# OPTIMAL FOLDING OF A PAGING DRUM IN A THREE LEVEL MEMORY SYSTEM

by

Lee J. Scheffler

Massachusetts Institute of Technology
Department of Electrical Engineering and Project MAC

## ABSTRACT

This paper describes a drum space allocation and accessing strategy called "folding", whereby effective drum storage capacity can be traded off for reduced drum page fetch time. A model for the "folded drum" is developed and an expression is derived for the mean page fetch time of the drum as a function of the degree of folding. In a hypothetical three-level memory system of primary (directly addressable), drum, and tertiary (usually disk) memories, the tradeoffs among drum storage capacity, drum page fetch time, and page fetch traffic to tertiary memory are explored. An expression is derived for the mean page fetch time of the combined drum-tertiary memory system as a function of the degree of folding. Measurements of the MULTICS three-level memory system are presented as examples of improving multi-level memory performance through drum folding. A methodology is suggested for choosing the degree of folding most appropriate to a particular memory configuration.

## 1.0 Introduction

Many computer systems today employ automatically managed multi-level memory systems of successively larger, slower, and cheaper storage devices to provide rapid access to large address spaces. This paper is concerned with a three-level paged virtual memory system with a rotating drum as the intermediate level of storage between primary (directly addressed) and tertiary memories. Specifically, this paper describes a drum space allocation and accessing strategy called "folding", whereby the effective storage capacity of the drum can be traded off for reduced drum access time. Several identical copies of each page are maintained on the drum, spaced equally around the drum circumference. A request to fetch a page from the drum is served by reading the copy of the page closest to the read heads.

With a simple modification to Coffman's drum analysis [3], an expression is derived for the mean page fetch time* of a drum as a function of its storage capacity, speed, degree of "folding", and mean page fetch request arrival rate. Using a general success function for the probability that any randomly chosen page fetch request finds a copy of the requested page on the drum, an expression is derived for the overall mean page fetch time of the combined drum-tertiary memory system as a function of the degree of folding. The "linear paging" model of Saltzer [5] is employed in an illustrative example of the application of this analysis to a real computer system. In appendix A, measurements of the MULTICS system [10] under a repeatable benchmark user load for several very different configurations of number of CPUs and size of primary memory provide some experimental verification of the analysis, but more importantly, demonstrate the value of the flexibility of being able to fold the drum in system tuning. Consideration is given to the practical limitations on the degree of drum folding. Finally, a method is suggested for the system designer to take advantage of the three-way tradeoff of drum size, speed, and degree of folding, to improve the cost-effectiveness of a multilevel memory system.

* For purposes of this paper, mean page fetch time of a memory subsystem is taken as the mean or expected time between the arrival of a demand page fetch request at the subsystem access algorithm queues and the arrival of the last bit of the requested page at the main memory of the system. This does not include any software overhead times involved in page exception handling.

This research was supported by the Advanced Research Projects Agency of the Department of Defense under ARPA Order No. 2095, which was monitored by ONR Contract No. N00014-70-A-0362-006.

## 2.0 The Three-Level Memory System

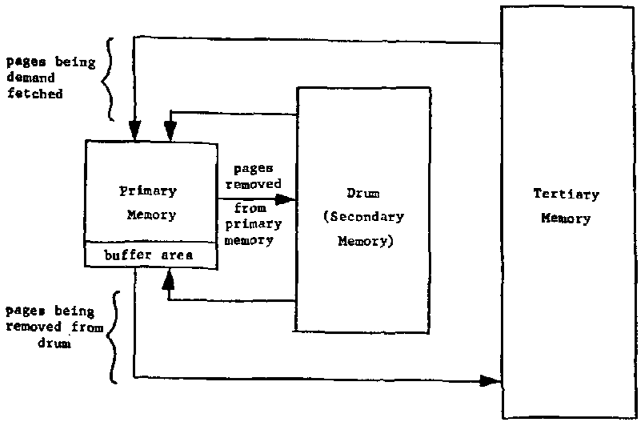
Our model of a three-level paged virtual memory system consists of:

1. a primary memory of size M page frames (usually core), that is directly referenced by programs, into which pages not already present are loaded as they are referenced;

2. an intermediate paging device of size D page frames (drum, in this paper) on which copies of pages being used by currently executing programs may reside when they are not in primary memory; and

3. a tertiary memory (usually disk) on which each page of program or data in the system maintains a permanent residence.

The flow of pages in this system is shown in figure 1. This flow is managed automatically (by



pages being demand fetched
Primary Memory
buffer area
pages removed from primary memory
Drum (Secondary Memory)
Tertiary Memory
pages being removed from drum

Figure 1

Flow of pages in a 3-level paged virtual memory

system programs invoked when a page needs to be moved) according to the following rules:

1. When a page not present in primary memory is referenced, if a valid copy of the page (a copy that includes the most recent modifications to the page) exists on the drum, the drum copy is fetched into primary memory. If no valid copy of the page exists on the drum, the copy in tertiary memory is fetched.

