal to 3. This may be considered as an upper bound on the actual number of accesses when inserting a record if α is not too large ($\alpha \leq 1.5$).

Our next task is to evaluate the costs of searching and deleting a record. Consider the primary bucket of chain m. It may be viewed as an M/M/b/b

queueing system with Poisson (λ) input, exponential ($1/\mu$) service time, b servers and a total capacity of b customers. A record will be inserted into this bucket if and only if it does not already contain b records. Otherwise (the customer balks the queue), the record is inserted into the first overflow bucket if the latter is not full. The primary bucket and the first overflow bucket taken together may be viewed as a single M/M/2b/2b queue. When this queue is full, then an entering customer balks the queue, i.e. if upon an insertion the two first buckets are both full, the third bucket is tried, etc...

Let $P_b^i(t)$ denote the probability that the first i buckets are full at time t. We have [5] :

$$(12) \qquad P_b^i \triangleq \lim_{t\to\infty} P_b^i(t) = \frac{(\lambda/\mu)^{ib}/(ib)!}{\sum_{k=0}^{ib} (\lambda/\mu)^k/k!} = \frac{(\alpha b)^{ib}/(ib)!}{\sum_{k=0}^{ib} (\alpha b)^k/k!}$$

In particular the probability that the primary bucket contains b records is given by :

$$P_b^1 = [(\lambda/\mu)^b/b!] \Big/ \sum_{k=0}^{b} (\lambda/\mu)^k/k!$$

This result was obtained in [2] by directly formulating the difference equations for the distribution of the number of records contained in the primary bucket, and in [3] by directly solving a discrete time Markov chain (which is precisely that of an M/M/b/b!).

Then the probability that, at time t, a new record is inserted into bucket i is equal to $P_b^{i-1}(t) - P_b^i(t)$ (with $P_b^0(t) \triangleq 1$). The expected number of records actually stored in bucket i at time t is :

$$(13) \qquad \bar{n}_i(t) = \lambda/\mu \, (P_b^{i-1}(t) - P_b^i(t))$$

Indeed $\bar{n}_i(t)$ is equal to the expected number of customers in an M/M/ib/ib ($\frac{\lambda}{\mu}[1 - P_b^i(t)]$, see [5]) minus the expected number of customers in an M/M/(i-1)b/(i-1)b. Then, using the same approach as in Section 3, we have :

$$C_{per} = \lim_{t\to\infty} \lim_{dt\to\infty} \frac{\mathcal{C}_{per}(t+dt) - \mathcal{C}_{per}(t)}{dt}$$

$$(14) \qquad \mathcal{C}_{per}(t+dt) - \mathcal{C}_{per}(t) = \frac{M}{\Lambda} \{3\lambda dt + \sum_{i=1}^{\infty} i\bar{n}_i \sigma dt + \sum_{i=1}^{\infty} (i+1)\bar{n}_i \mu dt\}$$

The three terms into brackets in Eq. (14) respectively correspond to insertion, searching and deletion [2].

Substituting Eq. (13) into Eq. (14), we get :

$$\mathcal{C}_{per}(t+dt) - \mathcal{C}_{per}(t) = \frac{M}{\Lambda} \{3\lambda + \sigma \frac{\lambda}{\mu} \sum_{i=0}^{\infty} P_b^i(t) + \lambda[\sum_{i=0}^{\infty} P_b^i(t) + 1]\}dt$$

Dividing by dt and taking the limit as $dt \to 0$ and $t \to \infty$, we finally obtain the following expression :

$$(15) \qquad C_{per} = (1 + \frac{\sigma}{\mu}) \sum_{i=0}^{\infty} P_b^i + 4$$

where P_b^i is given by Eq. (12).

$\sum_{i=0}^{\infty} P_b^i$ is plotted versus b for serveral values of α, in Figure 6.

$\sum_{i=0}^{\infty} P_b^i$ (= $\sum_{i=1}^{\infty} i\bar{n}_i$, see Eqs. (13) and (14)) is the expected number of accesses to find all records in chain m. Then, obviously, for a given value of b, $\sum_{i=0}^{\infty} P_b^i$ and therefore C_{per} increase as α increases, since the chain size increases with α. From Figure 6, $\sum_{i=0}^{\infty} P_b^i$ (and thus C_{per}) is a decreasing function of b, for a given value of α (see Appendix B for a formal proof). From Eq. (1), for a fixed mean file size ($\frac{\Lambda}{\mu}$), increasing b implies :

- either decreasing α (M kept constant), then obviously the access performance is improved but we incur the risk of underutilizing memory space if α is to small,

- or decreasing M (α kept constant), then the higher the bucket's capacity is, the smaller $\sum_{i=0}^{\infty} P_b^i$ is and the better the access performance is.

---

2) Observe the second and third terms are an upper bound on respectively the searching and deletion costs, obtained when assuming no reorganization occurs. However, the longer the time between two successive reorganizations is, the tighter these bounds are.

