International Journal of Computer and Information Sciences, Vol. 2, No. 4, 1973

A Theory of Dynamic File Management in a Multilevel Store

Lawrence L. Rose[1,2] and Malcolm H. Gotterer[1,3]

Revised June 1973

This paper describes the concept of dynamic file management. The objective of the theoretical model defined is to minimize the throughput time of any computer system with a multilevel store. A measure of throughput time is derived to be a function of job run time, file movement time, and file configuration choice time. This, plus locally imposed exogenous policies, provides a basis for dynamically moving files within the multilevel store to minimize throughput time.

1. INTRODUCTION

The workload for many computer systems is ever-changing and difficult to predict far in advance. Yet secondary storage file allocation is static in almost every computer system. By this we mean that file position does not change with respect to changes in system or user demands. This results in a slowing down of the system whenever the workload requires extensive use of files in the lower levels of the multilevel store. It would be advantageous to have a system which dynamically moves files within the multilevel store to accommodate the current load. Denning[1] notes that "... a centralized resource allocation method is necessary to monitor the entire system and control resource usage to satisfy objectives both of good service and system efficiency."

[1] Department of Computer Sciences, Pennsylvania State University, University Park, Pennsylvania.

[2] Present address: School for Applied Technology, State University of New York at Binghamton, Binghamton, New York.

[3] Present address: Department of Mathematical Sciences, Florida International University, Tamiami Trail, Miami, Florida.

The model to be derived shall act as a guide for a Dynamic File Manager (DFM) whose task is to keep the files under reference optimally located at all times in the multilevel store. For the formal model, file position in the multilevel store shall be considered optimal if it minimizes throughput time. First we shall formally describe those portions of a computer system that are relevant to dynamics secondary storage file management. Then we shall derive a definition of throughput time. Finally, we shall derive the sequence M^* of file storage configurations which minimizes throughput time.

2. MODEL DEFINITION

This model is applicable only to computers having multilevel secondary storage. The basic theoretical model assumes a pure store: Each file in the system resides on one and only one device in the multilevel store. This is in no way a restriction; rather, it simplifies the basic definitions so they may be more easily understood.

A *Computer File System* (CFS) shall consist of a finite number of files f_i, secondary storage devices d_j, and jobs e_m:

$$\begin{aligned} CFS &= \{F, D, E\} \qquad \text{given} \\ F &= \{f_1, f_2, \ldots, f_p\} \\ D &= \{d_1, d_2, \ldots, d_q\} \\ E &= \{e_1, e_2, \ldots, e_r\} \end{aligned} \tag{1}$$

A *Demand Set* consists of the job file references placed upon *CFS* during work period k: $S_k \subseteq F \times E$,

$$S_k = \{(f_i, e_m)\colon \ e_m \ \text{ references } \ f_i \ \text{ during work period } \ k\} \tag{2}$$

Given a *CFS*, a *Configuration Set* is a snapshot of all files on the devices at the start of work period k: $C_k \subseteq F \times D$,

$$C_k = \{(f_i, d_j)\colon \ f_i \ \text{ is on } \ d_j \ \text{ at start of work period } \ k\} \tag{3}$$

While the definitions of S_k and C_k are concise, it is not clear exactly what a work period k constitutes. We assert that, given a *CFS*, one can start the system processing the jobs in E (batch, multiprogramming, whatever) and derive S_k and C_{k-1} for $k = 1, 2, \ldots$ until E is exhausted by the following algorithm.

Demand-Configuration Algorithm (see Fig. 1). Let N denote the maximum allowable size for any S_k.

```
1. $k \leftarrow 1$; ($S_t = \lambda$ for $t = 1, 2, \ldots$).
2. $C_{k-1} \leftarrow$ current file configuration [see (3)]; $L = 0$.
```