2. When the primary memory page replacement algorithm decides to remove a page from primary memory, the page is written onto the drum.

3. When the drum page replacement algorithm decides to remove a page from the drum, the page is first read into a buffer in primary memory and then written out to its permanent tertiary memory address.

## 3.0 The Drum

The drum to be considered has T tracks, each with its own set of read-write heads, and each divided into S sectors, as shown in figure 2. The total storage capacity of the drum is thus $D = S \cdot T$ page frames. Successively numbered sectors on

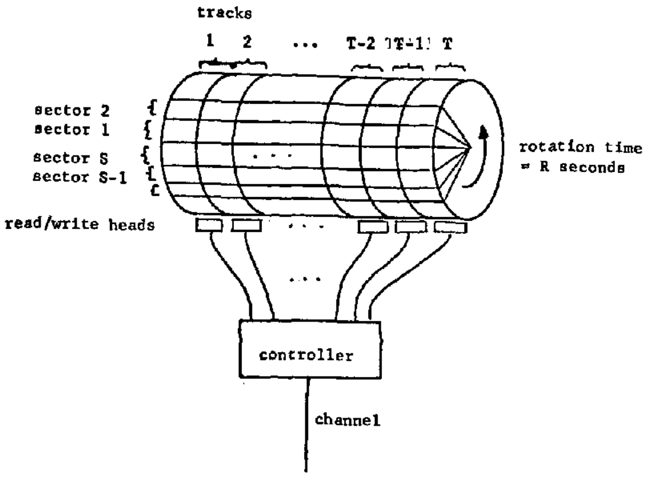


Figure 2

Organization of a sectored drum

different tracks can be accessed (either read or written) without loss of a revolution. The drum rotates in R seconds, so that page transmission time is fixed at R/S seconds.

A pair of first-come-first-served (FCFS) queues of arriving access requests is maintained for each drum sector, as shown in figure 3. Arriving demand page fetch requests for pages on drum sector k go into "fetch queue" k; other drum access requests go into "write queue" k. When sector k arrives under the heads, the page for the first request in fetch queue k is read. If fetch queue k is empty, the page for the first request in write queue k is accessed.

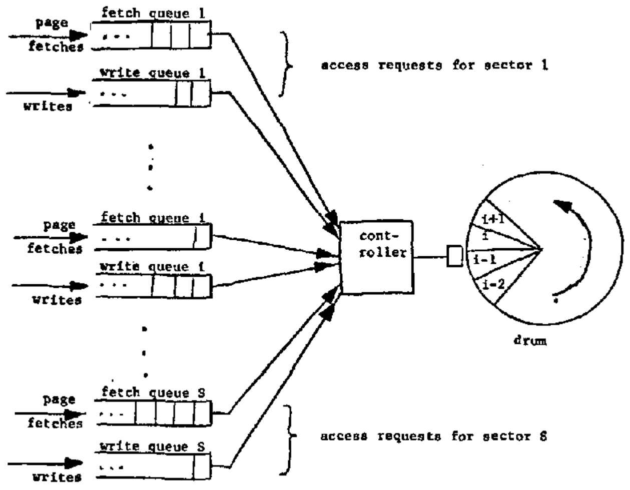


Figure 3

Queues for sectored drum scheduling algorithm

## 3.1 Sectored Drum Analysis

In [3], Coffman analyzed a model of a sectored paging drum where all arriving drum access requests for each sector are queued in a single queue in first-come-first-served order. In the model of the previous section, since all page fetch requests for each sector are served before any page write requests, and since the service of a write request never causes the service of a page fetch request to be delayed or prolonged, Coffman's model and analysis are directly applicable to studying drum behavior in our model with respect to page fetch requests. The major results of his analysis relevant to this paper are summarized here.

Let the inter-arrival times between page fetch requests to the entire drum be independent random variables distributed as $\lambda_f e^{-\lambda_f t}$, where $\lambda_f$ is the mean arrival rate of page fetch requests to the drum. Let the page fetch requests be distributed uniformly over S fetch queues, so that arrivals of page fetches to any sector k also form a Poisson process, with mean arrival rate $\frac{\lambda_f}{S}$. Then, according to Coffman, the mean page fetch time of the drum (time between arrival of the page fetch request at a fetch queue and completion of reading) is given by

$$\mathrm{mpft}_{\mathrm{drum}} = \overbrace{\frac{R}{S}}^{\text{page transmission time}} \tag{1a}$$

$$+ \underbrace{R[\rho(\frac{S+2}{2S}) + \frac{\rho^2}{2(1-\rho)} + \frac{(1-\rho)}{\rho} e^{\rho(1-1/S)}(1-e^{-\rho}) - \frac{1}{2}]}_{\text{queue waiting time}}$$

where $$\rho = \lambda_f \frac{R}{S} \tag{1b}$$

is the "utilization fraction" of the drum, or the expected fraction of drum page transfer capacity (the drum can transfer, by virtue of its rotational speed, S/R pages per second) that is used to transfer pages being fetched.*

## 3.2 The Folded Drum

Suppose that, instead of using every page frame of the drum to contain a different page, we use N page frames, spaced equally around the drum circumference, to contain identical copies of the same page. Clearly, this reduces the storage capacity of the drum quite drastically. However, if the drum access algorithm is modified to serve page fetch requests by reading the copy of the page closest to the heads, we find that page fetch time of the drum is also dramatically reduced. This strategy of maintaining N copies of each page on the drum and reading the one closest to the heads we will call "folding the drum N times". Let us attempt to quantify this tradeoff of size for speed.

