UNIVERSITÉ DE MONTRÉAL

MEMORY STRUCTURES FOR VIDEOTEX SYSTEMS

by

R. Nigel Horspool*, Gregor v. Bochmann**

and G. Eugene Saunders**

Publication #454

DÉPARTEMENT D'INFORMATIQUE

ET DE RECHERCHE OPÉRATIONNELLE

Faculté des arts et des sciences
Université de Montréal
C.P. 6128, Succursale "A"
Montréal, P.Q.
H3C 3J7

MEMORY STRUCTURES FOR VIDEOTEX SYSTEMS

by

R. Nigel Horspool*, Gregor v. Bochmann**
and G. Eugene Saunders**

Publication #454

* School of Computer Science, McGill University
** Département d'informatique et de recherche opérationnelle,
Université de Montréal

This report is prepared under contract no. 20SU.36100-1-0169 of
between the University of Montreal and the Department of Communications, Canada.

Département d'informatique et
de recherche opérationnelle
Université de Montréal

Octobre 1982

INDEX

1. INTRODUCTION

Videotex systems have three basic components: (a) a service providing host computer, usually including a page-oriented database, (b) a data transmission system, and (c) user terminals. This paper concentrates on the organization of the memory structures that may be found in the host computers to support the database (the information pages and various directory structures) and the data buffering required for transmission. The memory structures for distributed data base systems are also considered.

As far as the transmission system is concerned, two major approaches are considered: (a) "interactive videotex" and (b) "teletext". In addition, two major transmission media considered for interactive videotex: telephone lines and TV transmission channels. Interactive videotex differs from teletext in that pages are transmitted only in response to requests from users. In particular, a page may be transmitted only to the particular user who requested it. There are currently two major methods for transmitting videotex pages. One method is to use standard telephone lines and the other method is to use spare channels on cable television networks. These two methods have quite different transmission rates (1200 bps versus 4,000,000 bps). Therefore, we can expect different implementations and hardware configurations for the videotex computer system. However, many of the issues and implementation decisions are similar.

While initial videotex databases only included menu selection user interfaces, keyword and other kinds of user interfaces are currently considered in addition for newly developed videotex systems [41]. Sections 2 and 3 of this paper investigate issues related to the memory structures related to simple menu-oriented databases of information pages. In section 4, issues related to the support of keyword access (the simple kind of keyword interface [17]), and full keyword facilities with boolean search are reviewed.

Like traditional database systems, videotex databases may be viewed at different levels of the architecture [42], for instance the internal, conceptual, and external level. While the user interface is related to the database structures at the conceptual and external levels, the internal level is concerned with the physical organization of the data within the database, and the mechanisms used to provide the conceptual data structures to the application programs and users.

In section 2, we discuss the memory structures in central memory and secondary storage used to support the storage of the information pages and support their retrieval in an efficient way. A memory hierarchy and different ways of managing it are considered. Using some empirical mesurements on usage patterns, the optimization of the storage structures are demonstrated for some examples.

In section 3, the memory requirements for the page transmission to the users are considered, distinguishing the different transmission approaches over telephone and TV facilities. An additional problem in the case of teletext with a broadcast cycle of pages is the question of which pages to put into that cycle, in particular in such systems where additional pages may be available to the user through an alternate transmission scheme. An analysis similar to the one applied to the memory hierarchy in section 2 is shown to be useful here.

In section 4, extensions to the user interface are considered with their impact on the memory structures for supporting efficient access in the case of a large number of users. It is in this context that special purpose database machines could possibly be useful.

In section 5, the ideas of section 2 are extended in considering the configuration of a distributed videotex system. In particular the design and performance of the user interface machine is considered.

2. STORAGE STRUCTURES FOR VIDEOTEX

2.1 The Memory Hierarchy

Let us first consider the largest use that memory will be put to. The system will normally include a large store of pages. One source [18] estimates that a minimum of 50,000 pages are needed for the initial implementation of a videotex system and that the number would have to approach 500,000 pages if mass market success is wanted. We estimate that the average size of a page will be between 1000 and 2000 bytes. It is hard to be more precise than this because there will be many index pages with a small size and there will be an indeterminate number of pages containing picture descriptions and will therefore be quite large. For the moment, we will use 1000 bytes as a lower bound on the average page size. Hence, 50000 * 1000 = 50 Mbytes is an estimate of the minimum storage capacity of the system. Since 50 Mbytes is rather large for the main memory of a computer, it is clear that a mass storage device such as a disk must be used. However, it is reasonable to assume that a fairly high proportion of pages can be kept in the main memory or some other fast memory device.

Thus, we have obtained a convincing reason for why a standard videotex system must contain at least two kinds of memory. These would be the main memory of the computer and its disk memory. As we will discuss later, it would probably be cost-effective to attach a separate mass memory device with an

access time that is intermediate between main memory and disk access times. The configuration of these three kinds of memory would form a memory hierarchy. It is important to utilize the hierarchy efficiently, meaning that we should carefully choose which memory level is to contain which items of information (either program or data).

If we consider just the placement of videotex pages into appropriate memory levels, there may, logically, be a fourth kind of memory to consider. If we consider a videotex system that uses a broadcast cycle, such as Ceefax or Oracle, then we must determine which pages are to be included in the cycle. We can logically consider this cycle to be another kind of memory that is more exclusive than main memory (because of the limited capacity of the cycle) but that gives faster access to subscribers.

2.2 Memory Required for User Context Information

Additional main memory must be allocated for user context information. For each active user, we can reasonably assume that the following details must be retained:

a. User identification.

b. User address (I/O address or routing information).

c. Accounting information.

d. Current page number and page header information.

e. Trace of previously accessed pages, etc.

We can attempt some minimum estimates for the storage requirements as follows. The user identification need consist of no more than an account number and 4 bytes would be more than adequate for this purpose. (Presumably, the account number indexes a master file of subscribers that is held on a non-volatile storage medium such as a disk.) The user I/O address could probably be packed into 4 bytes too. Accounting information for an active session should, at least, include the time of day that the user signed on (requiring 6 decimal digits or 3 bytes) and the number of page requests that have been made so far (2 bytes, say). The current page number is about 7 bytes. The 7-byte figure is calculated according to the assumption that page numbers are comprised of up to 13 decimal digits followed by, perhaps, a decimal point and three more digits. Page header information includes cross-references to other videotex pages where the Telidon hierarchical tree structure is violated. Perhaps space for up to 2 such cross-references should be allocated, but this figure is a pure guess. Assuming 2, we have two more page numbers (of 7 bytes each) to store. The minimum total storage thus works out to be 34 bytes per user or 34 Kbytes for 1000 active users. Compared to the resident page directories that we will be looking at next,

this is an insignificant amount of storage.

However, the 34 byte estimate represents an absolute minimum. It may be expedient for a practical system to retain more information about the user status. Also, there is provision in Telidon for "action pages". These action pages correspond to computer programs which interact with the user. Such a program and its data must be retained in memory until the user has finished with it (unlike a normal text or picture page that can be deleted once it has been sent to the user). If several subscribers are using the same action page, there need be only one copy of the program resident in memory but each subscriber would need his own data space. Without any experience in this regard, it is impossible to guess the popularity of action pages and to estimate their storage requirements.

2.3 The Page Directory Structure

Another general issue for videotex systems concerns the implementation of the page directory. A user can enter a page number as a decimal number. However, because the page numbers follow a special kind of hierarchy, the page numbers are quite sparsely distributed. If, as we have just discussed, a page number can consist of upto 16 decimal digits then the space of page numbers is 10**16 in size and is immense when compared to the total number of pages (between 50,000 and 500,000). A videotex system will necessarily have to provide a directory that would be

used to translate from the page number to a memory address for the page. This directory will have to be implemented by some method that is efficient in storage. The possibilities include hash tables [36], tree structures [36] and trie memory [16]. A hash table implementation appears to be the most economical in storage at the expense of some unpredictability in access times to look up entries.

For an example, let us suppose that we wish to provide for 50,000 pages. A hash table with 75,000 entries would enable us to look up an entry with two probes on average. (62,500 entries would give an average of three probes to the table.) Thus the amount of CPU time expended in table look-up need not be significant. The storage occupied by the table is of far greater importance. We would expect that each entry in the table would contain (at a minimum) the following information:

a. The page number (16 decimal digits is equivalent to 51 bits).

b. The memory residence of the page, main memory or disk (1 bit).

c. The address of the page, either a main memory or a disk address (24 bits seems a reasonable estimate).

d. The size of the page in bytes (about 16 bits).

e. Usage statistics or priority information for this page (about 16 bits).

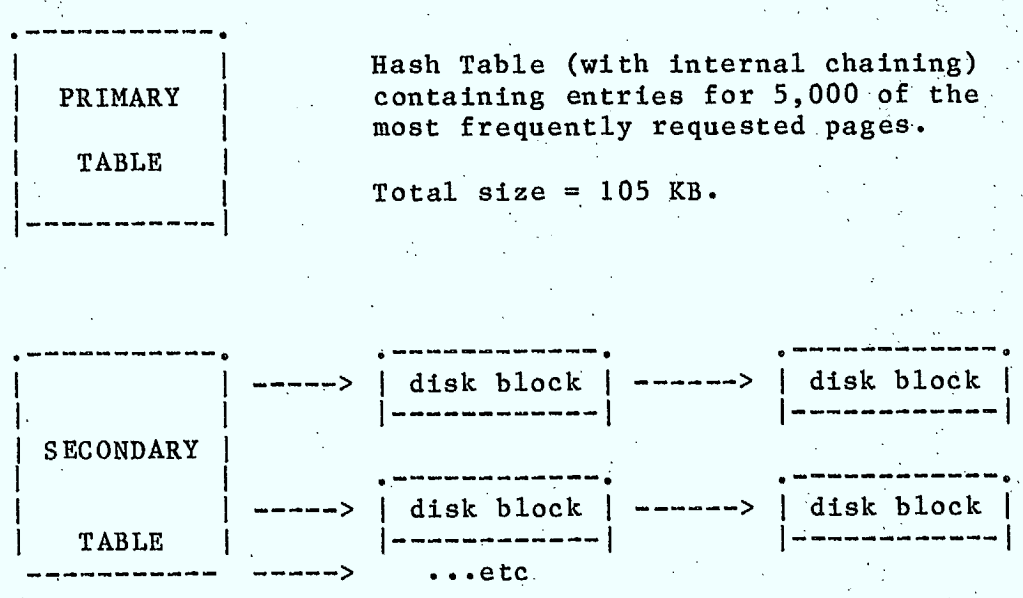
This gives a total of about 108 bits or 14 bytes. Thus, the size of the entire table would be approximately 14 * 75,000 = 1,050,000 bytes, that is, about 1 Mbyte. Although a megabyte of memory might be available, the table would occupy memory that could potentially hold 1000 videotex pages. Keeping an extra 1000 pages in main memory could conceivably reduce the number of pages read from disk by a large factor. However, if we do not keep the table in main memory, it would have to be held on disk and every page look-up would require an extra disk read. Probably, some compromise strategy is best. Analysis of such compromise strategies is difficult because it is very open-ended and because it is dependent on the usage statistics of the system. The required statistical information about the usage of existing videotex systems is usually not publicly available. Therefore, we will look in the following at just one viable possibility for splitting the directory between the two memory levels.

Let us suppose that the 90-10 rule applies to page requests so that 90% of all page requests are made to only 10% of the pages in the system. This rule appears to be approximately true for one experimental system (which is discussed in more detail later (39)). We will also assume that the system has 50,000

pages. If we set up a hash table that contained entries for only the 5,000 pages that account for 90% of all requests, its size would be only 105 Kbytes. To handle the other 10% of requests, we can set up a second table which corresponds to hashing with external chaining. The number of entries in this second table can be freely chosen, so let us use N to represent the number of entries. Each entry is a pointer to a hash bucket. Since the hash buckets are held on disk, the pointer is a disk address (requiring about 24 bits). The hash buckets on disk can be implemented as a linked-list of blocks, where each block contains information about some number of videotex pages and a pointer to the next block. The size of the block would probably be chosen to suit the hardware, let us assume 512 bytes which is sufficient to hold information on 36 pages. A look-up can be performed by a linear search through the entries in a bucket. A diagram of the data structure is shown in Figure 2.1. If N is chosen to be 50,000/36, then the average number of pages per bucket will be 36 (or one disk block). This implies that the second table will require 4,175 bytes. Probably, it is better to choose N to be larger, say double, so that we will rarely need to search through more than one disk block in a bucket. In fact, the entries in a bucket should be sorted according to their request frequencies, with the most popular pages at the fronts of the lists. This would tend to make the probability of having to read a second disk block even smaller. Therefore we can trade a 1 Mb table in main memory for about 115 Kb of tables in main memory, but where 10% of the look-ups will require a disk read (a very small proportion of

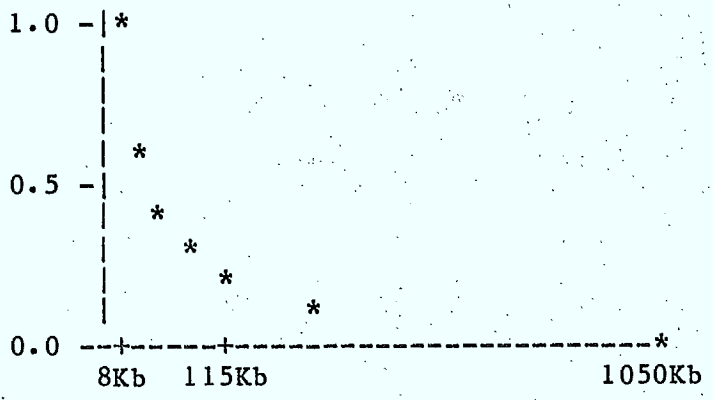
look-ups may require two disk reads). By varying the storage allocated to the tables in memory, we can change the trade-off as desired. A picture of the kind of trade-off that can be expected is shown in Figure 2.2.

It must be stressed that the selection of which page entries of the directory should be held in main memory as opposed to on disk is theoretically independent of the selection of which pages (the data itself) to keep in the fast memory and which should be held on disk. In practice, we would want to keep the entries for the pages that are the most often accessed in the primary table and the same consideration applies for selecting the pages to be held in fast memory. However, there is not necessarily any correlation between the number of entries in the primary table and the number of pages that are held in fast memory. It is not unreasonable to have directory information in the main memory about a page that normally resides on a disk. The converse is also possible but probably inefficient. (It seems unreasonable to have to access the disk to find an address in main memory for a page, this must surely represent an inefficient allocation of memory.)



Hash Table Organization for Page Directory

Figure 2.1



Expected number of disk accesses for a directory look-up versus main memory allocation (in Kbytes).

