A SIMULATION MODEL FOR DYNAMIC FILE MANAGEMENT

Lawrence L. Rose,[1] Malcolm A. Gotterer[2]

and Jack C. Hayya[3]

[1]State University of New York at Binghamton

[2]Florida International University

[3]The Pennsylvania State University

ABSTRACT

This paper describes a FORTRAN simulation program written to carry out Dynamic File Management (DFM) strategies. The DFM model provides effective management of secondary storage while enhancing throughput time. The main problems we address are: how data is moved from level to level; the determination of criteria to justify file movement; and the implications of implementing the DFM techniques. It is demonstrated that DFM strategies are feasible and provide the manager with a viable mechanism with which to govern auxiliary storage.

I. INTRODUCTION

Dynamic File Management (DFM) techniques are memory management strategies which move data in the secondary store in response to user and system demands for data. The objective of these strategies of data movement is to optimize secondary storage usage with respect to throughput time.

A simulation model is constructed to allow the comparison of various strategies of Dynamic File Management. File process time for any simulated input job stream is obtained for the system, with and without DFM management. The output is statistically analyzed to determine the potential for time savings under Dynamic File Management.

II. MODEL BACKGROUND

A memory hierarchy is composed of devices that have ascending "transport speeds" dependent upon wait, access, and transmission times. Memory hierarchies consisting of a main level (usually high speed core memory) and the auxiliary store (drum, disc, tape, etc.) were created because of the excessive cost of internal fast core storage. The higher a set of data (file) is in the auxiliary store, the faster it can be transported to or from the main store. However, the more complex a memory hierarchy, the more difficult it is to use efficiently.

The problem is one of dynamic assignment, since the access demand is stochastic; thus, assignments to hierarchial levels of storage should change to meet new demands. This is our area of research: it is the determination of "optimal" file position in secondary storage in response to changing demand. Storage management techniques must be developed for efficient use of configurations such as hierarchical or multilevel storage. The DFM strategies described herein are designed to do that, the criterion of efficiency being minimum throughput time.

The value of a memory management technique is its ability to improve performance. Thus, we should examine current techniques used to evaluate the performance of computer systems. We are searching, in particular, for a viable method to measure the effectiveness of the Dynamic File Management techniques. One may consider six major evaluation techniques: hardware monitors, software monitors, benchmarks, synthetic programs, analytic models, and simulation models.

It is not feasible (in a University environment) to obtain a computer system for experimenting with memory management strategies. Because of this, DFM techniques are not implemented in an actual system. The only evaluation techniques available are those using analytic and simulation models.

An analytic model is a mathematical representation, usually through queuing theory, of a computer system. Normally, the model does not evaluate an entire system, rather portions of it, such as channel interference and disk throughput.

It may appear that an analytic model such as Shedler's [22] could be developed to examine hierarchial storage, assuming a probabilistic level changer. But one

goal of this research is to examine the suitability of criteria for determining Dynamic File Management strategies in response to changes in demand. A proper modeling of this would require dynamic changes of the probabilities for file movement. Another goal of our research is to determine the reasons why files should be moved; therefore, a level changer governed only by probabilities is not considered. It follows that an analytic model is not the applicable evaluation tool to use in this research. Consequently, we use simulation.

III. SIMULATION

Simulation is acknowledged to be the most powerful of all computer evaluation techniques [2, 6, 14, 16, 19]. Input to the model consists of data characterizing the computer to be simulated and its normal job stream. The computer system is described by a set of parameters dependent upon the simulation model. Some models will have no system descriptors because these are constant, while other models may have a large requirement for system descriptors.

The job stream can be described in various ways [7, 14, 16]. Event-oriented software traces can be made if the projected job stream exists and is expected to remain constant. This is expensive; it contributes to the degradation of the system; and it creates voluminous output. More computation would be required to reduce the software traces to their minimal characterization. Often, however, if a system undergoes drastic change, the use of the system will also change to take advantage of the new features. Thus, extreme care must be taken in projecting job stream data. As an alternative, analytical data can be derived to describe the attributes of the job stream. One can produce distributions such as the normal, exponential, and Weibull, and test the operation of the system under these distributions.

Although many languages are used for simulation, no single language is preferred [11, 18]. A simulator written in ASSEMBLER, FORTRAN, or MAD, for instance, uses input parameters to describe the system. Other languages such as SIMSCRIPT, SOL, CSS, GPSS, SIMTRAN, and MDL are programmed as subroutines to describe the system, while a main program controls the system events.

