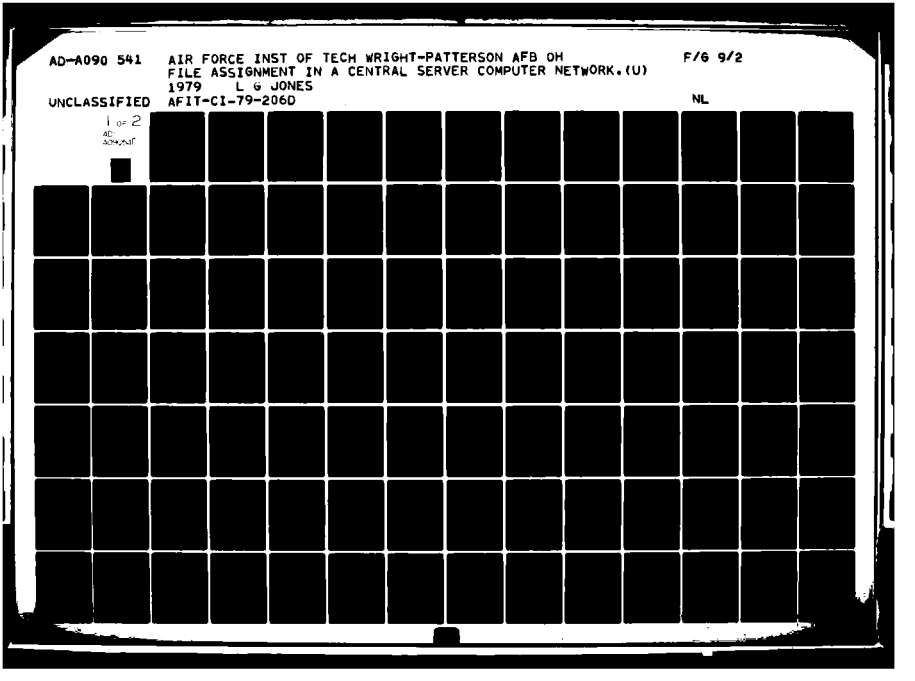
AD-A090 541 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH F/G 9/2
FILE ASSIGNMENT IN A CENTRAL SERVER COMPUTER NETWORK.(U)
1979 L G JONES
UNCLASSIFIED AFIT-CI-79-206D NL



LEVEL II

(1)

FILE ASSIGNMENT IN A CENTRAL SERVER COMPUTER NETWORK

LAWRENCE GENE JONES

~~Captain, USAF~~

Doctor of Philosphy

in

Computer Science

1979

Vanderbilt University

Nashville, Tennessee

162 Pages

UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 79-20 D ✓	2. GOVT ACCESSION NO. AD-A096 541	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* File Assignment in a Central Server Computer Network		5. TYPE OF REPORT & PERIOD COVERED ~~THESIS~~/DISSERTATION 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Capt Lawrence Gene Jones		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: Vanderbilt University		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS AFIT/NR WPAFB OH 45433		12. REPORT DATE 1979 13. NUMBER OF PAGES 162
14. MONITORING AGENCY NAME & ADDRESS*(If different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASS 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE: IAW AFR 190-17

25 SEP 1980

Fredric C. Lynch

FREDRIC C. LYNCH, Major, USAF
Director of Public Affairs
Air Force Institute of Technology (ATC)
Wright-Patterson AFB, OH 45433

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

ATTACHED

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

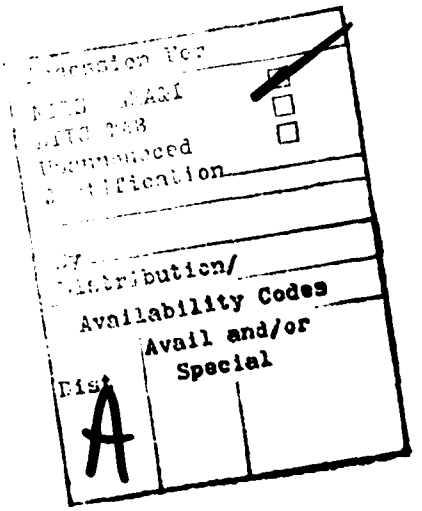
The placement of files among the nodes of a computer network can have a significant impact on the performance of the network. The problem of determining the optimal file placement is known as the file assignment problem. This work extends previous performance oriented file assignment research in two major areas. We first present a method to obtain file assignments for read-only files that allows replication of file copies. We then extend this method to read-write files with both non-replication and replication of file copies.

The network topology considered is the central server or star network topology. The star network is analyzed using techniques that model the system as a network of queues. The measure of file assignment optimality used is maximum central node throughput.

For the read-only model, we introduce the concept of replicated files with split access as a method to improve central node utilization. The problem is formulated as a multiple objective integer linear programming problem. We solve the problem with a polynomially time bounded heuristic. The solution is within proven bounds on the total storage required for an optimal solution.

The ideas used in the read-only model are extended to allow read-write files. We show that the read-write problem without file replication reduces to the read-only problem. We then analyze the read-only solution technique as a heuristic method for solving the read-write problem. Bounds are established for the error introduced by this

approximation. Case studies are presented and analyzed. They provide evidence that the method gives reasonably good answers.



REFERENCES

[Abra73] Abramson, N. and Kuo, F. F. Computer Communication Networks, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

[Aho74] Aho, A. V., Hopcroft, J. E., and Ullman, J. D. The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974.

[Ames77a] Ames, J. E. Dynamic file assignment in a distributed database system. Ph. D. dissertation, Dept. of Computer Science, Duke University, Durham, NC, 1977.

[Ames77b] Ames, J. E., and Foster, D. V. Dynamic file assignment in a star network. Proceeding Computer Networks Symposium, Gaithersburg, MD, 1977.

[Ande75] Anderson, G. A., and Jensen, E. D. Computer interconnection structures: taxonomy, characteristics, and examples. Computing Surveys 7,4 (Dec 1975), 197-213.

[Aror73] Arora, S. R. and Gallo, A. Optimization of static loading and sizing of multilevel memory systems,JACM 20, 2 (April 1973), 307-319.

[Aspl76] Asplund, C. L. ASQ User's Manual, Department of Computer Science Tech. Report 78-4, Vanderbilt University, Nashville, TN, 1976.

[Bard78] Bard, Y. The VM/370 performance monitor. Comuting Surveys 10, 3 (Sept 1978), 333-342.

[Bond76] Bondy, J. A., and Murty, U. S. R. Graph Theory with Applications. American Elsevier, New York, NY, 1976.

[Brin73] Brinch-Hansen, P. Operating System Principles. Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

[Buze71a] Buzen, J. P. Analysis of system bottlenecks using a queueing network model, Proceedings of ACM Sigops Workshop on System Performance Evaluation, Cambridge, Mass, April 1971.

[Buze71b] Buzen, J. P. Optimizing the degree of multiprogramming in demand paging systems, Proceedings IEEE Computer Science Conference, Sept 1971, 139-140.

[Buze71c] Buzen, J. P. Queuing network models of multiprogramming, Ph. D. Th., Div. of Engineering and Applied Physics, Harvard University, Cambridge, MA, May 1971.

[Buze73] Buzen, J. P. Computational algorithms for closed queueing networks with exponential servers, CACM 16, 9 (Sept 1973), 527-531.

[Buze74a] Buzen, J. P. and Chen, P. P. -S. Optimal load balancing in memory hierarchies, Information Processing 74.

[Buze74b] Buzen, J. P. and Goldberg, P. S. Guidelines for the use of infinite source queueing models in the analysis of computer system performance, AFIPS Vol 43 (1974), 371-374.

[Buze75] Buzen, J. P. Cost effective analytic tools for computer performance evaluation, Proc. IEEE COMPCON 1975, IEEE, New York, New York, 293-296.

[Buze78] Buzen, J. P. A queueing network model of MVS, Computing Surveys 10, 3 (Sept 1978), 319-331.

[Case72] Casey, R. G. Allocation of copies of a file in an information network, Proc AFIPS 1972 SJCC 40, 617-625.

[Case73] Casey, R. G. Design of tree networks for distributed data, Proc AFIPS 1973 NCC 42, 341-348.

[Chan76] Chandy, K. M. and Hewes, J. E. File allocation in distributed systems, Proc. Internat. Symp. on Computer Perf. Modeling, Measurement, and Eval., Cambridge, Mass, March, 1976,10-13.

[Chan78] Chandy, K. M., and Sauer, C. H. Approximate methods for analyzing queueing network models of computer systems, Computing Surveys 10, 3 (Sept 1978), 281-317.

[Chen75] Chen, P. P. -S. Queueing network model of interactive computing systems, IEEE Proceedings 63, 6 (June 1975),954-947.

[Chu69] Chu, W. W. Optimal file allocation in a multiple computer system, IEEE Transactions on Computers C-18,10 (Oct 1969),885-889.

[Chu73] Chu, W. W. Optimal file allocation in a computer network, In [Abra73], 82-94.

[Denn78] Denning, P. J., and Buzen, J. P. The opertional analysis of queueing network models, Computing Surveys 10, 3 (Sept 1978), 225-261.

[Depp76] Deppe, M. E., and Fry, J. P. Distributed data bases; a summary of research. Computer Networks 1,2 (Sept 1976).

[Dowd77] Dowdy, L. W. Optimal branching probabilities and their relationship to computer network file distribution, Ph. D. Th., Department of Computer Science, Duke University, Durham, NC, 1977.

[Dowd78] Dowdy, L.W. and Foster, D.V. The file assignment problem in perspective, Department of Computer Science Tech. Report 78-3, Vanderbilt University, Nashville, TN, 1978.

[Eswa74] Eswaran, K. P. Placement of Records in a File and File Allocation in a Computer Network, IFIP Conf. Proc.,Stockholm, Sweden (Aug 1974),304-307.

[Ferr78] Ferrari, D. Computer Systems Performance, Prentice-Hall, Englewood Cliffs, NJ, 1978.

[Fish78] Fisher, M. L. and Hochbaum, D. S. Data base location in computer networks. TR 78-10-09, Wharton School, University of Pennsylvania, Philadelphia, PA, October 1978.

[Fost74] Foster, D. V. File assignment in memory hierarchies, Ph. D. Thesis, University of Texas, Austin, Texas, August 1974.

[Fost76] Foster, D. V. and Browne, J. C. File assignment in memory hierarchies, Proc. Modeling and Perf. Eval. of Computer Systems, North-Holland Publishing Co., Amsterdam, (Oct 1976), 119-127. Also, Computer Science Tech. Report CS-1976-11, Duke University, Durham, NC, 1976.

[Fost77] Foster, D. V., Dowdy, L. W., and Ames, J. E. File assignment in a star network, Forthcoming in CACM. Systems and Information Science Tech. Rep. 77-3, Vanderbilt University, Nashville, TN, 1977.

[Gall76] Gallie, T. M., and Ramm, D. Computer Science/I: an introduction to structured programming, Kendall/Hunt, Dubuque, IA, 1976.

[Giam76] Giammo, T. Validation of a computer performance model of the exponential queueing network family, Acta Informatica 7, 2 (1976), 123-136.

[Gord67] Gordon, W. J. and Newell, G. P. Closed queuing systems with exponential servers. Operations Research 15,2 (April 1967), 254-265.

[Grah78] Graham, G. S., ed. Queueing network models of computer system performance, special issue, Computing Surveys 10,3 (Sept 78).

[Grap77] Grapa, E. and Belford, G. G. Some theorems to aid in solving the file assignment problem. CACM 20,11 (Nov 1977), 878-882.

[Hugh73] Hughes, P. H. and Moe, G. A structural approach to computer performance. Proc. AFIPS 1973 SJCC, 109-120.

[Horo76] Horowits, E., and Sahni, S. Fundamentals of Data Structures. Computer Science Press, Woodland Hills, CA, 1976.

[Igni76] Ignizio, J. P. Goal Programming and Extensions. Lexington Books, D. C. Heath and Co., 1976.

[Jack57] Jackson, J. R. Networks of waiting lines. Operations Research 5 (1957), 518-521.

[Jone78] Jones, L. G., Foster, D. V., and Krolak, P. D. A goal programming formulation to the file assignment problem. Department of Computer Science Technical Report 78-5, Vanderbilt University, Nashville, TN, 1978.

[Jone79a] Jones, L. G., and Foster, D. V. A heuristic solution to the file assignment problem. Proceedings IEEE SOUTHEASTCON 79, Roanoke, VA, April, 1979.

[Jone79b] Jones, L. G., and Foster, D. V. Toward a multiple copy file assignment model for files in a computer system. Proceedings Southeast Regional ACM Conf., Orlando, FL, April, 1979.

[Kane74] Kaneko, T. Optimal task switching policy for a multilevel storage system, IBM J. Res. Dev, July 1974, 310-315.

[Kane75] Kaneko, T. Optimization of a storage hierarchy under multiprogramming environment, IBM Research report RC 5222, Jan 1975.

[Klei75] Kleinrock, L. Queueing Systems, Volume 1: Theory. John Wiley and Sons, New York, New York, 1975.

[Klei76] Kleinrock, L. Queueing Systems, Volume 2: Computer Applications. John Wiley and Sons, New York, New York, 1976.

[Lee78] Lee, E. Y. S. Distributed data base design methodology. TRW Defense and Space Systems Group Tech. Report 30451-78-025, Redondo Beach, CA, 1978.

[Levi75] Levin, K. D. and Morgan, H. L. Optimizing distributed data bases- a framework for research. Proc AFIPS 1975 NCC 44, 473-478.

[Lips77] Lipsky, L. and Church, J. D. Applications of a queuing network model for a computer system. Computing Surveys 9, 3 (September 1977), 205-221.

[Mahm76] Mahmoud, S. and Riordon, J. S. Optimal allocation of resources in distributed information networks. ACM Transactions on Database Systems 1, 1 (Mar 1976), 66-78.

[Mill65] Miller, I., and Freund, J. E. Probability and Statistics for Engineers, Prentice-Hall, Englewood Cliffs, NJ, 1965.

[Moor72] Moore, F. R. Computation model of a closed queueing network with exponential servers. IBM J. Res. Dev. 16,6 (Nov 1972), 567-572.

[Morg77] Morgan, H. L. and Levin, K. D. Optimal program and data locations in computer networks. CACM 20,5 (May 1977), 315-322.

[Munt78] Muntz, R. R. Queueing networks: A critique of the state of the art and directions for the future. Computing Surveys 10, 3 (Sept 1978), 353-359.

[Peeb78] Peebles, R., and Manning, E. System architecture for distributed data management, Computer 11, 1 (Jan 1978), 40-47.

[Pric74] Price, T. G. Probability models of multiprogrammed computer systems. Ph. D. Th., Dept. of Electrical Engineering, Stanford University, Palo Alto, CA, Dec. 1974.

[Rama70] Ramamoorthy, C. V. and Chandy, K. M. Optimization of memory hierarchies in multiprogrammed systems. JACM 17,3 (July 1970), 426-445.

[Rose73] Rose, L. L., and Gotterer, M. H. A theory of dynamic file management in a multilevel store,International Journal of Computer and Information Sciences,Dec 1973.

[Rose75a] Rose, L. L., and Gotterer, M. H. File evaluation in auxiliary storage, International Journal of Computer and Informations Sciences, Sep 1975.

[Rose75b] Rose, L. L., and Gotterer, M. H. An analysis of file movement under dynamic file management strategies, BIT 15, 3 (1975),304-313.

[Rose78] Rose, C. A. A measurement procedure for queueing network models of computer systems. Computing Surveys 10, 3 (Sept 1978), 263-280.

[Schw78] Schwetman, H. D. Hybrid simulation models of computer systems. CACM 21,9 (Sept 1977), 718-723.

[Svob76] Svobodova, L. Computer Performance Measurement and Evaluation Methods: Analysis and Applications, Elsevier North-Holland, New York, New York, 1976.

[Taha71] Taha, H. A. Operations Research - an Introduction, The Macmillan Co., New York, New York, 1971.

[Taha76] Taha, H. A. Integer Programming - Theory, Application, and Computations, Academic Press, New York, New York, 1976.

[Wagn75] Wagner, H. M. Principles of Operations Research. Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

[Wong78] Wong, J. W. Queueing network modeling of computer communication networks. Computing Surveys 10, 3 (Sept 1978), 343-351.

COMPUTER SCIENCE

FILE ASSIGNMENT IN A CENTRAL SERVER COMPUTER NETWORK

LAWRENCE GENE JONES

Dissertation under the direction of Professor Derrell V. Foster

The placement of files among the nodes of a computer network can have a significant impact on the performance of the network. The problem of determining the optimal file placement is known as the file assignment problem. In this work, we present formulations of the problem and present methods for solving it.

This work extends previous performance oriented file assignment research in two major areas. We first present a method to obtain file assignments for read-only files that allows replication of file copies. We then extend this method to read-write files with both non-replication and replication of file copies.

The network topology considered is the central server or star network topology. This topology also applies to a single computer system with the central processor unit as the central node and the peripheral input/output storage devices as the outlying nodes. The star network is analyzed using techniques that model the system as a network of queues. The measure of file assignment optimality used is maximum central node throughput.

For the read-only model, we introduce the concept of replicated files with split access as a method to improve central node utilization. The problem is formulated as a mixed integer linear goal programming problem. A primary objective is to assign files to match the optimal branching probabilities as determined by queuing network analysis. A secondary objective is to use as little storage as possible to obtain the optimal performance. We prove bounds on the total storage required for an optimal solution. Since an exact solution would be computationally costly, a heuristic solution technique is given to solve the problem in a tractable amount of time. An algorithm is given for computing an upper bound on the performance of the heuristic, and case studies demonstrate the accuracy of the method to be reasonably good.

The ideas used in the read-only model are extended to allow read-write files. Two major concepts arise from this extension. First, we represent update overhead as a decrease in the number of jobs (customers) in the network. Then, we show that file access patterns are not constant, as in the read-only case, but are dependent upon the file assignment. We show that the read-write problem without file replication reduces to the read-only problem. Several possible formulations are given for the read-write problem allowing file replication. We then analyze the read-only solution technique as a heuristic method for solving the

read-write problem. Bounds are established for the error introduced by this approximation for both the worst cases and for less severe cases. Case studies are presented and analyzed. They provide evidence that the method gives reasonably good answers.