Figure 2.2

2.4 Page Placement in the Hierarchy

Choosing the best distribution of pages between two kinds of memory is a standard problem for memory hierarchies. However, the characteristics of the videotex system are unlikely to be very similar to other systems where memory hierarchies are provided. The videotex system is envisaged as having a large number of active users (say in excess of 1000) and each user is requesting pages relatively infrequently (about one every 10 seconds). Therefore, if we look at all the requests received over a short period of time (several seconds), we are unlikely to discover correlations between page requests.

For example, the fact that page 123 has just been requested by one user does not increase the probability that it will also be requested by another user in the near future. Also,

by the time that the first user has finished reading page 123 and now wishes to traverse the tree of pages to page 1237, say, there will have been hundreds of other requests received from the other videotex subscribers. Although there may be a strong probability that a user will request page 1237 after page 123, this correlation will go almost unnoticed in a system with a thousand or more users and it is not worthwhile for the system to try to exploit such correlations. It is simpler to consider page requests to be independent random selections drawn from some underlying probability distribution. In this situation, the optimal implementation is to simply place the pages with the highest request frequencies in the fast main memory and put all other pages in the slower backing memory (1). We would, however, expect page request frequencies to depend on the time of day and on other external factors. For example, weekend users might request information pages on sports scores the most whereas weekday afternoons may be dominated by users concerned with stock market closing prices. This factor suggests that the distribution of pages between the two kinds of memory should be allowed to change. There are various page replacement policies that may be suitable for automatically determining the distribution of pages. Some prime candidates are:

LRU The least recently used pages are kept in the slower backing memory. One implementation of this policy keeps a record of the time of last use for each page in the main memory. When a request is made for a page that is not in main memory, the page is transferred to main memory. If space in main memory

must be obtained first, it is obtained by deleting one or more pages that have not been used for the longest period of time.

LFU The least frequently used pages are kept in the slower memory. The implementation requires that a count of uses be kept for each page in the system. When space in main memory must be found, pages currently in memory with the lowest usage counts are deleted.

CLIMB The pages in the system are sorted into a priority order that is continually updated. Whenever a page is used, it is advanced one position higher in the priority list (overtaking one page). Obviously, a page at the top of the list cannot be advanced any higher when it is used. When space in main memory must be obtained, the page or pages in memory that are lowest in the priority list are deleted.

The LRU and LFU policies are standard paging strategies and are described in many references [13]. CLIMB is not so well-known, it is described and analyzed in only a few references [15, 32]. LFU would be the optimal management policy if there were no time factor present in the probability distribution function for page requests. When there is time dependence, LFU can be quite poor because an increased page request probability will not be acted upon until the usage count of that page has overtaken the counts for other pages in the system. Informally, we might say that LFU is poor because it is too stable. LRU would be expected

to operate more satisfactorily because it responds to changed page request probabilities very quickly. However, LRU can be criticised for being too unstable. That is, an isolated request for an unpopular main memory for a adapt as fast as LRU. That is, CLIMB is like a stable version of LRU but not as stable as LFU. CLIMB is named the "transposition heuristic" in [32] because it can be implemented by transposing adjacent elements in a priority list of the pages.

However, CLIMB has a big drawback compared to LRU. To implement LRU we need an ordered list of only the pages that are currently held in the fast memory. To implement CLIMB, we need an ordered list of all pages in the system and this list is too long to be practical. Even if the pages are numbered from 1 to 50,000, we would need 100Kb of storage for the list. This amount of storage would be better employed for holding an extra 60 to 100 pages in memory. It should be possible to implement a policy that is a compromise in both storage and performance between LRU and CLIMB. The ordered list should be longer than that required for just the pages resident in memory but much shorter than the size needed to list all pages. When a page not on the list is accessed, it is inserted into the list and the page that was previously in the last position is deleted from the list. To prevent a page from being pushed out of the list too soon after it has been inserted in the list, we should not insert a newly referenced page at the very bottom of the list. It should be inserted higher up. Also, each time a page is referenced, it

should advance more than one position in the list ordering. We suggest that the number of positions to advance should be expressed as a fraction of the page's current position in the list. However, the selection of values for the free parameters in this policy will require some experience or some detailed simulation experiments.

2.5 Most Cost-Effective Memory Configuration

2.5.1 Discussion of possible configurations

If the computer system has only main memory and a single disk unit, the disk will limit the overall system performance. To justify this assertion, consider a system with 1250 users who, collectively, generate 125 page requests per second. The disk can handle only about 25 page transfers per second (assuming typical disk speeds). Therefore, 80% of all page requests would have to be served from main memory. But this would imply that the system requires a rather large amount of main memory. If the 90-10 rule holds for a system with 50,000 pages, we would need to retain about 5,000 pages in main memory to satisfy 90% of page requests. These 5,000 pages would occupy 5 to 7.5 Mbytes, which is a quite feasible but large amount of main memory by current standards.

Any improvement in the transfer rate of pages from secondary storage into main memory would be reflected in a reduced requirement for main memory. There are two obvious ways of

improving the effective transfer rate:

a) attach additional disk drives to the system, or

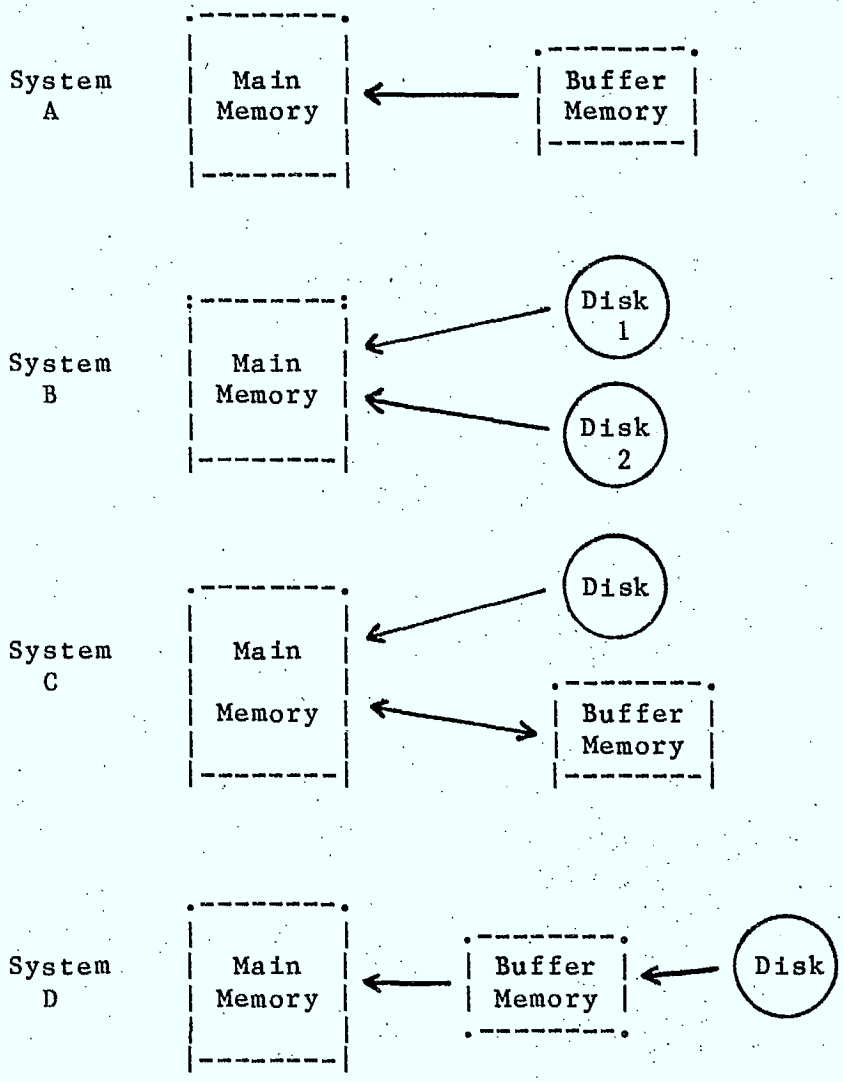
b) attach a semiconductor mass storage device. Such devices are often provided as direct replacements for disk or drum units but use LSI, CCD or Bubble technology and are therefore much faster. (We will refer to this kind of device as a buffer memory.)

These possibilities lead to four general system architectures to be evaluated from economic, cost-effectiveness, considerations. The various configurations are diagrammed in Figure 2.3. System A represents a system where the buffer memory simply replaces the disk drive. System B represents a system with replicated disk drives (we will consider two disks to be representative of systems with three or more disks). System C shows both a disk and a buffer memory connected to the main memory. System D shows a memory hierarchy where pages may only be transferred between the disk and the buffer memory and between the buffer memory and the main memory. A fairly sophisticated controller (independent of the main CPU) is needed to manage the page transfers.

Now system D can be immediately discounted as inferior to system C (given identical disk units and buffer memories in the two systems). This is essentially because pages normally resident

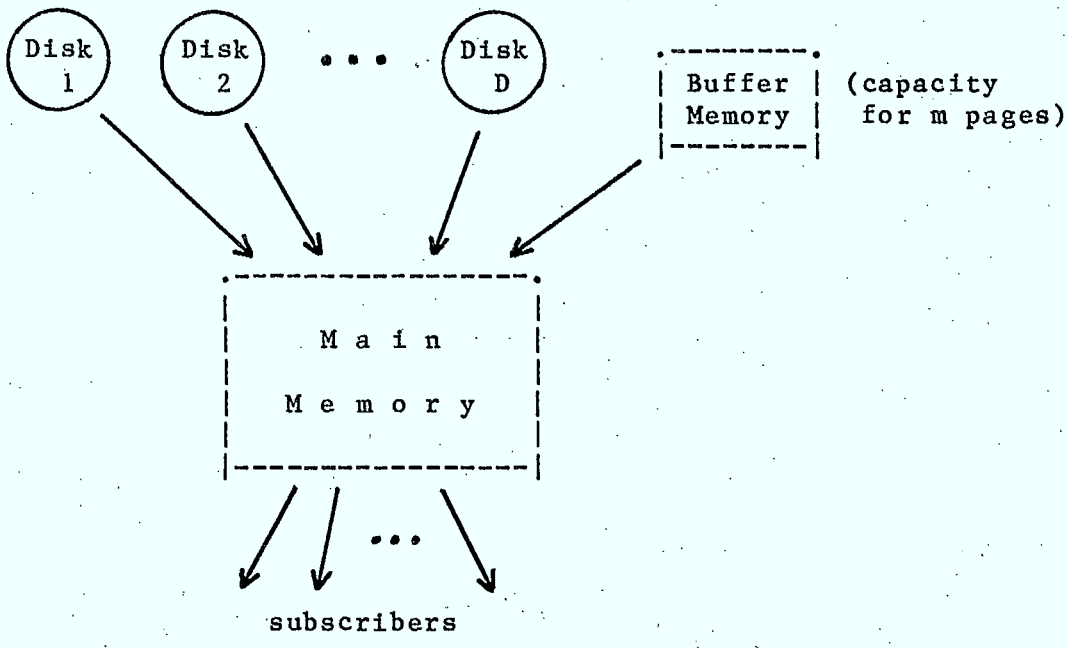
on the disk must be transferred in two steps for system D, whereas only one step is needed for system C. Also, some slots in the buffer memory of system D must be reserved for pages passing through into main memory, whereas no such provision need be made in system C. In fact, there are simulation studies for conventional virtual memory systems that compare the two configurations corresponding to C and D [8, 9]. The results show that configuration C can be used much more effectively than D and achieves a large improvement in the efficiency of memory usage.

We can consider systems A and B as variations on system C. Let us consider a generalized version of C which has a buffer store (with unspecified capacity) and D disk drives attached as drawn in Figure 2.4. Let us consider only the problem of determining page placement in the hierarchy.



Possible Memory Hierarchies

Figure 2.3



General Memory Hierarchy

Figure 2.4

2.5.2 Cost-effectiveness for the general memory hierarchy

We consider now the general memory hierarchy of figure 2.4, and assume that transfer rate of the buffer memory is sufficient to serve all user page requests. We also assume that the transmission rate to the subscribers is faster than the transfer rate from the buffer memory. (Additional output buffers are required in the opposite case, as discussed in section 3.3 below).

It is first noted that no pages need be held in the main memory. It is possible to fetch a page from the buffer memory

while another page is being transmitted to a subscriber. Thus we could theoretically make do with just enough main memory to hold two pages. (One memory slot contains the page being transmitted and the other slot is used to receive the page being read from the buffer memory.) In practice, the timing constraints imposed by this ideal situation may be too onerous and additional main memory would be appropriate. However, it is clear that no significant amount of main memory should be reserved for pages if buffer memory is cheaper (as would be the case) and provided that the CPU and I/O channels have enough unused capacity.

The problem therefore reduces to determining the most desirable quantity of buffer memory and the most desirable number of disk drives, where all other characteristics of the system are given. Let us define the following quantities:

n = request rate for pages from all users (in pages per second).

N = total number of pages in the database.

m = the number of pages that are held in the buffer memory.

rd = the transfer rate of information from a disk to the main memory (in pages per second).

cd = the cost of one disk drive (in dollars).

cp = the cost of sufficient buffer memory to hold one page (in dollars).

fb(m) = the probability that a request for a page can be satisfied from the buffer memory (rather than from one of the disks).

Now, the major requirement of the system is that it must be able to handle the processing load that is imposed upon it. Thus, we can write the following approximate equation which relates the available transfer rate from all disks to the page demands by the subscribers:

rd * D >= [1 - fb(m)] * n (Eqn. 1)

This equation is inexact because disk usage actually becomes more efficient as the load is increased. Thus, two disks will yield less than twice the overall transfer rate of one disk. (The increase of efficiency is due to the fact that disk seeks can be scheduled better if there are more requests to choose between in the request queue.) Also, we have not considered whether the database of pages should be totally replicated on each disk or whether each disk should hold only a portion of the database. (Our formula is more appropriate for the replicated database case.)

Experimental observations on human behaviour and on computing systems have repeatedly discovered that measurements appear to follow Zipf's Law [38, 31]. The pattern of usage for videotex pages is no exception. We have obtained data from one (small-scale) videotex system [39] and have found a very close agreement between the observed page reference frequencies and the frequencies that would be predicted by Zipf's Law. This law predicts that the frequency of access to the k-th most popular page should be proportional to 1/k. Equivalently, the cumulative probabilities of access to the k most frequently accessed pages should be a linear function of log(k). We have graphed the cumulative probability function for the experimental Telidon system against log(k) in Figure 2.5. The graph is indeed linear until about 400 pages are accounted for (representing 83% of all requests). After this point, the probability fails to increase as rapidly as the formula predicts. This is actually a common occurrence for phenomena that seem to follow Zipf's Law. The dropping away of the curve from the straight line is known as the "Groos Droop" [19, 7]. The curve corresponds to a 88-12 rule (i.e. 88% of requests are to only 12% of the pages).