5 - DISCUSSION

Consider first the case where the updating activity is negligeable compared to the searching activity. Then, from Eqs. (10) and (15) we have asymptotically, as $\sigma/\mu \to \infty$:

$$C^{\infty}_{dyn} \simeq \frac{\sigma}{\mu} \{ \sum_{i=0}^{\infty} Q_i(\lambda/\mu) - \frac{1}{\alpha} \sum_{i=1}^{\infty} i\, Z_i(\lambda/\mu) \} \tag{16}$$

$$C^{\infty}_{per} \simeq \frac{\sigma}{\mu} \sum_{i=0}^{\infty} P_b^i \tag{17}$$

One easily shows [see Appendix C] that

$$C^{\infty}_{per} > C^{\infty}_{dyn}$$

This was to be expected, since the chains being longer in the case of a periodical reorganization than in the case of a dynamic reorganization, the expected number of accesses for searching a record is higher in the former case.

Observe however (Eqs. (16) and (17)) that $(C^{\infty}_{per} - C^{\infty}_{dyn})/C^{\infty}_{dyn}$ is independent of σ/μ and that for all values of α (≤ 1.8), this ratio is less than 10 % (Table 1), with a maximum occuring around $\alpha = 1$. Indeed if $\alpha > 1$, even in the dynamic case, the chains get longer and the discrepancy due to marked records in the periodical case is less significant (see Figure 3).

We turn now our attention to the main case of interest of our study, namely the case where the file is subject to a frequent insertion/deletion activity.

Table 2 illustrates the comparison between Periodical and Dynamic Reorganization obtained from Eqs. (10) and (15) for various values of σ/μ and various values of α.

Three remarks may be drawn for Table 2. First note that as expected both C_{per} and C_{dyn} increase as the load factor α increases. Second, when the updating activity is small compared to the searching activity ($\sigma/\mu \geq 100$), the dynamic reorganization performs better than the periodical one, although the

difference is not substantial. Finally, if the updating activity is important ($\sigma/\mu \leq 1$), when the file is heavily loaded ($\alpha > .9$) a dynamical reorganization of the file performs not as well as a periodical one : this is not surprising since under heavy load conditions, (1) the expected cost when inserting a record is the same in the periodical case and in the dynamic case, since when α increases ($\alpha > 1$), in both cases on the average, two buckets are scanned when inserting a record ; (2) the slight searching access performance discrepancy due to a periodical relocation of records (see Table 1) is less and less significant as σ/μ decreases ; (3) as α increases, the deletion cost is eventually significantly higher under the dynamic policy than under the periodical one.

This is more clearly illustrated in Figure 7 where C_{per} and C_{dyn} are plotted versus α, for $\sigma = \mu$ (Figure 7.a) and $\sigma = 100\mu$ (Figure 7.b).

In conclusion, we have compared in this paper two reorganization policies for a hash-coded file whose records are equal in length, by studying the dynamic behaviour of the file subject to a frequent updating activity. In order to take into account elementary operations such as records insertion, deletion and searching, a queueing model was used and both the insertion searching and deletion processes were assumed to be Poisson. Such a modelling approach allows to study the file's behaviour with time. The file was assumed to be under steady-state (this is the case of a file whose size does not vary much with time).

Under these assumptions, it was shown that a policy which delays the relocation of records (Periodical Reorganization) does not significantly affect the search performance (see Table 1) as well as the global access performance (searching + updating) (see Table 2) compared to a policy which relocates records upon each deletion. If the updating acitivity is high, the former strategy performs even better than the latter under heavy load conditions (Figure 7.a).

Besides, a periodical reorganization presents the advantage of being more suitable for records variable in length. In the latter case, dynamic storage allocation is an uneasy task rendering a dynamic reorganization as described above unpractical and less performant than it is when records are equal in length (holes remain after a relocation). On the contrary, it is easier to mark deleted records and to insert a new record in the first sufficiently large hole encountered when scanning the chain.

ACKNOWLEDGEMENTS

The author gratefully acknowledges D. Potier for the hours he spent with him discussing this work.

APPENDIX A

(Proof of Eq. (11))

THEOREM : The expected time before the queue M/M/∞ includes for the first time S customers given that the queue is initially empty is given by :

$$(A1) \qquad E_S^0(t) = \frac{1}{\lambda} \sum_{i=0}^{S-1} \frac{\sum_{k=0}^{i} (\lambda/\mu)^k/k!}{(\lambda/\mu)^i/i!}$$

where λ and $1/\mu$ denote respectively the input rate and the expected service time. Observe that (A1) can be written as :

$$(A2) \qquad E_S^0(t) = \frac{1}{\lambda} \sum_{i=0}^{S-1} \frac{1}{P_1^i}$$

where P_1^i (Eq. 12) is the probability that the queue M/M/i/i is full [5].

Eq. (A2) can be written as :

$$(A3) \qquad E_S^0(t) = E_{S-1}^0 + \frac{1}{\lambda P_1^{S-1}}$$

Eq. (A3) may be interpreted as follows : the expected time to reach cell S for the first time is equal to the expected time to reach cell (S-1) for the first time plus the expected time τ_0 until cell S is occupied for the first time given there were initially S-1 customers in the queue. τ_0 is also the expected time until the first arrival of a customer who balks the queue M/M/S-1/S-1/ given this queue is initially full. Since arrivals are Poissons, τ_0 is precisely equal to the expected interarrival time of two successive balks (the mean time between two arrivals which find the queue full) [6].