Approved _____ Date 4/16/79

Advisor

FILE ASSIGNMENT IN A CENTRAL SERVER COMPUTER NETWORK

By

Lawrence Gene Jones

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

May, 1979

Nashville, Tennessee

Approved:	Date:
[illegible]	[illegible]
Patrick D. Krolak	4/16/79
Judson F. Parker	4/16/79
Robert J. Hemminger	4/16/79
[illegible]	[illegible]

ACKNOWLEDGEMENTS

I would like to thank all the members of my committee for their time in reviewing this work and for their helpful suggestions. In addition to this general thanks, I want to add some special thanks. Derrell V. Foster, my advisor, conceived the original research topic and provided me with the background and special tools to pursue the idea. I appreciate his time, effort, and counsel. Robert L. Hemminger displayed enormous patience with me and spent many hours to insure the readability and correctness of the formal proofs in this work. I am grateful for his help. Patrick D. Krolak originally suggested that goal programming was an appropriate method for formulating these problems. This suggestion was very valuable and helped me to focus my thinking.

Several of my contemporaries helped in ways that should qualify them as committee members. Robert P. Cook was very helpful in the early stages of research. He proofread my preliminary results and offered much useful criticism. Joseph L. Linn was a constant and valuable source of help. His inputs are reflected many places and they helped improve the overall quality of the work. Cathy J. Linn provided valuable input in several areas. Early in the research, she provided a key proof that gave confidence in an important

aspect of the research. Joe and Cathy, together, were very generous with their time, expertise, and opinions. They materially affected the outcome of the research with their help, and I am indebted to them. I would also like to acknowledge the help of Glenn A. Mitchell. Conversations with him also helped me to obtain insights during critical phases of the research.

It is very appropriate for me to thank John P. Wittry. He had the initial faith to nominate me for this Ph. D. program, and thus made everything else possible.

Finally, I acknowledge the support of my wife, Chris, and my children, Steve and Vicki. Their patience, love, and understanding through this period allowed me to grow emotionally as well as intellectually.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1. Introductory Remarks

Because computers are generally an expensive resource, the design of efficient computer systems is a traditional area of research. Within this framework, researchers have long known that systems can be greatly improved without adding more and more expensive hardware, but by making better use of the existing facilities. This approach is popular because many solutions obtained this way are cheap and easy to implement compared to the purchase of new equipment. The result is that the user gets more performance for his investment.

In this work, we take such an approach and analyze an important problem related to system performance, the file assignment problem. The problem has great applicability both in the tuning of existing systems and the design of future systems, including networks of computers. This work will provide the reader with a better insight into this important problem.

1.2. The File Assignment Problem (FAP)

A universal characteristic of computer systems is that information flows among the components. In most systems a

significant portion of this traffic is due to file access, and in many applications this traffic may be of the utmost importance. An I/O bound system, possibly a large data base query system or a point of sale inventory system, fits this description very well. However, most general data processing installations have a moderate to heavy amount of this activity, and there is much evidence to indicate that inquiry into the effects of this traffic is valid [Klei76, Grah78].

The location of files among system storage devices (i.e., what files go on which devices) obviously affects the traffic flow, and it has been shown that file placement can be a critical factor in system performance [Hugh73, Fost74]. We can see this intuitively. If we attempt to place all files on the fastest system devices in an effort speed up access, performance is likely to degrade due to queuing delays. Likewise, underloading the fast devices is obviously suboptimal because the faster hardware is not being used to capacity. Obviously, there is value in determining the "best" placement. The file assignment problem (FAP) is the problem of where to place files in a computer system in order to optimize some optimality measure.

One frequently used optimality measure is minimum cost. Models that take this approach generally account for query, update, and storage costs. These models tend to ignore queuing delays or introduce them indirectly as constraints.

The other common optimality measure is optimal performance. Performance oriented models tend to rely on queuing theory for solutions. Examples of optimal performance measures would be maximum throughput or minimum response time. Both cost optimization and performance optimization produce answers that are useful depending on the application.

1.3. The Branching Probabilities Problem (BPP)

The branching probabilities problem (BPP) is a different problem but closely related to the file assignment problem. To approach this problem we model a computer system as a network of queues. This approach has served very well in the past as a method for modeling systems [Grah78]. Additionally, it can provide a theoretical bound on the performance we can obtain.

Consider the queuing network of Figure 1.1. Here jobs (customers in standard queuing terminology) travel among the service facilities (nodes) and may have to wait in a queue before receiving service. This particular model is called a central server model since all traffic is funneled through a central node, n(0) (the central server). This network is also said to be a closed network because the total number of customers in the system is finite and constant. The branching probabilities, P(j), j=1,...,n, represent the probability that a job from n(0) will request service at

node n(j). The probability that a job from n(j) j=1,...n will request service at n(0) is 1 for the central server model.

We can see intuitively that the values of the branching probabilities may significantly affect performance measures. For example, if p(j) > p(i) and the service rate of n(i) >> the service rate of n(j), we would expect that increasing p(i) at the expense of p(j) might result in increased throughput and reduced expected waiting time.

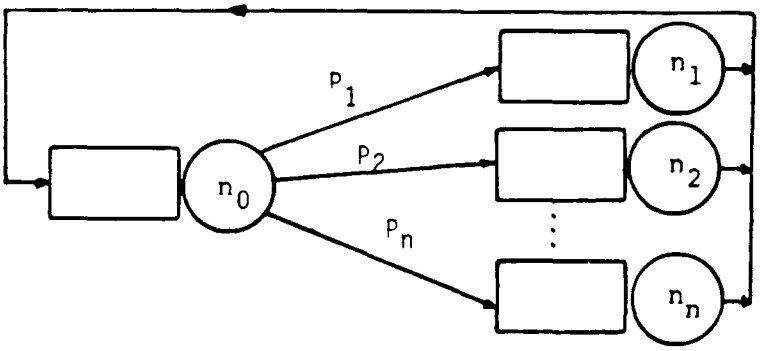


Fig. 1.1. Central server queuing network.

We now have the foundation to define the problem known as the branching probabilites problem (BPP). This problem is to find a set of optimal branching probabilities, P*(j)'s, that yields the optimum performance. Typical performance measures are maximum throughput at the central node, or minimum expected response time for a job.

1.4. FAP-BPP Relationship

We may demonstrate a close relationship between the file assignment problem and the branching probabilities problem. Consider the queuing network model of a computer system in Figure 1.2. This represents a computer system with two input/output (I/O) devices and a central processing unit (CPU). The CPU in Figure 1.2 corresponds to n(0) in Figure 1.1 since all I/O requests are processed by the CPU. Note that a particular assignment of files to devices affects the traffic flow, and hence the P(j)'s, from the CPU to the I/O devices. A file assignment that routes traffic in an attempt to match the optimal branching probabilities may be viewed as a realization of the BPP solution in an I/O bound system. A simple example will serve to clarify the concept.

Consider the simple system of Figure 1.2 together with the data in Figure 1.3. Assume we have an oracle that solves the BPP as: P*(1) = .7, P*(2) = .3 for maximum CPU throughput (i. e., send 70% of the traffic from the CPU to device 1, and 30% to device 2. How the p*'s are actually determined will be discussed in Chapter II. They are a nonlinear function of the service rates and other parameters.) In Figure 1.3, the access frequencies for the three files may be considered as normalized empirical values obtained by monitoring file accesses during a period of time.

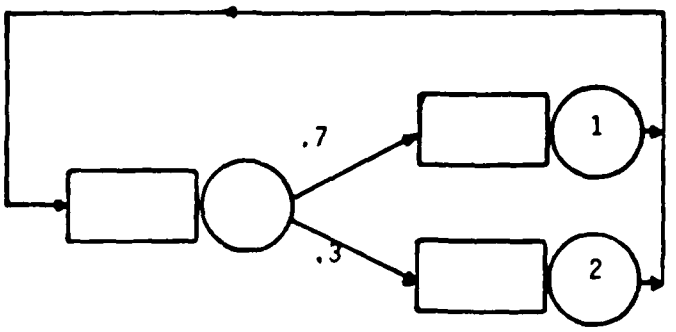


Fig. 1.2. Queuing network model of a computer system.

BPP solution :
P*(1) = .7, P*(2) = .3.

FILE	ACCESS FREQUENCY
A	.3
B	.5
C	.2

An optimal assignment :

DEVICE 1	DEVICE 2
Files B,C	File A

Fig. 1.3. An optimal file assignment.

In this example the accesses to file A equal 30% of all file accesses, file B is accessed 50% of the time, and file C is accessed 20% of the time. Note that the optimal file assignment routes 70% of the file access to device 1 by placing files B and C on that device, and routes 30% of the accesses to device 2 by storing file A on that device. In

this assignment the total file access frequencies assigned to each device matches the optimal branching probabilities exactly, therefore CPU utilization is maximized.

1.5. Justification for research

In this section we will provide evidence that this problem is worthy of a signficant research effort. We will do this by demonstrating that the problem is difficult to solve and that its solution produces useful benefits.

1.5.1. Difficulty of the problem

The file assignment problem is a computationally difficult problem. Eswaren has shown a simple formulation to be NP-complete [Eswa74]. The NP-complete problems are a set of problems for which there are no known algorithms that execute in less than exponential time with respect to the length of the input. This result strongly implies that it may not be feasible to seek exact FAP solutions for realistically sized problems, and that good heuristics should be developed. Researchers seeking exact results have typically had to solve integer programming problems or use other computationally costly enumerative methods.

Dowdy also empirically demonstrated that good heuristic solutions may be difficult to obtain [Dowd77]. Dowdy had eight computer scientists "eyeball" FAP solutions for a completely connected, three node network with ten files to assign. Performance measures were computed for all possible

assignments. The results showed that the average of all possible solutions was slightly better that the average solution produced by the computer scientists.

1.5.2. Benefits from problem solution

While the problem is difficult to solve, the benefits from its solution may be substantial. Central processor throughput improvements of up to 24% have been realized in actual practice merely by relocation of files [Hugh73, Fost74]. Hughes and Moe, experimenting with a UNIVAC 1108, realized a 24% improvement in CPU utilization by reallocation of files. Foster, working with a CDC 6600, realized a 20% improvement. These are significant gains indeed when we consider that they were obtained without any additional hardware or complications to the operating system.

1.6. The Problem in Perspective

In this section we will show where the file assignment problem fits with respect to other computing issues. The file assignment problem is part of the overall problem of how to design "better" computing systems. The file assignment problem impacts both on the improvement of existing systems and also on the design process. We will look at how the FAP fits into the analysis of both stand alone systems and networks of computers.

1.6.1 Single System Applications

File assignment is generally recognized as an important factor in system tuning [Ferr78, Svob76]. This puts file placement in the same category with specification of operating system parameters (e.g. quantum size, degree of multiprogramming), selection of resource management algorithms, and even pricing policies. From this we see that FAP solutions may impact on both improvement studies and selection studies.

FAP solution (and variations on the problem) may also impact on the design process. For example, file assignment studies may prove useful in selecting speeds and capacities for a system storage hierarchy. It is also a factor that may help a designer choose among a set of optimally tuned alternatives.

1.6.2 Network Applications

When we expand our horizons to networks of computers, FAP solutions are still important as a system tuning tool. As we have demonstrated, the stand alone system may be represented as by a star network model. As we move to more complicated structures, FAP solutions become more difficult to obtain, but we can easily envision how the solutions can impact on network performance.

However, with regard to the design process, the FAP takes on added importance in a network. Lee has determined

that file placement is one of the three major dimensions for the design of a distributed data base system [Lee78]. His study lists file placement, real time data control, and data base structure as the primary areas to consider in a design methodology. Peebles and Manning also list file placement among their critical design issues [Peeb78]. Given the current trend toward networking, it appears that the FAP will take on additional importance.

1.7. Approach

In this section we will consider the scope of the problem to be considered, note the major analytic tools to be employed, and state our major assumptions.

1.7.1 Scope

Here we define the bounds on the problem to be investigated. We will establish the network topology to be considered, define our measure of optimality, and state the generality of our model with respect to the types of file accesses modeled.

This work will be confined to analyzing a star (central server) network topology. There are three important justifications for this restriction. First, as demonstrated, the star topology can represent a stand alone, single CPU computer system. This obviously has great applicability. Second, the results obviously also apply to a star configured network of computers, and this is a

commonly used interconnection structure [Ande75]. Finally, since there are open questions for the relatively simple star topology, it is appropriate to more fully explore this structure before moving to the more complex structures.

With regard to the choice of an optimality measure, we noted in Section 1.2 that some models optimize costs while some models optimize performance. In this work we choose to optimize performance. Specifically, we use maximum CPU (or central node) throughput as our optimality criterion.

We choose to optimize performance for three primary reasons. First, performance oriented models have great applicabiliy. They produce answers that are directly useful for system tuning studies. Second, for most applications, peformance models are more realistic. They explicitly account for the queuing delays that are significant in multi-user environments. Finally, while we choose performance as our main thrust, this does not mean that cost considerations must be thrown out the window. The converse is not necessarily true. One way cost factors may be included is as constraints in the assignment formulation. Another obvious alternative is that a designer may have several performance optimized alternatives to choose among, and he can make a cost/performance study to assist him.

Given that performance is to be optimized, we must then determine which performance measure to use. Throughput has been chosen, first of all, because it is a very standard

performance metric. For this particular problem it also is a very appropriate choice. By maximizing throughput we are also maximizing the utilization of the central processor. Note that this increased utilization is not in the form of overhead but derives from useful work performed more efficiently. It is useful to know the maximum throughput we can obtain as a function of file placement since the files must be placed somewhere.

Finally, our study will allow full generality of file access. We will deal with files that can be queried (read files), updated (write files), or both. Some studies have dealt only with read-only files. This is due to the additional complexity introduced when updates are allowed. For completeness of presentation, we begin by considering read-only files, but we ultimately generalize to account for read-write files.

1.7.2. Analysis Tools

Most of the analytic tools come from operations research and mathematics. Queuing theory and nonlinear programming are used in the solution of the BPP. File assignments are performed using mixed integer programming and/or linear programming. Many problem formulations are represented as goal oriented programming problems (see Chapter II). Finally, inductive proofs are used to provide useful lemmas and theorems.

1.7.3. Assumptions

In this section we identify our major assumptions and present justification for these assumptions. The major assumptions deal with the work load on the systems to be modeled, and the assumptions that are necessary to allow a reasonable attempt at analyzing the systems.

First of all, we assume that the systems to be analyzed are not compute (CPU) bound. Obviously this model is most appropriate for systems that have heavy I/O traffic. However, the studies by Hughes and Moe [Hugh73] and Foster [Fost74] were based on systems with a wide mix of jobs. Their studies demonstrate that models such as these can provide useful results for general purpose systems.

We will model our system as a closed, central server queuing network with the usual attendant assumptions to allow closed form solutions. These assumptions will be more carefully detailed in Chapter II. Briefly, they include the assumptions of exponential service times, first-come-first-served queuing discipline, a fixed number of customers, and a steady state condition of the system. These standard assumptions have proven to work well in practice (e.g., see [Klei76] or [Lips77]) and in any case allow computation of good answers that are otherwise unattainable.

Finally, we assume that the file access patterns are essentially static. Thus, if a significant environment

change occurs, it may be appropriate to determine file access patterns under the different conditions. Separate FAP solutions could then be computed for each configuration. For example, we might expect different file access patterns during different shifts in a production environment.

1.8 Overview

The organization of the remainder of this thesis is as follows. Chapter II is a background chapter. First, it provides a literature review of work relevant to the FAP and the BPP. It then introduces some of the notation and basic concepts that will be used throughout the remainder of the thesis. Chapter III deals with the file assignment problem assuming that we are dealing with read-only files. Chapter IV then generalizes the results to include read-write files. In both Chapters III and IV we will address both replicated and nonreplicated file assignments. Finally, Chapter V summarizes the contributions and points to further areas of research.

CHAPTER II

BACKGROUND

In this chapter, we provide background material for the file assignment problem. In Section 2.1, we review the literature related to the FAP and the BPP. In Section 2.2, we present concepts and notation that are central to the rest of this work. We first discuss the goal programming concepts used in our FAP formulations. We then discuss some of the main ideas from queuing network theory. Finally, we briefly discuss the BPP solution techniques we rely on for the central server model.

2.1. Previous Research

There are many possible ways to organize the existing literature related to the file assignment problem. I will adopt a variation of Dowdy's [Dowd77] taxonomy which classifies the literature by problem formulation and solution techniques. The main divisions will be among the FAP, the BPP, the combined problems, and other related work. The FAP is further subclassified into linear programming models and queuing models as well as whether the models consider more than one file.

Recently Dowdy and Foster [Dowd78] have produced a comprehensive survey of FAP and BPP research. They have

produced a comparative table that summarizes some major features of the various models. This table can be found in Appendix A.

2.1.1. File Assignment Problem

2.1.1.A. Linear Programming Models

The largest body of literature is devoted to this approach. Generally these models are cost, rather than performance, oriented and as such may be somewhat artificial for many applications. Sometimes important variables must be known in advance when they are more appropriately decision variables. Queuing delays are either ignored or dealt with indirectly.

2.1.1.A.1. Single File Linear Programming FAP

These models optimize single file placement independently of the other files and obviously queuing delays are ignored.

Casey [Case72] formulates a linear cost model including update, query, and storage costs to determine the number and placement of copies for a single file. He notes that his problem is related to the warehouse location problem. Assuming update and query costs are equal, he proves an upper bound on the number of copies and shows monotonicity of a "cost graph" used to heuristically determine a local optimum solution. Eswaran [Eswa74] shows that this problem

is NP complete by reducing the minimum set cover problem to Casey's problem.

Levin and Morgan [Levi75, Morg77] also formulate a cost model including processing, update, query, and storage costs. They claim to have solved the multiple file problem but in reality they optimize file placements independently of each other. This means that queuing delays cannot be accounted for correctly. Their contributions include accounting for file access dependencies, dynamic access patterns, and probabalistic access patterns. Their inclusion of response time constraints ignore the fact that the variables are not independent of the file assignment.

Chandy and Hewes [Chan76] formulate a model similar to [Case72]. They solve the problem with a heuristic using linear programming that relaxes the integer constraints on file copy assignment to obtain a lower bound cost measure. They then utilize a hill climbing heuristic to search "near by" for a near optimal integral assignment and to provide an upper bound cost measure. Computational case studies show that in almost all cases the lower bound and upper bound solutions were the same. They recognize that this model only represents a partial solution to the problem and that queuing aspects must be considered.

2.1.1.A.2. Multiple File Linear Programming FAP

These models are generally more realistic than the single file models. Some tend toward performance optimization but most still ignore queuing delays.

Ramamoorthy and Chandy [Rama70] consider the assignment problem in a memory hierarchy. Given a program, its data requirements, and access profile, and a hierarchy of storage devices, they determine a set of memory sizes and types in order to minimize average access time subject to cost constraints. They model for a multiprogramming, multi-file environment with integer constraints on the memory modules. Queuing delays are ignored and the model is most appropriate for very predictable programs.