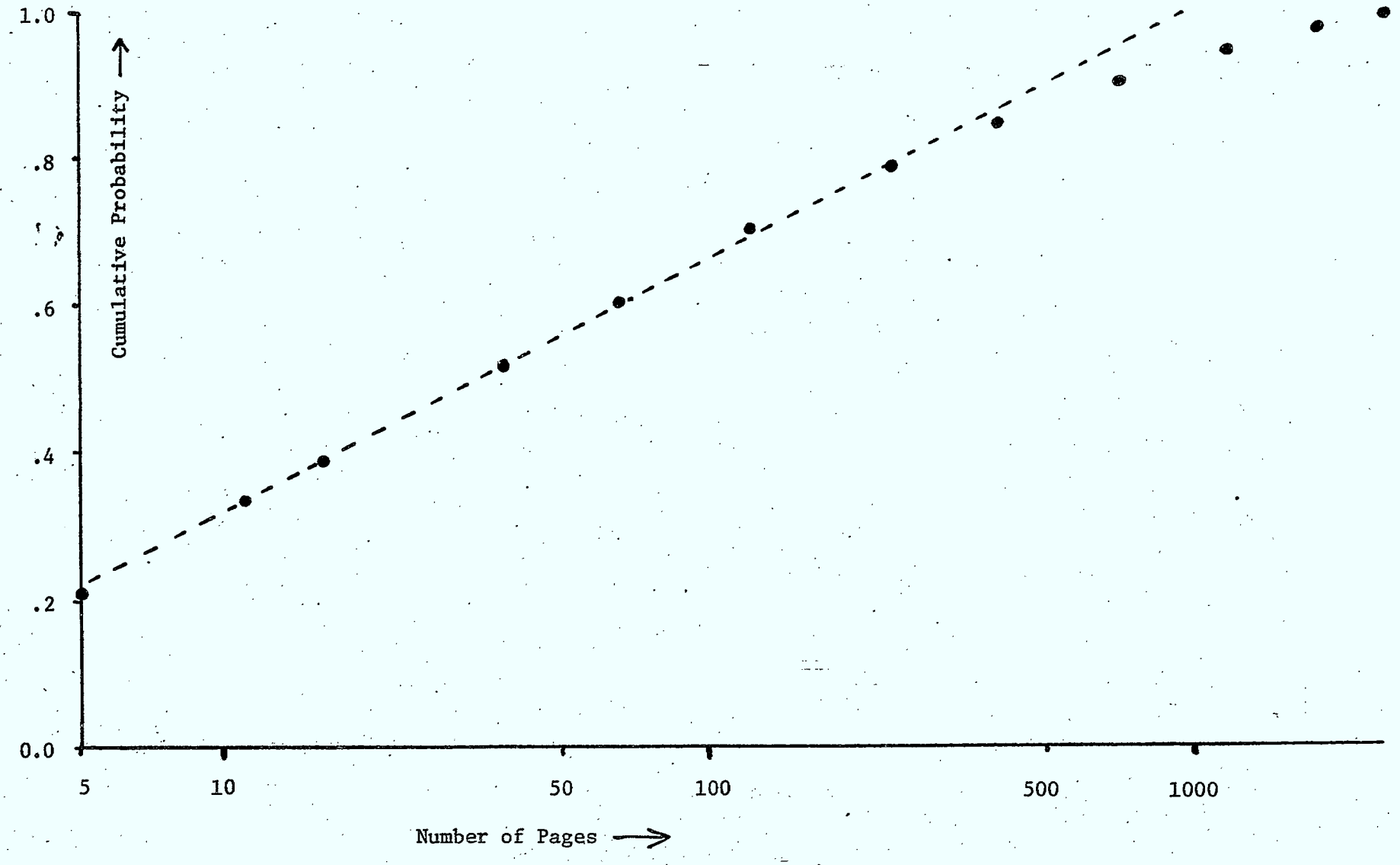

1.0
.8
.6
.4
.2
0.0
Cumulative Probability
5
10
50
100
500
1000
Number of Pages

Figure 2.5

If storage in the Telidon system is allocated optimally, we would retain only the most frequently accessed pages in a fast memory. Less frequently accessed pages would be relegated to the slower but cheaper disk memory. Therefore, a sensible storage allocation policy, as discussed in section 2, implies that the cumulative probability function, fb(m), as defined earlier, fits the Zipf Law. More formally, the Bradford-Zipf distribution specifies the following form for the cumulative probability function:

$$fb(m) = h \log(m/u + 1) + fb(0) \qquad \text{(Eqn. 2)}$$

The parameters, h and u, represent constants. The logarithms are to base e. For the system being modelled, fb(0) ideally should be zero but may not be, due to an imperfect match between the equation and reality. (This particular form of the Bradford-Zipf distribution is known as the "Yield Formula" [20].) The values of h and u cannot easily be predicted for a full-scale videotex system. Our data from the small system fits the following equation:

$$fb(m) = 0.151 \log(m / 0.731 + 1) - 0.091$$

This formula is accurate only for values of m that do not exceed 400. For larger values of m, the Groos Droop appears and the equation becomes inaccurate. (Therefore, we should be suspicious of any results that we derive where larger values for m are explicitly or implicitly assumed.) The method for calculating h, u and fb(0) is given in Appendix A1.

We can estimate the memory cost of our system, C, in dollars with:

$$C = D*cd + m*cp \qquad \text{(Eqn. 3)}$$

Note that C excludes the cost of memory needed to hold programs, page directories and other system data.

Supposing for the moment that D need not be an integer, the requirement of minimal cost would force equality to hold in Equation 1. Therefore, we can rewrite the equation as:

$$D = \frac{n * [\ 1 - fb(m)\ ]}{rd} \qquad \text{(Eqn. 4)}$$

This gives D as a function of m. Substituting for D in Equation 3 and differentiating with respect to m yields:

$$\frac{dC}{dm} = \frac{-\ n * h * cd}{rd * (\ m + u\ )} + cp \qquad \text{(Eqn. 5)}$$

It is clear that the second derivative of C is positive for all m (since n, h, u and rd are all positive quantities). Therefore, if we equate the first derivative to zero and solve for m, we obtain a unique minimum in the cost function. This solution for m is:

$$m = \frac{n * h * cd}{rd * cp} - u \qquad \text{(Eqn. 6)}$$

If we numerically determine m from Equation 6 and substitute for m in Equation 4, we will certainly find the optimal number of disks, D, to be a non-integer. Therefore, we will have to round D up and down to the two adjacent integers and check to see which gives the lower system cost. (This procedure yields the true minimum cost system because of the concave shape of the cost function.)

As an example, let us consider the small system for which we have obtained the page access probability function. We shall assume the following data:

cd = \$10000 (a low estimate),

cp = \$8 (approximately 0.1c per bit),

N = 4000,

rd = 25 (i.e. 40msec per disk transfer),

n = 100.

Inserting these values into Equation 6 leads to the result that the optimal value of m is 754. This is in a region where our fitted equation is a little inaccurate. However, we cannot actually use this value of m anyway. Substituting this value of m into Equation 4 yields D=0.17. We clearly cannot purchase 17% of a disk (with 17% of its performance for 17% of the cost) and so we consider D=0 and D=1 to find an achievable minimum cost system. (We ignore here the possibility of sharing part of the disk space with other applications). The memory cost when D=0 is given by Equation 3 as 4000*8 = \$32,000. To find the

cost when D=1, we must first determine from Equation 1 that we want a value of m that yields fb(m) = 0.75. From Equation 2 or from Figure 2.5, we see that m=191 yields the desired value of fb(m). Therefore the memory cost for D=1, m=191 is $11,528. (Since m=191 lies within the range of applicability for the probability function, we can trust this result.) Therefore, the memory configuration with m=191 and D=1 optimizes the system cost.

3. THE IMPACT OF THE TRANSMISSION NETWORK

3.1 Introduction

With interactive videotex, there are communications in two directions. Requests may be transmitted from the user to the central system and pages are transmitted from the central system to the user. The requests from the user can be transmitted over telephone lines or over bidirectional cable TV connections. The pages may be transmitted to the user over telephone lines or as television signals (through the air or by using cable TV distribution; we assume in the following a full channel (not only the return interval) allocated to videotex). These two choices for each direction of transmission result in four useful combinations:

a) Telephone requests, telephone page transmission,

b) Telephone requests, television page transmission,

c) Bidirectional cable TV for both requests and pages,

d) Telephone requests, both TV and telephone page transmission.

Since thousands (even millions) of subscribers can all receive the same television signals, it may be economical to use a broadcast cycle for page transmission. There is no corresponding

possibility for telephone transmission of pages where, presumably, only one subscriber is attached to the end of each telephone line.

3.2 Television Page Transmission

There are three possibilities to consider. First, a pure broadcast cycle approach may be used, as in Ceefax or Oracle. Subscribers may view only pages that the server elects to place in the cycle. Secondly, a pure request approach may be used. Only pages that subscribers have requested are transmitted. Third, a combination of the two approaches may be used. Presumably, the most popular pages are automatically included in the broadcast cycle. However, a limited number of slots in the cycle may be filled with pages that have been requested by subscribers.

3.2.1 Pure Broadcast Cycle

The performance of a videotex system that uses the pure broadcast cycle approach depends critically on the size of the cycle. Let us consider a system that fully uses the capacity of one television channel. The figures given for the transmission rate of videotex information varies between 3.89 and 5.8 million bits per second depending on which field trial or Telidon standard one looks at [10]. We will assume an average figure of 4 Mbs.

Following our earlier estimates that an average page contains between 1000 and 1500 bytes of data and assuming that

about 10 bits must be transmitted per byte (to allow for error detection and other system overhead), we can estimate that the page transmission rate lies between 267 and 400 pages per second.

If we follow guidelines that 90% of requests must be responded to within 10 seconds [23], we must limit the broadcast cycle length to 11.1 seconds (i.e. 10 / 90%). This implies that the cycle can contain between 3000 and 4400 pages. The videotex database can contain no more pages than this figure. It falls far short of the estimate given previously that at least 50,000 pages must be in the database. Also, the average response time would be 5.5 seconds which might be unacceptably slow to many users.

With such a small number of pages, it would be technically feasible to hold them all in the main memory of a computer. However, as we have argued previously, it would be cheaper and just as effective to hold almost all these pages in a mass memory backing store if the data transfer rate is sufficient. Pages can be retrieved from the backing store just before they are due to be transmitted in the broadcast cycle. Thus the main memory is used solely for temporary buffering of the pages.

The possibility of using one or more disks is also interesting. Because the pattern of page transmissions is totally determined by the cycle, the information on the disks can be formatted in an optimal manner. We could read many consecutive pages in the cycle by performing a sequence of disk reads to

consecutive tracks and adjacent cylinders. Disk seek times can be reduced to an absolute minimum and, with careful timing, latency delays can be reduced. Here is some data for a modern disk, the IBM 3350 disk unit [26]:

Time to seek to an adjacent cylinder	=	10 msec.
Average seek time	=	25 msec.
Maximum seek time	=	50 msec.
Number of tracks per cylinder	=	30.
Maximum capacity of each track	=	19069 bytes.
Time for one full rotation	=	16.7 msec.

For this disk, we could transfer 30 consecutive tracks of data to the main memory of the computer in 30 consecutive disk revolutions. Then there is a delay while the disk head seeks to the next cylinder (10 msec). As the seek is being performed, the disk continues rotating. Therefore, after the seek we must wait for the disk to rotate back to the beginning of the track in the new cylinder. Unless the tracks have a staggered organization (which is hardly worth the trouble), the additional delay is 6.7 msec. The overall data transfer rate (assuming data on consecutive tracks and on consecutive cylinders) is therefore 30 tracks for every 31 revolutions. This works out to be 1107 KBytes/sec or 8.8 Mbs. Thus a single disk can theoretically keep up with television channel transmission (and have about 50% spare capacity). It would require a very carefully tuned computer system to achieve this disk transfer rate in practice (there must be almost immediate response to each disk interrupt), but is not impossible.

Another interesting possibility is the use of CCD (Charge Coupled Device) logic for the backing memory. A CCD memory is implemented as multiple shift registers, where stored data circulates around closed loops. This would appear to exactly match the requirements of a pure broadcast cycle videotex system. There is a possibility that such a system could have simple hardware requirements. A general-purpose computer would not be required to handle page transmission, instead a small microprocessor would be adequate. However, a general-purpose computer would probably still be required in order to update the database of pages. The Texas Instruments TMS 3064 CCD memory device has the following characteristics [5]:

Capacity	= 64K bits
Cycle size	= 4K bits
Maximum transfer rate	= 5 Mbs
Maximum latency delay	= 820 microsecs.

The chip is organized as 16 shift registers, with each shift register holding 4K bits. The transfer rate matches the television transmission speed very closely. The biggest drawbacks with CCD memory are its cost and the volatile nature of the storage. According to current estimates [12], CCD memory costs about $10,000 per Mbyte (about 100 times as much as a disk).