PROOF : Let us consider the following birth-death process with absorbing state S whose state transition diagram is depicted in Figure A1 :

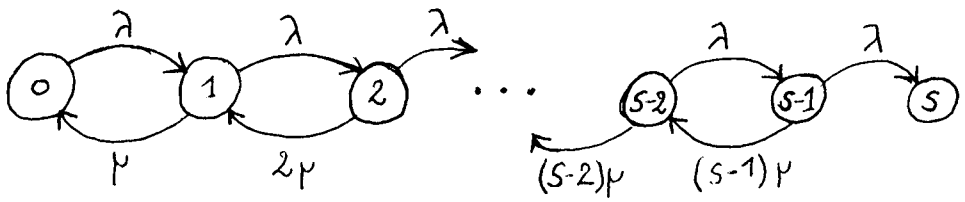


Figure A1 : State transition diagram

Let $P_i(t)$, i = 1, 2,...,S denote the probability that the system is in state i (i customers are in service) at time t. We have :

$$\frac{dP_0(t)}{dt} = -\lambda P_0(t) + \mu P_1(t)$$

$$\frac{dP_1(t)}{dt} = -(\lambda+\mu)P_1(t) + \lambda P_0(t) + 2\mu P_2(t)$$

$$\frac{dP_i(t)}{dt} = -(\lambda+i\mu)P_i(t) + \lambda P_{i-1}(t) + (i+1)\mu P_{i+1}(t), \quad i = 2,\ldots,S-2$$

$$\frac{dP_{S-1}(t)}{dt} = -(\lambda+(S-1)\mu)P_{S-1}(t) + \lambda P_{S-2}(t)$$

$$\frac{dP_S(t)}{dt} = \lambda P_{S-1}(t)$$

Taking the Laplace Transform of $P_i(t)$ denoted by $P_i^*(S)$, we have :

$$(s+\lambda)P_0(S) = \mu P_1^*(S) + 1$$

$$(s+\lambda+\mu)P_1^*(S) = \lambda P_0^*(S) + 2\mu P_2^*(S)$$

(A4) $$(s+\lambda+i\mu)P_i^*(S) = \lambda P_{i-1}^*(S)+(i+1)\mu P_{i+1}^*(S) \qquad i = 2,\ldots,S-2$$

$$(s+\lambda+(S-1))P_{S-1}^*(s) = \lambda P_{S-2}^*(s)$$

$$sP_S^*(s) = \lambda P_{S-1}^*(s)$$

$Q_i \triangleq -\frac{dP_i(s)}{ds}\Big|_{s=0}$ is the expected time spent in state i. We have :

(A5) $$E_S^0(t) = \sum_{i=0}^{S-1} Q_i$$

By taking the derivative at s=0 in Eq. (A4) we obtain the following system of equations which solves for the Q_i's :

$$\lambda Q_0 = \mu Q_1$$

$$(\lambda+\mu)Q_1 = \lambda Q_0 + 2\mu Q_2$$

(A6) $$(\lambda+i\mu)Q_i = \lambda Q_{i-1} + (i+1)\mu Q_{i+1} \qquad i = 2,\ldots,S-2$$

$$\lambda+(S-1)\mu\ Q_{S-1} = \lambda Q_{S-2}$$

$$1 = \lambda Q_{S-1}$$

One easily shows that the solution to the system of Equations (A6) is :

(A7) $$Q_i = \frac{1}{\lambda}\left(\frac{\lambda}{\mu}\right)^i / i! \sum_{k=i}^{S-1} \left(\frac{\mu}{\lambda}\right)^k k! \qquad i = 0, 1,\ldots,N-1$$

Substituting Eq. (A7) into Eq. (A5), we have :

(A8) $$E_S^0(t) = \frac{1}{\lambda} \sum_{k=0}^{S-1} \left(\frac{\mu}{\lambda}\right)^k k! \sum_{i=k}^{S-1} \left(\frac{\lambda}{\mu}\right)^i / i!$$

Eq. (A8) reduces to :

$$E_S^0(t) = \frac{1}{\lambda} \sum_{k=0}^{S-1} \left(\frac{\mu}{\lambda}\right)^k k! \sum_{i=k}^{S-1} \binom{i}{k}$$

$$E_S^0(t) = \frac{1}{\lambda} \sum_{k=0}^{S-1} \left(\frac{\mu}{\lambda}\right)^k k! \binom{S}{k+1}$$

Observing that $\binom{S}{k+1} = \binom{S-1}{k+1} + \binom{S-1}{k}$, we have :

(A9) $$E_S^0(t) = E_{S-1}^0(t) + \frac{1}{\lambda} \sum_{k=0}^{S-1} \left(\frac{\mu}{\lambda}\right)^k k! \binom{S-1}{k}$$

Observing that the second term of the right-hand side of Eq. (A9) reduces to :

$$1/\lambda \left[\sum_{k=0}^{S-1} (\lambda/\mu)^k/k!\right] / \left[(\lambda/\mu)^{S-1}/(S-1)!\right] = 1/\lambda P_1^{S-1},$$

We finally obtain Eq. (A3). Q.E.D.

COROLLARY : The expected time $E_S^j(t)$ before the queue M/M/∞ includes for the first time S customers given that the queue size is initially equal to j ($< S$) is :

$$E_S^j(t) = \frac{1}{\lambda} \sum_{i=j}^{S-1} \frac{\sum_{k=0}^{i} (\lambda/\mu)^k/k!}{(\lambda/\mu)^i/i!} \tag{A10}$$

Indeed $E_S^0(t) = E_j^0(t) + E_S^j(t)$.