First, let us clearly state the operation of the queue arrival and service disciplines with respect to demand page fetches. Page writes are discussed in a later section. An arriving page fetch request has the choice of entering any one of N different fetch queues for the N different sectors where copies of the page are maintained Thus, the S sector fetch queues are partitioned into S/N (assume that N divides S exactly) classes such that two fetch queues for sectors i and j are in the same partition if and only if i mod S/N = j mod S/N. The policy for choosing the fetch queue into which an arriving fetch request will go is quite simple: put it into the queue from which it will be serviced earliest. This amounts to finding the subset of fetch queues in the partition of this request that have the smallest number of requests in them, and then putting the new request into the queue for the sector that will arrive under the heads soonest. (There are far simpler ways of implementing this strategy than indicated here. However, this description relates the essential effect of the algorithm.) Figure 4 depicts this partitioning of sector fetch queues.

* Coffman defines "drum utilization" as the average number of sectors accessed (fetched, in this case) per drum revolution. $\rho$, as defined here, is Coffman's "drum utilization" divided by S, the number of drum sectors.

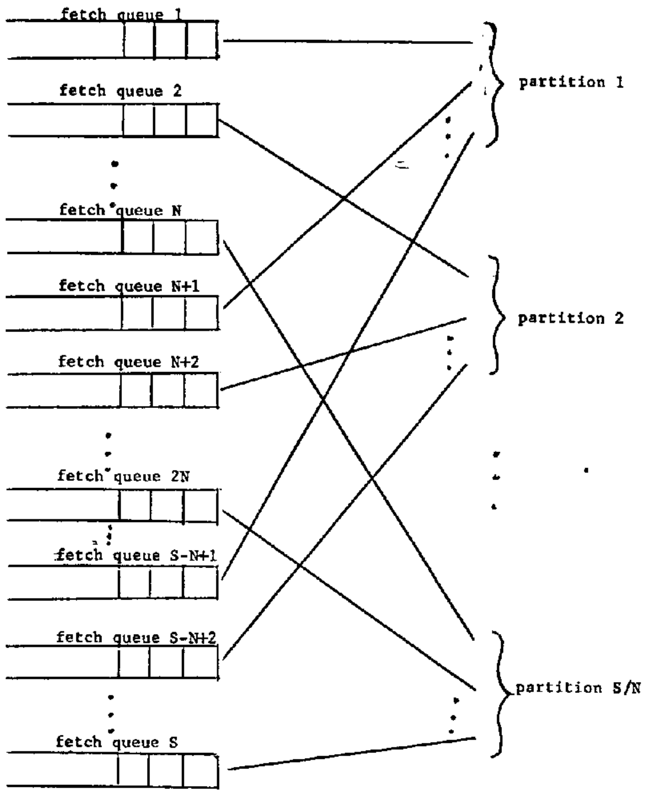


Figure 4
Partitioning the S sector fetch queues into S/N partitions for a drum folded N times

Note that arrivals to individual fetch queues no longer constitute a Poisson process, since the placing of a request in a queue is dependent on both the number there already, and the current drum position. Thus, Coffman's results are not directly applicable.

This difficulty is circumvented by noting that, for fetch requests, this model can be transformed into the following equivalent model. Let the drum in our new model rotate in R/N seconds and have S/N sectors and T tracks. The storage capacity of this new drum is thus D/N pages, the effective storage capacity of the larger drum folded N times. The time a single sector is under the heads is (R/N)/(S/N) = R/S seconds, the same as in the old model, so page transmission time is unchanged. Each partition of sector fetch queues in the old model is mapped into a single sector fetch queue in the new model whose number is the residue modulo (S/N) of the number of any one of the sectors in this partition in the old model. Figure 5 depicts the queue organization of this new drum model.