No matter what language is chosen for writing the simulator, the main problem is determining the level of detail of the model. How close must it resemble the actual machine? There is much thought on this issue [3, 9, 12, 13, 14]; but it is generally agreed that the finer the level of detail, the more costly the simulation model. On the other hand, if the level of detail is coarse, the results may be misleading.

The output of a simulation model describes the performance of the simulated system. Criteria must be selected to determine whether the simulated performance is better or worse than that of the actual system. MacDougall [15] uses theoretical times to simulate secondary storage accesses as we do. These are well documented, acknowledged average device access times [4]. Using these theoretical times, one can compute the throughput time for a collection of jobs, a measure used by others [8, 10, 12].

The linear objective function which we propose is similar to those used by others [7, 10] to measure performance. The advantage of such a measure is its versatility: it can easily be changed to reflect changes in endogenous or exogenous policies. Kimbleton [13] agrees that exogenous criteria such as user satisfaction are pertinent criteria for performance evaluation. He suggests the idea of system tuning, jobset by jobset.

Also, effective simulation need not imply a one-to-one correspondence between the system actions and those of the model. The goals of the model must be held in the proper perspective. Kimbleton [13,p. 589] sums it up best: "The goal of model building is not to construct a faithful reproduction of the system but to determine potential causal relationships between input and output."

We have said that proper criteria are required so that systems can be compared. There is no general agreement in the profession upon one criterion. There are, however, several criteria worth consideration:

a. The number of data units processed per unit time under a program load [1].
b. An objective function representing read access time, read resolution time, write time, size, cost, and reliability [21].
c. A combination of throughput time, turnaround time, and availability of the computer [3].

d. A figure of merit based on throughput time for the basic system [6]. First one computes the throughput time for the basic system, TPT*. Then one uses the following ratio as a figure of merit for any other model i:

$$TPT^*/TPT(i). \qquad (1)$$

IV. THE DFM SIMULATOR

The theory of Dynamic File Management is described in [20]. The definition of throughput time (TPT) for the theoretical model demonstrates the essentials of Dynamic File Management:

$$TPT(E,M,k+1) = TPT(E,M,k) + Z(k) + U(C'_k, C_{k+1}) + N(C_{k+1}, W_{k+1}). \qquad (2)$$

Equation (2) states that the throughput time required to process the first k+1 Worksets is equal to:

a. TPT(E,M,k): the time to process k Worksets; plus
b. Z(k): the time to decide if any files should change levels in the hierarchy; plus
c. $U(C'_k, C_{k+1})$: the time to move from the present file configuration set C'_k to a new file configuration C_{k+1}; plus
d. $N(C_{k+1}, W_{k+1})$: the process time for Worksets W_{k+1} under the new file configuration set C_{k+1}.

If Z and U in (2) are zero, then (2) reflects the "natural" throughput time for a system (characterized by auxiliary storage M) required to process the set of jobs E. The functions Z (decision) and U (move) reflect the overhead of an auxiliary memory manager. To ensure generality of the model, the only restriction on job processing is that there exists one CPU. The concept of a Workset (W_k) is a renaming of the job portions of set E processed during the k-th period.

THE DECISION FUNCTION

Input to the decision function Z consists of current and past data. The output of this function is a list of file moves. The more information a decision function takes into consideration, the better the decision; however, the more complex the function, the more overhead.

The input data for the decision function are classified into three categories: file priority, F(n); job priority, J(n); and system priority, S(n). Each of these measures has a final value ranging from 0 to 1. Adding adjustable weights of α, β, and γ to F, J, and S, respectively, we can define the decision function as:

$$Z(n) = \alpha F(n) + \beta J(n) + \gamma S(n). \qquad (3)$$

The weights α, β, and γ are user-defined exogenous variables. This enables the manager to utilize the DFM to achieve the goals of his installation more efficiently. These exogenous variables separate the DFM model from other memory management strategies. Given the definition for Z(n), the measures F, J, and S must be better defined so they can be implemented into the DFM model. The DFM builds a history of file activities in increments, as each Workset is processed. Most of the measures relate only to the files referenced during the most recent Worksets. The DFM has no need to evaluate those files that have been inactive over a long period. This reduces the number of files needing evaluation; it is justifiable since only those files referenced by the processed Worksets have been active in the system.