Eq. (A10) is precisely Eq. (11).

APPENDIX B

Theorem :

$\sum_{i=0}^{\infty} P_b^i(\alpha b)$ is a decreasing function of b integer for a given value of α.

Proof :

It suffices to show that for all $i > 0$, $P_b^i(\alpha b) > P_{b+1}^i(\alpha(b+1))$. From Eq. (12), we must have for i=1 :

$$\frac{(\alpha b)^b}{b!} \sum_{k=0}^{b+1} \alpha(b+1)^k/k! > \frac{[\alpha(b+1)]^{b+1}}{(b+1)!} \sum_{k=0}^{b} (\alpha b)^k/k!$$

Then it is enough to show that :

$$\frac{(\alpha b)^b}{b!} \frac{\alpha^k (b+1)^k}{k!} > \frac{\alpha^{b+1}(b+1)^{b+1}}{(b+1)!} \frac{\alpha^{k-1} b^{k-1}}{(k-1)!} \qquad k=1,\ldots,b+1$$

or equivalently :

(B1) $$\frac{b}{k} > \frac{b+1}{b}^{b-k} \qquad k=1,\ldots,b+1$$

B1 is easily verified by induction.

Therefore $P_b^1(\alpha b)$ is decreasing with b increasing. Observing that

$$P_b^i(\alpha b) = P_{ib}^1(\frac{\alpha}{i} \times ib) > P_{i(b+1)}^1(\alpha(b+1)) = P_{b+1}^i(\alpha(b+1)),$$

the result follows for $i \geq 1$.

APPENDIX C

Lemma : $P_b^i(\alpha b) > 1 - \frac{i}{\alpha}$ for all $i < \alpha$

Proof : For $i = 1$, one must have from Eq. (12) :

$$(\alpha b)^b/b! > \sum_{k=0}^{b} (\alpha b)^k/k! - \frac{1}{\alpha} \sum_{k=0}^{b} (\alpha b)^k/k!$$

or equivalently :

$$\sum_{k=0}^{b-1} (\alpha b)^k/k! < \frac{1}{\alpha} \sum_{k=0}^{b} (\alpha b)^k/k! = \frac{1}{\alpha} + \sum_{k=1}^{b} (\alpha b)^k/k!$$

It is enough to show that for $k=0,\ldots,b-1$

$$\frac{\alpha^k b^k}{k!} < \alpha^k \frac{b^{k+1}}{(k+1)!}$$

which is true since $b \geq k+1$.

Therefore $P_b^1(\alpha b) > 1 - \frac{1}{\alpha}$. The result follows for $i > 1$ since

$$P_b^i(\alpha b) = P_{ib}^1 \left(\frac{\alpha}{i} \times ib\right) > 1 - \frac{i}{\alpha}$$

Theorem : $C_{per}^{\infty} > C_{dyn}^{\infty}$

Proof : From Eqs. (16) and (17), we must have :

$$\alpha \sum_{i=0}^{\infty} [Q_i(\alpha b) - P_b^i(\alpha b)] \leq \sum_{i=1}^{\infty} i\, Z_i(\alpha b)$$

where Q_i, Z_i and P_b^i are respectively given by Eqs. (8), (9) and (12). It suffices to show that for all $i > 0$:

$$\alpha(Q_i - P_b^i) \leq i\, Z_i$$

or equivalently

$$\alpha Z_i \frac{(1 - Q_i)}{1 - Z_i} \leq i Z_i$$

$$(C1) \quad \alpha(1 - Q_i) < i(1 - Z_i)$$

(C1) is trivially verified if $\alpha \leq i$.

Otherwise ($\alpha > i$) (C1) is equivalent to :

$$(C2) \quad \alpha \sum_{k=0}^{ib-1} (\alpha b)^k/k! < i \sum_{k=0}^{ib} (\alpha b)^k/k!$$

Substituting Eq. (12) into (C2) and dividing both sides, by $\sum_{k=0}^{ib} (\alpha b)^k/k!$ we get :

$$\alpha[1 - P_b^i] < i$$

which is true by the above Lemma. Q.E.D.