3.2.2 Combined Approach

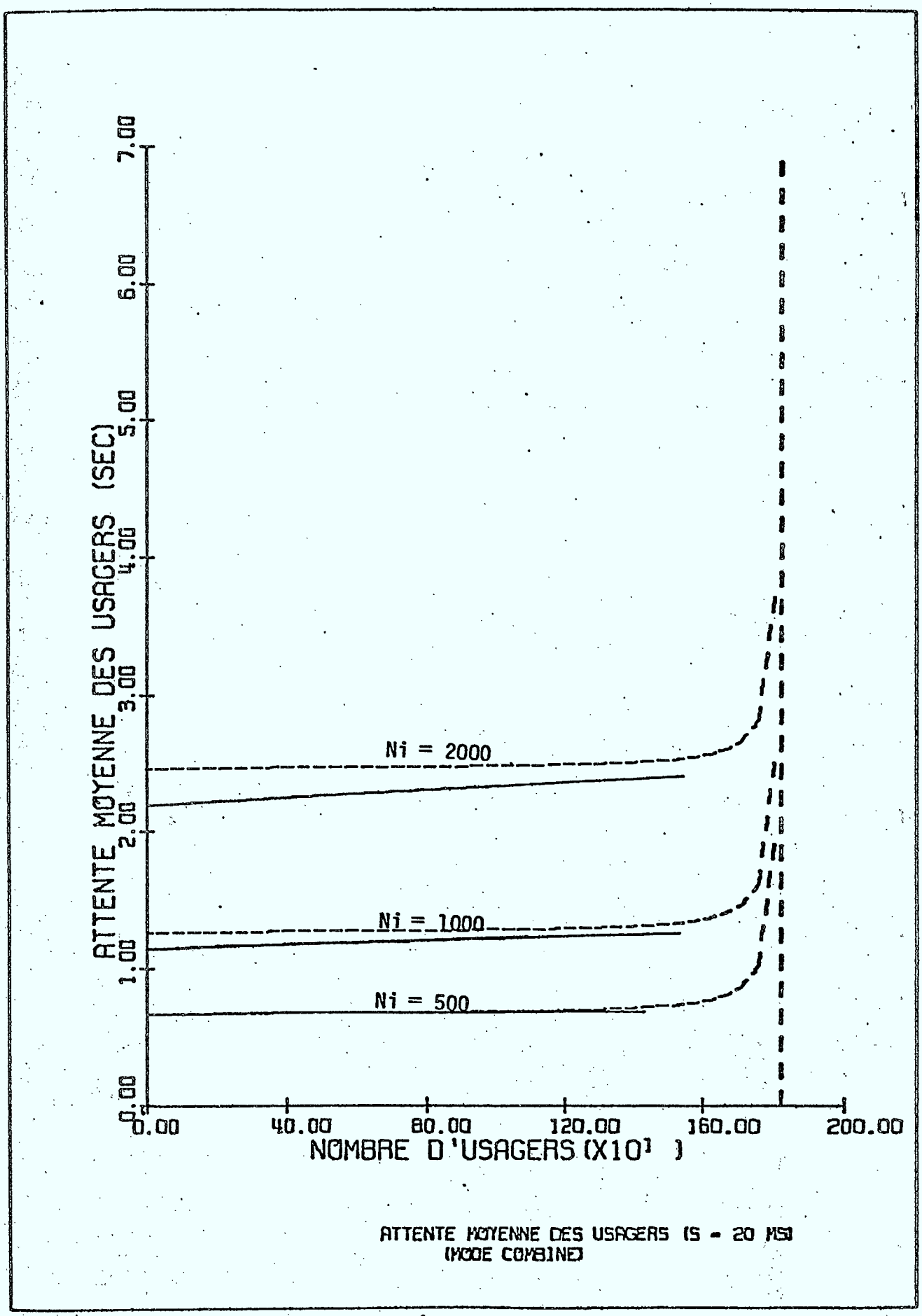

ATTENTE MOYENNE DES USAGERS (SEC)
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
Ni = 2000
Ni = 1000
Ni = 500
0.00
40.00
80.00
120.00
160.00
200.00
NOMBRE D'USAGERS (X10¹)
ATTENTE MOYENNE DES USAGERS (S = 20 MS)
(MODE COMBINE)

Figure 2.6

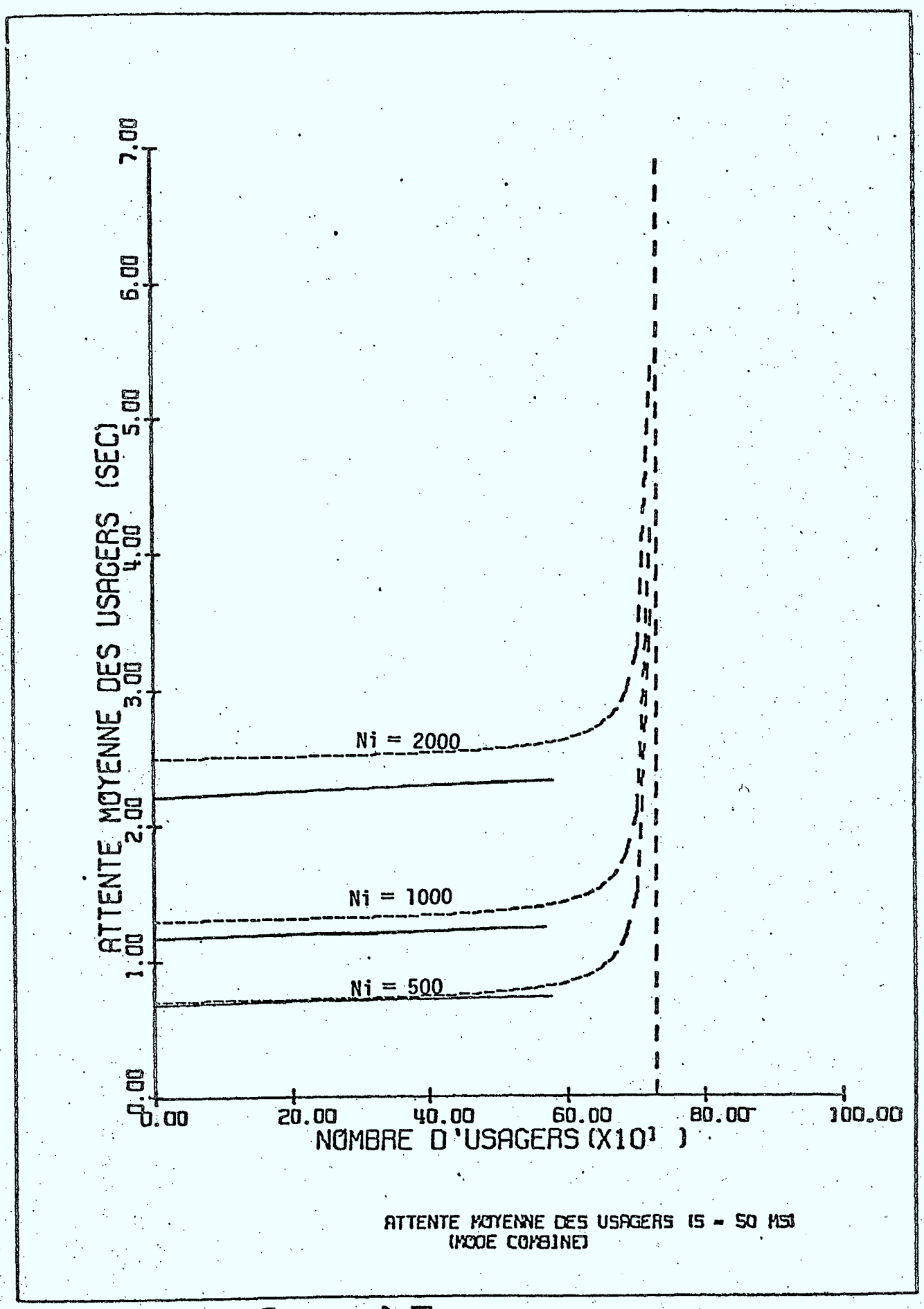

ATTENTE MOYENNE DES USAGERS (SEC)
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
Ni = 2000
Ni = 1000
Ni = 500
0.00
20.00
40.00
60.00
80.00
100.00
NOMBRE D'USAGERS (X10³)
ATTENTE MOYENNE DES USAGERS (S = 50 MS)
(MODE COMBINE)

Figure 2.7

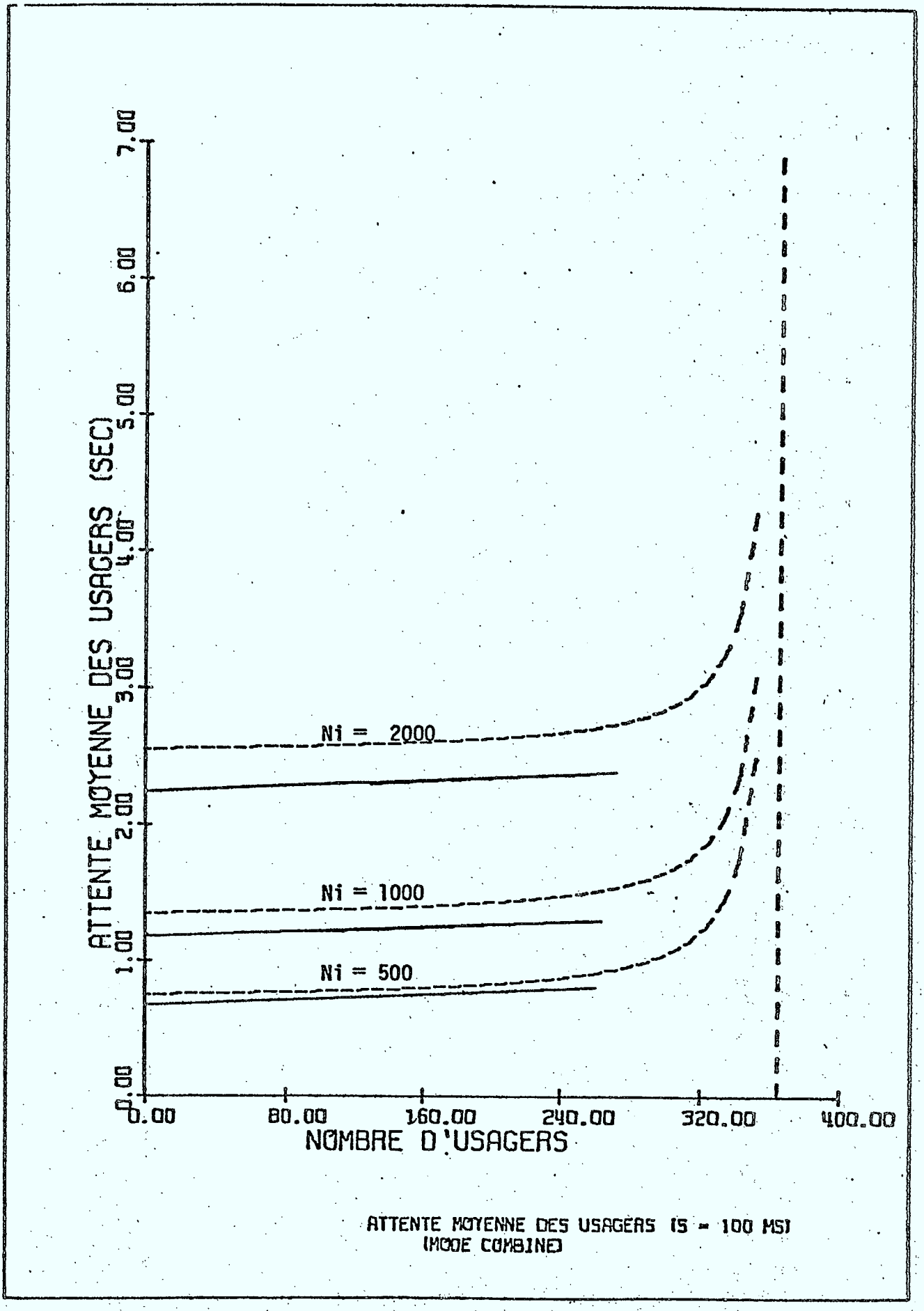

ATTENTE MOYENNE DES USAGERS (SEC)
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
Ni = 2000
Ni = 1000
Ni = 500
0.00
80.00
160.00
240.00
320.00
400.00
NOMBRE D'USAGERS
ATTENTE MOYENNE DES USAGERS (S = 100 MS)
(MODE COMBINE)

Figure 2.8

A combined broadcast cycle and request approach has been simulated for the case where a cable television broadcast medium is available [24]. The simulation covers various assumptions about the hardware configuration and its timings. Some of its fixed assumptions are the following:

Fraction of requests to pages in the cycle	= 40%
Fraction of requests to pages not in cycle, but are available from local disk(s)	= 55%
Fraction of requests to pages not in cycle, obtained from a non-local database	= 5%
Transmission speed over cable television	= 1.5 Mbs.
Transfer rate from non-local database	= 4800 bps
Average size of a page	= 1000 bytes.

The other important assumptions, and assumptions that were varied, were the following. First, it was assumed that each request received by the system involved a certain amount of processing. This processing might include CPU activity and searching page directories for information about the page. Three figures for the amount of processing needed for each request were simulated: 20ms, 50ms and 100ms. Secondly, the total number of pages in the database was assumed to be 500, 1000 and 2000. These three possibilities for each of two parameters lead to nine simulation curves. The nine curves from the study [24] are reproduced in Figures 2.6, 2.7 and 2.8 (corresponding to the figures numbered 5.23 - 5.24 in the study). They show how the expected waiting time for a request to be answered increases with the number of users on

the system.

There is nothing surprising in these curves. Their general shape would be predictable without any simulation experiments. The most important observation is that the system has a maximum capacity. If more users than this attempt to use the system, the expected delays in waiting for a page become intolerably large. This maximum number of users can be calculated analytically from the values of the system parameters (as, indeed, Lafitte did [24]). Similar calculations could be performed for a real system after the necessary measurements had been performed.

In order to maximize the capacity of a combined broadcast cycle - request mode system, it is obvious that we should only repeat very popular pages in the cycle. Thus, the selection of which pages are to be regularly repeated is analogous to the problem of sharing the pages between two kinds of memory. The crucial problem is determining the proportion of pages in the cycle that are regularly repeated versus the proportion of pages that are included in the cycle <u>once</u> in response to a specific subscriber request. (Note that the size of the broadcast cycle is constrained by a calculation similar to one given in section 3.2.1.)

If a page is sufficiently popular that there would almost always be at least one user desiring to see that page during a cycle, then the page should be regularly repeated as part of the

cycle. However, it seems hard to be more precise with this statement. Given the stochastic nature of the system, it can never be guaranteed that there will actually be a user who wishes to access any particular page during a given cycle.

Some crude analysis of the problem can be carried out as follows. We define M and x as:

M = maximum number of pages transmitted per second.

x = number of pages per second that are part of the cycle and regularly repeated (x<=M).

The other quantities that we will be using, n and fb(x), have the same meanings as before in Section 2.5.2. That is, n represents the rate at which users make requests to access pages and fb(x) is the cumulative probability distribution function for page requests. Now, we are permitting only M-x pages in each cycle to be transmitted in response to special requests. Thus we have a basic requirement:

$$[\ 1 - fb(x)\ ] * n \ <= \ M - x$$

Equality in this equation would hold if the system were operating at maximum capacity. Rearranging the equation and assuming that the system is fully loaded yields:

$$n = \frac{M - x}{1 - fb(x)} \qquad \text{(Eqn. 7)}$$

Presumably our goal is to maximize the capacity of the system, that is, maximize the number of users it can handle. We can choose the value of x that maximizes n and so maximizes the number of users. When n is a maximum, x satisfies the following equation:

$$(M - x) * \frac{d}{dx}\{ fb(x) \} = 1 - fb(x)$$

If we use the same form for fb(x) as before (Equation 2 in Section 2.5.2), we have the following equality:

$$\frac{(M - x) * h}{x + u} = 1 - fb(0) - h \log(x/u+1) \qquad \text{(Eqn. 8)}$$

By substituting known values for M, h, u and fb(0), we can numerically solve Equation 8 for x. Any solution for x in the range $0<x<M$ necessarily corresponds to a maximum for n (and thus gives the maximum request rate that the system can handle).