File priority consists of three parameters: the time since the last activity of the file, the duration of the last file activity, and the average activity of the file in question. Job priority consists of the run priority of each job, the time allocation for a job, and the user file priorities of the job's file reference set. Finally, we have the system priority which considers three parameters: the response priority for each Workset, the splitability of the files referenced by each Workset, and the size of those files.

The DFM is invoked after each Workset is processed, as defined in (2). Since a Workset comprises a collection of many user and system jobs, the invocation is dynamic as opposed to real-time; hence, the name Dynamic File Management. DFM monitoring, nonetheless, is performed in real-time; it is the background portion of the DFM.

Given an auxiliary store under DFM control, the level of DFM timeliness must be examined. Let "gap" represent the number of Worksets processed between each DFM invocation. The smaller the value of "gap", the more overhead the DFM must make up since it is more active. Also, the smaller "gap" is, the more timely the DFM can be in response to changes in the job environment. The more often the DFM is invoked, the less new information it has; thus, the likelihood for performing proper

file movement diminishes. But, the larger the value of "gap", the lesser the DFM response. This problem is solved by creating two separate levels of timeliness for the DFM: the temporary "gap" (tgap) and the permanent "gap" (pgap). These two manager-defined variables enable the DFM to have short range (tgap) and long range (pgap) response capabilities. File moves are made often (each tgap) to respond temporarily to recent changes in the workload; other file moves are made less often (each pgap) to respond more firmly to long range trends.

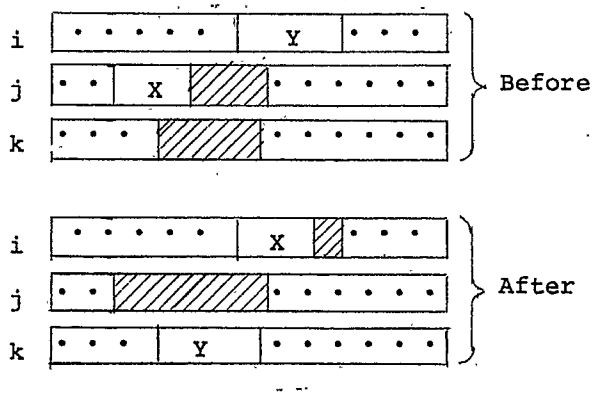
THE MOVE FUNCTIONS

Given two levels of timeliness, one can examine the DFM capabilities to move data in the auxiliary store. When the DFM is invoked after tgap Worksets, it responds with data movement designed to enhance throughput in the short run. This is accomplished by making a series of Temporary moves (T-moves). DFM data movement after pgap Worksets focuses upon long range trends and is accomplished by Permanent moves (P-moves) of data in the store. These two DFM move capabilities constitute the primitives necessary for data movement in secondary storage.

A "permanent" (P-move) change is the movement of a file from a spot B to spot D, with B now available for use by other files. If a file occupies location D prior to the P-move, it is moved to an open spot in the storage hierarchy before the file from location B is moved to its destined location D. Illustration 1 shows the action of a typical P-move.

ILLUSTRATION 1

P-move: $X_j \rightarrow X_i$

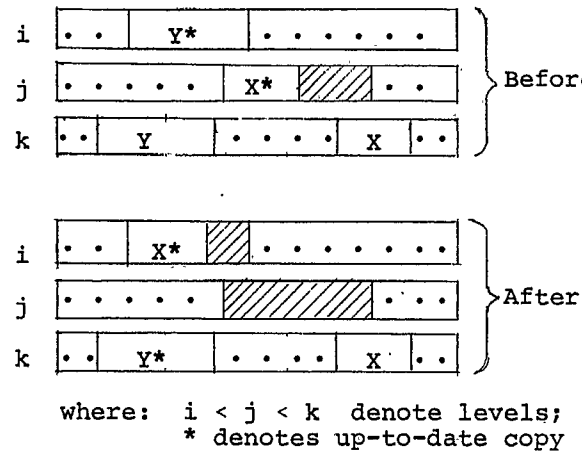


where: i,j,k denote levels and k > i

Consider a "temporary" change (T-move) to be the replacement of location D with file X from spot B. File X now occupies both spots D and B. In addition, if there exists another copy of file X at some location lower than B, it is deleted from location B. If spot D was previously occupied by some file Y, that copy of Y is lost at D. However, another copy of file Y still exists, else Y would have been an illegal destination for the T-move. Illustration 2 shows the action of a typical T-move.