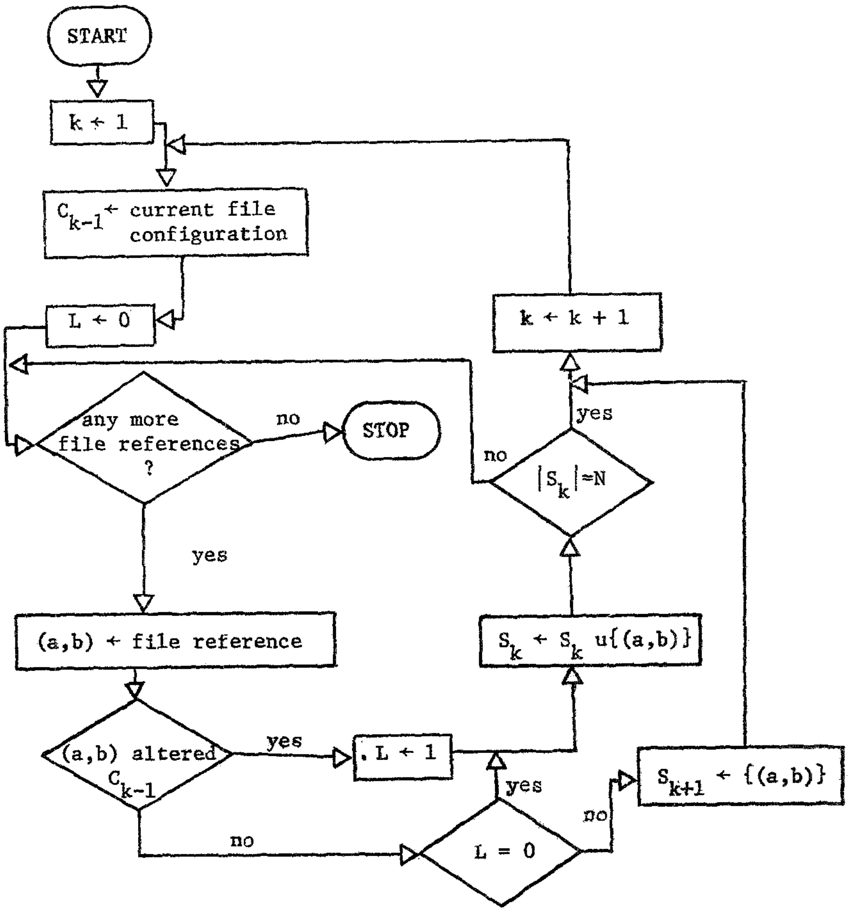

Fig. 1. Flowchart of demand-configuration algorithm.

3. $(a, b) \leftarrow$ next file reference [see (2)]; if no more activity, STOP.
4. If (a, b) altered C_{k-1}, then $L \leftarrow 1$ and go to 6.
5. If $L \neq 0$, then $S_{k+1} \leftarrow \{(a, b)\}$ and go to 8.
6. $S_k \leftarrow S_k \cup \{(a, b)\}$.
7. If $|S_k| < N$, then go to 3.
8. $k \leftarrow k + 1$; go to 2.

Using this procedure, one can construct two sets:

$$M = \{C_0, C_1, C_2, \ldots, C_{k-1}\}, \qquad R = \{S_1, S_2, S_3, \ldots, S_k\} \tag{4}$$

The subset of $M \times R$ consisting of the ordered pairs (C_i, S_j) such that $j = i + 1$ can be used to characterize the evolution of file placement in the multilevel store with respect to the file demands of the job sets in a pairwise sequential manner.

Let us examine the work processed by *CFS* during arbitrary period m. If $(e_m, f_i) \in S_t$, then some portion of job e_i was executed during period t. Let $w_{m,t}$ represent that portion of job e_m processed during period t. Then the total amount of work done by *CFS* during period t is

$$W_t = \{w_{m,t} : \ \forall_i \quad (e_m, f_i) \in S_t \quad \text{for some} \quad i\} \tag{5}$$

We can then redefine the set of work processed by *CFS* to be

$$B = \{W_1, W_2, W_3, \dots, W_k\} \ni k = |M| \tag{6}$$

Assertation. Given *CFS*, $B = E$.

Proof. Let $e_m \in E$; then $e_m = \{w_{m,k_1}, w_{m,k_2}, \dots, w_{m,k_n}\}$ for some finite n. By definition of W_{k_i}, each $w_{m,k_i} \in W_{k_i}$; hence $e_m \in B$. Thus $E \subseteq B$. Let $w_{m,k} \in B \to (e_m, f_i) \in S_k \to w_{m,k} \in e_m \subseteq E$. Hence $B = E$. QED

Now we can look at the subset $M \times B$ consisting of the ordered pairs $(C_i, W_j) \ni j = i + 1$ which characterize the stepwise processing by the system *CFS* of the workload E. Thus set B identifies the manner in which the set of jobs E is processed.

Definition 1. The time function $N(a, b)$ computes how long it takes to process workset b under initial configuration a.

Now we can initially define the *throughput time* of system *CFS* to process workload B under the configuration sequence M:

$$\text{TPT}(B, M, k) = \sum_{i=1}^{k} N(C_{i-1}, W_i) \tag{7}$$

This sets the stage for the basic theoretical model. Given a *CFS*, how does one optimize $M = \{C_0, C_1, C_2, \dots, C_{k-1}\}$ to minimize the run time for each $W_i \in B$ which in turn will minimize throughput time for B?