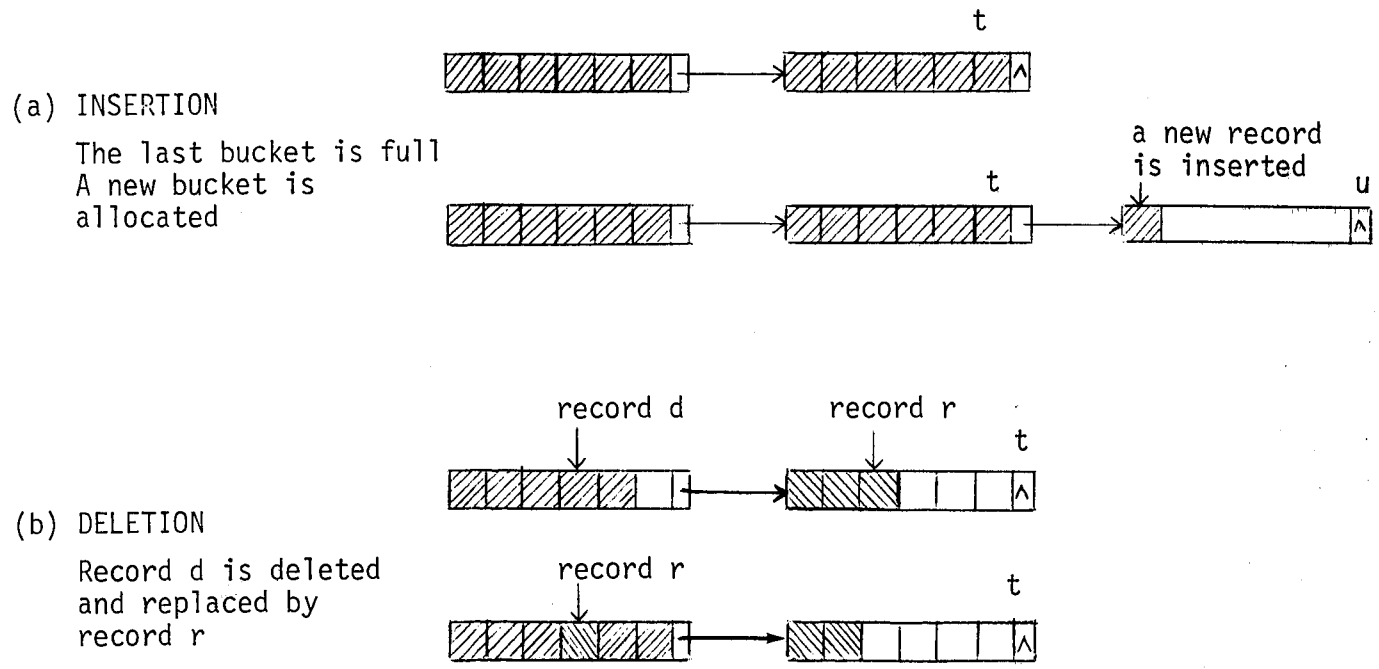


Figure 1 : Dynamic Reorganization ; Insertion/Deletion in chain m :
H(K) = m, m=1,...,M, b=6

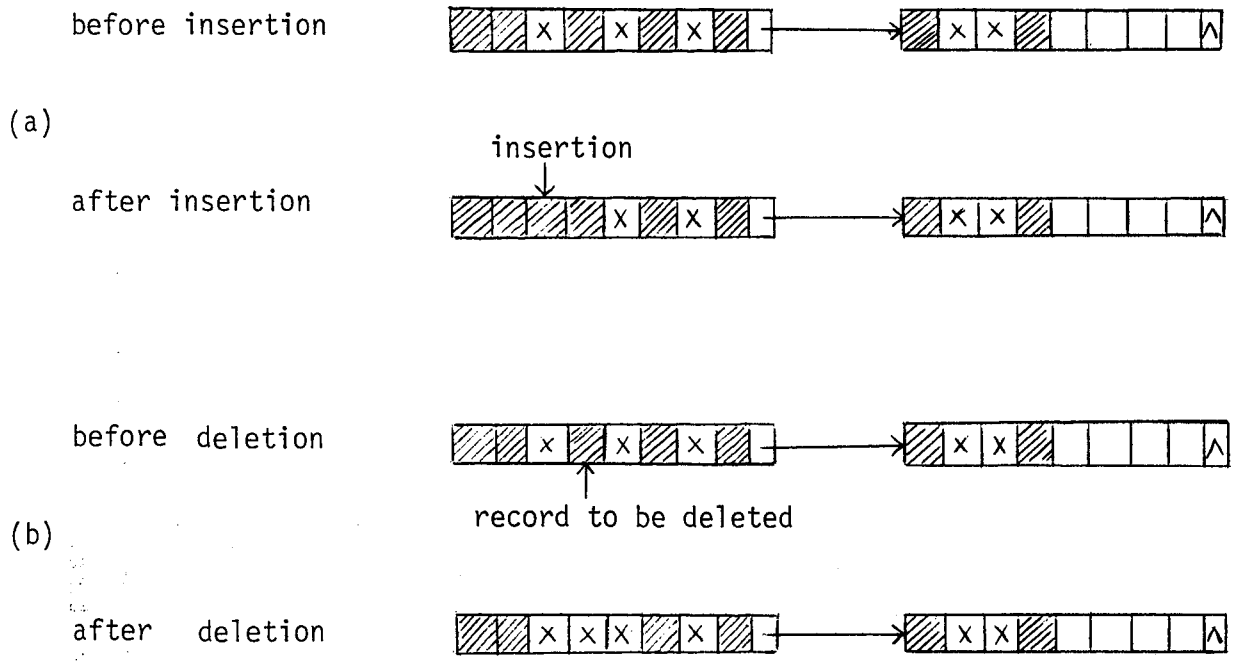


Figure 2 : Periodical Reorganization : Insertion/deletion in chain m, m=1,...,M
record actually stored, record deleted, empty cell