As a brief numerical example, we will use the values of h, u and fb(0) that were determined for the experimental system. For various values of M, the capacity of the system is maximized as follows:

M = 100, x = 20, n = 136

M = 200, x = 49, n = 333

M = 300, x = 87, n = 578

M = 400, x = 132, n = 877

Of course, only the constraints imposed by the broadcast cycle approach are being considered. Even if there is room in each cycle to handle a system with 877 requests per second, there is no guarantee that the system can retrieve pages from the disk at this rate. In other words, we have to also take the analysis of Section 2.5.2 into account. If we take the M=400 example a little further, we can see that the system must transmit 400-132 = 286 pages per second in response to subscriber requests. If every such page needed to be retrieved from a disk, we would need about 10 disk units on the system.

There is also a feedback problem. How can the system know which are the most popular pages (and which should be regularly repeated) if requests for these pages are not transmitted to the central site? Perhaps the system should occasionally drop a page from the cycle just to find out how many requests this generates for the page from users? However, it would force all users to be connected to the system via telephone (or two-way cable), whereas this would previously have been unnecessary for users who were content to reference only popular pages.

3.2.3 Pure Request Approach

When every page must be specially requested by users, there is considerably more processing overhead imposed on the system than with the combined approach, as just discussed. It

will therefore be very important to look up page numbers in directories and to retrieve pages from backing store quite efficiently. The analysis given previously in Sections 2.3, 2.4 and 2.5 is very applicable.

In addition to efficiency problems with directory structures and page placement in storage, there are some scheduling decisions that are worth considering. We contend that responses to requests should not be sent too quickly. Our argument is as follows. The initial assumption is that users are just as likely to be satisfied with 0.5 second response times as with 0.1 second response times. Perhaps users will even perceive a one second delay in response to be fast. (The worst response time that we should consider is 10 seconds [23].) Suppose, for sake of example, that the videotex system is aiming for a response time of one second. Now, if a request for some page arrives and the system waits one second before transmitting the page, there is some chance that another request for the same page will have arrived in the meantime. If this does happen, both requests can be answered with a single page transmission. The more popular a page is, the more likely it is that requests for it can be combined.

Of course, it would be ridiculous for the system to stop transmitting pages at any point just because it has no pending requests that are approaching the one second target delay. However, when the system has a choice between two pages to

transmit next, the selection rules appear to be:

1. If neither page has had an outstanding request for one second or more, transmit the less popular page first.

2. If one of the pages has had an outstanding request for one second or more, transmit that page next.

3. Otherwise (both pages are overdue for transmission), transmit the page that has the greater number of requests (to annoy fewer subscribers). If both pages have the same number of requests, transmit the less popular page first.

Our scheduling criteria are therefore biased towards sending less popular pages first. This is simply because the chance of being able to pool requests for these pages are not so good.

When the system has a large queue of outstanding requests (perhaps hundreds of requests pending at any moment), the best scheduling policy is hard to pin down. The system would have to build a tentative list of pages in the order that they are to be transmitted and estimate the time at which each page should be sent. Thus the system will be able to plan ahead so as not to leave too many pages to be transmitted over a short interval. When there is slack in this schedule, the system will be able to apply our scheduling criteria and so improve system efficiency. The more heavily loaded the system is, the less freedom there will

be in shuffling the schedule.

The precise formulation of this scheduling problem and its solution must remain a research problem for now.

3.3 Telephone Transmission of Pages

The proposed standard for transmission of videotex pages over telephone lines specifies a speed of 1200 bits per second. At this rate, it would take about 10 seconds to send an average page. Perhaps the slow speed of transmission will cause pages to be kept small in size, and so our estimate of the average page size is too high. The pages of the British Ceefax system are smaller than our estimate.

The transmission speed has a consequence for memory usage and storage organization. The user's "think time" between receipt of one page and making a request for the next is likely to be quite small compared to page transmission times. For one thing, the user can read a page as it is being received. Therefore each user's telephone line is likely to be occupied nearly all the time with page transmission.

If there are around 1000 users connected to the system then, perhaps, as many as 800 of them are receiving pages at any moment. This implies that the system has to maintain output buffers for 800 telephone lines. If the smallest unit read from a

disk and transmitted is a page, then around 800 Kbytes of memory would be devoted to telephone output buffering.

One solution that would reduce the memory requirement is to split pages into smaller blocks. With the slow telephone transmission rate, it would be possible to fetch blocks from disk as they are needed for transmission. The penalty attached to using this approach is the greater volume of disk accesses that would be required.

A better solution might be to adopt the use of Videotex Access Machines (VAM's) [40]. Each VAM would use relatively cheap hardware and could service the majority of page requests from a group of subscribers. A network of VAMs linked with one main computer, to handle less popular page requests and to interface with other computer systems, would be a relatively economical solution. The design and performance analysis of a VAM machine is provided in section 5 of this report.

3.4 Combined Telephone and Television Transmission

This scheme for transmitting pages does not appear to have been suggested before. It is a simple idea that maximizes the capacity of the system to transmit pages. A television channel is used to transmit just a broadcast cycle of regularly repeated pages. If a user wishes to see one of the pages in the cycle, there is no difficulty. The user's videotex adaptor will

capture the page from the air as usual. If the user wishes to see some page that is not in the broadcast cycle, his adaptor transmits a request for it over the telephone line and the page is subsequently received over this same line.

As discussed previously, a pure broadcast cycle can be transmitted very efficiently. The cycle should contain as many pages as possible (consistent with limits on acceptable response times), so as to minimize the need for special telephone requests.

The installations providing the broadcast cycle and the installation responding to telephone requests could be separate entities. However there are benefits to having a single facility or to at least linking the two services. By monitoring the frequencies of requests for pages over the telephone, the system can learn which pages should be added to the broadcast cycle. (Knowing which pages to remove from the cycle is another problem, as discussed in section 3.2.2.)

The only disadvantage appears to be that the videotex adaptor needs to be a little more complicated than with any of the other approaches that we have considered. However, we think that the idea is worth further consideration.

4. EXTENDING THE USER INTERFACE

4.1 Introduction

A basic videotex system provides relatively primitive facilities for locating information in the database. Unless the user has access to a directory of page numbers which he can peruse and so plan his sequence of inputs, the user can easily get "lost" while traversing the database. A logical next step in making videotex more usable is to provide some more sophisticated access methods for the information.

One possible approach is to provide keyword access to pages. For example, a user wishing to see a list of restaurants may only have to type the keyword "RESTAURANT". The list is likely to be rather long. Perhaps the user would be permitted to type two keywords, such as "GREEK RESTAURANT", and then he would be shown only the few pages that give him access to information on Greek Restaurants. An even more sophisticated interface might permit the user to construct simple queries, similar to "RESTAURANT: GREEK AND (COST<10.00)". This might retrieve information on a few affordable Greek restaurants for the user. The example of the Greek Restaurant and the associated menu tables has been discussed more fully elsewhere [17, 41].

Whatever facilities are added to assist the user in finding appropriate pages in the database, they will require

computing resources. All keyword schemes require some form of searching, some keyword schemes requiring more searching than others. A more powerful facility such as a simple query language may require much computation. For example, a query that asks for a list of all towns with ten or more restaurants would probably require information on every restaurant in the database to be read. Even more powerful query languages, such as those based on natural language, may be envisaged but they currently belong to the realm of artificial intelligence. A range of different user interfaces for videotex are discussed in [41].

Other possibilities for extending videotex involve general interactive software. This would correspond to the "action page" concept. User services that require interaction include message forwarding systems (as are provided in many computer systems), reservation taking or seat sales (e.g. for the theatre), computer aided instruction and game playing (e.g. checkers or pacman). If more than a few usertake advantage of such services, there will be a sub stantial processing load on the system.

In this section of the report, we will look into the possibility of providing for keyword access to pages and for more general user queries. The possibility of using special-purpose hardware to assist these extensions will be considered. There has been much research into designing special hardware for database systems. However, the hardware solution is not necessarily very

applicable to videotex systems because it may not be as fast as would be required and is relatively expensive.

4.2 Survey of Database Machines

The term "Database Machine" describes any computer system with special-purpose hardware to assist the processing of database operations. The operation that is most frequently automated is that of searching through large volumes of data. Because of the large volume, the data would normally reside on a disk or drum and the special hardware would consist of processors that scan data as it passes beneath the read/write heads. To make the searching as fast as possible, each track has its own read/write head and its own processor. These kinds of storage devices are frequently called "logic-per-track" devices.

An early logic-per-track device was a modified version of a disk built by Burroughs Corp. [30]. It was a disk with 1000 information tracks, each track having its own read/write head. Tracks were subdivided into quadrants that held information. The information was tagged as being "keyword", "associated data" or "unused". When the computer issued a search request, the processors associated with each read/write head would check the data in every track simultaneously for a match against a specified keyword. When a keyword is successfully matched, the associated data is read by the read head and transmitted to the computer. (This is known as "associative search".)

More recent developments of logic-per-track devices are similar to the Burroughs device. The main improvements have been in the search processors, which are now capable of handling complicated conjunctions of search conditions. One of the simpler schemes is used in CASSS [21] and involves multiple searches through the data and tag accumulators that are associated with every data record on the disk. For example, if CASSS is to search for records containing both the keywords "GREEK" and "RESTAURANT", it proceeds by first searching for matches against "GREEK". In each record that matches, the associated tag accumulator is incremented (from zero to one) and rewritten to the disk. Subsequently, the same process is repeated for the keyword "RESTAURANT". Finally, the processors are instructed to search for and output any record whose tag accumulator contains a value of two or more. Thus the disk would need to rotate at least three times to output the desired list of Greek restaurants. More rotations than three are needed if two processors try to output a record simultaneously (one of them will be forced to wait for a subsequent disk revolution).

The RAP device [28] is more powerful than CASSS because there are k Boolean comparators associated with each head. (k is not determined because only prototypes of RAP have been built.) As each data record passes underneath the read/write head, it can be checked for conformity with k terms of a general Boolean function that describes the desired records. If the selection function contains k or fewer terms, matched records can be output

immediately. If the function has more than k terms, records that satisfy the first k terms are rewritten with some mark bits set. On the next disk revolution, the Boolean comparators can check for the next k terms of the selection function, and so on.

There are many other database machine designs that support searching. These include LEECH [27], CASSM [37], CAFS [2], RARES [25], DBC [3], RAPID [29] and STARAN [34].

Searching is not the only operation appropriate for a database system. Other operations suitable for the relational database model include projection, join, union, intersection and difference. However, according to one reference [35], it is search and join that are the most interesting for hardware enhancement. The join operation is more difficult to implement than a search (or selection) operation and not all designs for database machines support the join operation. Some machines that do provide it include CASSM, RAP, LEECH and CAFS. However, the time required to complete the operation is very data dependent. It is typically proportional to the number of tuples in one of the relations participating in the join operation. Thus, machines such as RAP and CAFS will usually require many disk revolutions to complete the processing.

Another operation which is useful but not required for a relational database system is sorting. A few machine designs do provide for fast sorting of tuples (i.e. records) in relations.

These are RARES [25], the Chen Machine [11] and the Intelligent Memory [14]. Sorting is nice for preparing output to pass on to a user. Also, sorting hardware may be driven by software in order to achieve the effect of a join operation (as is done in RARES).

Special hardware has been provided or proposed for text retrieval applications. In a bibliographic system, the user is usually permitted to search through the keywords associated with the papers in the database or to search through the abstracts of the papers. The user might search for all papers where a particular combination of words appears. A full text system would permit users to search through the actual texts of the documents.

The hardware used to implement text retrieval is not very different to that used for searching in database systems. The major differences only arise in the keyword matching process. The specialized processors attached to read/write heads scan text as it passes underneath. However, finite state machine logic may be used to perform pattern matching on the text. One simple form of pattern matching makes provision for "don't care" characters. For example, if a "don't care" is indicated by "?" then the pattern "BE?T" would match any of "BEAT", "BEST", etc. in the text. A more general possibility is to provide "variable length don't care" characters. If "*" denotes such a VLDC then the pattern "BE*T" would match "BET", "BEAST", "BETTERMENT", etc. By careful use of "don't care" characters, some controlled imprecision can be added to a search. This can enable the user to circumvent problems

caused by many variants of a word being used in documents. A topical example would be the group of words: "computer", "computing" and "computation".

Although there are many proposals for text retrieval hardware, there appear to be few actual implementations. One implementation is the Associative File Processor [6].

4.3 Simple Keyword Access

The simplest approach to keyword look-up would use precoordinated keyword access. With this approach, the system implementors (or the information providers) must initially extract keywords from documents and form them into table structures. Since this is done in advance of any user input, the term "precoordinate" is used (to suggest prior coordination of the keywords with the documents). When the user specifies a keyword, the system will only look for it in one or more of the previously constructed tables. Thus the users are constrained by the contents and the organization of these tables. It takes very careful design by the system implementors to make the keyword tables useful and easy to use by subscribers.

A simple precoordinate keyword scheme has been implemented in the experimental "Montreal Keyword System" [17] at the Universite de Montreal. Keyword tables are associated with some of the more important and strategically placed pages in the

database. As an example, suppose that a user is currently viewing page 12345.67 in the database when he inputs a keyword. The system will search the table associated with page 12345, if there is one. If there is no table, the system will check the ancestor page in the hierarchy, page 1234, and search its table, if there is one, etc. That is, the closest table along the path back to the root of the page hierarchy is searched. If the table contains the specified keyword, the associated information will refer to a menu page for the appropriate subject and the system will display this automatically. If the keyword is not found, the table can specify up to 10 other keyword tables to be searched next.

Because the tables in the Montreal Keyword System are associated with particular pages, the tables can take account of context. For example, the tables can be set up so that specifying the keyword "GREEK" while looking at any page concerned with restaurants will bring the user to information on Greek restaurants. However, specifying "GREEK" while looking at a page on the subject of travel may bring the user to information on travel agents who provide vacations to Greece.

As described above, the keyword look-up is carried out as the following logical steps:

1. Find the nearest page in the hierarchy (along the path to the root) that has an associated keyword table.

2. Locate this keyword table in memory or, if necessary, fetch it from disk.

3. Search the table for the desired keyword. If it is found, we are finished.

4. If the keyword was not found, we look up the location of the next keyword table to try and proceed as for step 2.

Some hardware assistance with steps 2 and 3 is possible. Some of the logic-per-track devices discussed earlier would permit the look-ups to be performed without actually having to read tables into main memory first. However, there would be little speed advantage to this. The speed of the search is mostly limited by the rotation rate of the disk. If the tables are short, very little CPU time is consumed.

Economies of scale are possible if all the small keyword tables are combined into a single large table. We may still logically view the data structure as representing many small tables. There need not be a close correspondance between a logical view of data and its physical realization. We propose revising the steps needed to look-up a keyword as follows:

1. As before, determine the page p that has an associated (logical) keyword table.

2. Search the combined table for the combination <p,keyword>.

3. If found, the associated information is retrieved and we are done. Otherwise, we determine the next page p whose associated (logical) table is to be searched and go back to step 2.