3. APPROACHING THE PROBLEM

Let us first isolate the jobset B from all else. This set represents the work to be done. Given B, then $\exists M' = \{A_1, A_2, \dots, A_k\}$ of configurations defined as was M [Eq. (4)] that is optimal with respect to processing each $W_i \in B$. In other words,

$$TT_k = \sum_{i=1}^{k} N(A_k, W_i) \quad \text{is minimal} \tag{8}$$

In Eq. (7) the configuration transformations C_0 to C_1 to C_2, etc., were as a result of doing required work in B, so no extra time was required or requested to produce M. At first glance it may appear that TT_k constitues a valid definition of TPT(B, M', k). However, M' is not necessarily the natural set of configurations and TPT(B, M', k) must take this into consideration. Only for the trivial case where $A_i = C_{i-1} \forall_i$, $1 \leqslant i \leqslant k$, is TT_k equal to TPT(B, M', k). Otherwise let t be the least $i \ni C_{t-1} \neq A_t$. Then in order to process W_t against A_t, we must first transform the memory configuration from C_{t-1} to A_t. We thus have an additional time requirement to transform C_{t-1} to A_t.

Definition 2. The time function $U(a, b)$ computes how long it takes to transform files in the multilevel store from file configuration a to file configuration b.

Then $U(C_{t-1}, A_t)$ shall denote the time required to leave the C_{t-1} configuration to go into the A_t configuration. Certainly TPT(B, M', k) must take this time into consideration.

In addition, from periods $t + 1$ through k we shall also have the additional time requirement of $U(A_{i-1}, A_i)$, $t < i \leqslant k$, to achieve the next A_i needed to process W_i. Now a proper definition for TPT under M' can be made:

$$\text{TPT}(B, M', k) = \sum_{i=1}^{k} U(A_{i-1}, A_i) + TT_k \qquad \ni A_0 = C_0 \tag{9}$$

Note that if $M' = M$, then $U(A_{i-1}, A_i) = 0$ for $i = 1, 2, \ldots, k$. Thus Eq. (9) is consistent with TPT as previously defined [Eq. (7)]. Let us retain only Eq. (9) for the definition of TPT since Eq. (7) holds just for the trivial case of M, the natural sequence of configurations.

Now let us examine Eq. (9). By assumption, M' is the optimal sequence to reduce the second term of TPT(B, M', k). What about the first term, the time required between each jobset to readjust the file configuration? Since M' did not take the function U into consideration, we cannot assume any longer that TPT(B, M', k) is minimal.

Again, given B and C_0, then $\exists$ set $M^* = \{Q_1, Q_2, \ldots, Q_k\}$ such that $Q_1 = C_0$ and TPT(B, M^*, k) is minimized.

How does one, in practice or in theory, find this set M^* that minimizes TPT for jobset B? The thesis of dynamic secondary storage file management is that one can, through a heuristic model, closely approximate this sequence of optimal configurations M^*.

We have tried to lay a foundation to fully describe the problem to be solved. Also, it is clear that, for a given solution M, TPT will measure the goodness of that solution by accurately determining the throughput time required to process jobset B under configuration sequence M.

4. SOLUTION TO THE PROBLEM

Unfortunately, the fact that there exists an optimal sequence M^* which, given B, minimizes TPT(B, M^*, k) gives no insight into how one might find this sequence M^*. It is clear that function TPT consists of two components: file move time U and jobset process time N. How can we minimize both of these functions simultaneously for all $Q_i \in M^*$?

Let us first decide upon an optimization strategy. What assumptions are to be made about B, the set of work demands to the system? In pure theory we may assume that we know (or the Oracle can tell us) all of B. Under this assumption then $\exists$ a set M^* of optimal configurations such that TPT(B, M^*, k) is minimal.

Note that k is arbitrarily large, but certainly finite, if we consider the set B to terminate when the input to our system CFS stops or the system halts itself for one reason or another. We might try to characterize the input stream B and use queueing theory to optimize M to produce or derive M^*, using a model such as Shedler's.[2]

Consider now that we wish to design a practical model to simulate the mathematical solution to our problem. Our problem then should have practical assumptions or it shall be unusable. We do not wish to impose any restrictions upon this problem that would remove it from reality. Therefore we shall not assume any knowledge of any input stream whatsoever. The DFM must react to what happens, not what it knows will happen.

This means that we cannot minimize TPT to a global optimum. The only way TPT can be optimized globally is in a nondeterministic manner and the DFM model must be deterministic to be practically implemented. We must do our best, using heuristics, to derive the locally optimal set of configurations as B is processed, workset by workset.