ILLUSTRATION 2

T-move: $X_j \rightarrow X_i$



where: i < j < k denote levels;
* denotes up-to-date copy

These move functions produce at most two file copies: the Home copy, which is the initial copy, and the Away copy, the result of a T-move. By definition, an Away copy is always at a higher level in the storage hierarchy than is the Home copy. Any updating is done only to the Away copy if one exists. If the Away copy is ever overlaid, the file's Home copy is first updated.

These DFM move functions offer some distinct advantages. The first advantage of the two move functions is that there are at most two copies of any file in the auxiliary store. Normal use of a hierarchical store of N levels allows as many as N copies of any file to exist in the store. For large files, the immediate implication is that storage savings become large. The second advantage is that the two file copy types reflect the two levels of timeliness for the DFM. T-moves are quick and timely; P-moves are less timely, but more justified.

The following strategy is employed using these two move capabilities. On slight justification, perform T-moves to create other file copies higher in the multilevel store. If this is a poor decision, little time is lost. Less often do P-moves (followed by device compaction of Home file copies only) to increase long-run system throughput. This allows the DFM to be dynamic; yet, it minimizes overhead.

A file is accessed at its highest position in the store. Changes are made only to the highest file copy which is the most accessible and up-to-date. Updates to the extra file copies are done only when the upper file is to be deleted. This method preserves information and minimizes update time with only a slight addition in bookkeeping overhead.

V. DFM EXPERIMENTS

The objective of the DFM model is auxiliary memory efficiency. The figure of merit for the DFM simulation model is throughput time. The reasons for choosing throughput time are that it is a valid measure of system efficiency [8, 10, 12, 15] and that it is defined by the function TPT, equation (2).

Testing of the DFM model under all possible input streams would be a costly task. Instead, tests are performed using a uniformly distributed input stream.

THE SIMULATOR EXPERIMENT

The input to the DFM simulator describes the computer system auxiliary store to be simulated. The other necessary input is a definition of the weights in the DFM decision function Z for both Temporary and Permanent moves. These weights are described by the vector V:

$$V = (\alpha_T, \beta_T, \gamma_T, \alpha_P, \beta_P, \gamma_P). \quad (4)$$

Changes in weights result in a new DFM strategy. These changes do not alter the manner of file movement; rather, they alter the reasons for it.

The output of the DFM model is file throughput time (FTT): auxiliary storage cumulative time spent during the processing of the input stream accesses. When we add the CPU time to these times, we have throughput time (TPT).

The DFM model is simulated 82 times under uniform input accesses to storage, varying randomly the six major parameters (4). One run is executed without DFM intervention to provide the base file throughput time, FTT_0, for the work periods 1,2,...,48. File throughput time after k work periods for DFM model i is denoted $FTT_i(k)$. This represents the total auxiliary storage time spent during the processing of k Worksets. From file throughput time, we can derive throughput time for these simulation runs.