To implement step 1, there need only be a single bit in the entry for each page in the page directory. The bit would simply indicate whether the corresponding page has an associated (logical) table. For example, if the user is currently viewing page 12345, the system would check the bit for page 12345, then for 1234, then for 123, etc. until a page with the bit set is found.

To implement step 2, a wide choice of data structures for the table are possible. We could certainly use a hash table. That is, a hash function is computed from the <p,keyword> combination and used to index the hash table. If the hash table is not heavily loaded, the desired entry will be found almost immediately.

Step 3 can be implemented by putting some extra entries into the combined table. If the look-up with <p,keyword> fails then we could perform a look-up with a special combination such as <p,"???"> which retrieves the number(s) of page(s) whose

associated table(s) are to be searched next.

Now a single combined table is certain to be very large. We can get around this by implementing the table in two levels as was suggested for the page directory in Section 2.3 of this report. Thus some of the most frequently used <p,keyword> combinations could be kept in main memory and the remainder are kept on disk. Alternatively, the whole table could be held on disk and if it is not compressed into too small a volume, almost all keyword look-ups would require just a single disk read.

Again, there is little need for special purpose hardware to speed up the look-up. The only advantage of logic-per-track devices is that they have read/write heads for each track. The processing capability associated with each head would be largely wasted. We can obtain the same search rate by using a (cheaper) fixed-head disk or drum device, which also has one head per track.

4.4 Postcoordinate Keyword Access

Most bibliographic retrieval systems use postcoordinate keyword access. Their mode of operation is not difficult to describe. Initially, every document in the bibliographic database is provided with a list of keywords. (Some systems would require abstracts to be supplied for every document.) The keyword lists are normally provided by the authors of the documents. If a user wishes to retrieve documents relating to Greek Restaurants, say,

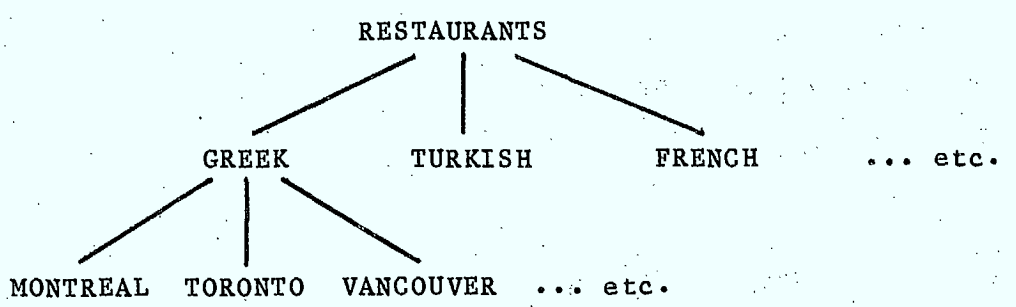
he would specify the keywords that he thinks would define the kind of document he wants. For example, he might simply type 'RESTAURANT GREEK'. The system then searches through all the keyword lists to find one or more documents where both keywords appear in the associated keyword list.

It is possible that the author of the document did not provide a good selection of keywords or that he used some synonyms. For example, an author might have used the keyword "CAFETERIA" and our user's search would not locate this document. Some systems permit the user to specify choices (disjunctions) and this mechanism could be used by a suspicious user to widen his search. He may, for example, enter his keyword list as "(RESTAURANT or CAFETERIA) (GREEK or GREECE)". This possible lack of appropriate keywords (or keywords that the user would expect) is, perhaps, the biggest problem for postcoordinate keyword systems.

The idea of postcoordinate keyword searches is easily adapted to videotex systems. Every information provider would, presumably, be required to supply keyword lists for his most important pages. Such pages would be "gateways" into particular subjects and would usually be menu pages. These keyword lists are held on-line in the videotex system. When a user specifies some keywords, the lists are searched. Very often, there is more than one page in the system whose associated keyword list matches the search criteria. In this situation, an appropriate action seems

to be to display the matched page that is highest in the page hierarchy (and consequently has the shortest page number). From this page, the user should be able to use normal menu selection to find the specific information page that he desires.

We will explain why the highest page out of many that match the search criteria is the most appropriate to display. Suppose that the hierarchy of pages on the subject of restaurants is structured as below:



That is, the menu page on the subject of restaurants displays the choice of cuisine. After a user has selected which cuisine (Greek, Turkish, etc.) he wants, he is next shown a menu page that displays a choice of locations (Montreal, Toronto, etc.). After he has specified the location, he may then be able to access particular restaurants.

Now, one of the lower level pages would likely have all three of the keywords "GREEK", "RESTAURANT" and "MONTREAL". The menu page at the level above (where the user is asked to select

location) would just have the keywords "GREEK" and "RESTAURANT". Therefore, if the user requests a search on the keywords "GREEK RESTAURANT", one menu page at the second level and many pages at the third level of this sub-tree in the hierarchy would match. Clearly, it is the highest page in the hierarchy that is the most appropriate (in the absence of any information that specifies the desired location).

On the other hand, if the user were to specify the keyword combination "RESTAURANT MONTREAL", there would be many pages at the same level in the hierarchy that match. That is, the search should find pages on Greek restaurants, French restaurants, Turkish restaurants, etc., all in Montreal. Possibly the user should be shown the lowest common ancestor of all these pages in the hierarchy (the initial menu page for "RESTAURANT"). The user would probably prefer to be shown the first of the pages that match and be permitted to step through all of these matching pages. However, this capability would either require searches to be repeated or it would require a (possibly) large amount of extra context information to be retained (the list of page numbers).

A postcoordinate keyword system is relatively easy for the user to understand and is quite flexible. Users are not constrained to predefined access paths to information, as with menu selection or with precoordinate keyword systems. However, the flexibility comes at a price. Let us first estimate the amount of storage occupied by the keyword lists and then estimate the

search times.

In a minimal size system of 50,000 pages, there might be about 10,000 menu pages (assuming an average branching degree of 5 at each level). Perhaps not all menu pages would have associated keywords, so we will use a conservative estimate of 5,000 keyword lists. An average of five keywords per list is another conservative estimate. We will guess that the average keyword contains 8 characters (keywords tend to be longer words than an average word in English prose). These numbers multiply to give a lower bound of 200,000 characters. In addition, there would be extra storage needed to delimit the keywords and the corresponding page numbers must be stored. Presumably, total storage requirements that reach a few megabytes and are not unreasonable for keyword lists of larger videotex systems.

According to one source [33], an implemented bibliographic search system that runs on an IBM-370/158 computer searches through text at a rate of 100,000 characters per second. Even with our minimal estimate of a 200,000 character list, a simple keyword search would take 2 seconds. If the keyword list can be held entirely in main memory and if efficient search algorithms programmed in assembly language are used, faster search rates are possible. In one experiment on an Amdahl V7 computer [22], search rates between 2 and 9 million characters per second were observed. (The range reflects dependance on the length of the keyword.) Much of the improvement in speed over the previous

measurement is due to the fact that an Amdahl V7 is about six times as fast as an IBM-370/158. At a search rate of 5 million characters per second, it would take 0.04 seconds of concentrated CPU activity to search the small list.

These search speeds are much too slow for a system like videotex where hundreds of users will be active at any time. If postcoordinate keyword search services are to be provided to the majority of users, it is clear that some hardware assistance is essential. We should now consider how fast database machines, as described earlier in Section 4.2, can perform keyword searches.

If a modified disk, a logic-per-track device, is used then all the devices described in the literature require at least one disk revolution per search. Most devices require one revolution per keyword if a conjunction of keywords is specified. The RAP system is an exception because it would be able to handle k keywords in one revolution. Now most disks revolve 60 times per second. Thus, the logic-per-track devices reported in the literature could handle fewer than 60 keyword look-ups per second. Possibly, this search rate can be improved if the keyword lists are replicated on the disk. For example, if the keyword lists occupy less than 50% of the disk capacity, the disk surfaces could be divided into two semicircles, each containing all the information. In this case, each search would require only half of a revolution.

For faster search rates, it seems necessary to use solid state memory devices, such as CCD or Bubble memory. In fact, a prototype of the RAP machine [28] has been built with CCD memory. These memory devices are normally too expensive for use in large database systems. However, if our estimates are correct, less than a megabyte of storage would be needed to support keyword access in a moderate size videotex system. With this quantity, the cost is no longer an issue (about $10,000 for one megabyte of CCD memory).

With such memory devices, the time required to locate, retrieve and search one 32 Kbyte segment of memory should be between 200 and 500 microseconds [4]. The time depends on the particular memory configuration in use and on how complex the search is. At this rate, the system could perform between 250 and 600 searches per second through 256 Kbytes of keyword information.

Without any prior experience, it is difficult to guess how frequently users will request keyword searches. Presumably, users will tend to rely on a keyword search only to get to a suitable starting page in the hierarchy. From that point on, normal menu selection and sequencing through pages should be sufficient. If we envisage a system with 1000 users who collectively access pages at a rate of 100 per second then a search request rate of around 20 per second should be reasonable. This would probably be too much for the standard logic-per-track devices used in database machines (at best, it would be marginal).

However, corresponding devices that use CCD or Bubble memory should be quite fast enough.

5. VIDEOTEX ACCESS MACHINES

5.1 Introduction

Any videotex system considered so far has been physically implemented as a single CPU and data base serving a population of users. Like any on-line computer system, such a configuration has a practical maximum number of simultaneous users, beyond which the response time becomes intolerably long. The most obvious way of serving a larger population of users is to duplicate the entire system as many times as necessary, which leads to update problems and offers no economies of scale, since the cost per user remains the same. A better solution is to have users connected to a number of satellite machines which in turn are connected to a central machine containing the data base. The satellites, or Videotex Access Machines (VAM's) would contain temporary storage for the most frequently accessed pages using the memory hierarchy ideas of section 2. In this case however, the memory hierarchy exists between the VAM and the central machine with the small subset of most frequently accessed pages stored in the VAM. For example, consider once again a videotex data base whose pattern of accesses follows the 90-10 rule. The VAM could therefore satisfy 90% of all requests made to it by storing the most frequently accessed 10% of the data base. The remaining 10% of requests are passed on to the central machine, and the pages when received are used to keep the most popular page list up to date using one of the page replacement policies described in section 2.4.

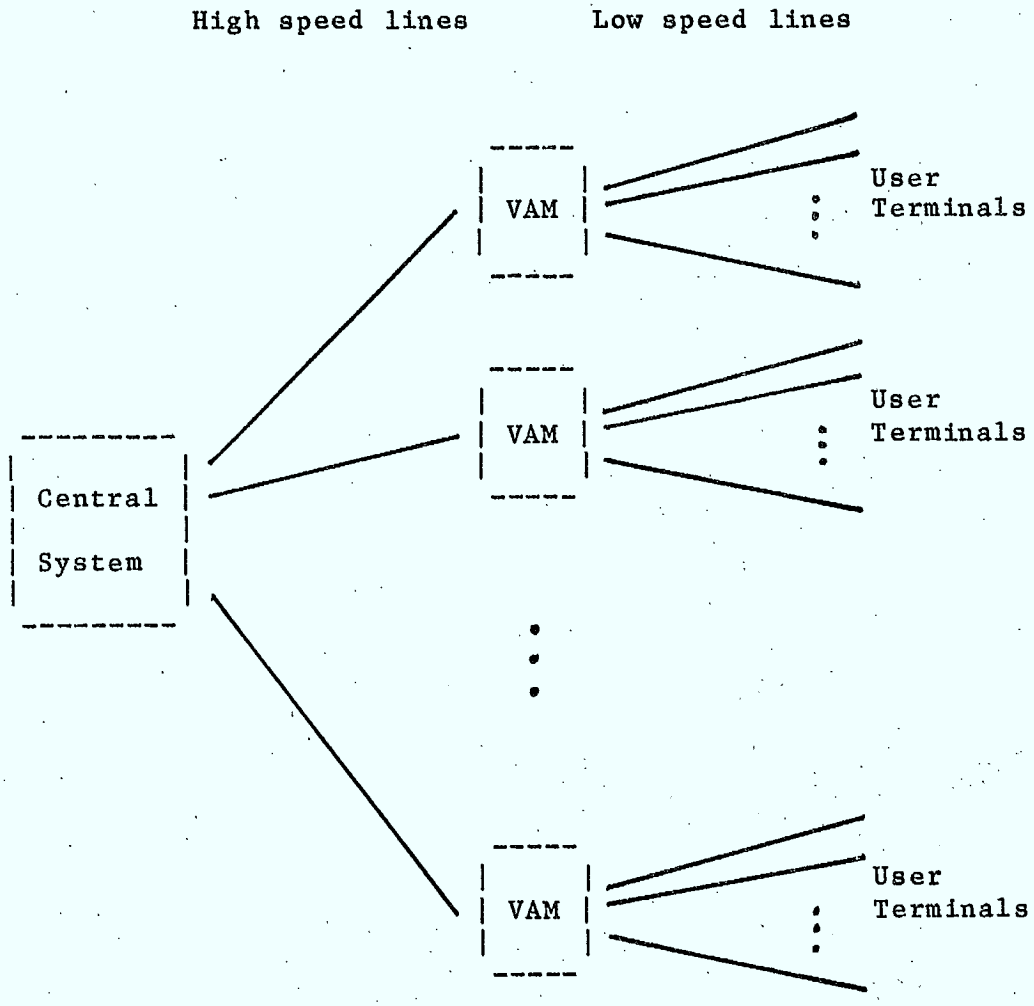
Considering the user population as a whole, the cost per user for the VAM configuration can be shown to be lower than for the replicated system configuration. For simplicity, assume that the VAM and central system are identical except that the VAM disc storage is only 10% as large. Using again the 90-10 rule, the central system in the VAM configuration can then support 10 VAM's since each one passes on 10% of the requests it receives. Therefore eleven systems are used to increase the user population tenfold. This is however less expensive than using ten copies of the central system because the smaller disc storage on the VAM is considerably cheaper.

In addition many of the advantages of distributed computing apply to the VAM configuration. The VAM's can be located physically close to the users, saving communication costs, and the single connection to the possibly distant central machine is effectively shared by all users of the VAM. The VAM does all direct user interfacing and buffer handling as well as housekeeping tasks such as logon, user identification and usage statistics, so that the central machine need only accept pre-processed page requests and return the pages demanded. Such a distribution of functions over two machines allows each to be optimized for its specific task, resulting in greated overall efficiency.