Let us examine the recursive definition of TPT:

$$\mathrm{TPT}(B, M, k) = \mathrm{TPT}(B, M, k-1) + U(C_{k-1}, C_k) + N(C_k, W_k) \quad (10)$$

Between work sets W_{k-1} and W_k in B we must alter configuration C_{k-1} to C_k and then process file work W_k. Note the dilemma here. One cannot assume that minimizing U and N for each W_k will minimize TPT. For example, say the system CFS is at configuration C_k. The Oracle sees that work set W_{k+n} for arbitrary n will be most time consuming, thus $C_t = C_k$, $k \leqslant t \leqslant k + n$, is the optimal sequence of configurations. How can we know that without an n-lookahead potential? Clearly we cannot; thus set $n = 0$ for the DFM model for a look ahead of $n = 1$ or 2 or k does us no good for $n > k$. Given no look ahead, there the DFM's job shall be to predict what configuration should be best, based upon past demands to the system.

5. DFM DECISION TIME

Looking at definition (10) of throughput time, let it be clear that the DFM is charged with choosing and constructing C_k from $C_{k-1} \forall_k$. We must consider this DFM overhead: the time required at each step to choose C_k given C_{k-1}. This should be reflected in TPT if it is to be a true measure of our efforts since the DFM needs CPU time.

Two further definitions are necessary. First let us define the function R, short for the DFM *decision function*:

$$R(C_k) = C_{k+1} \tag{11}$$

Second, this decision must be timed, so define function Z as the timer function as follows.

Definition 3. $Z(k)$ is the time required for the DFM function R to choose C_{k+1}.

Now we can make the final adjustment to TPT so that it properly measures the time required to process jobsets under DFM management:

$$\begin{aligned}\mathrm{TPT}&(M, B, k+1)\\ &= \mathrm{TPT}(M, B, k) + Z(k) + U(C_k, R(C_k)) + N(R(C_k), W_{k+1})\end{aligned} \tag{12}$$

At each stage the DFM must choose the next configuration, produce it, and then process the next jobset under the new configuration. This entire process, over arbitrary jobsets, is to be minimized.

For any system, given CFS, we can compute its basic throughput time $\mathrm{TPT}(M, B, k)$. The objective of the DFM is to minimize $\mathrm{TPT}(M^*, B, k) \forall_k$ by deriving the set M^*, element by element for each time period.

A simulation model (the DFM) is necessary in order to exemplify the heuristics for deriving a close approximation to M^*. Also, it was found that $\mathrm{TPT}(M^*, B, k) \ll \mathrm{TPT}(M, B, k)$, thus showing the feasibility and usefulness of such a model.

6. THE DFM SIMULATOR

The DFM decision function $R(C_k)$ as defined in Eq. (11) has no knowledge of any future jobsets $W_t \forall t > k$. Thus the configuration $C_{k+1} = R(C_k)$ chosen may very well not minimize $N(C_{k+1}, W_{k+1})$. Not only is this detrimental to the minimization of $\mathrm{TPT}(B, M^*, k)$, but the time spent choosing C_{k+1}, $Z(k)$, and the time spent proceeding to C_k, $U(C_k, C_{k+1})$, are all for naught. In other words, DFM decision errors can be very costly with respect to the throughput time function (12).

Let us assume that at time period k the DFM evaluates the entire system; i.e., the decision function R takes configurations $C_0, \ldots, C_{k-1}$ and jobsets $W_1, \ldots, W_k$ into consideration to determine the next configuration C_k. Then at time period $k + 1$ must the DFM decision function R consider the total past history again? Clearly, the further back toward C_0 and W_1 the decision function looks, the better its final decision should be. Also, however, the further back R looks, the longer the time spent making the decision (Z).

The DFM model must exhibit balance between U and N while keeping Z to a minimum if TPT is to be minimized. If $Z(k)$ is large for all k, then $R(C_k)$ will have to be correct to minimize N. Looking at the three components of TPT, Z and U represent DFM work time. The time the DFM is trying to save is the difference between running jobset W_{k+1} under configuration C_k; or choosing a new configuration C_{k+1}, moving memory files from C_k to C_{k+1}, and then running W_{k+1} under C_{k+1}; i.e. all the DFM activity must be justified by shortened run times for W_{k+1} under the new configuration C_{k+1}.

Equation (12) provides a basis for the physical movement of files in a multilevel store. Local policies, as they effect both the set of users and the system itself, must form an integral part of the implementation of the DFM model. The better the DFM strategies, the closer its set of chosen configurations will approach the optimal configuration sequence.

REFERENCES

1. Peter J. Denning, "Third generation systems," *ACM Computing Surveys* 3(4):178 (1971).
2. G. S. Shedler, "A queueing model of a multiprogrammed computer with a two-level storage system," *CACM* **16**(2):3–10 (1973).