ILLUSTRATION 3

DFM Output Times

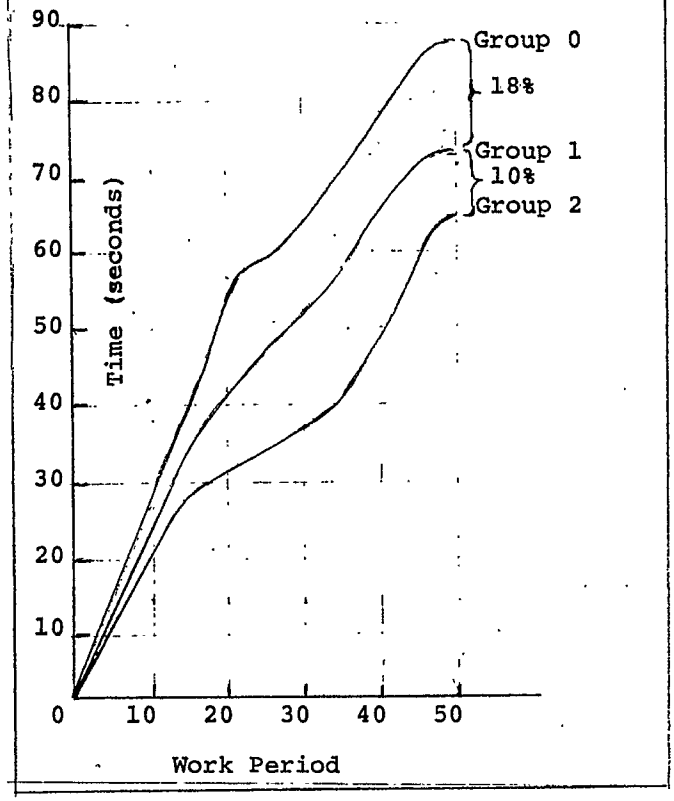


Illustration 3 represents three FTT curves: Groups 0, 1, and 2. The Group 0 curve represents FTT file throughput time for the system with no intervention under uniform random input. This curve represents the upper bound for DFM output.

The curve labeled Group 1, Illustration 3, constitutes the mean of the high DFM output for 48 Worksets. The Group 2 curve represents the mean of the low DFM output. Forty-six of the eighty-one DFM output curves are classified in either Group 1 or Group 2.

Illustration 3 shows that, for certain DFM strategies (defined by the V vector for each run), time savings from 18% - 28% can be made under random selection from a uniform distribution. Expansion from FTT to TPT reveals a 13% - 23% increase in throughput.

File throughput data represents the necessary output from the DFM simulator. Each of the 82 DFM data sets is

associated with a vector of weights. These data sets are then fitted to polynomials [5]. In all cases, the linear coefficient was statistically significant at $\alpha = 0.01$.

STATISTICAL ANALYSIS

Inspection of the eighty-two polynomials indicates three groups. The grouping yields significant differences in time savings for Group 2 (low times) over the scale from work period t=0 and t=k. This is verified by a multivariate test of equality for means of the regression coefficients of Group 2 and Group 0 [17, p. 117-120].

That is, we test the hypothesis,

H_0: $\mu_2 = \mu_0$, where μ_0 represents the Group 0 mean,

versus

H_1: $\mu_2 \neq \mu_0$, where μ_0 represents the Group 2 mean.

The above test involves the use of the covariance matrix [23, p. 418] of the Group 2 vectors. This matrix is found to be near singular. Singularity of the covariance matrix implies that the vectors are very nearly linearly dependent or approximately the same, eliminating the need for further computation. Results derived from polynomial slope comparison indicate that the Group 2 vector statistically dominates the base vector. This implies that for arbitrary work period k and vector t of Group 2: $TPT_0 > TPT_t$.

Another test of hypothesis is made using the data of Group 1 and that of Group 2. To wit,

H_0: $\mu_2 = \mu_2$, where μ_1 = mean Group 1,

versus

H_1: $\mu_1 \neq \mu_2$, where μ_2 = mean Group 2.

The same results are obtained as before. The mean Group 2 vector dominates the mean Group vector statistically.

We have shown that the DFM decision function enhances throughput. We next show that statistically significant differences between groups can be attributed to the weights in (4). We also determine which of these weights contribute to the efficiency of the DFM model.

DECISION FUNCTION WEIGHTS

The six weights in the decision function are exogenous variables which the manager defines. They are in the decision functions so that the manager can define the priorities for file movement during run-time. Now benefits, such as job priority, may be more important to the manager than globally minimizing throughput time. Hence, our purpose is not to find the "optimal" set of weights for the decision function, but to determine the bounds within which the manager can easily work without adversely affecting throughput time.

Stepwise discriminant analysis [23, p. 413] is applied to partition the associated decision weight vectors into inferior and superior groups by time saving differentials of the polynomial regressions of time on the weight vectors. The objective of discriminant analysis in this test is to identify these significant differences in throughput time with the input decision function weights, and subsequently, to see if some restrictions might be removed from those weights not contributing to savings.

The stepwise discriminant analysis yields the function

$$U = 19.87093 + 1.33809\alpha_T - 2.62676\beta_T + 2.228270\alpha_P + 4.38766\gamma_P \qquad (5)$$

significant at the 0.01 level. The resulting classification table shows that the classification procedure was adequate.

The stepwise discriminant analysis indicates that the superior weight vectors can be identified easily. This means that there exists a procedure whereby the vector (4) can be classified as superior or inferior.

The stepwise procedure also indicates that variables α_T and β_P can be eliminated as they do not contribute significantly to the classification power. It is then concluded that system priority for T-moves (short run) and job priority for P-moves (long run) will have little effect upon the resulting throughput time of the model.