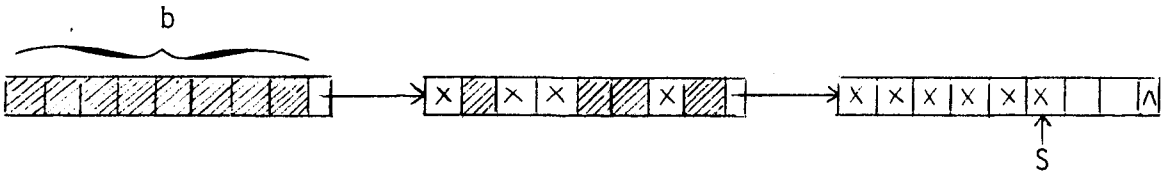


Figure 3 : Periodical Reorganization : S = 3b-2 cells are occupied in chain m

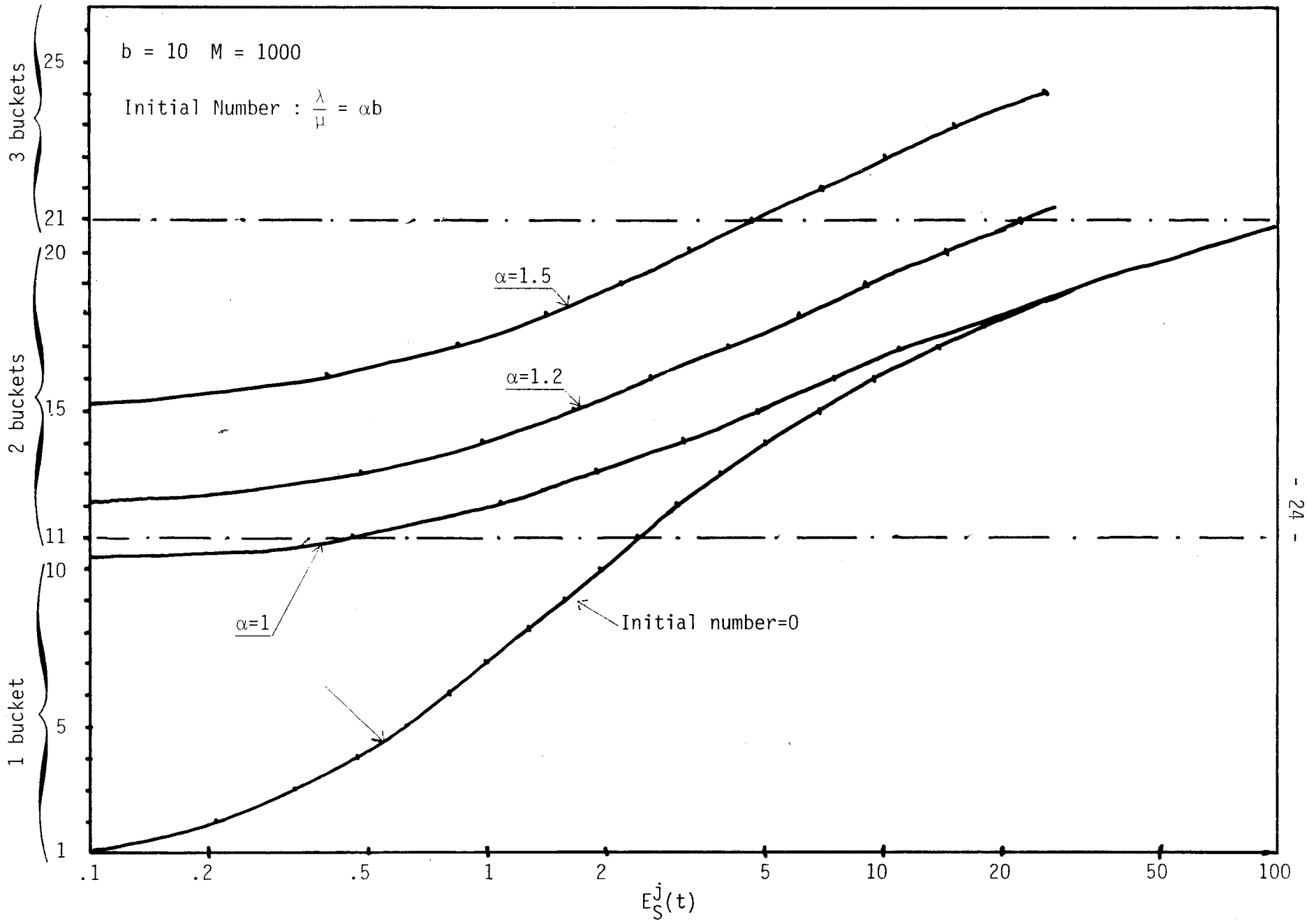


Figure 4 : $E_S^j(t) \triangleq E$ {time before occupation of the S^{th} cell/j 1st cells are occupied at time 0}
Time unit : $\frac{1}{\mu}$ = mean record life time.