Assume that inter-arrival times of page fetch requests to the folded drum are still independent random variables obeying the exponential density function $\lambda_{f,N}\, e^{-\lambda_{f,N} t}$, where $\lambda_{f,N}$ is the mean arrival rate of page fetch requests to the N-folded drum, and that page fetch arrivals are uniformly distributed over the sectors of our

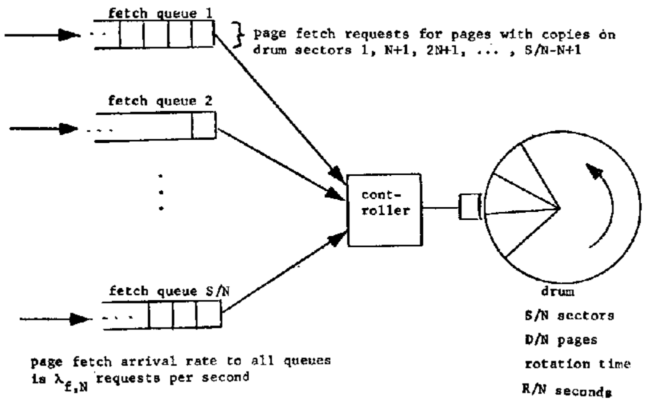

fetch queue 1
page fetch requests for pages with copies on drum sectors 1, N+1, 2N+1, ... , S/N-N+1
fetch queue 2
fetch queue S/N
controller
drum
S/N sectors
D/N pages
rotation time
R/N seconds
page fetch arrival rate to all queues is $\lambda_{f,N}$ requests per second

Figure 5

Queue Organization of N-Folded Drum Model

new smaller drum.* Then page fetch arrivals to each individual sector fetch queue in our new model constitute a Poisson process. We thus regain the use of Coffman's drum analysis results, to give us an expression for $mpft_{drum}(N)$, the mean page fetch time of the drum folded N times:

## 3.3 Evaluating Folded Drum Mean Page Fetch Time

In order to evaluate this rather forbidding expression for a particular drum, we need to know $\lambda_{f,N}$, the mean arrival rate of page fetch requests to the drum folded N times, or equivalently, $\rho_N$, the fetch utilization fraction of the N-folded drum. These can be measured directly, or predicted from knowledge of the combined mean page fetch arrival rate of both drum and teriary memories.

Consider the three-level memory system of Figure 6. $\mu_f$ is the mean arrival rate of page fetch requests to the combined drum-tertiary memory system. $\lambda_{f,N}$ of these $\mu_f$ page fetches per unit time have valid copies of their pages resident on the drum; the remainder require tertiary memory accesses. Let us make the simplifying assumption that $\mu_f$ is independent of D/N; i.e. that the mean page fault rates remain constant as effective drum size, and derivatively, mean page fetch time of the combined memory system, changes due to greater or lesser drum folding.** Let $P(M,\delta)$ represent the probability that a randomly chosen page fetch request finds a copy of the missing page on the drum, for main memory size M page frames, and effective drum size $\delta$ page frames. Setting $\delta = D/N$, we obtain

$$\lambda_{f,N} = P(M,\ D/N)\ \mu_f \tag{3a}$$

* A sufficient but not necessary condition for these assumptions to be true, given that they are true for the unfolded drum (N=1), is the following: Page fetch requests directed to the drum before folding, but directed to tertiary memory after folding due to decreased drum size, are chosen at random from the page fetch requests that went to the drum before folding.

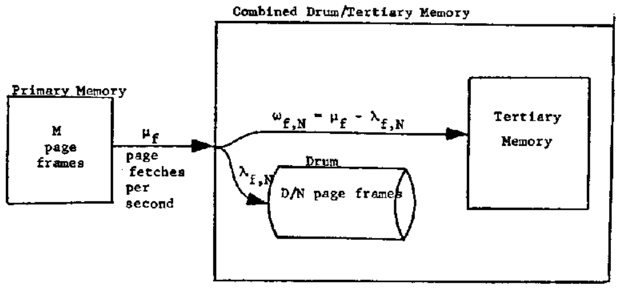

Combined Drum/Tertiary Memory
Primary Memory
M page frames
$\mu_f$ page fetches per second
$\omega_{f,N} = \mu_f - \lambda_{f,N}$
Tertiary Memory
$\lambda_{f,N}$
Drum
D/N page frames

Figure 6

Page fetch arrival rates in a three-level memory

and

$$\rho_N = \frac{R}{S}\ P(M,\ D/N)\ \mu_f \tag{3b}$$

Several existing models for individual program and multiprogramming paging behavior supply functions similar to $p(M,\delta)$. [1,2,4,7] Given an explicit function for $P(M,D/N)$, equation 3b may be substituted into equation 2a to produce the desired expression for $mpft_{drum}(N)$, the mean page fetch time of the drum folded N times.

As an illustrative example, let us employ Saltzer's linear model for multi-level memory demand paging performance† to obtain $\lambda_{f,N}$. According to Saltzer, the mean headway‡ between references (mhbr) to a page for which no copy exists in memory levels 1 through i is directly proportional to the sum of the sizes of memory levels 1 through i. Assuming that $\mu_f$ is proportional to 1/mhbr (i.e. that mean instruction execution time is constant), then Saltzer's model claims that

$$\mu_f = \frac{a}{M} \tag{4}$$

where a is a constant of proportionality. Focusing on the fraction of the $\mu_f$ page exceptions+ that are not satisfied on the drum we have that

** In the experiments on the MULTICS system reported in appendix A, it was found that, under multiprogramming of degree between 4 and 8, the mean page fault rate $\mu_f$ varied less than 25% over a wide range of drum foldings, so long as secondary memory did not become overloaded. Thus, although this is certainly not a generally valid assumption, it is sufficient for investigating the degree of drum folding that minimizes overall mean page fetch time.