Arora and Gallo [Aror71] seek to minimize the sum of execution, access, and transfer times in a memory hierarchy subject to size constraints. Queuing delays are ignored in this portion of the model. They also consider a separate queuing model to determine optimal memory sizes from a set of alternatives. These two procedures are used in an iterative fashion to improve the cost/performance ratio.

Chu [Chu69, Chu73] deals with queuing delays and file allocation in a totally connected network. He seeks to minimize update, query, and storage costs subject to device capacities, a minimum response time, and a predetermined number of redundant file copies. This is formulated as a 0-1 nonlinear programming problem and suffers greatly from

intractability. Also it does not seem appropriate to have to provide the number of file copies. This should be determined by the solution technique.

Casey [Case73] tackles the problem of finding a file assignment, a tree network topology for data routing, and data line capacities to handle the traffic while minimizing line rental and file storage costs. The problem is a nonlinear, mixed integer programming problem and solved heuristically. No measure of optimality is provided.

Mahmoud and Riordon [Mahm76] address the problem of file allocation and communication link capacity for a fixed topology. They seek to minimize storage and communications cost subject to a ceiling on message delay and a file availability requirement (a reliability constraint). Solution to the file copy aspect is difficult and therefore approximated. No bounds are given to measure the "goodness" of the heuristic. The remaining problem is a nonlinear integer programming problem. Computational results suggest that the technique may be intractable for realistically sized problems. A shortcoming of the formulation is the exclusion of storage capacity constraints.

2.1.1.B. Queuing Models

The standard assumptions for most queuing analysis apply here. Service and arrival times are exponential and the queue disciplines are restricted, usually FIFO. These

assumptions have proven to work well enough in practice, and without them, analytic solutions are usually impossible to obtain.

Hughes and Moe [Hugh73] use a queuing model in a very pragmatic fashion to analyze a specific computer system. Their objective is to maximize CPU throughput on a stand alone system with four primary storage devices. Among other things they analyze the effect of reallocating the I/O load by file redistribution. They use a rule of thumb formalized by Buzen [Buze71a,Buze71c]: 1) when the degree of multiprogramming is high, load devices proportionate to their speeds; 2) when the degree of multiprogramming is low, overload the faster devices. They demonstrate the importance of the file allocation in a case study whereby CPU throughput is improved by 24%.

Buzen and Chen [Buze74a] consider a queuing model to balance the message traffic in the memory hierarchy of a central server system in order to minimize response time. The primary drawback is that they solve for what corresponds to an open system with an infinite number of customers. Buzen and Goldberg [Buze74b] show that this is a poor estimate for a closed system except for very low device utilization and very high degrees of multiprogramming.

Lipsky and Church [Lips77] apply known queuing theory results to a case study of an IBM 360/65 J system. Their objective is to obtain maximum productive (non-overhead)

throughput. The decision variables are the branching probabilities to a nonhomogeneous set of disks. Their model shows a 30% difference in productive utilization between the best and worst case tests. As a result they place the most heavily accessed data sets on the faster disks.

2.1.2. Branching Probabilities Problem

Buzen [Buze71a, Buze71c, Buze73] considers the BPP as applied to the central server model. The objective is to find the optimal branching probabilities to the peripheral devices so as to maximize CPU throughput. Buzen formally developes the load distribution rules of thumb used by Hughes and Moe [Hugh73].

Price [Pric74] provides the important result that CPU utilization as a function of the branching probabilities is unimodal for the central server model. This result verifies the solution techniques of Buzen [Buze71c], Foster and Browne [Fost76], and Foster, Dowdy, and Ames [Fost77].

Dowdy [Dowd77] solves for the optimal branching probabilities in a general network with the objective of maximizing overall network throughput (netput). Netput is defined as the sum of all the throughputs for all nodes in the network. The problem is solved by the same nonlinear programming technique used by Foster, Dowdy, and Ames [Fost77]. The unimodality result of Price [Pric74] is unfortunately not applicable to general networks.

Therefore, it is unknown whether netput is unimodal in general. Dowdy provides empirical evidence that netput may be unimodal by exhaustively examining several cases for a three node network. In all cases, netput was shown to be unimodal.

2.1.3. Combined FAP and BPP Problems

The solution techniques here recognize that a particular file assignment impacts on the branching probabilities. In particular the number of files stored on a device affects the average service rate of the device. Therefore there is an iterative approach between BPP solution and FAP realization of the probabilities.

Foster and Browne [Fost76] seek to maximize throughput in a central server model. Queuing delays, device capacities, and file reusability are included in the model. The optimal probabilities are calculated via an exhaustive search. The new file assignment is determined heuristically. Dowdy [Dowd77] reports that the technique is nearly intractable.

Foster, Dowdy, and Ames [Fost77] extend the work of Foster and Browne [Fost76] with a technique that iterates between a BPP solver (Micro model) and an integer programming file allocator (Macro model) that seeks to realize the optimal branching probabilities (BP's). The network topology is a central server model also called a

star network. The Micro model solves the BPP via nonlinear programming. The Macro model solves an integer programming problem that minimizes the sum of the differences between the optimal BP for a device and the file traffic assigned to the device. The assignment is subject to device capacity constraints, and integrality of files (no file splitting). The Macro file allocator provides exact, optimal answers but takes a long time to compute moderately sized problems. The overall process iterates until CPU throughput is maximized.

Jones and Foster [Jone79a] apply heuristic techniques to the general approach of [Fost77]. They sacrifice some model accuracy to obtain more reasonable computational speeds. First, service rates are assumed constant and are computed on the basis of overall average message length. This simplification introduces some inaccuracy but eliminates the costly iteration between the Micro and Macro models. Second, the integer restrictions are relaxed and a similar linear programming problem is solved. The answer is rounded to a feasible integral solution and the possible error introduced is calculated. While no error bounds are proven, test cases demostrate that this approach can lead to reasonable success.

2.1.4. Related Work

Here we reference a pertinent queuing analysis tool called ASQ that is not specifically related to the FAP or

BPP but may be useful in obtaining solutions to these problems. ASQ is a large FORTRAN IV program that was developed at the University of Texas and was later extended at Duke and Vanderbilt Universities [Aspl78]. ASQ stands for Arithmetic Solutions to Queues and provides the capability to anlyze queuing networks. Users may interactively design, build, and analyze networks of queues with a great deal of flexibility. Among other things, it can be useful for determining approximate solutions to complex BPP problems.

Ames and Foster [Ames77b] address the problem of when to reassign files assuming the file access patterns are dynamic rather than static. They assume the usage patterns are unknown and vary with time. A predictive algorithm is used that bases its prediction on past history. When it is established that the predicted gain minus the overhead involved with a file reassignment is greater than the current performance, then a file reassignment is made. Various strategies were analyzed using an event driven simulation. Adaptive file reassingment policies were found to yield around 50% better performance.

2.2. Notation And Some Basic Concepts

This section will explain most of the basic concepts needed for the rest of this work. First, we give a brief overview of the notation and concepts used in multiple

objective goal programming. Then, we present the basic notation for queuing networks and discuss the significant assumptions involved. Finally, we discuss the method of computing the optimal branching probabilities.

2.2.1 Goal Programming concepts

In many formulations of the file assignment problem, as well as in many "real world" problems, we encounter the need to optimize multiple conflicting objectives and find that a weighted objective function is inappropriate. Goal programming offers a method of formulating and solving such problems. In this section we offer an explanation of goal programmming terms based on the definitions by Ignizio [Igni76].

2.2.1.A. What is goal programming?

Linear goal programming has its basis in the more familiar linear programming but it differs in two significant ways. One primary extension allows the model builder to specify multiple, preemptive objective functions. The list of preemptive objectives are grouped into what is called an "achievement function". The other major difference is in the nonabsolute nature of constraints.

By allowing a multiple objective achievement function, the modeler is not forced to choose one objective to the exclusion of other possibly desirable objectives. Instead of stressing the optimization of a single objective, goal

programming stresses the satisfaction of multiple objectives. For example, a manufacturer might obviously wish to maximize his profits. However, in the same context he might also wish to minimize overtime operation, and have a marketing objective of selling the maximum number of a particular widget. This type of problem is a good candidate for a goal programming formulation.

The computation of a goal programming solution proceeds through the achievement function optimizing objectives in order of priority. At any given level of optimization, the solution mechanism allows only those changes that do not affect the optimality of a higher priority objective. In effect, at each level we solve a problem that has additional constraints based on the higher priority objective function values.

Another aspect of goal programming that differs from traditional linear programming is the treatment of constraints. Traditionally, constraints determine the boundaries of a system, and a solution must satisfy every constraint absolutely for it to be considered feasible. Goal programming substitutes nonabsolute "goals" for constraints and attempts to minimize deviation from a specified level rather than imposing an absolute requirement. This is particularly useful when you consider that many real world constraints are not truly absolute. Goal programming returns an answer that gives a quantitative

measure of deviation from a goal. The analyst may then make his own judgements concerning absolute feasibility.

While goal programming offers many advantages over traditional linear programming, there is a catch. The problem with solving goal programs is that computational overhead is increased. This may be particulary significant when we are solving large problems or a lot of them as in a branch and bound type approach. For this reason goal programming may be an unaffordable luxury in many applications.

In this work we attempt to borrow the good aspects of goal programming and discard the computationally costly aspects. The primary strength of goal programming from our point of view is in the ability to formulate problems. Goal programming provides a notation and a way of talking about complex multiple objective problems. We will take advantage of this strength. However, we recognize the computational difficulties for this application and therefore do not necessarily propose to solve all the problems we formulate by goal programming methods.

2.2.1.B. Goal programming notation

We will introduce the basic goal programming notation by demonstrating how to construct the objectives, and how to create the list of goals for the achievement function. The next section will give a small example to demonstrate the concepts.

Each objective will be expressed as a function of the decision variables, that is:

$$O(i) = f(i)$$

where O(i) represents objective i, and f(i) represents the ith function f(X) where $X = (x_1, x_2, \ldots, x_j)$, the vector of decision variables.

Every objective will have a right hand side of the form

$$f(i) \begin{matrix} >= \\ = \\ <= \end{matrix} b(i) .$$

We then introduce a <u>negative</u> deviation variable, n(i), and a <u>positive</u> deviation variable, p(i), so that each objective has the form

$$f(i) + n(i) - p(i) = b(i) .$$

We must next group objectives and assign priority levels to the these groupings. As we do this, we must keep in mind the <u>preemptive</u> nature of the goal programming solution mechanism. This means that for priorities Pi and Pj, i > j, Pi is preferred to Pj regardless of <u>any</u> multiplier applied with Pj. Thus, if any objectives are of an absolute nature, they should be placed in the highest priority level. The remaining nonabsolute objectives should be prioritized by the analyst.

The final step in model development is to establish a list of goals as an achievement function. Given the general form of an objective

$$f(i) + n(i) - p(i) = b(i),$$

we may achieve the three possibilities listed in Table 2.1 by minimizing a linear function of the deviation variables.

	Objective	Procedure	Comments
(a)	$f(i) \geq b(i)$	minimize $n(i)$	Min + deviation
(b)	$f(i) \leq b(i)$	minimize $p(i)$	Min - deviation
(c)	$f(i) = b(i)$	minimize $n(i) + p(i)$	Min total deviation

TABLE 2.1. Procedure for achieving objectives.

For each priority level we then group linear functions of the deviation variables into an achievement goal, a_i, where $a_i = g_i(N,P)$ and where $g_i(N,P)$ is the ith priority level linear function of the deviation variable vectors,

$$N = (n(1), n(2), \ldots, n(m)),$$

$$P = (p(1), p(2), \ldots, n(m)).$$

Then the overall achievement function is given as

$$A = (a_1, a_2, \ldots, a_k).$$

We may thus give the form of the general goal program as:

$$\text{Find } X = (x_1, x_2, \ldots, x_j)$$

so as to minimize

$$A = (a_1, a_2, \ldots, a_k)$$

such that

$$f(i) + n(i) - p(i) = b(i) \quad i=1, \ldots, m$$

and

$$X, N, P \geq 0.$$

2.2.1.C. Example

In this section we present a simple problem and give both the goal programming and the linear programming formulations for a problem. In comparing the two formulations, keep in mind that the two cannot be exactly the same due to the fundamental differences.

PROBLEM STATEMENT

A manufacturing company produces two products, product 1 and product 2. Product 1 requires 2 hours of assembly time and yields a net profit of $12. Product 2 requires 1 hour of assembly time and yields a net profit of $8. The nonovertime production time available per week is 40 hours. The demands per product per week are 30 units of product 1 and 15 units of product 2.

The decision variables are the amount of each product to be produced per week. In order of priority, the objectives are:

(1) Production must not exceed demand;

(2) Maximize profit;

(3) Minimize overtime necessary.

Assume that it is possible to product a fractional amount of the products.

LINEAR PROGRAMING FORMULATION

Minimize: $W * (-(8x_1 + 12x_2)) + (x_1 + 2x_2)$

subject to:

(a) $x_1 \leq 30$

(b) $x_2 \leq 15$

(c) $x_1 + 2x_2 \geq 40$

$x_1, x_2 \geq 0$.

We must assume that W can be a weighting factor such that the profit maximization objective is preemptively preferable to the overtime minimization objective.

GOAL PROGRAMMING FORMULATION

Find x_1, x_2 so as to

Minimize: $A = \{ (p(3) + p(4)), (n(1)), (p(2)) \}$

such that:

(1) $8x_1 + 12x_2 + n(1) - p(1) = 10000$

(2) $x_1 + 2x_2 + n(2) - p(2) = 40$

(3) $x_1 + n(3) - p(3) = 30$

(4) $x_2 + n(4) - p(4) = 15$

$X, N, P \geq 0.$

Constraints (a) and (b) from the LP formulation correspond to objectives (3) and (4) and are treated as priority 1 goals. Profit maximization (objective (1)) is priority 2, and overtime minimization (objective (2)) is priority 3. The right hand side of objective 1 was chosen arbitrarily but large enough to be an upper bound on weekly profit.

2.2.2. Queuing concepts

2.2.2.A. Notation

As mentioned before, we will model our computer systems as closed queuing networks. Consider Figure 2.1. In this model there are a total of "NDEVS" I/O devices, each with a mean service rate MU(j), j=1,...,NDEVS. Likewise, the CPU has a mean service rate LAMDBA. The average degree of multiprogramming (or in standard queuing terminology, the number of customers) in the system is DMP. The branching probability, P(j), j=1,...,NDEVS, represents the probability that a job from the CPU will request service at I/O device j. This model is called a central server model since all traffic is funneled through the CPU (the central server). The network is also said to be a closed network because the total number of customers in the system is finite and constant.

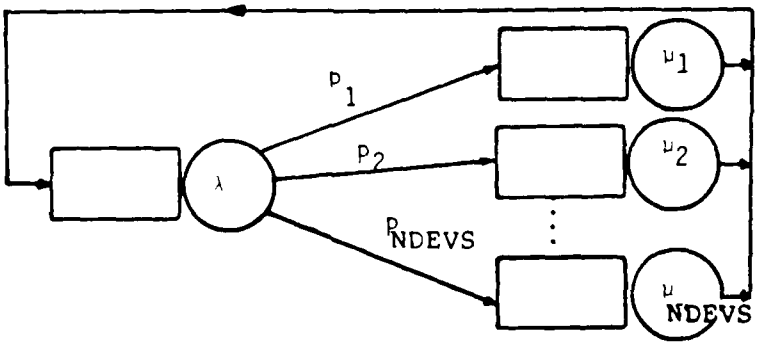


Figure 2.1. Queuing network model.

2.2.2.B. Assumptions

Our model is based on Markovian queuing theory assumptions. These assumptions are standard in the literature and are necessary to allow tractable analysis in most cases. The assumptions include: the service rates are exponentially distributed and are independent of queue length; the system has a fixed number of customers (DMP); the queue discipline is first-come-first-served; each queue provides customers for a single server; and, the system is assumed to be in a steady state condition (i.e., no transient effects of system initialization are present).

How significant are these assumptions? At first glance (and indeed after some research), the assumptions seem so harsh as to render the model useless. However, there is a much experimental evidence to suggest that good answers can be obtained using these assumptions [Buze75, Fost74, Giam76, Hugh73, Lips77, Rose78]. This has puzzled many analysts since some studies have shown that certain of the assumptions (e.g., exponentially distributed service rates [Brin73]) do not generally hold. This has led some researchers to search for underlying relations in the Markovian queuing network theory that might relate to directly verifiable assumptions. This approach is called operational analysis [Denn78]. The application of operational analysis principles have led to the same mathematical equations as the Markovian assumptions. This

is a significant result that adds confidence to the Markovian models.

2.2.3 Branching Probabilities Problem Solution

A prerequisite for solving the branching probabilities problem is the ability to compute the utilization of queuing network nodes. The key to this computation is the calculation of a nonlinear normalization factor, G, formulated by Gordon and Newell [Gord67]. Buzen [Buze73] and Dowdy [Dowd77] devised efficient methods for computing this factor, with Dowdy's method having the computational edge. We may compute the utilization of the CPU, given NDEVS, DMP, LAMBDA, MU(j), and P(j), j=1,...,NDEVS, as:

UTILIZATION(CPU) = G(DMP-1)/G(DMP).

The details of the computation are given in Appendix B.

Price [Pric74] has shown that central node utilization in a star network is a unimodal function of the branching probabilities. Dowdy [Dowd77] then uses nonlinear programming to compute the maximum CPU utilization by solving the unconstrained problem:

Minimize -G(DMP-1)/G(DMP).

Associated with this value is the set of optimal branching probabilities, P*'s. Since the space is known to be unimodal, the answer obtained is a global optimum.

CHAPTER III

MODELS FOR READ-ONLY FILES

3.1. Introduction

If the models in this work were classified according to the literature review in Chapter II, our approach would be under the heading of "Combined FAP and BPP Solutions" (Section 2.1.3). The model devised by Foster, Dowdy, and Ames [Fost77] for nonreplicated, read-only files serves as a point of departure for this research. Thus, our fundamental approach is similar in that we solve a BPP and then solve a FAP that attempts to realize the BPP solution. This is the approach illustrated by Figure 1.3 in Chapter 1. In this chapter we extend the model of [Fost77] by formulating a FAP that allows replicated copies of read-only files.

The organization of this chapter is as follows. Before presenting our model, we first define a new concept of file access that we will use in our model. After this, we present our model and give both an exact solution formulation and a heuristic technique for solving the problem. We then prove bounds on the total storage required by any solution to our model, and finally, we evaluate the effectiveness of the heuristic.

3.2 File access concepts

There have been two primary concepts of file storage in previous work: integral file storage, and split file storage. In this section we define a new concept, integral storage with split access. These ideas are illustrated in Figure 3.1.

Integral file storage is the standard method used in current computer systems. The idea is simply that a single, integral copy of a file is stored on a single storage device. The access mechanics are simple and are built into conventional operating systems. With respect to system performance, the disadvantage of this method is is that it may not be possible to meet the optimal branching probabilites exactly.