Extended videotex features can also be naturally handled by the VAM configuration. Telesoftware and action pages could be

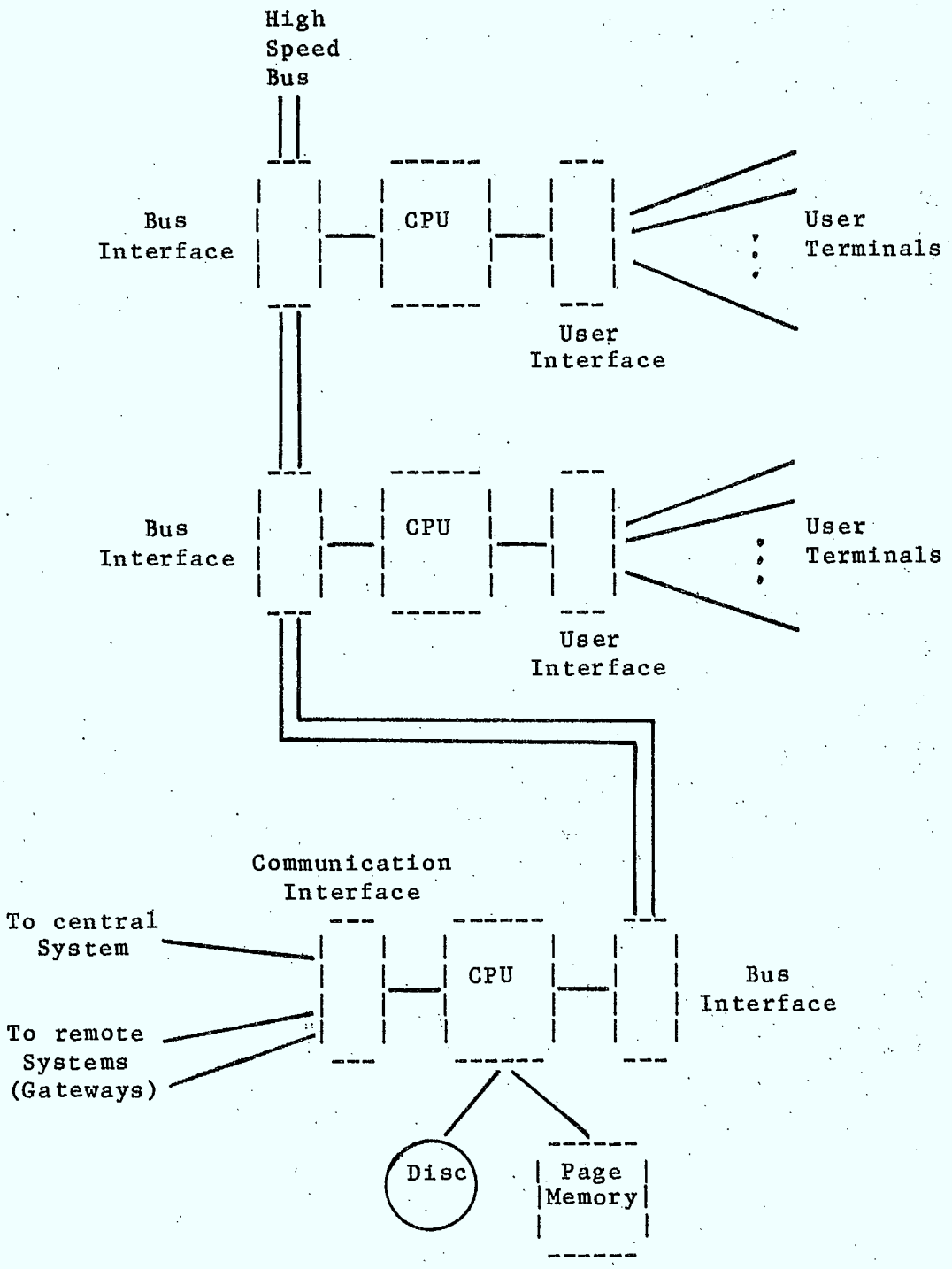
executed on the VAM where extra CPU power can be provided as needed. Keyword searches could be partially implemented on the VAM using the same policy as used for temporary page storage, that is the most popular keyword indices would be stored in the VAM so that only a small portion of keyword lookups or searches would be done on the central machine. The configuration also allows more general networking since a VAM can access several central machines as easily as one. This provides users with a gateway function where access to remote data bases is provided as naturally as access to the central machine.

The design and implementation of a Videotex Access Machine is a current research topic at the Université de Montréal. It is to be implemented as a multi-micro-CPU system with a single disc and programmed in a concurrent programming language. The disc implements the local storage of most popular pages, and as many CPU's are provided as needed to handle the user processing. Modern concurrent languages such as Concurrent Pascal or Modula 2 are designed to allow efficient multi-programming on a "bare" machine, that is without the use of a general purpose operating system and the overhead it requires. In this way the machine is dedicated precisely to the task represented by the concurrent program. In the remainder of this section we will describe how such a system might function and provide an analytic model of the system capacity for a given hardware configuration.



Overall VAM System Configuration

Figure 5.1



A Possible VAM Configuration

Figure 5.2

5.2 System Configuration

The VAM machine is physically implemented as a single disc drive and enough CPU's to handle user processing and I/O. If more than one CPU is used then the system components are linked together using a shared high speed communication link. The components in this case are complete micro-computer systems consisting of a CPU, memory and usually some sort of I/O facility. The concurrent program implementing the VAM system as a whole consists of several concurrent processes that communicate amoung themselves and in a given implementation these processes are distributed over the CPU's. There is no conceptual difference between inter-process communication taking place on one CPU or between two CPU's so the structure of the overall concurrent program is independent of the physical realization on one or more CPU's. It is practical considerations that determine the distribution of processes over the CPU's, for instance the processing and I/O capacity of a single CPU and the speed penalty of communicating between two CPU's.

Since as many CPU's are used as needed, the throughput of the system is limited by the average service time of the disc. If the requests to the disc are treated in the order received, this time is dominated by the average seek time of the disc. However disc scheduling algorithms, which take into account the current head position of the disc, allow a reduced average service time at the cost of a greater variance since requests are not treated in

the order in which they are received. According to Teorey [43] the SCAN algorithm, where the head shuttles back and forth servicing requests waiting along the way, is the best method for low to moderate disc loading. We will assume in our model that all VAM disc accesses are made using this method.

Although the VAM stores only the most popular pages in the system, the memory hierarchy ideas of section 2 apply as well to the VAM taken by itself. Of the pages stored on the VAM, the most popular of these in turn can be stored in memory to reduce the load on the disc. However the directory storage cannot be handled exactly as considered before because the set of pages on the VAM is continually being updated. When a new page is stored in the VAM, an old one must be deleted. If the corresponding directory entries were stored on disc, this would require two extra disc writes to bring them up to date. Considering also that the set of pages on the VAM is already fairly small, we propose storing the entire VAM page directory in memory.

Estimating the number of CPU's required for a given configuration is difficult because the CPU processing required is low while the I/O throughput required is relatively high. For example, if each user request needed 5 ms of CPU time (1000 instructions on a slow machine) then 200 users each generating one request every 10 seconds on the average would present only a 10% load on the CPU. On the other hand, assuming an average page size of 1000 bytes, the same population of users would present an I/O

load of two times 20 Kbytes per second, assuming the CPU must receive the data from somewhere (disc or another CPU) and send it to the user terminals. While this is well within the overall DMA range of even a slow machine, it is still not necessarily true that a given CPU could support that many users, given such practical constraints as the number of I/O boards that can be plugged in or how many devices can contend for DMA. In addition, extended features such as telesoftware and keyword searches require CPU processing. However it is difficult to estimate how much they require until usage statistics on these features become available. It is for these reasons that the VAM is conceived as a multi-CPU system with the flexibility of adding processors to the system as needed.

Figure 5.2 gives a possible configuration where the user terminals are divided between two CPU's and a third CPU is used to manage the local storage and interface with the central systems. In this instance additional user interface CPU's may be added without changing the basic structure.

5.3 System Modelling

The VAM system model is based on three input parameters. Firstly, the capacity of the disc determines what proportion of requests can be satisfied locally without accessing the central system. The larger the disc, the greater this proportion and so greater is the number of VAM's the central system can support.

Secondly, the size of page storage in memory determines what proportion of requests serviced by the VAM can avoid accessing the disc. Increasing the size of this storage reduces disc accesses and therefore increases the potential number of on-line users. Finally, the disc characteristics determine the practical limit on the rate of accesses to the disc which along with the previous point determines the number of users that the system can handle. We use queuing system analysis to model the performance of the system for various user loads. To determine the relationship between the size of local storage and the proportion of requests that it can satisfy, we use the Yield formula, introduced in section 2.5.2.

Taking the VAM and central system as a whole, the response time for various user request rates can be modelled by queuing system analysis. The usual measures are average response time and 90% time, the time in which 90% of all requests are satisfied. Putting a given limit on the response time imposes through the model a limit on the user request rate. In the VAM system, the response time distribution is a weighted sum of two disjoint cases: the response time of the VAM for pages it stores locally and that of the central system for requests it receives from the VAM.

The response time of the VAM for locally stored pages is the more important of the two since we expect most page requests to be satisfied locally. We will model this as just the disc

response time since the CPU processing time is insignificant in comparison. Although a queuing model analysis of a disc using the SCAN algorithm is very complicated, Teorey [43] provides open form equations that can be used to numerically determine the mean and variance of the response time as a function of the request rate, given the characteristics of the disc drive in question.

For requests made to the central system, we will model the response time as the sum of the central disc response time and the communication time between the two systems, again ignoring the CPU processing time. Since we are just concerned here with the configuration of the VAM and since this class of requests represents only a small proportion of the total we will simply estimate a value for the central system disc response time. The communication time will be twice the transmission delay between the two systems, for sending the request and receiving the page demanded.

Letting $\overline{tv}$ be the average response time of the VAM for local requests , $\overline{tc}$ the average response time of the central system and pv the proportion of requests serviced by the VAM, then the average combined response time is:

$$\overline{tr} = pv \cdot \overline{tv} + (1 - pv) \cdot \overline{tc} \qquad \text{(Eqn. 9)}$$

If we specify an acceptable response time for the system, the value of $\overline{tv}$ can be found given pv and an estimate of $\overline{tc}$. The queuing model then gives the corresponding disc access rate from which the maximum number of on-line users can be determined.

In order to calculate the 90% time we use Allen's extension to Martin's estimate which states that the 90% time is equal to the mean plus 1.3 times the standard deviation [44]. For this we need the variance of the combined response time which is:

$$var(tr) = pv \cdot Var(tv) + (1 - pv) \cdot Var(tc) + pv \cdot (1 - pv) \cdot (\overline{tv} - \overline{tc})^2 \qquad \text{(Eqn. 10)}$$

In order to determine how much local page store should be allocated on the VAM, we need to know the frequency distribution of page accesses. For a small set of test data, this was modelled quite well using the Yield formula, at least up to about the 85th percentile of accesses (see section 2.5.2). However, no data was available to the authors for a more realistic sized data base of say 50,000 pages. Therefore we present in Appendix A2 a method of extrapolating the formula already obtained to apply to a larger sized data base.

Here we repeat the new Yield formula derived for a data base of 50,000 pages:

$$fb2(m) = 0.114 \log(m/0.731 + 1) - 0.069 \qquad \text{(Eqn. 11)}$$

Note that as in the case of the original formula, if we set fb2(m) to 0.85 or greater and solve for m , we get values that are too small because of the non-linearity of the data in this range. This new formula corresponds however to a 92-8 rule, so we will revise this downward to the more conservative 90-10 used in previous examples. Obviously more work is needed to fit a formula

to the non-linear part of the curve in order to determine such values with some confidence.

Letting pv be the proportion of requests to be serviced locally by the VAM then the Yield formula gives the amount of disc storage necessary. Similarly, letting pm be the proportion of requests to be serviced directly from VAM memory, avoiding disc accesses, we can calculate the amount of page storage necessary.

Given the above proportions, we can calculate the average number of disc accesses per user request. This gives us the maximum user request rate given the maximum disc access rate determined from queuing analysis. The maximum number of on-line users can then be found from the average rate at which each user makes requests to the system. Each user request generates (pv - pm) page reads on the average because pm of the reads go to memory and (1 - pv) go to the central system. In addition it generates (1 - pv) disc writes because pages from the central system are kept on the VAM disc to keep its pages list up to date. The total number of disc accesses on the average is then:

$$pd = (pv - pm) + (1 - pv) = 1 - pm \qquad \text{(Eqn. 12)}$$

This turns out to be independent of pv, decreasing as pm increases.

5.4 A System Example

In modelling a hypothetical VAM system, we will use the following parameters:

Data base size	= 50,000 pages
Average page size	= 1,000 bytes
Average user request rate	= 1 page per 10 seconds per user
Average system response time = $\overline{tr}$	= 1 second
90% response time	= 2 seconds

We specify the proportion of requests to be serviced by the VAM and the smaller proportion to be serviced from VAM memory

pv = 0.9

pm = 0.4

As mentioned earlier, increasing pv increases the number of VAM's that can be connected to the central system but costs VAM disc storage and directory space, while increasing pm increases the number of users that can be connected to the VAM but costs VAM memory space. If the costs of all system components were known, including the central system and the communication links, it should be possible to minimize the cost per user as a function of pv and pm . However we simply specify what seem to be reasonable values, since we are concerning ourselves here with the VAM system alone.

Given the value of pv , we can use the 90-10 rule to predict that the VAM disc must have a capacity of 5,000 pages or 5 Mbytes. This is in the range of Winchester type discs now available, which also have the virtue of being compact and fairly inexpensive. Using the estimate in section 2.3 of 14 bytes per directory entry then we also need 70 Kbytes of memory for directory space. To find the memory space needed for pages we solve fb2(m) = pm (Equation 11) which gives a value of m = 44. Therefore the 44 most popular pages in the data base will satisfy 40% of all accesses. This requires a memory space of 44 Kbytes.

To model the disc response time we will use the characteristics of the IBM 2314 type disc because Teorey [43] provides complete data on it and because its average access time of 72 ms is within the range of current Winchester type discs. A more detailed analysis based on a particular disc would use the explicit formulas Teorey provides.

Using a central system response time of 1 second and a data transmission delay of 1/4 second we have $\overline{tc}$ = 1.5 second. Using the values of $\overline{tr}$ and pv given we can solve Equation 9 to yield $\overline{tv}$ = .94 second, the required average response time of the VAM disc. From Teorey's Figure 8 this gives a disc access rate of 27 requests per second. His formulas give also var(tv) = .602 $\overline{tv}$. Estimating the variance of the central system to be 1 second (exponential case), Equation 10 gives var(tr) = .607. The 90% time is then $\overline{tr}$ + 1.3 s.d.(tr) = 1.95 seconds, which is within

the required range.

Equation 12 gives pd = 0.6 disc accesses on the average per user request. Therefore the system will support 27/0.6 = 45 user requests per second. Using the user request rate given of one per 10 seconds, then the system can support 450 on-line users. With the estimate in section 2.2 of 34 bytes for the user context this requires a total of 15.3 Kbytes of storage.