† A discussion of the linear paging model and validating experiments appears in reference [3].

‡ mean number of machine instructions executed

+ instances of reference to a page not in primary memory

$$\mu_f - \lambda_{f,N} = \frac{a}{M + D/N} \qquad (5)$$

which, after the substititution of $\mu_f M$ for a (from equation 4), and some rearrangement, becomes

$$\lambda_{f,N} = \frac{D}{NM + D}\,\mu_f \qquad (6a)$$

suggesting that

$$P(M,\ D/N) = \frac{D}{NM + D} \qquad (6b)$$

for this model. By equation 2b, then,

$$\rho_N = \frac{R}{S}\,\lambda_{f,N} = \frac{R}{S}\,\frac{D}{NM + D}\,\mu_f \qquad (7)$$

($\rho_N$ is the "fetch utilization fraction" of the drum folded N times), which, upon substitution into equation 2a, gives the desired expression for $\mathrm{mpft}_{drum}(N)$.

## 3.4 Optimal Folding

We take the mean page fetch time ($\mathrm{mpft}_{total}$) of the combined drum-tertiary memory system (Figure 6) as the measure of memory performance that we wish to minimize. Given the first-order independence between mean page fetch request generation rate $\mu_f$ and mean page fetch time $\mathrm{mpft}_{total}$ (discussed in section 3.3), minimizing mean page fetch time for a constant page fetch rate minimizes the real time a program spends waiting for pages to be fetched This has a generally desirable effect on overall system performance (e.g. response time to users' requests for computation and throughput of users' useful work per unit time).

$\mathrm{mpft}_{total}$ is given by

$$\mathrm{mpft}_{total} = P(M,\ D/N)\ \mathrm{mpft}_{drum}(N) + [1-P(M,\ D/N)]\ \mathrm{mpft}_{tertiary}\ (\omega_{f,N}) \qquad (8a)$$

where

$$\omega_{f,N} = \mu_f - \lambda_{f,N} = [1-P(M,\ D/N)]\ \mu_f$$

is the mean arrival rate of page fetch requests to tertiary memory with the drum folded N times. $\mathrm{mpft}_{tertiary}$ is a function of $\omega_{f,N}$ because, in general, the mean page fetch time of a tertiary memory system will depend on the arrival rate of page fetch requests.

With $\mathrm{mpft}_{drum}(N)$ given by equations 2, P(M, D/N) given by the character of the specific system, and $\mathrm{mpft}_{tertiary}(\omega_{f,N})$ specified by the particular tertiary memory system, a curve of $\mathrm{mpft}_{total}$ versus N for possible values of N* can be constructed. In general, such a curve will either have a single minimum, indicating that $\mathrm{mpft}_{total}$ can be enhanced by folding the drum, or will have positive slope for its whole length, indicating that an unfolded drum is best. Investigating the minimum of equation 8a with respect to N is of little merit, since the value of N that produces this minimum will in general be non-integral, and thus will not correspond to a possible folding. We are really only interested in comparing the values of $\mathrm{mpft}_{total}$ corresponding to possible values of N.

* See section 3.6 for constraints on the possible values of N.

## 3.5 Example of Optimal Folding Using Linear Paging

To illustrate the process of evaluating equation 8a for a specific system, we again make use of Saltzer's linear paging model. The necessary equations are reproduced below:

For $\mathrm{mpft}_{drum}(N)$:

$$\mathrm{mpft}_{drum}(N) = \frac{R}{S} + \frac{R}{S}[\rho_N(\frac{S+2N}{2S}) + \frac{\rho_N^2}{2(1-\rho_N)} + \frac{(1-\rho_N)}{\rho_N}\, e^{\rho_N(1-N/S)}(1-e^{\rho_N}) - \frac{1}{2}] \qquad (2a)$$

From the linear paging model:

$$P(M,\ D/N) = \frac{D}{NM + D} \qquad (6b)$$

so that

$$\rho_N = \frac{R}{S}\,\frac{D}{NM + D}\,\mu_f \qquad (7)$$

To simplify this example, we take

$$\mathrm{mpft}_{tertiary}(\omega_{f,N}) = A \qquad (9)$$

(by which we assume that tertiary memory page fetch time remains constant over the range of page fetch loads placed on it by successive drum foldings).

Then, from equation 8a, $\mathrm{mpft}_{total}$ is given by:

$$\mathrm{mpft}_{total} = \frac{1}{NM + D}\cdot \Big[D\Big\{\frac{R}{S} + \frac{R}{N}[\rho_N(\frac{S+2N}{2S}) + \frac{\rho^2_N}{2(1-\rho_N)} + \frac{(1-\rho_N)}{\rho_N}\, e^{\rho_N(1-\frac{N}{S})}(1-e^{-\rho_N}) - \frac{1}{2}]\Big\} + NM\cdot A\Big] \qquad (10)$$

## 3.6 Constraints on Folding

There are several obvious constraints on N, the number of drum folds. N must be integral. It must be true that $1 \le N \le S$. And N must divide S exactly. (One could conceive of more complex drum folding schemes where N does not divide S exactly; these are not considered here.)