The split file storage concept advocates the physical splitting of a file in order to be able to match the branching probabilities exactly. The disadvantage of this approach is that no operating system supports this type of access and the problems of implementing this could be enormous.

The concept we introduce in this work is the idea of integral file storage with split access. With this concept we store integral copies of a file but there may be copies of any given file stored on different devices. By splitting the access we mean that the operating systems does not always have to access the same copy of a replicated file.

Thus, the access for any such file is said to be split among devices. By allowing the assignment of a fraction of the total file access to a given copy of a file, it is much more likely that the optimal branching probabilities may be met exactly.

We can see that the optimal branching probabilities can be met exactly using this access method if there are no storage capacity restrictions. Since there are no capacity restrictions, we may put a copy of each file on all the devices. If we denote the empirically observed and normalized access frequency to a file i by FREQ(i), and the portion of file i access assigned to device j by FREQFR(i,j), then we will match all P*(j)'s exactly by assigning FREQFR(i,j) = FREQ(i) * P*(j) for i=1,NFILES (number of files), j=1,NDEVS (number of devices). To see this, note that $\sum_{i=1}^{NFILES}$ FREQ(i) = 1 by definition, and since each device contains all files,

$$\sum_{i=1}^{NFILES} FREQFR(i,j) = \sum_{i=1}^{NFILES} FREQ(i) * P^*(j) \text{ for all } j.$$

Obviously, the file access traffic on each device equals the optimal branching probability for each device.

While this storage method is not currently practiced on standard operating systems, it has a potentially simple implemementation. If we add entries in the system file directory to indicate multiple copies and the percentage of access to be given to a particular copy, the system may generate a random number and, based on a probability

distribution, access the appropriate copy. In the long run the correct access frequencies would tend to be realized.

BPP solution :
$P^*(1) = .7$, $P^*(2) = .3$.

FILE	ACCESS FREQUENCY
A	.5
B	.15
C	.35

Accessing Method	Optimal assignment Dev. 1	Dev. 2
Integral Storage	Files A & B (.65)	File C (.35)
Split Storage	File A 20/35 File C (.7)	File B 15/35 File C (.3)
Integral Storage, Split Access	File A File C * (.7)	File B File C ** (.3)

* Access frequency = .2
** Access frequency = .15
Values enclosed in parenthesis are TOTFREQ values.

Fig. 3.1. File accessing methods.

Figure 3.1 compares optimal file assignments for the various access methods. The values enclosed in parenthesis are the total file access frequencies assigned to the device, TOTFREQ. Note that the optimal branching probabilities can be met exactly using either split storage

or integral storage with split access. The integral storage method misses the match by (.7-.65) = .05 for device 1 and (.35-.3) = .05 for device 2.

3.3. Read-only, replicated file FAP model

In this section we describe our solution to the model at a high level. First, we outline the algorithm. We then give the goal programming formulation for the file assignment portion of the model, and finally we outline the heuristic techniques uses to obtain an approximate solution to the problem.

3.3.1. Outline of Algorithm

The high level structure of the algorithm is given below. We then describe the major steps in more detail.

(A) Read Input;

(B) Compute Device Service Rates;

(C) Solve BPP;

(D) Assign Files.

3.3.1.A. Read Input

The Read Input step reads the following data which is summarized in Figure 3.2:

(a) File information: the number of files (NFILES), file access frequencies (FREQ), physical file lengths (LEN), average message lengths (MLEN);

(b) Storage device information: the number of devices (NDEVS), device capacities (CAP), mean latency times (MLT), transfer times per word (TT);

(c) CPU mean burst time (LAMBDA), and

(d) The average number of core active jobs (DMP).

```
NFILES    :    Number of files
FREQ(i)   :    Frequency of accessing file i, 0<=FREQ(i)<=1.
LEN(i)    :    Length (in words) of file i.
MLEN(i)   :    Average message length of file i.

NDEVS     :    Number of storage devices.
CAP(j)    :    Capacity (in words) of device j.
MLT(j)    :    Mean latency time of device j.
TT(j)     :    Transfer time per word for device j.

LAMBDA    :    CPU speed.

DMP       :    Degree of multiprogramming.

   The limits for i and j are:
          i = 1,...,NFILES, and
          j = 1,...,NDEVS.
```

Fig. 3.2. Input to model.

3.3.1.B. Compute Device Service Rates

Service rates are computed based on a frequency weighted average message length. Specifically we may compute the average message length as

$$AML = \sum_{i=1}^{NFILES} FREQ(i)*MLEN(i).$$

We then compute the service rate, MU(j), for device j as MU(j) = 1 / (MLT(j) + TT(j) * AML).

3.3.1.C. Solve BPP

This computes the set of branching probabilities, P*(j)'s, that maximize CPU utilization. It is solved using non-linear programming as discussed in Section 2.2.3. Recall that the result is known to be optimal for the central server model [Pric74]. For details see [Fost77].

3.3.1.D. Assign Files

Subject to device storage capacities, we assign files in an attempt to match the P*(j)'s. The "closest match" is defined as minimizing the sum of the differences between the assigned file access to a device, and the P*(j) of the device. This problem is formulated as an integer linear goal programming problem. The formulation is given in Section 3.3.3.

3.3.2. Comments on the Basic Algorithm Structure

Since we state that our approach is similar to the approach in [Fost77], it is appropriate to point out any significant differences. The chief difference lies in the basic structure of the algorithm. In [Fost77] the algorithm structure in pidgin ALGOL [Aho74] is:

```
Read-input;
UNTIL Convergence DO
     BEGIN
          Compute-Service-Rates;
          Solve-BPP;
          Assign-Files;
     END;
```

The algorithm is structured this way based on a recognition that service rates are assignment dependent. This, in turn, may result in a different BPP solution. We may see this dependence intuitively. Consider that if a device has only files with very short message lengths assigned to it, it can send more messages per unit time than if the device has files with very long message lengths assigned to it. Thus, the assignment determined by the Assign-Files algorithm may indeed affect the service rates and the BPP solution.

The most serious shortcoming of this method is that there is nothing inherent in Assign-Files or Solve-BPP to force convergence. This obviously means that the algorithm may not terminate. This cannot be viewed as totally acceptable.

We can view the algorithm proposed in this work as a method of forcing convergence. Here we approximate the actual service rates by using an overall average message length. In some cases perhaps this sacrifices some

accuracy. However, we gain the assurance that our algorithm terminates. As further justification, the use of constant service rates has been standard in much of the literature.

3.3.3. Goal Programming Formulation

We present the following informal formulation of the file assignment problem. For the formal representation, refer to Figure 3.3. The objectives of the problem are given below with supplementary explanations when necessary. We then list the prioritized goals of the achievement function.

OBJECTIVES:

a. Assign 100% of file accesses. (This is a housekeeping constraint to insure that the totality of file accesses is accounted for).

b. Assess a fixed charge for the allocation of a file copy. (Another housekeeping constraint to charge for the allocation of a complete file copy even when access is split).

c. Do not exceed the physical capacity of a device.

d. Match the P*(j)'s as closely as possible. (Try to make the sum of the differences over all devices = 0).

e. Minimize total storage used.

f. 0-1 constraint for file copy allocation. (File copies are allocated in integral units).

g. All variables >= 0. (Standard feasibility constraints).

ACHIEVEMENT FUNCTION:

Priority	Goals	
1	Objectives a,b	(Assign all accesses, charge for copy)
2	Objective c	(Stay within capacities)
3	Objective d	(Match the P*(j)'s)
4	Objective e	(Minimize storage used)

VARIABLES: FREQ,LEN,CAP,NFILES,NDEVS same as Fig. 3.2.

ASMTFR(i,j) : Access frequency for file i on device j divided by FREQ(i), 0 <= ASMTFR(i,j) <= 1.

ASMT(i,j) : the ceiling of ASMTFR(i,j).

$n(k)_{..}$: Negative deviation variable for goal k.

$p(k)_{..}$: Positive deviation variable for goal k.

Limits on i and j: i=1,...,NFILES, j=1,...,NDEVS.

ACHIEVEMENT FUNCTION:

$$\text{Min } a=\Big(\Big[\sum_{i=1}^{NFILES} (n(1)i + p(1)i) + \sum_{i=1}^{NFILES} \sum_{j=1}^{NDEVS} (p(2)ij) \Big], \Big[\sum_{j=1}^{NDEVS} p(3)j \Big], \Big[\sum_{j=1}^{NDEVS} n(4)j \quad p(4)j \Big], \Big[\sum_{j=1}^{NDEVS} p(5)j \Big] \Big)$$

OBJECTIVES:

$$\text{(a): } \sum_{j=1}^{NDEVS} ASMTFR(i,j) + n(1)i - p(1)i = 1 \qquad \text{for all } i$$

$$\text{(b): } ASMTFR(i,j) + n(2)ij - p(2)ij = ASMT(i,j) \qquad \text{for all } i \text{ for all } j$$

$$\text{(c): } \sum_{i=1}^{NFILES} ASMT(i,j) * LEN(i) + n(3)j - p(3)j = CAP(j) \qquad \text{for all } j$$

$$\text{(d): } \sum_{i=1}^{NFILES} ASMTFR(i,j) * FREQ(i) - P^*(j) + n(4)j - p(4)j = 0 \qquad \text{for all } j$$

$$\text{(e): } \sum_{i=1}^{NFILES} ASMT(i,j) * LEN(i) + n(5)j - p(5)j = 0 \qquad \text{for all } j$$

(f): ASMT(i,j) = (0,1) for all i, j

(g): ASMTFR(i,j) >= 0 for all i, j

Fig. 3.3. Goal programming formulation

3.3.4. Heuristic Solution Outline

The problem formulation for the Assign-Files problem given in Section 3.3.3 is precise, but unfortunately contains integer variables. This leads to computational intractability for realistically sized problems. Therefore, we resort to heuristic techniques to obtain approximate solutions in a reasonable amount of time. These heuristics provide a solution that is optimal for achievement goals 1 and 2 and will be optimal or "near" optimal for goals 3 and 4. Sections 3.3.5 and 3.4 will furnish measures for establishing how good the heuristics are.

For simplicity, we assume there is at least one large mass storage device, j_L, that can hold all of the files. This is a reasonable assumption given that off-line storage such as a tape library can easily fit this requirement. In pidgin ALGOL the Assign-files problem is formulated as follows:

```
Match-BPs;
IF Any-device-capacity-exceeded
     THEN Obtain-capacity-optimal-solution;
Minimize-storage-used;
```

3.3.4.A. Match-BPs

This solves the computationally easier linear programming problem that is formally stated in Figure 3.4. The problem is to minimize the sum over all devices of the

differences between P*(j) and the file accesses assigned to device j (this is goal 3 in the goal programming formulation). This is subject to device capacity constraints (goal 2), the requirement to assign all file accesses (this is contained in goal 1), but ignoring the integral charge for storing a file copy (also contained in goal 1).

$$\text{Minimize : } \sum_{j=1}^{NDEVS} EPSLN(j)$$

Subject to:

$$\text{(a)} \quad \sum_{i=1}^{NFILES} ABS(ASMTFR(i,j) * FREQ(i) - P^*(j)) <= EPSLN(j) \quad \text{for all } j$$

$$\text{(b)} \quad \sum_{i=1}^{NFILES} ASMTFR(i,j) * LEN(i) <= CAP(j) \quad \text{for all } j$$

$$\text{(c)} \quad \sum_{j=1}^{NDEVS} ASMTFR(i,j) = 1 \quad \text{for all } i$$

$$\text{(d)} \quad ASMTFR(i,j) >= 0 \quad \text{for all } i, j$$

Limits on i and j: i=1,...,NFILES, j=1,...,NDEVS.

Fig. 3.4. Match-BP's formulation.

By solving this problem we obtain a solution that is essentially goal 1 optimal (compliance with the integral requirement is automatically met when we perform the subsequent steps), but since the integer restriction was ignored, the solution might not be goal 2 optimal. Thus we require the test, "IF Any-device-capacity-exceeded", to insure that we optimize goal 2 at the expense of goal 3, if

necessary. This solution also gives a lower bound on goal 3 but not necessarily a greatest lower bound.

3.3.4.B. Obtain-capacity-optimal-solution

If any device capacity is exceeded, then the solution is goal 2 suboptimal since we may trivially meet this condition by assigning no files to a device (Recall that we can place all of the files on the large mass storage device, j_L). The general idea of this procedure is to reallocate files from the offending devices in an intelligent fashion trying to "stay close" to the unrestricted goal 3 optimum. The offending devices are considered in the following order: underloaded devices, perfectly loaded devices, overloaded devices. The underloaded devices (if any) are devices that are already somewhat capacity constrained, otherwise they would not be underloaded. These devices then tend to limit our ability to match the optimal P*(j)'s and should be dealt with while we have the greatest flexibility to reallocate files.

Within each of the categories, the offending devices are examined in decreasing order of P*(j) value, i.e., largest P*(j) first. The rationale is that this is roughly the order in which the devices can do the most "harm" to optimality, so we deal with them while we are most flexible.

The actual reallocation of files is a localized swapping of files and file accesses. No attempt is made to

try all possible recombinations because that would be an exponentially difficult problem, thus defeating the purpose of the heuristic in the first place. We defer discussion of the reassignment details until Section 3.3.6 because we require some notation and concepts from Section 3.3.5.

3.3.4.C. Minimize-storage-used

Using the methods described in Section 3.3.5, we may shuffle the assignment of shared files so that we do not affect goal 3 optimality, but we can decrease the total number of file copies below a proveable bound. The bounds are proven in Section 3.3.5 and are:

Lower bound: Sum of the length of all files

Upper bound: Lower bound + (# devices - 1) x length of the largest file.

We again defer some of these details until Section 3.3.6.

3.3.5. Proof of Storage Bounds

In this section we will prove upper and lower bounds on storage required to meet goals 1, 2, and 3. First, we provide the basic definitions, then an intuitive overview and the basic equations. Finally, we provide the formal proofs.

3.3.5.A. Defintions

1. TOTFREQ(j) : The sum of all file access frequencies assigned to a device j. TOTFREQ(j) = $\sum_{i=1}^{NFILES}$ FREQ(i) * ASMTFR(i,j), where ASMTFR is defined as in Figure 3.3.

2. File copy matrix (FCM) : An NFILES x NDEVS matrix, M, with entries f(i,j) such that:
if f(i,j) = 0, then file i is not stored on device j, and if 0 < f(i,j) <= 1, then file i is stored on device j and f(i,j) = FREQ(i) * ASMTFR(i,j).
Furthermore, we call a FCM a feasible FCM if

$\sum_{i=1}^{NFILES}$ f(i,j) = TOTFREQ(j), j=1,...,NDEVS, and

$\sum_{j=1}^{NDEVS}$ f(i,j) = FREQ(i), i = 1,...,NFILES.

If 0 < f(i,j) < 1, then we say f(i,j) is a "proper" entry.
Informally, this is a matrix that defines a file assignment, i.e., what files are stored on which devices and what the access frequencies are for all file copies. The significance of a proper entry is that more than one copy of the file it represents will exist in the assignment. If f(i,j) = 1, then only a single copy of file, i, will be present.

3. File copy walk (FCW) : In a FCM, a sequence of proper f(i,j)'s such that the pattern is $[f(i_0,j_0);\ f(i_0,j_1),\ f(i_1,j_1);\ f(i_1,j2),\ldots;\ f(i_{n-3},j_{n-2}),\ f(i_{n-2},j_{n-2});\ f(i_{n-2},j_{n-1})]$ and, in no case does ik = ik+1. For each column in a FCW, except perhaps column j_0, we say there is a "way in" to the column. Likewise, there is a "way out" of each column, except perhaps column j_{n-1}. Hence, we say P "goes through" column j_k, $0<k<n-1$. Note that the first two entries are in the same row. Likewise, the last two entries are in the same row. A FCW that goes through n columns is called an n-FCW.

4. File copy path (FCP) : A FCW with at most one way in and at most one way out for each row or column. A FCP that goes through n columns is called an n-FCP.

5. File copy cycle (FCC) : A FCP such that $j_0=j_{n-1}$ and i_{n-2} does not equal i_0. A FCC that goes through n columns is called an n-FCC (n-cycle).

6. Two FCM's, M1 and M2, are row equivalent iff for all i, $\sum_{j=1}^{NDEVS}$ (f(i,j) for M1) = $\sum_{j=1}^{NDEVS}$ (f(i,j) for M2). M1 and M2 are column equivalent iff for all j, $\sum_{i=1}^{NFILES}$ (f(i,j) for M1) = $\sum_{i=1}^{NFILES}$ (f(i,j) for M2). M1 and M2 are row-column equivalent iff M1 and M2 are row equivalent and column equivalent.

7. An nxn submatrix, m, is formed from an NxN matrix, M, by eliminating N-n complete rows and N-n complete columns. Note that if m contains a cycle, then M contains a cycle.

3.3.5.B. Overview and basic equations

In Section 3.3.5.C we present the formal proofs of the storage bounds, but since the concepts are basically simple, we first present an intuitive view of the main ideas of the proofs.

Consider the feasible FCM, M, in Figure 3.5. Note that M contains a file copy cycle, namely, (f(1,1);f(1,2),f(3,2);f(3,1)). We may compute a row-column equivalent matrix, M', in at least two ways. First, we could recompute the f(i,j) as $f'(i_k,j_k) = f(i_k,j_k) - V1$, $f'(i_k,j_{k+1}) = f(i_k,j_{k+1}) + V1$ as long as V1 <= .11. If we choose V1 = .11, we obtain the matrix in Figure 3.6. Note that it has one less file copy (non-zero f(i,j)) than Figure 3.5. We could also recompute the f(i,j) as $f'(i_k,j_k) = f(i_k,j_k) + V2$, $f'(i_k,j_{k+1}) = f(i_k,j_{k+1}) - V2$, as long as V2 <= .09. If we choose V2 = .09, we obtain the matrix of Figure 3.7. Note that it also has one less file copy than Figure 3.5.

Files	Devices			FREQ(i)
	1	2	3	
1	.21	.19	0	.4
2	0	0	.3	.3
3	.09	.11	.1	.3
P*(j)	.3	.3	.4	

Fig. 3.5. A file copy matrix (FCM), M.

Files	Devices			FREQ(i)
	1	2	3	
1	.1	.3	0	.4
2	0	0	.3	.3
3	.2	0	.1	.3
P*(j)	.3	.3	.4	

Fig. 3.6. M', computed using V1.

Files	Devices			FREQ(i)
	1	2	3	
1	.3	.1	0	.4
2	0	0	.3	.3
3	0	.2	.1	.3
P*(j)	.3	.3	.4	

Fig. 3.7. M', computed using V2.