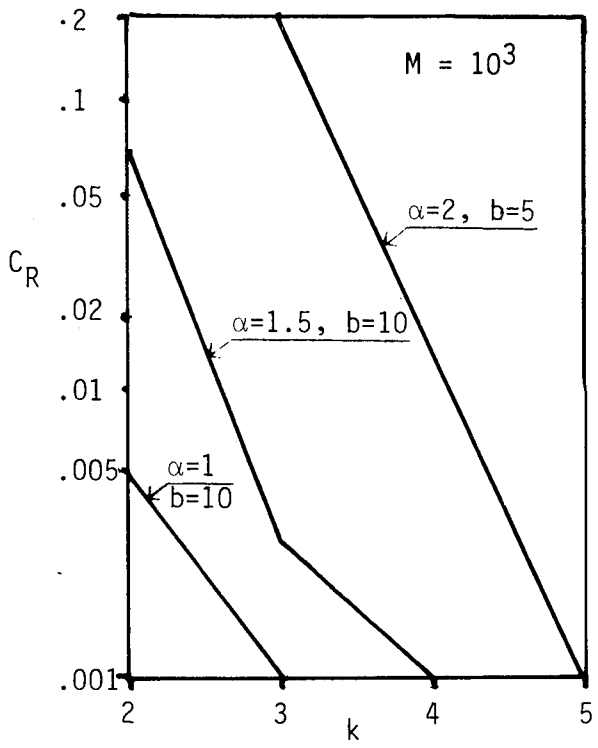


Figure 5 : Expected reorganization Cost versus k (Time unit : $1/\Lambda$)

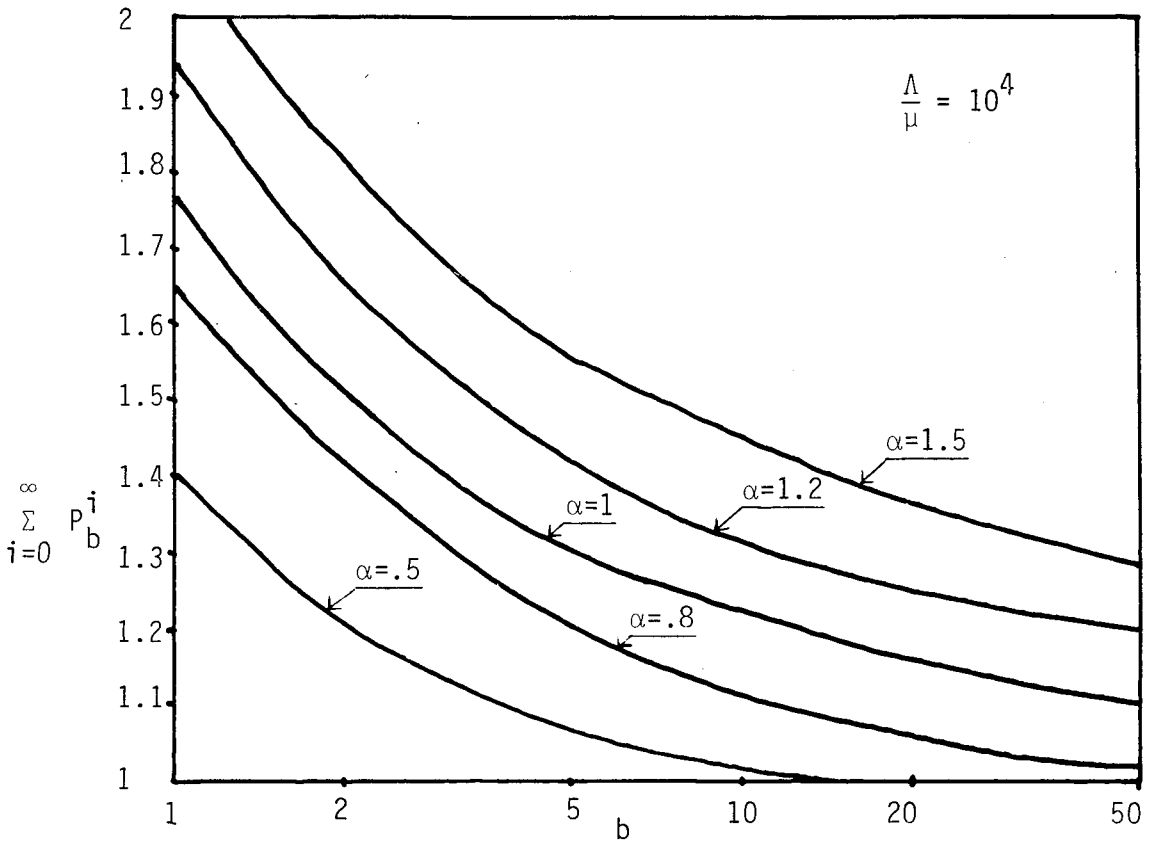
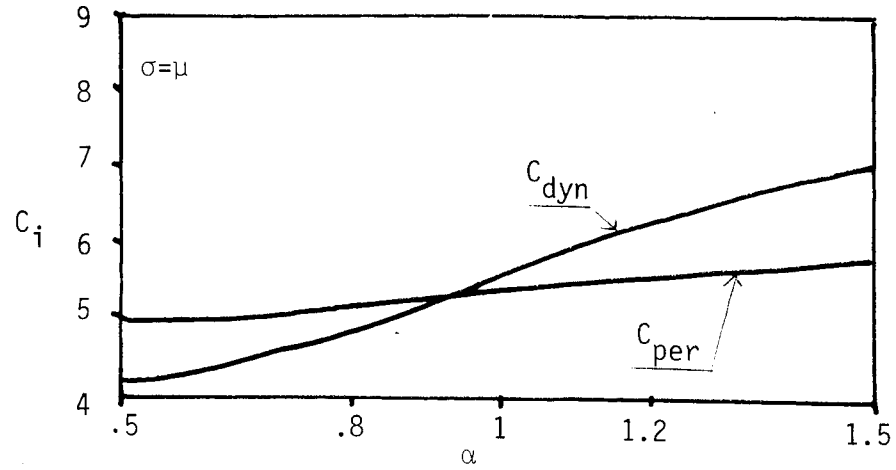
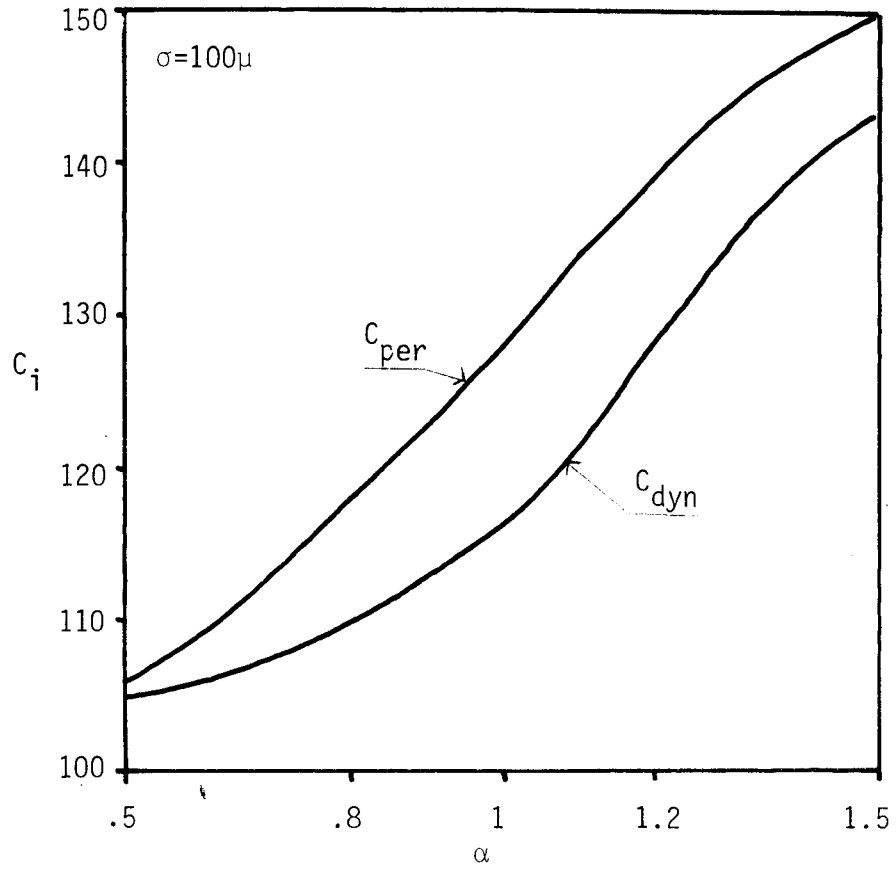