The total combined arrival rate of demand page fetch and write requests for the drum must not exceed the page transfer capacity of the drum. If $\lambda_{f,N}$ and $\lambda_{w,N}$ represent the arrival rates of demand page fetch and page write requests to the drum folded N times, then this condition is expressed as

$$\lambda_{f,N} + N\,\lambda_{w,N} < \frac{S}{R} \qquad (11)$$

($\lambda_{w,N}$ is multiplied by N because each request to write a page onto the drum blossoms into N requests to write N copies of the page on N different sectors.) If $\mu_f$ and $\mu_w$ are the respective combined arrival rates of page fetch and write requests to the drum-tertiary memory, and $P_f(M,\delta)$ and $P_w(M,\delta)$ are the respective probabilities that a particular page fetch or write request will go to the drum with capacity $\delta$ pages, then this condition becomes

$$\mu_f P_f(M, D/N) + N\,\mu_w P_w(M, D/N) < \frac{S}{R} \qquad (12)$$

which, when solved for N with specific functions for $P_f$ and $P_w$, gives an upper bound on the number of folds so as not to overload the drum. In practice one should choose N somewhat less than this bound because, as drum capacity is approached, the time to complete the writing of all N copies of a page will increase markedly, and primary memory may become tied up with copies of old pages that have not yet been completely written out.

Finally, the increase in access traffic to tertiary memory must not overload it. If C is the maximum average number of accesses per unit time that tertiary memory can support, then this condition is

$$\mu_f\,[1 - P_f(M, D/N)] + \mu_w\,[1 - P_w(M,D/N)] \le C \qquad (13)$$

### 3.7 Write Interference

When a drum is folded N times, each time a page is written, N different drum sectors must be written. Thus, although the number of distinct pages to be written onto the drum can be expected to decrease as the effective size of the drum is decreased by folding, the total number of sectors to be written increases somewhat faster with increasing N. Equations 11, 12 and 13 explore one aspect of this increase, defining upper bounds on N such that neither the drum nor tertiary memory become overloaded. However, since demand page fetch requests are given priority over write requests, the mean sector write time, and therefore the mean page write time, will increase drastically as maximum drum page transfer capacity is approached. The implication is that, when the primary memory page replacement algorithm decides to remove a particular page from primary memory, it will be some time before all N copies of that page are completely written onto the drum, and the primary memory space taken up by that page can be used for some incoming page. The danger is that a page fetch request may be forced to wait for the completion of writing of the primary memory page it will replace, thus effectively lengthening page fetch time.

This interference is viewed as a second-order effect. Paging systems often try to maintain a buffer of a small number of free primary memory page frames into which incoming pages may be read without waiting for a page to be written out. If the probability that a new page fetch request finds no free primary memory page frames into which it can be read is significant, that is an indication that the buffer being maintained is too small for the system operating point, and should be made larger. Since the buffer sizes we are considering are generally only a small fraction of the size of primary memory, their size can be increased without seriously reducing the primary memory available for paging.

## 4.0 Drum Cost Versus Mean Page Fetch Time

The introduction of the folded drum allocation and accessing strategy adds new complexity to the system design activity of choosing the drum to be used as the paging device of a multilevel memory. Where the system designer previously had to trade off only drum speed and drum size (which are primary determinants of drum cost) against mean page fetch time of the multi-level memory, he now may wish to consider the possibility of folding the drum. He now has the option, for example, of choosing a small, fast drum, to be used unfolded, or a larger and slower drum, to be used folded, to meet the same mean page fetch time objective. It is thus appropriate to suggest a methodology for dealing effectively with this three-way tradeoff.

The two most common questions one might expect a designer to ask are:

1. What is the minimum mean page fetch time that can be obtained for d dollars?
2. How much will it cost to keep mean page fetch time under m milliseconds?

The following methodology provides quantitative information to answer these questions.

1. For each drum on the market that satisfies interface specifications, find the possible folding that minimizes mean combined drum-tertiary page fetch time for the system (from equation 8).
2. On a single set of drum cost (vertical) versus mean page fetch time (horizontal) axes, construct a scatter plot with each point representing the cost and minimum combined mean page fetch time for each drum under consideration. (Remember to include the costs of implementing a folded drum algorithm, if they will be substantial.) Such a graph might look like that of Figure 7.

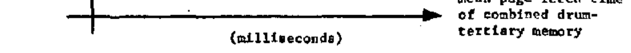


Figure 7

A scatter-plot of drum cost versus minimum attainable mean page fetch time for several hypothetical drums

The scatter-plot produced in step 2 becomes a source of cost-benefit information for choosing the drum and degree of folding that will be used.