We now formally present the computations just illustrated. Later, we will see that it may be useful to

perform reassignment along a FCP as well as a FCC. Therefore, both cases are illustrated.

Given a FCM, M, containing a FCP, P = [$f(i_0,j_0)$; $f(i_0,j_1)$,$f(i_1,j_1)$;...,$f(i_{n-2},j_{n-1})$;$f(i_{n-2},j_{n-1})$], where V1 <= MIN($f(i_k,j_k)$), 0<=k<=n-2, and V2 <= MIN($f(i_{k-1},j_k)$), 1<=k<=n-1, we may compute another FCM, M', as either:

$$\text{(a)}\quad f'(i_m,j_k) = \begin{cases} f(i_m,j_k) - V1\ , & m = k \\ f(i_m,j_k) + V1\ , & m = k-1, \text{ or} \end{cases}$$

$$\text{(b)}\quad f'(i_m,j_k) = \begin{cases} f(i_m,j_k) + V2\ , & m = k \\ f(i_m,j_k) - V2\ , & m = k-1. \end{cases}$$

Furthermore, the new matrix, M', is row equivalent to M and column equivalent except for columns j_0 and j_{n-1}, which differ as follows:

$$\text{(1)}\quad \sum_{i=1}^{NFILES} f'(i,j_0) - V1 = \sum_{i=1}^{NFILES} f(i,j_0),$$

$$\sum_{i=1}^{NFILES} f'(i,j_{n-1}) + V1 = \sum_{i=1}^{NFILES} f(i,j_{n-1})$$

if the f's are computed as in (a), or

$$\text{(2)}\quad \sum_{i=1}^{NFILES} f'(i,j_0) + V2 = \sum_{i=1}^{NFILES} f(i,j_0),$$

$$\sum_{i=1}^{NFILES} f'(i,j_{n-1}) - V2 = \sum_{i=1}^{NFILES} f(i,j_{n-1})$$

if the f's are computed as in (b).

In the case where M contains a FCC, C = [$f(i_0,j_0)$; $f(i_0,j_1)$,$f(i_1,j_1)$;...; $f(i_{k-2},j_{k-1})$, $f(i_{k-1},j_{k-1})$; $f(i_{k-1},j_0)$], and V1 and V2 are as before, we may compute a new FCM, M', using either (a) or (b) as above. The resulting FCM, M', is row-column equivalent to M.

The verification of these assertions is straightforward. By our choices of V1 and V2 we guarantee that no f(i,j) will be recomputed to be less than 0. Then, since we add and subtract the same terms exactly once for each row in the FCP or FCC, we maintain row equivalence. Note that for a feasible FCM no f'(i,j) will be greater than 1 since the sum along any row is <= 1 to start with. In the case of the FCP, we perform only one calculation in columns j_0 and j_{n-1}, hence the excesses/deficiencies follow as given. The other columns are recomputed like the rows and column equivalence is maintained. In the case of the FCC, the final computation occurs in the same column as the first computation. We thus have the case that the excesses/deficiencies cancel each other out yielding column equivalence for all columns for M' as well as row equivalence.

In Section 3.3.5.C we show that as long as we choose our V1's to be the Min ($f(i_k,j_k)$) term on a file copy cycle, or choose V2's to be the Min($f(i_{k-1},j_k)$) on a FCC, we may eliminate file copies until there are no more cycles. We establish necessary and sufficient conditions for the

existence of a cycle, and prove the maximum number of matrix entries (file copies) such that the matrix has no cycles. Finally, we establish the worst case storage condition to provide an upper bound on the storage required.

3.3.5.C. Proofs

LEMMA 1.

A file copy matrix, M, has a file copy cycle iff M contains an nxn submatrix, m, with n<= MIN(NDEVS,NFILES), such that each of the rows has 2 or more non-zero entries and each of the columns has 2 or more non-zero entries.

PROOF:

First we assume that each row and column of a FCM must contain at least one non-zero entry, otherwise we redefine the dimensions of M. We also note that for a cycle to have both a way in and a way out of a column (required if an entry of that column is contained in a cycle), the column must have at least 2 different rows with non-zero entries and each of these row entries has a corresponding non-zero entry in another column.

(1). Given an n-cycle, we must have an nxn submatrix with the specified properties.

This follows from the definition of a file copy cycle. We get the desired submatrix by choosing the rows and columns visited by the n-cycle.

(2). Given the n x n submatrix as specified, we will show that there must be a k-cycle, k <= n.

Any one of the n columns of the submatrix has at least 2 different non-zero entries, i.e., a non-zero entry in at least 2 rows. In turn, each of these rows has another non-zero entry in a different column. Therefore, we see that all n columns have a way in and a way out. We may build a cycle as follows. Start at a non-zero entry. Go to a non-zero entry in the same row. Then go to a non-zero entry in the same column as the previous entry. Then go to a non-zero entry in the same row, etc.. Repeat this process until a row or column is visited that has been previously visited. Delete the portion of the path from the beginning up to the repetition point. What is left is a cycle. Observe that this is a standard Eulerian type construction from graph theory.

LEMMA 2.

An n x m file copy matrix, M, with at least n+m non-zero entries, n and m >= 2, contains a file copy cycle.

PROOF:

We induct on s = n + m. The basis is s = 4. For s=4 we have a 2 x 2 FCM where all entries are non-zero. This obviously meets the Lemma 1 conditions for a FCC.

Assume that if 4 <= s' < s and M' is an n' x m' FCM with s' = n' + m', n' and m' >= 2, and with at least s' non-zero entries, then M' contains a FCC.

Let M be an n x m FCM with s = n + m, n and m >= 2, and with at least s non-zero entries. If every row and every column of M contains at least 2 non-zero entries, then M contains a FCC by Lemma 1. Therefore, without loss of generality, we assume at least one row has no more than 1 non-zero entry. Let R_1 be the set of all rows with <= 1 non-zero entries, and let r_1 be the number of rows in R_1. Now let M' = M excluding the R_1 type rows. So M' is a $(n-r_1)$ x m FCM with at least s' = $(n-r_1)$ + m non-zero entries. We need only establish that n' = n - r_1 >= 2. If n' = 0, i. e., n = r_1, then M had only n = r_1 non-zero entries, a contradiction. If n' = 1, i. e., n = r_1 - 1, then M had n - 1 non-zero entries in R_1 type rows and at most m non-zero entries in the other row for a total of <= (n-1) + m, a contradiction. Thus, n' >= 2 and M' meets the criteria of the induction hypothesis. Since M' contains a FCC, M contains a FCC.

THEOREM 1.

Given that at least one device has the capacity to hold all files at one time, the minimum storage necessary to optimize goals 1, 2, and 3 lies between

$$\sum_{i=1}^{NFILES} LEN(i) \quad \text{and}$$

$$\sum_{i=1}^{NFILES} LEN(i) + (NDEVS - 1) \times LEN(i_L),$$

where file i_L is the largest sized file in words.

PROOF:

LOWER BOUND:

Since all files must be assigned, the lower bound is trivially a single copy of each file. This requires $\sum_{i=1}^{NFILES} LEN(i)$ words of storage.

UPPER BOUND:

We know from Lemma 2 that by eliminating file copies along file copy cycles we can guarantee that no assignment need have more than NFILES + NDEVS - 1 copies (FCM dimensions are NFILES x NDEVS). Consider the worst case example where the largest file must be replicated NDEVS times. In the worst case we can have $FREQ(i_L) > 1 - Min(P^*(j))$ for all j. This forces NDEVS copies of i_L. The other files are assigned once, and the bound is as stated

3.3.6. Details of the Reassignment Heuristics

3.3.6.A. Obtain-Capacity-Optimal-Solution Reassignments

In this section we address the file reassignment heuristics used in the Obtain-capacity-optimal-solution procedure overviewed in Section 3.3.4.B. We will also give measures of the potential goal 3 optimality loss when we perform the localized file reassignment. The general idea of these heuristics is to reassign file accesses until all devices are goal 2 optimal (i.e., no capacities are violated) while trying to "stay close" to the non-integer-restricted goal 3 optimum determined by Match-BPs (Section 3.3.4.A.). Note that this optimum may be unrealizable if the integer variables had been taken into account.

We define the sets J-,J+,J0 as follows: J- is the set of access underloaded devices (TOTFREQ(j) < P*(j)); J+ is the set of access overloaded devices (TOTFREQ(j) > P*(j)); J0 is the set of devices having TOTFREQ(j) = P*(j). Note that after any heuristic reassignment step we must reclassify the devices into these sets. As mentioned in Section 3.3.4.B, we deal with devices in the following order: J-, J0, J+. Within each set we consider devices in decreasing order of P*(j). We treat devices in J0 as members of J- or J+ if they share a file with a device in the respective set. It is not possible for J- and J+ to

share a file and still be goal 3 optimal (see Section 3.3.6.B).

The high level structure of the reassignment process is given in pidgin ALGOL as:

```
WHILE Any-capacities-violated DO
     BEGIN
          Device := Next;
          WHILE Capacity-violated(Device) DO
               BEGIN
                    Initial-reassignment(Device);
                    Secondary-reassignments(Device);
               END;
     END
```

The "Next" function furnishes the next device for file reassignment according to the stated priority scheme. The following sections describe the other major procedures. All measures of goal 3 optimality loss are with respect to the non-integer-restricted solution to Match-BPs (Figure 3.4). In most cases these measures include a multiplictive factor of 2. This is because one device's total frequency loss must be one (or more) other device's total frequency gain. This is required by constraint (c) in Figure 3.4.

Regarding goal 3 optimality loss, there is a crucial point to keep in mind. Namely, the loss of goal 3 optimality will not necessarily mean a decrease in CPU utilization. A utilization decrease will definitely occur

if the service rates are truly assignment independent as in the case where all average message lengths (MLEN's) are the same. Surprisingly, where service rates are actually assignment dependent it is possible to obtain better CPU utilization. This phenomenon is difficult to predict due to the nonlinear nature of CPU utilization. This is demonstrated in Section 3.4.2.E.

3.3.6.A.1. Initial reassignments

The initial file reassignment considers only those files stored on an offending device. Exhaustive reassignment is not attempted for tractability reasons. The steps below define the initial reassignments used for a device j'. Only one of the initial reassignments is performed per iteration.

1. Eliminate all files, i, that cannot fit on device j', i.e., all i such that LEN(i) > CAP(j'). Place the files on device j_L (the large mass storage). The objective 3 optimality loss is 2*ASMTFR(i,j')*FREQ(i) for all files i involved.

2. Next we consider three different ways to reassign files from j' and pick the file that causes the least loss of goal 3 optimality. Denote this file by i'.

(a). Check for file copy cycles that include j'. If j' is in one or more FCC's and if any f(i,j') on the cycle equals V1 or V2, then we may eliminate file i on device

j' with no loss of optimality. Denote this file by A, and denote the loss to optimality by A-loss = 0. If there is no such file, let A-loss = 2 + e, e>0. The value of e is arbitrarily chosen. As long as e is greater than 0, A-loss is larger than the worst possible optimality loss (namely, 2.00) to indicate the non-existence of file A.

(b). Consider files on j' that are shared with other devices k. Denote by B the shared file which causes the least loss to optimality when all of f(i,j') is allocated to the sharing device k. The loss to optimality, B-loss, is:

i. If j' is an element of J'-, B-loss = (MIN (TOTFREQ(k) + f(i,j') - P*(k) for all devices k sharing copies of any i with j') + f(i,j')).

ii. If j' is an element of J+, 2*MIN(f(i,j')).

(c). Finally, we denote by C the file with the MIN(f(i,j')). Denote the potential loss of optimality as C-loss = 2*MIN(f(i,j'))

We now choose i' = the file corresponding to MIN(A-loss,B-loss,C-loss), and reassign i' as follows: if i' = A, the reassignment is along the cycle as in Section 3.3.5; if i' = B, the reassignment is to the device k sharing the file; if i' = C, the reassignment is to the mass storage device j_L.

3.3.6.A.2. Secondary reassignments

In this section we discuss two secondary reassignment schemes. The first method involves one-for-one file exchanges between devices in J- and devices in J+. The second method involves reallocation along a file copy path between devices in J- and devices in J+. First we will deal with the one-for-one exchanges.

If the initial reassignment procedure results in an optimality loss, then there may be some obvious one-for-one trades between devices in J- and devices in J+. Specifically, we look for a f(i+,j+) > f(i-,j-) where j- is an element of J- and j+ is an element of J+, such that swapping the two files does not violate either device capacity. We choose two files to exchange such that we minimize ABS(TOTFREQ(j+) - P*(j+)) after the exchange. The file exchange improves goal 3 optimality in a straightforward way with the optimality gain =

2*(TOTFREQ(j+) - P*(j+) before -

ABS(TOTFREQ(j+) - P*(j+)) after).

There is a special case of this idea where we simply move a file from J- to J+ without swapping a corresponding file to J-. We can consider this to be a swap where LEN(j+)=0 and FREQ(j+)=0.

A second straightforward method of improving goal 3 optimality exists if there is a file copy path between devices on J+ and J-. Note that if the initial reassignment

step did not sacrifice some optimality, this condition cannot occur. For example, consider two devices, j- an element of J- and j+ an element of J+, sharing a file i. We may obviously improve the goal 3 optimality by decreasing f(i,j+) by some V and increasing f(i,j-) also by V as long as TOTFREQ(j-) + V <= P*(j-) and TOTFREQ(j+) - V >= P*(j+). From our basic definition of this process we know that we can make this recomputation along a path of arbitrary length.

3.3.6.B. Minimize-Storage Reassignments

Since storage minimization is a lower priority goal (4) than P* matching (3), we are constrained to not change the goal 3 optimality measure. We may reassign files in two similar ways in order to obtain an assignment that has total storage within the proven bounds.

First of all, we may obviously reassign files along files copy cycles in as in Section 3.3.5 until there are no more file copy cycles. We will then be within the storage bounds. Since any resulting file copy matrices produced this way are row-column equivalent, we do not sacrifice any goal 3 optimality.

If the optimum goal 3 measure is not zero, i. e., we could not achieve exact P* matching due to capacity restrictions, we may have some additional flexibility to further minimize storage. We noted in Section 3.3.6.A that

if file sharing occured, it must be among device within their own relative loading sets in order to be goal 3 optimal, i. e., elements of J- cannot share files with elements of J+. Thus in some circumstances, we may be able to reassign shared file access among devices that are not along a cycle as long as the devices remain within their relative loading classification, i. e., J- or J+. Perfectly loaded devices (J0) can only do this reassignment along a cycle. Stated precisely, we may reassign all of the file i access, f(i,j), from device j to device k along a file copy path and not affect goal 3 optimality as long as

f(i,j) = Vl of path p

and either

(1) TOTFREQ(k) + f(i,j) <= P*(k),

if j and k are elements of J-,

or

(2) TOTFREQ(j) - f(i,j) >= P*(j),

if j and k are elements of J+.

These conditions insure that no device along the path is assigned an infeasible (negative) access to file i, and that no device along the path changes its membership from J- to J+ (J+ to J-). Since f(i,j) is equal to zero after the reassignment, we may delete that copy of i from j. This concept is illustrated in the following example when the file 6 access on device 4 is shifted to device 3.

3.3.7. Example

In this section we present an example to illustrate and review some of the major concepts previously presented. The file assignment matrix entries contain the values of ASMT(i,j) with ASMTFR(i,j) in parenthesis if the access is split.

INPUT

DMP = 4

LAMBDA = 0.001

NFILES = 7

NO:	FREQUENCY:	LENGTH:	MLENGTH:
1	0.29	1000.00	100.00
2	0.15	1000.00	100.00
3	0.14	1000.00	100.00
4	0.07	1000.00	100.00
5	0.09	1000.00	100.00
6	0.08	1000.00	100.00
7	0.18	1000.00	100.00

NDEVS = 4

NO:	MLT:	TT/W:	CAPACITY:
1	2000.00	5.00	3000.00
2	6000.00	20.00	1000.00
3	7500.00	10.00	10000.00
4	9000.00	10.00	10000.00

```
 AVERAGE MESSAGE LENGTH =  100.00

 COMPUTE-DEVICE-SERVICE-RATES OUTPUT:
 MU  = 0.40000E-03  0.12500E-03  0.11765E-03  0.10000E-03

 SOLVE-BPP OUTPUT:
 P*  = 0.76459E+00  0.95634E-01  0.83218E-01  0.56562E-01

 ASSIGN-FILES OUTPUT:
        1        0        0        0
        1        0        0        0
        0(.00)   1(.26)   0(.00)   1(.74)
        0        0        1        0
        0        0        0        1
        0(.00)   1(.74)   1(.17)   1(.10)
        1        0        0        0

DEVICE  2 CAPACITY EXCEED
CYCLIC ELIMINATION OF FILE 6
OPTIMALITY LOSS = 0.00

 NEW FILE ASSIGNMENT:
        1        0        0        0
        1        0        0        0
        0(.00)   1(.67)   0(.00)   1(.33)
        0        0        1        0
        0        0        0        1
        0(.00)   0(.00)   1(.17)   1(.83)
        1        0        0        0

 STORAGE MINIMIZATION
   ALLOCATE FILE  6  ACCESS
   FROM DEVICE    4
   TO   DEVICE    3
   NO LOSS IN OPTIMALITY

 FINAL FILE ASSIGNMENT
        1        0        0        0
        1        0        0        0
        0(.00)   1(.67)   0(.00)   1(.33)
        0        0        1        0
        0        0        0        1
        0        0        1        0
        1        0        0        0

TOTFREQ = 0.62000E+00  0.95634E-01  0.14962E+00  0.13475E+00
CPU UTIL =  0.381438E+00
STORAGE  =  0.800000E+03
```

3.4. Evaluating the heuristic

In order to evaluate the worth of a heuristic we must consider at least two things. We need to know if it offers computational advantages over an exact solution, and we need to know if it is accurate enough to be useable. In this section we evaluate the heuristic proposed in this chapter by these criteria.

Note that for an exact solution we must solve the large integer linear goal programming problem of Figure 3.3. These problems are known to be exponentially difficult [Taha76]. We thus seek to see if our methodology produces answers in polynomial time.

3.4.1. *Analysis of running times*

The thrust of this analysis is toward the heuristics developed for the Assign-Files portion of the algorithm. We do not address the problem of evaluating the Solve-BP portion of the solution. This is thoroughly discussed in [Dowd77]. Empirical results show Solve-BP to be quite tractable. For convenience we reproduce the basic Assign-files structure given in Section 3.3.4.

```
Match-BP's;
IF Any-capacity-exceeded
     THEN Obtain-capacity-optimal-solution;
Minimize-Storage-Used;
```

We will analyze first the worst case, asymptotic limits on the heuristic. As we will see, portions of the heuristic offer poor asymptotic limits. In these cases, we offer modifications that can be made to make the worst case time more tractable. Since worst case limits are not always a good indication of the practicality of a scheme, we also present some empirical evidence of what we could call "expected" case computational difficulty. We will see that the problem is quite tractable for the cases studied.