As we saw in section 5.2, even 450 users would not present much of a load to a single CPU, but practical considerations would probably cause the user interfaces to be divided amoung two or more CPU's.

In summary then a VAM system can service 90% of requests to a 50,000 page data base using a Winchester type disc with 5 Mbytes of storage. The system can support 450 on-line users using about 130 Kbytes of memory for data storage. The exact number of CPU's and the ammount of memory needed for program storage must be determined experimentally.

6. CONCLUSIONS

The main conclusions from this work may be summerized as follows:

Conclusions relating to storage structures of page-oriented databases:

The memory space required in central memory for the storage of the user context information for the active subscribers is small compared to the size of the page directory, and to the space that may be occupied by most frequently accessed information pages.

Among the different replacement strategies for deciding which pages to put from the fast buffer memory into the slower disk storage, such as "last recently used", "least frequently used", or "CLIMB", the CLIMB algorithm seems to be the the best. However, further reseach is required to determine how it could be best adapted to the videotex environment, including the determination of certain operational parameters.

Optimal allocation of the information pages onto fast buffer memory and slower disk storage can be determined based on the non-uniform access pattern of the subscribers to the data. However, not enough statistical data is currently available on such usage patterns to make general conclusions on such optimal

allocations.

A hashing technique seems appropriate for finding requested pages in the database. Part of the hashing directory may be resident in the main memory of the computer.

The memory allocation strategies discussed above are applicable for both telephone and TV channel transmission, as long as pages can be arbitrarily requested by the subscibers.

In the case of a broadcast cycle combined with directly requested pages, the strategy for placing certain pages on the broadcast cycle or not may be determined by the same method as for the question or placing pages in fast or slower memory (see above).

Allocating the memory storage hierarchy over two separate types of machine results in a distributed videotex system with several advantages over a centralized system. The user interface machines can service large numbers of subscribers using relatively cheap hardware.

Conclusions relating to keywords.

For the implementation of simple keyword access, the use of hashing techniques seems quite appropriate.

For the implementation of keyword search with boolean conditions, the use of special-purpose hardware, such as CCD devices or bubble memory may be useful, but it is not clear whether its use would be economically advantageous at this time.

BIBLIOGRAPHY

[1] Aho, A.V., Denning, P.J., and Ullman, J.D. "Principles of Optimal Page Replacement". Journal of ACM, 18, 1 (Jan. 1971), pp. 80-93.

[2] Babb, E., "Implementing a Relational Database by Means of Specialized Hardware". ACM Trans. on Database Systems, 4, 1 (March 1979).

[3] Banerjee, J., and Hsaio, D.K. "DBC - A Database Computer for Very Large Databases". IEEE Trans. on Computers, C-28, 3 (1979).

[4] Berra, P.B., and Oliver, E. "The Role of Associative Processors in Data Base Machine Architecture". Computer, 12, 3 (March 1979), pp. 53-61.

[5] Bhandarkar, D.P., Barton, J.B., and Tasch, A.F. Jr. "Charge-Coupled Device Memories: A Perspective". Computer, 12, 1 (Jan. 1979), pp. 16-24.

[6] Bird, R.M., Tu, J.C., and Worthy, R.M. "Associative Parallel Processors for Searching Very Large Textual Data Bases". Proceedings of 3rd Workshop on Comp. Arch. for Non-numeric Processing, (May 1977), pp. 8-16.

[7] Brookes, B.C. "Bradford's Law and the Bibliography of Science". Nature, vol. 224 (Dec. 1969), pp. 953-956.

[8] Bunt, R.B., and Harbus, R.S. "An Evaluation of Paging Storage Hierarchies". Tech. Report 80-8, Dept. of Computational Science, University of Saskatchewan, 1980.

[9] Bunt, R.B., Harbus, R.S., and Plumb, S.J. "The Effective Management of Paging Storage Hierarchies". Tech. Report 82-3, Dept. of Computational Science, University of Saskatchewan, 1982.

[10] Chang, E. "Sending Pages" (page 165) in The Telidon Book edited by D. Godfrey and E. Chang, Press Porcepic, Toronto, 1981.

[11] Chen, T.C., Lum, V.W., and Tung, C. "The Rebound Sorter: An Efficient Sort Engine for Large Files". Proc. 4th Intl. Conf. on Very Large Data Bases, (1978), pp. 312-315.

[12] Chi, C.S. "Advances in Computer Mass Storage Technology". Computer, 15, 5 (May 1982), pp. 60-74.

[13] Denning, P.J. "Virtual Memory". Computing Surveys, 2, 3 (Sept. 1970), pp. 153-189.

[14] Edelberg, M., and Schissler, L.R. "Intelligent Memory". Proceedings of 1976 NCC. AFIPS Press, pp. 393-400.

[15] Franaszek, P.A., and Wagner, T.J. "Some Distribution-Free Aspects of Paging Algorithm Performance". Journal of ACM 21, 1 (Jan. 1974), pp. 31-39.

[16] Fredkin, E.H. "Trie Memory". Communications of ACM, 3, 9 (Sept. 1960), pp. 490-499.

[17] Gecsei, J. "Montreal Keyword System". Technical Report 431, Département d'informatique et de recherche opérationnelle, Université de Montréal (Dec. 1981).

[18] Godfrey, D. "Hardware Configurations for Telidon Videotex Systems" (page 98) in The Telidon Book edited by D. Godfrey and E. Chang, Press Porcepic, Toronto, 1981.

[19] Groos, O.V. "Bradford's Law and the Keenan-Atherton Data". American Documentation, 18, 1 (Jan. 1967), p. 46.

[20] Haspers, J.H. "The Yield Formula and Bradford's Law". J. Am. Soc. Inf. Sci., 27, 5 (Sept. 1976), pp. 281-287.

[21] Healy, L.D. "A Character-Oriented Context-Addressed Segment-Sequential Storage". Proceedings of Third Ann. Symposium on Computer Architecture, (Jan. 1976), pp. 172-177.

[22] Horspool, R.N. "Practical Fast Searching in Strings". Software - Practice and Experience, vol. 10 (1980), pp. 501-506.

[23] Houle, J.L., personal communication, 1982.

[24] Lafitte, C. "Simulation de Réseaux de Télécommunications Domestiques". M.A.Sc. thesis, Département de Génie Electrique, Ecole Polytechnique, Montreal, May 1980.

[25] Lin, C.S., Smith, D.C.P., and Smith, J.M. "The Design of a Rotating Associative Memory for Relational Database Applications". ACM Trans. on Database Systems, 1, 1 (March 1976), pp. 53-65.

[26] McGill University Computing Centre "Introduction to Disk and Tape Usage", 1982.

[27] McGregor, D.R., Thomson, R.G., and Dawson, W.N., "High Performance for Database Systems". Systems for Large Databases, North-Holland, 1976, pp. 103-116.

[28] Ozkarahan, E.A., Schuster, S.A., and Smith, K.C., "RAP - An Associative Processor for Data Base Management". AFIPS Conference Proceedings, 1975 NCC, pp. 370-387.

[29] Parhami, B. "A Highly Parallel Computing System for Information Retrieval". AFIPS Conference Proceedings, 1972 FJCC, vol. 41, pp. 681-690.

[30] Parker, J.L. "A Logic per Track Retrieval System". Proceedings of IFIP Congress (1971), pp. TA-4-146 to TA-4-150.

[31] Peachey, J.B. "The Bradford-Zipf Distribution and Program Behaviour". Tech. Report 82-1, Dept. of Computational Science, University of Saskatchewan (1982).

[32] Rivest, R. "On Self-Organizing Sequential Search Heuristics". Communications of ACM, 19, 2 (Feb. 1976), pp. 63-67.

[33] Roberts, D.C. "A Specialized Computer Architecture for Text Retrieval". Proc. of Fourth Non-Numeric Workshop, (Aug. 1978), pp. 51-59.

[34] Rudolph, J.A. "A Production Implementation of an Associative Processor: STARAN", AFIPS Conference Proceedings, 1972 FJCC, vol. 41, pp. 229-241.

[35] Smith, D.C.P., and Smith, J.M. "Relational Data Base Machines". Computer, 12, 3 (March 1979), pp. 28-38.

[36] Standish, T.A. Data Structure Techniques, Addison-Wesley, 1980.

[37] Su. S.Y.W., and Lipovski, G.J. "CASSM: A Cellular System for Very Large Data Bases". Proceedings of Conf. on Very Large Data Bases, (Sept. 1975), pp. 456-472.

[38] Zipf, G.K. Human Behaviour and the Principle of Least Effort, Addison-Wesley, 1949.

[39] Bochmann, G.v., Gecsei, J., and Lin, E. (BC Telephone Comp.), "Keyword access in Telidon: An experiment", Proc. Videotex 82, to appear.

[40] Ball, A., Bochmann, G.v., and Gecsei, J., "Videotex Networks", IEE Computer, vol. 13, no. 12 (December 1980), pp. 8-14.

[41] Ball, A., Bochmann, G.v., and Gecsei, J., "User interfaces for videotex", Techn. Report, Univ. of Montreal, 1981.

[42] Tompa, F.W., Bochmann, G.v., and Gecsei, J., "Alternative database strategies for videotex", chap. 8.5 in The Telidon Book, ed. D.Godfrey and E. Chang, Press Porcepic Ltd., Toronto, 1981.

[43] Teorey, T.J., "Properties of disk scheduling policies in multiprogrammed computer systems", AFIPS Conference Proceedings, Fall Joint Computer Conference, vol. 41, 1972.

[44] Allen, A.O., "Elements of Queuing theory for system design", IBM System Journal, vol. 14, no. 2, pp. 161-187 (1975).

APPENDIX A1 - FITTING THE BRADFORD-ZIPF DISTRIBUTION

The following equations provide a convenient method of finding the parameters of the Yield Formula used to describe the Bradford-Zipf distribution.

The general form of the Yield Formula is:

$$fb(m) = h \log(m/u + 1) + fb(0)$$

Initially, we find three values for m such that the corresponding fb(m) values are equally spaced. That is, we find m1, m2 and m3 such that:

$$fb(m1) = A$$

$$fb(m2) = A + B$$

$$fb(m3) = A + 2B$$

Then we compute h, u and fb(0) as follows:

$$u = \frac{m2^{**}2 - m1*m3}{m1 - 2*m2 + m3}$$

$$h = B / \log(b) \text{ where } b = (m2+u)/(m1+u)$$

$$fb(0) = A - h*\log(m1/u + 1)$$

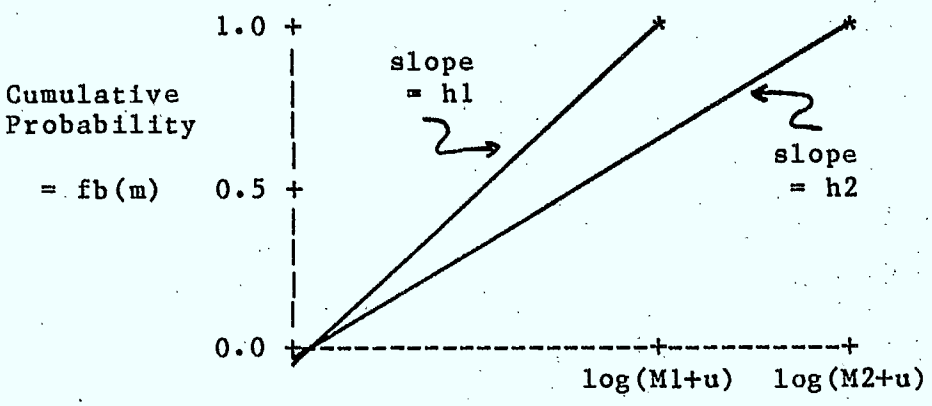
APPENDIX A2 - EXTRAPOLATING THE YIELD FORMULA

Given that we have a yield formula that applies to a small test data base, we propose to extrapolate this formula for a larger sized data base by changing the parameters of Equation 2.

First we note that the factor h appears in the calculation of fb(m) for all values of m except m=0. Setting b0 = fb(0)/h we have the more natural form

$$fb(m) = h\,(\log(m/u + 1) + b0)$$
$$= h\,(\log(m + u) - \log u + b0) \qquad \text{(Eqn. A1)}$$

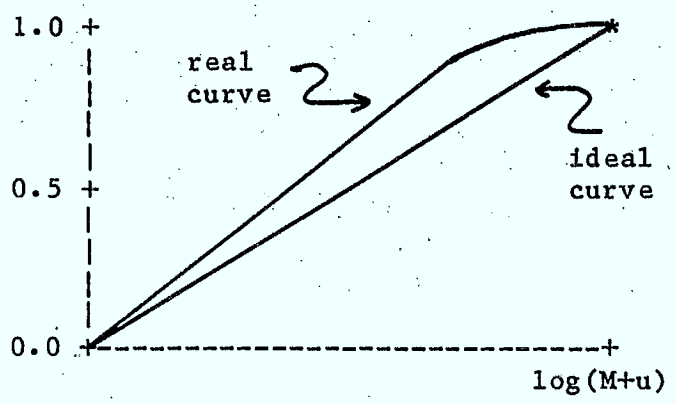
Here the equation is seen to be linear in log(m + u) with slope h. Letting M1 be the data base size, we should have fb(M1) = 1, a cumulative access probability of unity. For a larger data base size of M2 , we would have another Yield formula fb2 , such that fb2(M2) = 1. The second formula can then be derived from the first by changing the value of h.



Expected Yield formulas for data base sizes M1 and M2

Figure A1

If we keep u and b0 constant then h is determined solely from the value of M , the data base size. The real data however is only linear for low values of m , after which the curve falls off (the "Groos Droop"), so the slope h calculated from the data is larger than we would expect.



Comparison between the Yield formula and the actual data

Figure A2

If however we postulate that the slope calculated from the data is proportional to the ideal slope calculated from the data base size, then this proportionality can be maintained for the extrapolated equation by multiplying by the ratio of the ideal slopes. For this we need an expression for the ideal slope h′ as a function of the data base size M. Using fb(M) = 1 as before we have:

$$h' = \frac{1}{\log(M + u) - \log u + b0} \qquad \text{(Eqn. A2)}$$

Letting h1 be the slope calculated from the data for the test data base of size M1 , we calculate h2 the slope for a data base of size M2 using a factor k such that h2 = k h1. The factor k is then

$$k = \frac{\log(M1 + u) - \log u + b0}{\log(M2 + u) - \log u + b0} \qquad \text{(Eqn. A3)}$$

Substituting values from the equation in section 2.5 and using M1 = 3795 and M2 = 50,000 we get k = .755 which gives a new Yield formula applicable to a data base of 50,000 pages.

$$fb2(m) = 0.114 \log(m / 0.731 + 1) - 0.069 \qquad \text{(Eqn. A4)}$$