One comment is appropriate concerning this choice. A drum that produces optimum mean page fetch time when folded mid-way in its range of possible foldings probably provides more flexibility in an evolving system than one that produces optimum mean page fetch time when completely unfolded (N=1) or when maximally folded (N=S). As primary memory size changes, or as mean time between page fetch requests changes (due perhaps to changing the number of CPUs, faster CPUs, new scheduling algorithms or changing the character of user load), a drum that can be further folded or unfolded to mirror shifts in the system's operating point will be more valuable than one that cannot be folded or unfolded further.

## 5.0 Summary and Conclusions

The "folded" drum allocation and accessing algorithm offers the computer system designer the flexibility to trade off drum size for speed as needed to improve multi-level memory performance, and to keep pace with evolving system configurations, programming, or user load. Sections 3.2 through 3.7 explore the quantitative aspects of this tradeoff for a specific model of a three-level paged virtual memory system, culminating in equations 2 and 8 which relate combined drum-tertiary memory mean page fetch time to the number of drum folds.

A model of the redistribution of page fetch requests between the second and third levels of a three-level memory system is required to apply the analysis to a real system. The "linear" paging model of Saltzer was present as an example of such application. The folded drum analysis assumes that times between successive arrivals of page fetch requests to the drum are independent of each other, a condition which is met to a first approximation by multiprogramming over a small number of jobs.

Appendix A presents results of experiments with the MULTICS three-level memory. Although these results cannot be taken as validating this model and analysis, they do demonstrate the usefulness of drum folding in system tuning.

A methodology was presented for deriving comparative cost-benefit information for different drums used as paging devices in a multi-level memory. This information should be useful in choosing between competitive drums during system design and upgrading activities. The method could conceivably be generalized to multi-level memories of more than three levels, and to include other memory devices than drums.

## APPENDIX A Measurements of the MULTICS Three-Level Memory with Drum Folding

The Honeywell 645 MULTICS system is a time-shared multi-programmed computer system with an automatically managed three-level memory composed of core, drum, and disk. The drum presently used by MULTICS has 16 sectors and 256 tracks, giving it a capacity of 4096 pages. Drum rotation time is 33.3 milliseconds. The MULTICS drum scheduling algorithm is quite similar in function to the two-priority folded drum scheduling algorithm analyzed in this paper (although it is very different in form).

Measurements were taken on the three very different configurations shown in Figures A1, A2, and A3. In each configuration, a number of benchmark user processes* were run simultaneously to provide a reproducible realistic user load. In each configuration, the number of drum folds was varied from 1 to 16 (8 for the "minimum" configuration) in powers of 2, the range of possible foldings implemented. For each configuration and degree of folding, average frequency of page exceptions from main memory ($\mu_f$) and drum, disk, and combined drum-disk mean page fetch times** were measured over approximately 10-20 minutes of time after the system reached an equilibrium state. Using equations 2a and 7, a prediction of drum mean page fetch time ($mpft_{drum}(N)$) was obtained. To this was added an estimate of software overhead for drum queue management. This prediction, together with the measured value of disk mean page fetch time (which already includes software overhead) were substituted into equations 8, with P(M, D/N) given by equation 6b, to obtain a prediction of overall drum-disk mean page fetch time, $mpft_{total}$. The measured and predicted values of $mpft_{total}$ for each configuration and degree of folding are presented side-by-side in table A1.

These experi ments should be taken with a grain of salt, for several reasons. The model described in this paper is not an exact model of the MULTICS drum scheduling algorithm. Several concessions to simplicity and efficiency were made in its implementation. In particular, software overhead times for drum queue management are not constant with varying numbers of

* For purposes of system tuning, comparisons of performance of successive versions of the MULTICS software system, and conductin controlled experiments. Douglas H. Hunt of M.I.T. Project MAC Computer Systems Research Division developed a method of simulating a realistic user load on the system. Several independent processes, each representing a single user, are run simultaneously. All processes execute the same single control program which invokes commands in a random order from a subset of the more frequently used system commands involved by users. The commands consist of editing, compilation, trial execution; debugging, final execution, and file system manipulation requests. The frequency of usage of each type of command from each process is adjusted to match frequency-of-usage statistics kept during normal system operation, so that, under multiprogramming, these processes exhibit the same distribution of resource and system program usage as observed for a normal user load.

** The mean page fetch times measured include software overhead times for drum and disk queue management and interrupt handling.