3.4.1.A. Worst case asymptotic limits

Here we will analyze each major portion of the heuristic outlined above. In some cases where the worst case bound is exponential, we will present alternate methods that are polynomially bounded.

MATCH-BP's

The Match-BP's portion is the linear programming problem given in Figure 3.4. As with all linear programming problems, the worst case bound is not very good. If we have m equations with m+n unknowns, we may have to consider $\binom{m+n}{m}$ solutions to obtain the optimum. From a more practical standpoint, practitioners consider even large (1000 constraint) linear programming problems to be tractable taking roughly O((# of constraints)^3) to compute [Wagn75].

OBTAIN-CAPACITY-OPTIMAL-SOLUTION

The Obtain-capacity-optimal-solution is composed of several major portions, we therefore reproduce its structure for convenience.

```
Classify-devices;
WHILE Any-capacities-violated DO
     BEGIN
          Device := Next;
          WHILE Capacity-violated(Device) DO
               BEGIN
                    Initial-reassignment;
                    Secondary-reassignments;
                    END;
     END
```

In this representation we explicitly show the initial device classification step. The Classify-devices procedure can compute the classifications for the devices by computing the TOTFREQ values taking O(NDEVS*NFILES) operations. We mentioned earlier that we must reclassify the devices after each reassignment. However, once the initial sets are determined, reclassification will only involve a constant amount of time per assignment. This is because we reclassify at most 2 devices per reassignment. We thus absorb this cost into the cost of the reassignments.

We can see that the outer WHILE loop looks at O(NDEVS) devices to determine if they are capacity feasible. The

inner WHILE loop need look at no more that O(NFILES) files to run through all the possible ways to have a device be capacity infeasible. So far, we have a basic structure that takes polynomial time without having to resort to anything exotic. The reassignment procedures require a more careful look.

The Initial reassignments are given in detail in Section 3.3.6.A. The operation of eliminating files that cannot fit on a device obviously can take no more that O(NFILES) operations (step 1). Likewise, this is the bound for finding the smallest f(i,j) (step 2c) or smallest shared f(i,j) (step 2b). The other operation (step 2a) deals with tracing file copy paths or file copy cycles and requires a more detailed look.

The analogy between the file copy matrix and a graph of nodes and edges is readily apparent. If we make each proper FCM entry, (f(i,j) with $0 < f(i,j) < 1$), a node and connect two nodes if they occupy the same row or column, we have formed a supergraph that contains the relations defining file copy paths and cycles. While some of the paths and cycles of this graph do not meet the requirements for being FCP's or FCC's, the supergraph certainly contains all FCP's and FCC's. In the worst case then, we could have a FCM with all nonzero entries and the corresponding supergraph would have NFILES*NDEVS nodes. The worst case graph would also be totally connected. The problem then of finding all cycles

would be O(NFILES*NDEVS!), which is not considered tractable. However, we can offer a modification to the cycle and path tracing algorithms that can yield a polynomial time bound by limiting the number of paths or cycles that we evaluate. We can do this by limiting ourselves to considering the m shortest cycles (or paths) instead of all of them. This problem has an asymptotic bound that is polynomial and is given by $O(m*(NDEVS*NFILES)^3)$ [Horo76]. Thus, step 2a is the computational bolltleneck for the Initial-reassignment step.

For the Secondary assignments, we may again modify the path tracing portion to include only the m shortest paths. This yields the same bound as above. The one-for-one swaps are relatively inexpensive. We seek to find an f(i+,j+) > f(i-,j-) such that TOTLEN(j+) - LEN(i+) + LEN(i-) <= CAP(j+) and TOTLEN(j-) - LEN(i-) + LEN(i+) <= CAP(j-). There are at most NFILES*NDEVS f(i,j) entries. Thus, to compare each one against the others takes at most $O((NFILES*NDEVS)^2)$ operations. Again the path tracing asymptotic time dominates.

We will now put all the individual portions together to yield an overall bound. We will make the assumption that we have limited the path and cycle tracing portions to consider only the m shortest paths. Thus, the asymptotic limit for the Obtain-capacity-optimal-solution heuristic is given by:

$O(NDEVS*NFILES + NDEVS * NFILES * m*(NDEVS*NFILES)^3)) =$

$O(NDEVS*NFILES + m * (NDEVS*NFILES)^4)$ time.

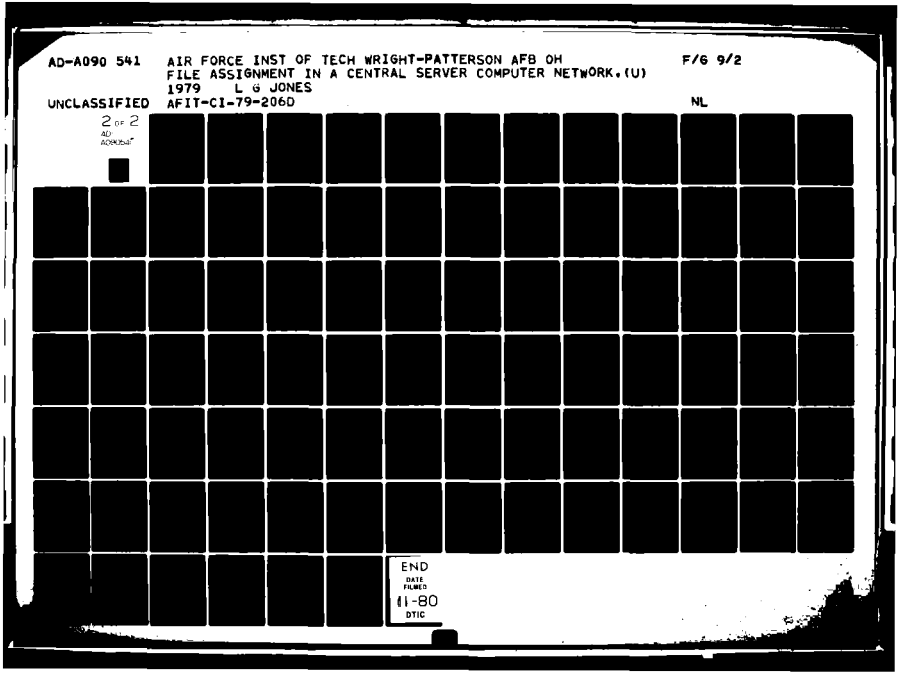
MINIMIZE STORAGE

Storage minimization is only involved with traversing FCP's and FCC's. Thus, the discussion and analysis for the Initial-reassignment step also apply here. The Storage minimization procedure thus is bounded by $O(m*(NDEVS*NFILES)^3)$ time.

In summary, we see that the Match-BP's portion of the algorithm is the theoretically limiting part of the heuristic. In practice, we know that linear programming problems are not usually considered intractable except for extemely large problems. If we then concern ourselves with practical matters, we need to determine what to expect from the Obtain-capacity-optimal-solution portion when solving actual problems. This is what we deal with in the next section.

3.4.1.B. Expected case running times

In the previous section we analyzed the worst case running times for the Assign-files heuristic. Here we take an empirical approach to give a practical feel for how the heuristic performs. As already stated, practitioners generally regard linear programming problems as tractable. We therefore will consider how the FCC and FCP portions of the heuristic perform on test cases.


AD-A090 541
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH
F/6 9/2
FILE ASSIGNMENT IN A CENTRAL SERVER COMPUTER NETWORK.(U)
1979 L G JONES
UNCLASSIFIED
AFIT-CI-79-206D
NL
2 OF 2
AD A090541
END
DATE FILMED
11-80
DTIC

It should be obvious that the computational difficulty of the FCP and FCC operations is directly related to the size of the associated graph. The size of the associated graph is in turn determined by the structure of the file copy matrix produced by Match-BP's. A graph node exists when we have proper f(i,j) entries, meaning that the access is split. We will refer to this as file fragmentation. If there are a lot of fragmented entries in the FCM, then the graph is large. The converse is also true. Therefore, a measure of the likely number of fragmented entries in the FCM, would be indicative of how long it takes to locate all cycles and paths.

We present the results of an empirical study of file fragmentation involving over 100 test cases. The problems considered included a ranges of parameter values. Problems of various sizes (NDEVS x NFILES) were tried. There was also an assortment of capacity constraint conditions (constraints active vs constraints inactive) and service rate ratios. The results are summarized in Table 3.1. The maximum number of nodes is computed as NFILES*NDEVS. The maximum number of edges is given as $\binom{\#\ \text{nodes}}{2}$ [Bond76].

Problem size	Max # nodes/edges	Avg # nodes/edges
3 x 10	30 / 435	1 / .5
4 x 7	28 / 328	5 / 4
4 x 8	32 / 496	4 / 3

TABLE 3.1. Summary of fragmentation study.

Based on these results, we can say that the actual file fragmentation encountered on the test cases is much less than the worst case. If we can extend these results to other cases, we can expect to be able to investigate all (or nearly all) of the cycles and paths. This would allow more flexibility in finding a good solution.

3.4.2. Accuracy of the heuristic

In addition to computation time, we must also consider the accuracy of the answers produced. Here, we provide two primary measures. First, we present an algorithm similar to those used in practice. Next, we provide methods of computing upper bounds and approximate upper bounds on the solution. Finally, we present some test cases to evaluate the heuristic against these measures empirically.

3.4.2.A. The "Industry" Algorithm

In this section we present an algorithm that will be used as a point of comparison for our heuristic. We call this algorithm the "industry algorithm" because it is

similar to file assignment algorithms that are used in practice. We will first present an intuitive look at the algorithm and give evidence that it is reasonably similar to the methods used by practictioners. We then present the algorithm formally and provide an example of its use.

Roughly stated, the industry algorithm is to load the most frequently used files on the faster devices subject to capacity restrictions. We may refine this by also providing the optimal branching probabilities and then loading the files subject to both capacity and the P*'s. This rough statement is a synthesis of the methods described in [Hugh73], [Fost76], and [Lips77]. The assignments generated will be limited to the integral storage method because this is the literature standard. We also disallow any rearrangement of the files after the assignment is made. However, we actually strengthen the algorithm by furnishing the P*'s as determined in [Fost77]. We therefore believe we are being realistic in using this algorithm as a benchmark.

We now present the algorithm in pidgin ALGOL.

```
Sort files by FREQ in decreasing order;
Calculate overall average message length;
Calculate P*'s;
Sort devices by P* in decreasing order;
Totlen(j) := Totfreq(j) := 0 for all j;
i := 1;
For j := 1 to NDEVS DO
     WHILE Totlen(j)+LEN(i) < CAP(j) AND
           Totfreq(j)+FREQ(i) < P*(j)  DO
          BEGIN
               Assign file i to device j;
               Totfreq(j) := Totfreq(j)+FREQ(i);
               Totlen(j)  := Totlen(j) +LEN(i);
               i := i + 1;
          END;
```

We now apply the algorithm to a simple example.

FILE	FREQ	LEN
A	.4	100
B	.2	100
C	.2	100
D	.2	100

DEVICE	P*	CAP
1	.7	300
2	.3	500

The assignment :

DEVICE 1	DEVICE 2
Files A,B	Files C,D

3.4.2.B. Upper bounds

It is difficult to obtain a non-trivial guaranteed upper bound for this problem. This is due in part to the fact that there is no known closed form solution for optimal CPU utilization as a function of DMP, P*'s, and MU's. Things are further complicated by the assignment dependence of the service rates, and the inclusion of device capacity constraints in our formulation. We can however present an algorithm to compute an upper bound on the problem. There are cases where this bound will be exact. However, for a given problem the bound may be somewhat loose. Therefore, we also provide method for obtaining an estimated upper bound. This estimated upper bound is heuristic in nature and may not truly bound the solution, but empirical results show that it can sometimes be exact and it seems to be a useful measure.

To help in establishing a bound, we have two significant facts. First, CPU utilization is a nondecreasing function of the device service rates for a fixed CPU service rate. Second, the service rates are assignment dependent, and with integral storage, there is a finite number of possible sets of service rates.

We may use the above facts to establish a simple upper bound on utilization as follows. First, we compute the upper bound service rates using the average message length to be MIN(MLEN(i)) for all i. This guarantees that all

devices will be as fast as is possible (maybe even faster) for a given set of files. We then compute the optimal P*'s and the corresponding CPU utilization using these upper bound service rates. We are thus assured that this value for CPU utilization is at least as good as any other assignment using the same set of files since any "real" assignment may not have any faster service rates, and may have a poorer match of the P*'s due to capacity constraints. However, with a little more work we may improve on this upper bound.

As we hinted above, we will relax some of the problem constraints to allow a tractable computation of an upper bound. Firstly, we will ignore device capacity contraints. Secondly, we will allow physical file splitting. These relaxations enable us to avoid the combinatorial difficulties that are otherwise present. We then estimate the shortest "realizable" average message length for the fastest device, j_F, and use this value for the AML of all devices.

To illustrate the above ideas, consider the following. Assume we use $AML(j_F) = MIN(MLEN(i))$ for all i as previously suggested, and designate all files that have MLEN(i) = MIN(MLEN(i)) to be in set I. Then assume that based on this AML, $P^*(j_F) = .6$. Consider what happens if the sum of the FREQ(i) for all i in I is less than 0.6. We are faced with the situation where in order to achieve a

TOTFREQ(j_F) = P*(j_F) we must include some files in the assignment to j_F with MLEN(i) > Min(MLEN(i)). This will result in an AML(j_F) > MIN(MLEN(i)). Thus, the service rate based on the assumption of minimum AML is said to be "unrealizable" for the given set of files. The actual service rate will be slower and the P*(j_F) is likely to be less.

We can achieve a realizable service rate for j_F in a relatively simple way. First, we sort the files in increasing order of MLEN, and sort the devices in decreasing order of service rate (using the overall average message length to compute service rates). We then deal out the files to the devices in order, until a device j has TOTFREQ(j) = P*(j), splitting the file if necessary. From this assignment we compute new services rates and P*'s and repeat the process until it converges. We then use AML(j) = AML(j_F) for all j, and we are thus guaranteed that no actual assignment can realize any shorter AML and hence any faster MU's.

One shortcoming of this method is that we were unable to prove convergence for this process. However, in all cases investigated the process always converged and converged fairly rapidly (5 or fewer iterations). Despite this shortcoming, we justify the use of this method by the superior bound it computes. If, however, a case occurs that does not converge, we can always take a fall-back position by assuming that AML(j) = MIN(MLEN(i)).

We now give a pidgin ALGOL representation of the algorithm.

```
Comment: Upper Bound Algorithm;
Sort files by MLEN in increasing order;
Sort devices by MU in decreasing order
     (using overall AML);
UNTIL no change in AML's DO
     BEGIN
          Compute P*'s;
          Assign files to devices in order until
             TOTFREQ(j) = P*(j) (split if necessary);
          Recompute AML's based on assignment;
     END;
For j := 2 to NDEVS DO
     AML(j) := AML(1);
```

The final step of assigning all devices the shortest, and possibly (for the slower devices) unrealizable, message length insures that no realizable assignment will have any faster service rates. Thus we have a bound on utilization as a function of both branching probabilities and service rates.

While the above algorithm indeed provides an upper bound, case studies show that this may be a very high upper bound. This motivates us to suggest a slight modification to the previous algorithm to produce an "approximate" upper

bound. To do this we simply eliminate the final step of making all AML's = AML(j_F). We are now using realizable service rates given that we are allowed to split files and ignore capacity constraints.

This approximate upper bound has been seen empirically to be an exact upper bound in cases where the service rates are "sufficiently" different, i. e., MU(1) $>_s$ MU(2) $>_s$ ... $>_s$ MU(NDEVS), where $>_s$ denotes "sufficiently greater than". While we cannot guarantee this bound to be an upper bound, case studies indicate that it appears to be a more realistic upper bound than the guaranteed bound. Obvicusly, when all message lengths are the same, the approximate upper bound and the guaranteed upper bound will be equal to the exact upper bound.

As a final bounding measure, we point out that we derived another kind of upper bound in Section 3.3.4 when we solved the linear programming problem, Match-BP's. This provides a bound on utilization for a given (not necessarily optimal) set of service rates assuming assignment independence. Since assignment independence of service rates is not usually true, this answer is only guaranteed to bound the solution if all average message lengths are the same. This solution recognizes physical capacity constraints, but allows physical file splitting. The measures of the heuristic's potential optimality loss against this bound were pointed out in Section 3.3.6. The

use of this bound in practical situations may prove to be of some value so we include it here.

3.4.2.C. Some Empirical Results

Here we present the results of 12 different experiments that cover a broad range of cases. We do not attempt to attach statistical significance to the findings, but we do believe the results offer insight into the problem. The experiments were factored as follows:

1. Two levels of DMP were used: 4 and 10. These were chosen to provide a low and high value. The DMP significantly affects the distribution of the P*'s.

2. Two different ratios of average mean latency time to average transfer time were used (Avg. MLT/ Avg. TT). For one choice the ratio was approximately 2, and for the other choice the Avg. MLT is 0. These levels were chosen to test the validity of ignoring the assignment dependence of the service rates. Obviously, as Avg. MLT becomes >> Avg. TT, this is a more accurate assumption. Thus, we are testing both the worst case (MLT=0) and a mid-range case.

3. Three different levels of device capacity activity were chosen:

a. Device constraints very significant;

b. Device constraints moderately significant;

c. Device constraints not significant.

Since our methodology is designed to recognize capacity constraints, these influences bear testing.

The constant factors in the experiments were the topology, the CPU burst rate, and the Avg. MU/LAMBDA ratio. The network topology was fixed at three I/O devices. The mean CPU burst rate, LAMBDA, was held at .001. Finally, the ratio of Avg. MU/LAMBDA was held at approximately .2. This was to insure an I/O bound situation.

To assist in identifying the properties of each experiment, we identify each experiment with a three character code. These codes are given in Figure 3.8. The actual data used is summarized in Figures 3.9 through 3.12. The results of the experiments are summarized in Table 3.2.

```
Character 1, Capacity constraint code.
     V means capacity constraints very significant
     M means capacity constraints moderately significant
     N means capacity constraints not significant

Character 2, DMP code.
     F means DMP = 4
     T means DMP = 10

Character 3, Avg. MLT/ Avg. TT ratio
     0 means Avg. MLT = 0
     2 means ratio    = 2
```

Fig. 3.8. Experiment naming convention.