Figure 6 : Periodical Reorganization : Expected number of accesses to find all records versus bucket capacity



(a) high insertion/deletion activity



(b) low insertion/deletion activity

Figure 7 : Dynamic versus Periodical reorganization, M=1000, b=10

α	Λ/μ	ΣP_b^i	ΣQ_i	$1/\alpha \Sigma i Z_i$	$(C^{\infty}_{per}-C^{\infty}_{dyn})/C^{\infty}_{dyn}$
.8	8000	1.13	1.28	.23	.08
1	10000	1.23	1.55	.44	.10
1.2	12000	1.34	1.78	.56	.09
1.5	15000	1.45	2.06	.70	.07
1.8	18000	1.60	2.34	.84	.06

Table 1 : Periodical versus dynamic reorganization : High searching activity ; b=10, M=1000.

σ/μ \ α		.8	1	1.2	1.5
.1	C_{per}	5.24	5.35	5.47	5.59
	C_{dyn}	4.90	5.65	6.23	6.90
1	C_{per}	6.26	6.46	6.68	6.90
	C_{dyn}	5.85	6.64	7.33	8.13
100	C_{per}	118	128	139	150
	C_{dyn}	110	116	128	143
1000	C_{per}	1135	1235	1345	1455
	C_{dyn}	1059	1110	1222	1367

Table 2 : Periodical versus Dynamic Reorganization ; b=10, M=1000

REFERENCES

[1] D.E. KNUTH - "The art of computer programming" - Vol. 3 - Sorting and Searching - Addison - Wesley - Reading - Mass. - 1973.

[2] C.A. OLSON - "Random Access File Organization for Indirectly Addressed Records" - Proceedings of the ACM National Conference - 1969 - pp. 539-549.

[3] J.A. VAN DER POOL - "Optimal Storage Allocation for a File in Steady State" - IBM J. Research Develop. - January 1973 - pp. 27-38.

[4] M. STONEBRACKER, E. WONG, P. KREPS, G. HELD - "The Design and Implementation of INGRES", ACM Transactions on Database Systems - Vol. 1 - N° 3 - September 1976 - pp. 189-222.

[5] L. KLEINROCK - Queueing Systems - Vol. 1 : Theory - Wiley - Interscience - New York - 1975.

[6] L. TAKACS - "Introduction to the Theory of Queues" - Oxford - University Press - New York - 1962.