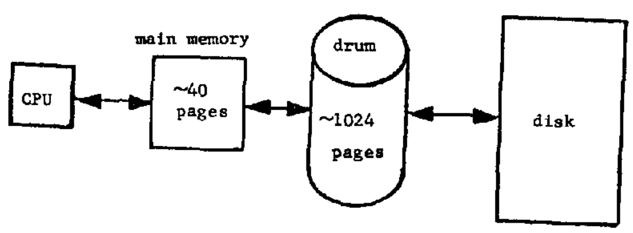


Figure A1. A "minimum" MULTICS configuration

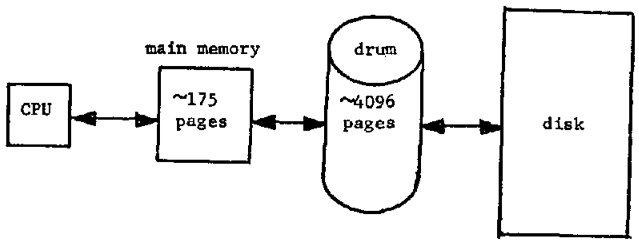


Figure A2. A "small" MULTICS configuration

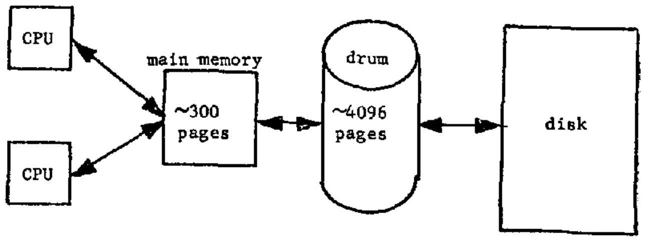


Figure A3. A "large" MULTICS configuration

| Configuration | No. Drum Folds | mean time between page faults | total mean page fetch time | |
|---|---|---|---|---|
| | | (msec) | measured (msec) | predicted (msec) |
| minimum (1 CPU, ~40 pages primary memory | 1 | 15.6 | 21.9 | 21.9 |
| | 2 | 12.4 | 14.9 | 13.8 |
| | 4 | 11.7 | 16.4 | 18.0 |
| | 8 | 18.0* | 23.2 | 25.5 |
| small (1 CPU, ~175 pages primary memory) | 1 | 11.8 | 23.5 | 22.0 |
| | 2 | 12.7 | 15.0 | 16.4 |
| | 4 | 12.2 | 16.2 | 18.4 |
| | 8 | 14.4 | 25.6 | 32.6 |
| | 16 | 27.7* | 71.5 | 76.1 |
| large (2 CPUs, ~300 pages primary memory) | 1 | 9.1 | 24.6 | 23.4 |
| | 2 | 11.3 | 15.3 | 20.2 |
| | 4 | 11.5 | 24.2 | 39.2 |
| | 8 | 27.7* | 84.4 | 88.4 |
| | 16 | 45.3* | 113.5 | 122.3 |

* = overload on secondary memory

Table A1

Measured and Predicted Combined Drum-Disk Mean Page Fetch Times for the MULTICS System

drum folds, and become a significant fraction of drum page fetch time for large numbers of drum folds (N=8 or N=16). The analysis in the body of the paper makes the assumption that software overheads are negligible, or at worst, not dependent on the number of drum folds. Also, Saltzer's linear paging model, used to obtain the drum page exception success funtion P(M, D/N), is not an exact model of paging in MULTICS. Nevertheless, the figures in table A1 demonstrate that folding a sectored drum in a three-level memory can result in measurable improvements in memory performance.

## ACKNOWLEDGEMENTS

Professor F. J. Corbató suggested folding the MULTICS paging drum and interested me in this project. Steven H. Webber programmed the MULTICS multi-level memory strategies and wrote many of the metering programs used in the experiments. Noel I. Morris implemented the folded drum algorithm on the Honeywell 645 MULTICS system. Douglas H. Hunt designed and implemented the benchmark script programs used in the experiments comparing different degrees of drum folding. Professors F. J. Corbató and J. H. Saltzer of M.I.T. and P. J. Denning of Purdue University provided guidance and critical appraisal of this work.

## REFERENCES

1. Aho, A. V., Denning, P. J., and Ullman, J. D., "Principles of Optimal Page Replacement", JACM, volume 18, number 1, January 1971, pp80-93.

2. Chow, C. K., "On Optimization of Memory Heirarchies", IBM research report RC 4015, September 5, 1972.

3. Coffman, E. G., "Analysis of a Drum Input/Output Queue Under Scheduled Operation in a Paged Computer System", JACM, volume 16, number 1, January 1969.

4. Denning, P. J., "Effects of Scheduling on File Memory Operation", Proceedings AFIPS, 1967 Spring Joint Computer Conference, volume 30, pp9-21.

5. Saltzer, J. H., "A Simple Linear Model of Demand Paging Performance", submitted for publication to Communications of ACM.

6. Toda, Iwao, "A Large-Scale Data Processing System: DIPS-1", proceedings of First USA-JAPAN Computer Conference, 1972, pp193-202.

7. Sekino, A., "Performance Evaluation of Multiprogrammed Time-Shared Computer Systems", Ph.D. thesis, MIT Project MAC TR-103, September, 1972.

8. IBM System/370 Model 158 Facts Folder, International Business Machines Corp., August 1972.

9. IBM System/370 Model 168 Facts Folder, International Business Machines Corp., August 1972.

10. The Multics Programmer's Manual, available from Honeywell Information Systems Inc., contains a complete bibliography of published papers and theses concerning the MULTICS system.