Character code 1 = V, Capacity constraints
very significant

DEVICE	CAPACITY:
1	2300.00
2	5400.00
3	7800.00

FILE	FREQ	MSG. LEN	LENGTH
1	0.23	120.00	1500.00
2	0.12	110.00	1400.00
3	0.10	100.00	1300.00
4	0.09	90.00	900.00
5	0.08	80.00	700.00
6	0.08	70.00	500.00
7	0.07	60.00	300.00
8	0.06	50.00	250.00
9	0.05	40.00	200.00
10	0.05	30.00	150.00
11	0.04	20.00	100.00
12	0.03	10.00	50.00

Fig. 3.9. File and capacity data for code V.

Character 1 code = M, capacity constraints
moderately signficant

DEVICE	CAPACITY
1	2300.00
2	5400.00
3	7800.00

FILE	FREQ	MSG. LEN	LENGTH
1	0.23	120.00	1200.00
2	0.12	110.00	1100.00
3	0.10	100.00	1000.00
4	0.09	90.00	900.00
5	0.08	80.00	700.00
6	0.08	70.00	500.00
7	0.07	60.00	300.00
8	0.06	50.00	250.00
9	0.05	40.00	200.00
10	0.05	30.00	150.00
11	0.04	20.00	100.00
12	0.03	10.00	50.00

Fig. 3.10. File and capacity data for code M.

Character 1 code = N, capacity constraints not significant.

DEVICE	CAPACITY
1	4200.00
2	7000.00
3	7800.00

FILE	FREQ	MSG. LEN	LENGTH
1	0.23	120.00	1200.00
2	0.12	110.00	1100.00
3	0.10	100.00	1000.00
4	0.09	90.00	900.00
5	0.08	80.00	700.00
6	0.08	70.00	500.00
7	0.07	60.00	300.00
8	0.06	50.00	250.00
9	0.05	40.00	200.00
10	0.05	30.00	150.00
11	0.04	20.00	100.00
12	0.03	10.00	50.00

Fig. 3.11. File and capacity data for code N.

Character 3 code = 0

DEVICE	MLT:	TT/W:
1	0.00	28.00
2	0.00	50.00
3	0.00	100.00

Character 3 code = 2

DEVICE	MLT:	TT/W:
1	1900.00	5.00
2	2850.00	10.00
3	5700.00	20.00

Fig. 3.12. Avg. MLT/ Avg. TT data.

Abbreviations:
ID: Experiment identification code
GUB: Guaranteed upper bound
AUB: Approximate upper bound
IND: Industrial algorithm
SS: Split storage algorithm
SA: Integral storage, split access algorithm.

Table entries are CPU utilizations.

ID	GUB	AUB	IND	SS	SA
VF0	.5535	.5057	.3594	.3283	.3999
VF2	.5448	.5256	.3078	.4506	.4577
VT0	.7524	.6487	.4155	.4008	.5257
VT2	.7133	.6777	.3958	.6058	.6176
MF0	.5812	.5228	.4345	.3943	.4569
MF2	.5448	.5256	.4196	.4893	.4885
MT0	.7524	.6487	.6096	.5262	.5978
MT2	.7133	.6777	.5042	.6708	.6695
NF0	.5535	.5057	.5106	.5021	.5065
NF2	.5448	.5256	.5264	.5254	.5261
NT0	.7524	.6487	.5986	.5712	.5712
NT2	.7133	.6777	.6825	.6727	.6727

TABLE 3.2. Experimental results.

While we recognize that this is a small sample on which to base conclusions, nonetheless, we believe that it is worthwhile to point out some trends. Throughout the following discussion, the numerical values given are CPU utilization figures. For example, if we say method A is

superior to method B by .1146, we mean that the CPU utilization obtained using method A minus the CPU utilization obtained using method B equals 11.46%.

First, we consider the upper bounds. The approximate upper bound seemed to do a reasonable job. It bounded all the solutions except cases NF0, NF2, and NT2, and in these cases it was only very slightly exceeded (average = .002). The average difference between the AUB and the GUB was about 5%, supporting the premise that the GUB may be unrealizable and somewhat high for many cases. We now consider the performance of the SA algorithm detailed in this work.

Compared against the simple industrial algorithm, the SA algorithm demonstrated better performance. It is instructive to note that the SA algorithm was superior (by over 11%) in cases where the storage constraints were relatively active. However, in the cases where storage constraints were relatively inactive, the industrial algorithm was as good or better (usually within 1%). This would suggest that while the SA algorithm is better for a wider range of cases, the industrial algorithm might have significant practical value in cases where storage capacities are not a problem.

Interesting results are obtained when we compare the SA algorithm to the SS algorithm. Recall that the SS answers bound the SA answers when service rates are assignment independent. This is not the case in these experiments

since the MLEN's are all different. Surprisingly, the SA algorithm outperformed the SS algorithm in most of these tests (average of about 4% better). This demonstrates that the loss of goal 3 optimality (detailed in Section 3.3.6) may not be a critical factor when we have a significant variation in the average message lengths.

In our final utilization comparisons, we compare the SA algorithm results to the upper bounds. Compared to the GUB results, the SA results are about 9% lower. This is not bad but not astoundingly good. If we accept the premise that the GUB results are usually too high, perhaps the AUB is a better yardstick. In this case the SA algorithm compares quite favorably, being within about 4 1/2% lower on the average. Practitioners would consider this to be a good result.

A secondary measure of effectiveness of the SA algorithm is the total storage required. The IND and SS algorithms, of course, require only the lower bound amount of storage since they do not replicate files. For code V experiments this lower bound is 7350 words and for code M or N experiments the value is 6450. The upper bounds are 10350 and 8850 for V and M/N respectively. The results for the individual experiments are given in Table 3.3. The "% Extra" column is computed as

(Total storage - lower bound)/(Upper bound - lower bound).

This indicates how close an answer is to the worst case. We see that when a replicated assignment results, the average extra storage is approximately 68% of the worst case. We believe that this high value is due to the fact that file length is proportionate to access frequency for these experiments. Thus, the more likely variables to be split are also the ones with the greatest lengths. A different relation between FREQ and LEN would probably result in less storage required.

ID	Total Storage	% Extra
VF0	7350	0
VF2	7350	0
VT0	7350	0
VT2	7350	0
MF0	6450	0
MF2	7450	42%
MT0	6450	0
MT2	7450	42%
NF0	7650	50%
NF2	8550	87%
NT0	8750	96%
NT2	8650	92%

TABLE 3.3. Storage require by SA assignments.

At this point it is appropriate to make some general statements regarding the use of the SA algorithm. It is apparent that we need to compare the IND and SA algorithms in terms of accuracy versus execution time, complexity, and size of the algorithms. Obviously, the SA algorithm is bigger, more complex, and requires more execution time than

the IND algorithm. In terms of accuracy, the SA algorithm is better considering all cases, but when storage constraints are not active, the IND algorithm seems to perform about as well.

To try to summarize the trade-offs, we will construct what appear to be the ideal scenarios for the use of these algorithms. We must recognize that for conditions that do not match these ideals, there is no clear cut answer as to which to use.

The IND algorithm is short and simple. It's use would be appropriate in a situation where storage constraints were not a significant problem. It would be more attractive in an environment where the access profiles changed frequently and it would be desirable to execute the file assignment algorithm frequently.

The SA algorithm is somewhat lengthy and complex. It seems to perform best when storage constraints are a significant consideration. It would be appropriate to use in an environment where access profiles changed infrequently, and new file assignments were not needed very often, say every new work shift.

In conclusion, we submit that although the number of experiments is relatively small, they were carefully chosen, and the results are encouraging. We tested our algorithm under conditions that lie outside its basic assumptions, and it proved to be reasonably robust and was found to yield

good answers. While it is dangerous to generalize based on a small sample, this study gives some evidence that the method could be considered useful.

3.5. Chapter summary

In this chapter we dealt with the file assignment problem assuming that the files were read-only files. This allowed us to replicate files without incurring any performance degradation due to multiple-copy updates. The requirement for file replication was justified on a system performance basis and was implemented with the concept of integral storage with split access.

We presented an exact formulation for the problem as a linear integer goal programming problem. Since this type of problem is generally costly to solve, we introduced a heuristic solution technique. We were able to bound the storage required by the technique and to present empirical evidence to sugggest that it obtains reasonably good answers.

The basic ideas introduced in this Chapter will be extended in Chapter 4 to include read-write files. In that case, we will be forced to consider the possibility that multiple-copy updates may indeed degrade performance.

CHAPTER IV

MODELS FOR READ-WRITE FILES

4.1. Introduction

In the previous chapter, we used the simplifying assumption that the files were read-only files. This is a common simplification in the literature, but it is obviously not completely realistic. In this chapter we further generalize the model to account for read-write files. Initially, we introduce the basic concepts. The main ideas include update overhead and assignment dependent access frequencies. After this, we explore several models. First, we investigate read-write files in the non-replicated case. Next, we allow replication and develop a model (and variations) that attempts to reduce update overhead. Then, we formulate a model that recognizes the basic inseparability of the P*-matching and overhead minimization goals. Finally, we examine the accuracy of using the simpler read-only model solution technique of Chapter III to approximate the read-write model solution.

4.2. The impact of adding update messages

Before proceeding we need to extablish some basic terminology. We will use the terms "read" and "query" interchangeably. Likewise, "write" and "update" will be

considered equivalent. Finally, "access" to a file may represent either a read(query) or a write(update).

The addition of update messages to the read-only model complicates the model. This is most obvious when we consider file assignments that allow replication of files. When we deal with read-only files, the proliferation of file copies primarily impacts on the total storage used by the system. A query message goes only to a single file copy and only modest overhead is incurred in the directory look up process. However, when we are dealing with read-write files, an update message may, in the worst case, have to be sent to all copies of a replicated file each time an update occurs. This is because of data consistency requirements.

While we recognize that there may be clever schemes devised to reduce the impact of updates to multiple copies, we will assume that an update message always goes to all copies. This worst case assumption provides an upper bound for any updating scheme and frees the analysis from any associated assumptions.

The differences between the read-only and read-write models may manifest themselves in at least two ways. First, we may treat multiple update messages as a redistribution of the file access frequencies. Second, we may treat extra update traffic as overhead that degrades performance. We will address these ideas in the following two sections.

4.2.1. Redistribution of file access frequencies

First, recall that we assume an update goes to all copies. Based on this assumption, we may show that file access frequency is not constant, as in the read-only case, but is a function of the file assignment. Define QlFREQ(i) as the estimated or empirically determined query frequency for a copy of file i in an assignment with no file replication. Likewise, define UlFREQ(i) as the update frequency for a copy of file i in an assignment with no file replication. Since these are normalized values, $\sum_{i=1}^{NFILES}$ (QlFREQ(i) + UlFREQ(i)) = 1.
We then define QnFREQ(i) and UnFREQ(i) as the query and update frequencies, respectively, for any copy of file i for an assignment allowing file replications. We then present the following equations:

(4.1) QnFREQ(i) = QlFREQ(i)/DENOM

(4.2) UnFREQ(i) = UlFREQ(i)/DENOM

where

(4.3) $$DENOM = \sum_{i=1}^{NFILES} QlFREQ(i) + \sum_{i=1}^{NFILES} \sum_{j=1}^{NDEVS} ASMT(i,j)*UlFREQ(i)$$

and as before

ASMT(i,j) = 1 if a copy of file i is on device j,

0 otherwise.

DENOM represents all queries and updates for all files. $\sum_{j=1}^{NDEVS}$ ASMT(i,j) equals the number of copies of file i and is included in DENOM to reflect the fact that an update goes to all copies of a replicated file. DENOM normalizes the frequencies such that

$$\sum_{i=1}^{NFILES} \left(UnFREQ(i) * \sum_{j=1}^{NDEVS} ASMT(i,j) + QnFREQ(i) \right) = 1.$$

Also note that these functions are piecewise linear since ASMT(i,j) is a (0,1) variable.

File	Update frequency	Query frequency
A	.6	0.0
B	.4	0.0

	ASSIGNMENT 1		ASSIGNMENT 2		ASSIGNMENT 3		ASSIGNMENT 4	
	Device 1	Device 2	Device 1	Device 2	Device 1	Device 2	Device 1	Device 2
Files	A	B	A,B	B	A	A,B	A,B	A,B
Totfreq	.6	.4	.71	.29	.37	.63	.5	.5

Fig. 4.1. Effects of assignment on access frequency.

Consider the example of Figure 4.1, in which we have two devices and two write-only files with the given single copy update frequencies. Define the total frequency assigned to the devices as

$$Totfreq(j) = \sum_{i=1}^{NFILES} ASMT(i,j)*(UnFREQ(i)+QnFREQ(i)),$$

and compare assignments 1 and 2. Notice that the replication of file B causes a redistribution of file accesses. This is because in Assignment 2, every update to file B now generates two messages.

We now demonstrate the computation of some of the Totfreq values of Figure 4.1. In Assignment 1, there is no file replication, therefore, DENOM = 1.0, and thus,

UnFREQ(A) = U1FREQ(A) = Totfreq(1), and

UnFREQ(B) = U1FREQ(B) = Totfreq(2).

This results from the fact that no files are replicated. In Assignment 2, however,

DENOM = (0 + 0) + 1*.6 + 0*.6 + 1*.4 + 1*.4 = 1.4,

UnFREQ(A) = .6/1.4 = .42,

UnFREQ(B) = .4/1.4 = .29,

Totfreq(1) = UnFREQ(A) + UnFREQ(B) = .71, and

Totfreq(2) = UnFREQ(B) = .29.

The computations for Assignments 3 and 4 are similar.

This redistribution of access frequencies is only one side effect of updates to replicated files. In particular note that unlike the read-only model, we may not always be able to match the P*(j)'s exactly even if we ignore storage constraints. Consider Figure 4.1 again. If P*(1) = .52 and P*(2) = .48, Assignment 4 yields the closest possible match. This is because split file access, which is possible for queries, is not possible for updates, because a single update message must go to all copies. A disturbing point about choosing Assignment 4 is that, intuitively, we would expect some performance degradation due to the increased traffic caused by multiple update messages. We address the problem of representing this increased overhead in the next section.

4.2.2. Update Overhead Traffic

We assume that file accesses are the dominant activity in our model. So for a given degree of multiprogramming (DMP), we may compute the average number of jobs accessessing a given copy of a file, i, as DMP*FREQ(i), where FREQ(i) = UnFREQ(i) + QnFREQ(i). If we consider update to more than one copy of a file as unnecessary overhead, then define DMP' to be "useful" DMP, or the average number of jobs not involved in extra update.

Define UTFREQ(i) to be the total update frequency for all copies of i. We can see that

$$\text{UTFREQ}(i) = \text{UnFREQ}(i) * \sum_{j=1}^{\text{NDEVS}} \text{ASMT}(i,j) ,$$

also

UnFREQ(i) * DMP = the average number of jobs updating a single copy of i in a given assignment;

UTFREQ(i) * DMP = the average number of jobs updating all copies of i in the same assignment.

Then (UTFREQ(i)-UnFREQ(i)) * DMP = average number of update overhead jobs for file i.

Then define

$$\text{OHEAD} = \sum_{i=1}^{\text{NFILES}} (\text{UTFREQ}(i) - \text{UnFREQ}(i)) * \text{DMP},$$ the average number of update overhead jobs in the system.

We then see that DMP' = DMP - OHEAD.

This result has important implications with regard to performance because CPU utilization is a non-decreasing function of DMP [DOWD77]. Therefore, if any overhead is

incurred, there will be a decrease in the "useful" CPU throughput.

4.3. Solution Methodology for a Non-replicated, read-write FAP

In this section we solve for the case of read-write files with no replication. This is an important model to solve because this is compatable with the file storage system for most standard operating systems. First, we will derive simplified expressions for QnFREQ(i) and UnFREQ(i). We then provide an outline of an algorithm to obtain an exact solution and give some of its more important details.

4.3.1. Simplification of Access Frequency Expressions

Given that we do not allow replicated files, we may significantly simplify the piece-wise linear expressions for QnFREQ(i) and UnFREQ(i). Non-replication of files means that the number of copies of file i = 1 for all i. Then,

$$\sum_{j=1}^{NDEVS} ASMT(i,j) = 1,\ i = 1,\ldots,NFILES.$$

We can see that DENOM now reduces to unity:

$$DENOM = \sum_{i=1}^{NFILES} QlFREQ(i) + \sum_{i=1}^{NFILES} UlFREQ(i) * \sum_{j=1}^{NDEVS} ASMT(i,j)$$

$$= \sum_{i=1}^{NFILES} (QlFREQ(i) + UlFREQ(i)) = 1.$$

Thus we have

$$QnFREQ(i) = QlFREQ(i)\ ,\ \text{and}$$

$$UnFREQ(i) = UlFREQ(i)\ .$$

Happily, we are left with expressions for QnFREQ(i) and UnFREQ(i) that are independent of the file assignment.

4.3.2. Algorithm Outline

The high level structure of the algorithm is given below. The major steps are then described in more detail. Note that this is the same basic structure as for the read-only model in Chapter III.

(1) Read Input;

(2) Compute Service Rates;

(3) Solve BPP;

(4) Assign Files;

4.3.2.A. *Read Input*

The input data is given below in Figure 4.2. Note that except for the subdivision of access frequencies and message lengths, this is the same as for the read-only model.

4.3.2.B. Compute Service Rates

As in the read-only case, service rates are computed based on a frequency weighted average message length. In this case we have both query and update frequencies and message lengths. Specifically we may compute the average message length as

$$AML = \sum_{i=1}^{NFILES} QlFREQ(i)*QLEN(i)+UlFREQ(i)*ULEN(i).$$

We then compute the service rate, MU(j), for device j as

$$MU(j) = 1 / (MLT(j) + TT(j) * AML).$$

```
NFILES    :     Number of files
QlFREQ(i):     Query frequency for a single copy of file i.
UlFREQ(i):     Update frequency for a single copy of file i.
LEN(i)    :     Length (in words) of file i.
QLEN(i)   :     Average length of query message for file i.
ULEN(i)   :     Average length of update message for file i.

NDEVS     :     Number of storage devices.
CAP(j)    :     Capacity (in words) of device j.
MLT(j)    :     Mean latency time of device j.
TT(j)     :     Transfer time per word for device j.

LAMBDA    :     CPU speed.

DMP       :     Degree of multiprogramming.

The limits for i and j are:
     i = 1,...,NFILES, and  j = 1,...,NDEVS.
```

Fig. 4.2. Read-write model input data.

4.3.2.C Solve BPP

This procedure is exactly the same as in the read-only model. Recall that Solve-BPP computes the set of branching probabilities, P*(j)'s, that maximize CPU utilization using non-linear programming. As stated before, the answer obtained is a global optimum.