SUMMARY AND CONCLUSIONS

In this paper we demonstrate that simulation is a viable and effective technique for dealing with the analysis of complex computer systems. The DFM simulator is tested with random uniform input. It is shown that throughput time can be decreased. The decrease depends upon the values assigned to four "bound" weights in a decision function. It is possible to classify a weight vector for a decision function as superior or inferior, depending upon the effect it has on throughput time. These results demonstrate the soundness of the theory of Dynamic File Management.

BIBLIOGRAPHY

1. Anacker, William and Wang, Chu Ping. Evaluation of Computing Systems with Memory Hierarchies." IEEE Transactions on Electronic Computers, Vol. EC-16, No. 6 (December, 1967), 670-679.
2. Calingaert, Peter. "System Performance and Evaluation: Survey and Appraisal." CACM, Vol. 10, No. 1 (January, 1967), 12-18.
3. Campbell, D. J., and Heffner, W. J. "Measurement and Analysis of Large Operating Systems during System Development." Proceedings AFIPS, 1968, FJCC, Vol. 33, Pt. 1, 903-914.
4. Denning, P. J. "Virtual Memory," ACM Computing Surveys, Vol. 2, No. 3 (September, 1970), 153-189.
5. Dixon, W. J., ed. BMD Biomedical Computer Programs. Berkely: University of California Press, 1968.
6. Drummond, M. E. "A Perspective on System Performance Evaluation." IBM Systems Journal, Vol. 8, No. 4 (1969), 252-263.
7. Gascei, J; Slutz, D. R.; and Traiger, I. L. "Evaluation Techniques for Storage Hierarchies." IBM Systems Journal, Vol. 9, No. 2 (1970), 78-117.
8. Herman, Donald J. "SCERT: A Computer Evaluation Tool." Datamation, Vol. 13, No. 2 (February, 1967), 26-28.
9. Huesmann, Lowell R., and Goldberg, Robert P. "Evaluating Computer Systems Through Simulation." Computer Journal, Vol. 10 (August, 1967), 150-156.
10. Hutchinson, George K. "A Computer Center Simulation Project." CACM, Vol. 8, No. 9 (September, 1965), 559-568.
11. _____, and Maguire, John Morris. "Computer Systems Design and Analysis Through Simulation." AFIPS Conference Proceedings 1965, FJCC, Vol. 27, Pt. 1.
12. Katz, Jesse H. "Simulation of a Multiprocessor Computer System." AFIPS Conference Proceedings 1966, SJCC, Vol. 28, 127-139.
13. Kimbleton, Stephen R. "The Role of Computer System Models in Performance Evaluation." CACM, Vol. 15, No. 7 (July, 1972), 586-590.
14. Lucas, Henry C., Jr. "Performance Evaluation and Monitoring." ACM Computing Surveys, Vol. 3, No. 3 (September, 1971), 79-92.
15. MacDougall, M. H. "Computer System Simulation: An Introduction." ACM Computing Surveys, Vol. 2, No. 3 (September, 1970), 191-209.
16. Merikallis, Reino A., and Holland, F. C. "Simulation Design of a Multiprocessing System." AFIPS Conference Proceedings 1968, FJCC, Vol. 33, Pt. 2, 1399-1410.
17. Morrison, Donald F. Multivariate Statistical Methods. New York: McGraw-Hill, 1967.
18. Parupudi, Murty, and Winograd, Joseph. "Interactive Task Behavior in a Time-Sharing Environment." Proceedings of the ACM, August 1972, 680-692.
19. Peters, Alan. "LCS at United Airlines." Modern Data, Vol. 5, No. 2 (February, 1972), 34-36.
20. Rose, Lawrence, and Gotterer, Malcolm. "A Theory of Dynamic File Management." International Journal of Computer and Information Sciences, scheduled for publication in December, 1973.
21. Scarrott, C. G. "The Efficient Use of Multilevel Storage." Proceedings of the IFIP Congress, 1965, Vol. 1, 137-141.
22. Shedler, G. S. "A Queueing Model of a Multiprogrammed Computer with a Two-Level Storage System." CACM, Vol. 16, No. 2 (January, 1973), 3-10.
23. Snedecor, George W., and Cochran, William G. Statistical Methods. Ames, Iowa: Iowa State University Press, 1967.