4.3.2.D. Assign Files

This procedure solves the integer programming problem given in Figure 4.3. Here, we may define FREQ(i) = QlFREQ(i) + UlFREQ(i) since the values for QnFREQ(i) and UnFREQ(i) are constant. We now have a formulation that is essentially the same model proposed in [Fost77] for read-only files. This is because the expressions for QnFREQ(i) and UnFREQ(i) were only assignment dependent if

the number of file copies is not fixed. The objective is to minimize the sum of the differences between the assigned frequencies and the optimal branching probabilities. Constraint 1 insures that the device capacities are not exceeded. Constraint 2 insures that all files are assigned once. Constraint 3 establishes that the assignment variables are 0,1 variables. Note that except for the integer variables, this formulation is essentially the same as the Match-BP's model of Figure 3.4.

$$\text{Minimize:} \quad \sum_{j=1}^{NDEVS} \sum_{i=1}^{NFILES} ABS(ASMT(i,j) * FREQ(i) - P*(j))$$

Subject to:

$$(1) \quad \sum_{i=1}^{NFILES} ASMT(i,j) * LEN(i) <= CAP(j) \qquad j=1,\ldots,NDEVS$$

$$(2) \quad \sum_{j=1}^{NDEVS} ASMT(i,j) = 1 \qquad i=1,\ldots,NFILES$$

$$(3) \quad ASMT(i,j) = (0,1) \qquad i=1,\ldots,NFILES,\ j=1,\ldots,NDEVS$$

Fig. 4.3. File assignment formulation.

4.3.3. Non-replicated FAP Example

We present a computer run of an example in Figure 4.4 to illustrate the model presented in this section. As in Chapter III, the file assignment matrix represents the allocation of the file (rows) to the devices (columns). A "1" represents that file i is assigned to device j.

```
INPUT

DMP      =  0.400000E+01
LAMBDA   =  0.100000E-02

NFILES   =     10
  NO   Q1FREQ   U1FREQ   QLENGTH   ULENGTH    LENGTH
   1    0.15     0.14     75.00     75.00     100.00
   2    0.09     0.06     25.00     25.00     100.00
   3    0.10     0.04     75.00     75.00     100.00
   4    0.04     0.03     75.00     75.00     100.00
   5    0.05     0.00     25.00      0.00     100.00
   6    0.04     0.00     25.00      0.00     100.00
   7    0.02     0.02     25.00     25.00     100.00
   8    0.04     0.00     25.00      0.00     100.00
   9    0.04     0.04     25.00     25.00     100.00
  10    0.06     0.04     25.00     25.00     100.00

NDEVS    =      3
  NO:        MLT:        TT/W:      CAPACITY:
   1         400.00       5.00        400.00
   2         500.00      10.00        400.00
   3       25500.00      10.00       1000.00

AVERAGE MESSAGE LENGTH =     50.00

SERVICE RATES:
MU   =  0.153846E-02   0.100000E-02   0.384615E-04

OPTIMAL BRANCHING PROBABILITIES:
P*   =  0.688281E+00   0.311719E+00   0.000000E+00

AN OPTIMAL FILE ASSIGNMENT:
     1     0     0
     1     0     0
     1     0     0
     0     1     0
     0     1     0
     0     0     1
     0     0     1
     0     1     0
     0     1     0
     1     0     0

CPU UTILIZATION           =  0.458534E+00

DEC 1099 (KL10) CPU TIME =  0:13.9  MIN:SEC
```

Fig. 4.4. Read-write non-replicated FAP example.

4.3.4. Comments on running time

The methodology just presented has obvious computational shortcomings. For large problems, the use of mixed integer programming techniques may well prove to be intractable. This difficulty also applies to the similar read-only technique in [Fost77]. Tractable heuristics are needed to solve large problems.

Based on the results in Chapter III, the industrial algorithm may be a good candidate. It provides integral answers and seems to be reasonably accurate if storage constraints are not overly active. Other bin packing techniques that can be applicable to this problem are given in [Fost76]. Yet another possibility is to solve the corresponding linear programming problem (i. e., drop contraint 3 in Figure 4.3) and round the answer. This is similar to the method used in [Chand76] and [Jone79a].

4.4. Models to minimize update overhead for replicated read-write file assignments

In this section we present our first models for read-write files with replication. In these models we use a goal programming formulation that is very similar to the one for the read-only model. However, it additionally includes the objective of minimizing update overhead. With this formulation, we present several possibilities for the arrangement of the achievement function. We do not propose

any solution techniques in this section, but defer this to Section 4.6.

Since we know that update overhead degrades performance, we wish to minimize its effect. The expression for update overhead is given by

$$\sum_{i=1}^{NFILES} \left(\sum_{j=1}^{NDEVS} ASMT(i,j) - 1 \right) * UnFREQ(i) * DMP .$$

Unfortunately, this is a nonlinear expression. However, note that minimizing the linear expression

$$\sum_{i=1}^{NFILES} \sum_{j=1}^{NDEVS} ASMT(i,j) * UlFREQ(i)$$

will also minimize update overhead. We can then add the following objective to the read-write formulation:

$$(h): \quad \sum_{i=1}^{NFILES} \sum_{j=1}^{NDEVS} ASMT(i,j) * UlFREQ(i) + n(6)i - p(6)i = 0 \quad \text{for all } i.$$

We must also change the P* matching objective (d) to account for the query and update frequencies. The expression we wish to minimize is

$$\sum_{i=1}^{NFILES} (ASMTFR(i,j)*QnFREQ(i)+ASMT(i,j)*UnFREQ(i)) - P*(j)$$

for all j. Unfortunately, we were unable to derive a simple linear expression that represents this nonlinear expression. In general, we can convert this problem to a (0,1) linear programming problem by greatly expanding the size of the problem [Chu73]. However, since we do not propose to obtain an exact solution to this computationally difficult problem, such a problem transformation would only serve to confuse the formulation. We choose to leave the problem in the present form for clarity.

This gives us the objectives in Figure 4.5 for our read-write goal programming formulation.

Objectives:

$$\text{(a):} \quad \sum_{j=1}^{NDEVS} ASMTFR(i,j) + n(1)i - p(1)i = 1 \qquad \text{for all } i$$

$$\text{(b):} \quad ASMTFR(i,j) + n(2)ij - p(2)ij = ASMT(i,j) \qquad \text{for all } i \text{ for all } j$$

$$\text{(c):} \quad \sum_{i=1}^{NFILES} ASMT(i,j) * LEN(i) + n(3)j - p(3)j = CAP(j) \qquad \text{for all } j$$

$$\text{(d):} \quad \sum_{i=1}^{NFILES} (ASMTFR(i,j)*QnFREQ(i) + ASMT(i,j)*UnFREQ(i) - P^*(j) = 0 \qquad \text{for all } j$$

$$\text{(e):} \quad \sum_{i=1}^{NFILES} ASMT(i,j) * LEN(i) + n(5)j - p(5)j = 0 \qquad \text{for all } j$$

$$\text{(f):} \quad ASMT(i,j) = (0,1) \qquad \text{for all } i, j$$

$$\text{(g):} \quad ASMTFR(i,j) \geq 0 \qquad \text{for all } i, j$$

$$\text{(h):} \quad \sum_{i=1}^{NFILES} \sum_{j=1}^{NDEVS} ASMT(i,j) * UlFREQ(i) + n(6)i - p(6)i = 0 \qquad \text{for all } i.$$

Fig. 4.5. Goal programming objectives for read-write models.

To the achievement function we add the objective

[p(6)i]

in order to minimize the expression. We are now faced with several choices for the ordering of the objectives within the achievement function. We will consider three cases.

CASE 1

Achievement priorities:

(1) assign all files;

(2) stay within device storage capacities;

(3) meet the optimal branching probabilities;

(4) minimize update overhead;

(5) minimize total storage.

The achievement function becomes:

$$\text{Min } a=([\sum_{i=1}^{NFILES}(n(1)i + p(1)i) + \sum_{i=1}^{NFILES}\sum_{j=1}^{NDEVS}(p(2)ij)\ ],$$
$$[\sum_{j=1}^{NDEVS}(3)j],$$
$$[\sum_{j=1}^{NDEVS} n(4)j + p(4)j\ ],$$
$$[\sum_{i=1}^{NFILES} p(6)i\ ],$$
$$[\sum_{j=1}^{NDEVS}(5)j\ ]\ ) .$$

If the problem is formulated this way, it may not be possible to reduce storage to the proven levels of Chapter III. Consider Figure 4.1 again. If P*(1) = .5 and P*(2) = .5, the priority 3 level forces replication of both files. The lower priority goal to minimize overhead cannot change this without changing goal 3 optimality and total storage becomes > NFILES + NDEVS - 1. This is not a totally satisfying result. Furthermore the overhead incurred by giving P* matching priority over minmizing overhead might

result in an actual decrease in CPU utilization.

CASE 2

Achievement priorities:

(1) assign all files;

(2) stay within device storage capacities;

(3) meet the optimal branching probabilities;

(4) minimize total storage.

(5) minimize update overhead;

The difference between Case 2 and Case 1 is that the last two priorities are interchanged. Thus, the achievement function becomes:

$$\text{Min } a=([\sum_{i=1}^{NFILES}(n(1)i + p(1)i) + \sum_{i=1}^{NFILES}\sum_{j=1}^{NDEVS}(p(2)ij)\ ],$$
$$[\sum_{j=1}^{NDEVS} p(3)j],$$
$$[\sum_{j=1}^{NDEVS} n(4)j + p(4)j\ ],$$
$$[\sum_{j=1}^{NDEVS} p(5)j\ ],$$
$$[\sum_{i=1}^{NFILES} p(6)i\ ]\ )\ .$$

The same objections that apply to Case 1 apply here. Since the ultimate objective in all our models is optimum CPU throughput, we should prefer Case 1 over Case 2.

CASE 3

Achievement priorities

(1) assign all files;

(2) stay within device storage capacities;

(3) minimize update overhead;

(4) meet the optimal branching probabilities;

(5) minimize total storage.

Here we have given overhead minimization priority over P* matching and the achievement function becomes:

$$\text{Min } a=([\ \sum_{i=1}^{NFILES} (n(1)i + p(1)i) + \sum_{i=1}^{NFILES}\sum_{j=1}^{NDEVS} (p(2)ij)\ ],$$
$$[\ \sum_{j=1}^{NDEVS} p(3)j],$$
$$[\ \sum_{i=1}^{NFILES} p(6)i\ ],$$
$$[\ \sum_{j=1}^{NDEVS} n(4)j + p(4)j\ ],$$
$$[\ \sum_{j=1}^{NDEVS} p(5)j\ ]\ )\ .$$

This formulation will result in an answer that will only replicate files that have UlFREQ(i) = 0 (read-only files). In a manner similar to Case 1, a disadvantage is that the loss of CPU utilization by perhaps not matching the P*'s exactly may be more than if a small amount of overhead is incurred. However, an advantage is that the storage used will be within the bounds proven for the read only model.

To see this, recall that only the files that had split access become involved in the storage minimization process. In this model the only files that may have split access are the read only files (UlFREQ(i) = 0). Therefore, this model reduces to the read only model when it comes to cyclic reassignment. For further insight we can demonstrate that this model has the property that QnFREQ(i) = QlFREQ(i), and UnFREQ(i) = UlFREQ(i) for all i. To see this, consider the equations for QnFREQ(i) and UnFREQ(i):

QnFREQ(i) = QlFREQ(i)/DENOM, and

UnFREQ(i) = UlFREQ(i)/DENOM, where

$$\text{DENOM} = \sum_{i=1}^{\text{NFILES}} \text{QlFREQ}(i) + \sum_{i=1}^{\text{NFILES}} \sum_{j=1}^{\text{NDEVS}} \text{ASMT}(i,j) * \text{UlFREQ}(i).$$

Since all files are assigned at least once,

$$\sum_{j=1}^{\text{NDEVS}} \text{ASMT}(i,j) >= 1.$$

We also know that the only files that are replicated are those with UlFREQ(i) = 0, therefore if

$$\sum_{j=1}^{\text{NDEVS}} \text{ASMT}(i,j) > 1, \text{ then UlFREQ}(i) = 0.$$

Hence, $\sum_{i=1}^{\text{NFILES}} \sum_{j=1}^{\text{NDEVS}} \text{ASMT}(i,j) * \text{UlFREQ}(i) = \sum_{i=1}^{\text{NFILES}} \text{UlFREQ}(i).$

By definition

$$\sum_{i=1}^{\text{NFILES}} \text{QlFREQ}(i) + \sum_{i=1}^{\text{NFILES}} \text{UlFREQ}(i) = 1,$$

therefore,

QnFREQ(i) = QlFREQ(i) and UnFREQ(i) = UlFREQ(i) for all i.

This tells us that

FREQ(i) = QlFREQ(i) + UlFREQ(i)

is assignment independent. This is the property that allows the cyclic reassignment of files in the read only model.

If we can obtain P* matching and don't have to split read-write files, then we can have all the accuracy of the solution for the read-only model and require no more storage than previously proven. If a specific application has an adequate supply (in terms of access frequency) of read-only files, then this formulation has to be viewed as the best of the three cases presented.

4.5. Weighted objective function model

One objection to the formulations of the previous section stemmed from the fact that neither P* matching nor overhead minimization truly preempted one another. This is due to the fact that they both are contributors to optimum CPU throughput. Since the two objectives are not preemptive, perhaps a weighted objective function is appropriate.

We present the following achievement function to be used in conjunction with the objectives of Figure 4.5:

$$\text{Min } a=\Big(\Big[\sum_{i=1}^{NFILES}(n(1)i + p(1)i) + \sum_{i=1}^{NFILES}\sum_{j=1}^{NDEVS}(p(2)ij)\Big],$$
$$\Big[\sum_{j=1}^{NDEVS} p(3)j\Big],$$
$$\Big[\sum_{j=1}^{NDEVS} W1 * (n(4)j + p(4)j) + W2 * \Big(\sum_{i=1}^{NFILES} p(6)i\Big)\Big],$$
$$\Big[\sum_{i=1}^{NFILES} p(5)i\Big]\Big) \quad .$$

Here W1 is the weighting factor to be applied to the P* matching goal, and W2 is the weighting factor to be applied

to the overhead minimization goal. The values of W1 and W2 should be a function of several things which might include DMP, NFILES, NDEVS, MU's, LAMBDA, and the total update frequencies. The analysis in Section 4.6 should provide insight into possible choices for the weights. We do, however, note that the results of Section 4.6 indicate that P*-matching seems to be a most critical factor in determining CPU utilization.

We choose not to go into a detailed analysis of possible weights here because of the computational difficulty of obtaining an exact answer. We intend to solve the problem heuristically, and our approach will not utilize these weights. The approach we take, discussed in the next section, is to use the read-only solution already developed as a heuristic approximation to the read-write problem.

4.6. Read-only solution applied to the read-write model

In this section we evaluate the use of the read-only solution as an approximation to the read-write model. Dowdy [Dowd78] considered the read-only model of Chapter III to be a way to solve the read-write problem. The read-only model may be considered an approximation to the read-write model because of the storage minimization process. Storage minimization contributes to the reduction of update overhead by reducing the total number of replicated files. In fact, we saw in Section 4.3 that in the limiting case (i. e., no

replication), the two models are equivalent. In view of this insight and the fact that we have a tractable heuristic for the read-only case, it seems worthwhile to analyze how well the read-only solution approximates the read-write solution.

To convert a read-write problem to a read-only problem we perform the following calculations.

$$MLEN(i) = (QlFREQ(i)*QLEN(i) + UlFREQ(i)*ULEN(i)) / (QlFREQ(i) + UlFREQ(i)) \quad \text{for all } i.$$

$$FREQ(i) = QlFREQ(i) + UlFREQ(i) \quad \text{for all } i.$$

If we are to use the approximation intellegently, we must have some notion of the errors that may be introduced. We must account for two types of errors. The first is the error introduced by ignoring update overhead. The second is the error introduced by assuming assignment independent access frequencies. We will first deal with these errors in the worst case and then address some more likely cases for the errors.

4.6.1. Worst case errors

4.6.1.A. Update overhead error

In section 4.2.2 we introduced the concept of update overhead. If we use the read-only model to approximate the read-write model, we are ignoring any overhead that may be incurred. Here we will derive some bounds on the error. It will be seen that the worst case bounds are not very useful.

Therefore, we will provide graphs showing the amount of error introduced for some more likely cases that might be encountered.

The read-only model produces a solution that guarantees at most NDEVS-1 extra copies of files. Let the most frequently updated file be file i_U. Then the worst case for update overhead occurs when we have NDEVS-1 extra copies of i_U. Thus, in the worst case

OHEAD = (NDEVS - 1) * UnFREQ(i_U) * DMP.

Which we can expand to get

OHEAD = ((NDEVS-1) * UlFREQ(i_U) * DMP) /

(1 + (NDEVS-1) * UlFREQ(i_U)).

We note that NDEVS-1 may be as small as 0, in which case the lower bound on overhead is 0. NDEVS may also approach infinity, and in this case overhead approaches DMP (i. e., all activity is overhead activity). This limit is not terribly useful or interesting for practical situations.

Now let us consider what happens if we hold NDEVS to a "reasonable" finite value. For a lower bound, UlFREQ(i_U) can be as small as 0. In this case OHEAD = 0, and we have reduced the model to the read-only case. For an upper bound, UlFREQ(i_U) can be as large as 1. In this case we can compute overhead as

OHEAD = (NDEVS-1)*DMP / (1+NDEVS-1)

= DMP - (DMP/NDEVS).

This case, too, is not terribly likely. From a practical standpoint, it is more useful to consider empirical values in a more likely range of values for UlFREQ(i_U) and NDEVS.

In Figure 4.6, we plot DMP'/DMP to get worst case values for the percentage of error introduced. Each curve is labeled with the number of I/O devices corresponding to the curve. As an example of the use of the Figure, we may see that for a system with six I/O devices, if the largest value of UlFREQ(i) = .05, then in the worst case, useful DMP is approximately 80% of the full DMP. Keep in mind that the worst case is realized only when file i_U is replicated the maximum number of times. We see that when either NDEVS or UlFREQ(i_U) is relatively small, the accuracy is reasonable. However, the DMP'/DMP error figures are not the final word. The ultimate issue is how this impacts on CPU utilization.

To assess the impact of DMP loss on CPU utilization, we conducted 6 experiments. The constant factors were a topology of three I/O devices, and a fixed CPU rate, LAMBDA = .001. Variable factors were as follows:

1. Two relations among the MU's were used:

   MU(1) = MU(2) = MU(3), and

   MU(1) = 2*MU(2) = 4*MU(3).

2. Three ratios of average MU/LAMBDA were used:

   Avg. MU/LAMBDA = .1,

   Avg. MU/LAMBDA = .3, and

   Avg. MU/LAMBDA = .5.

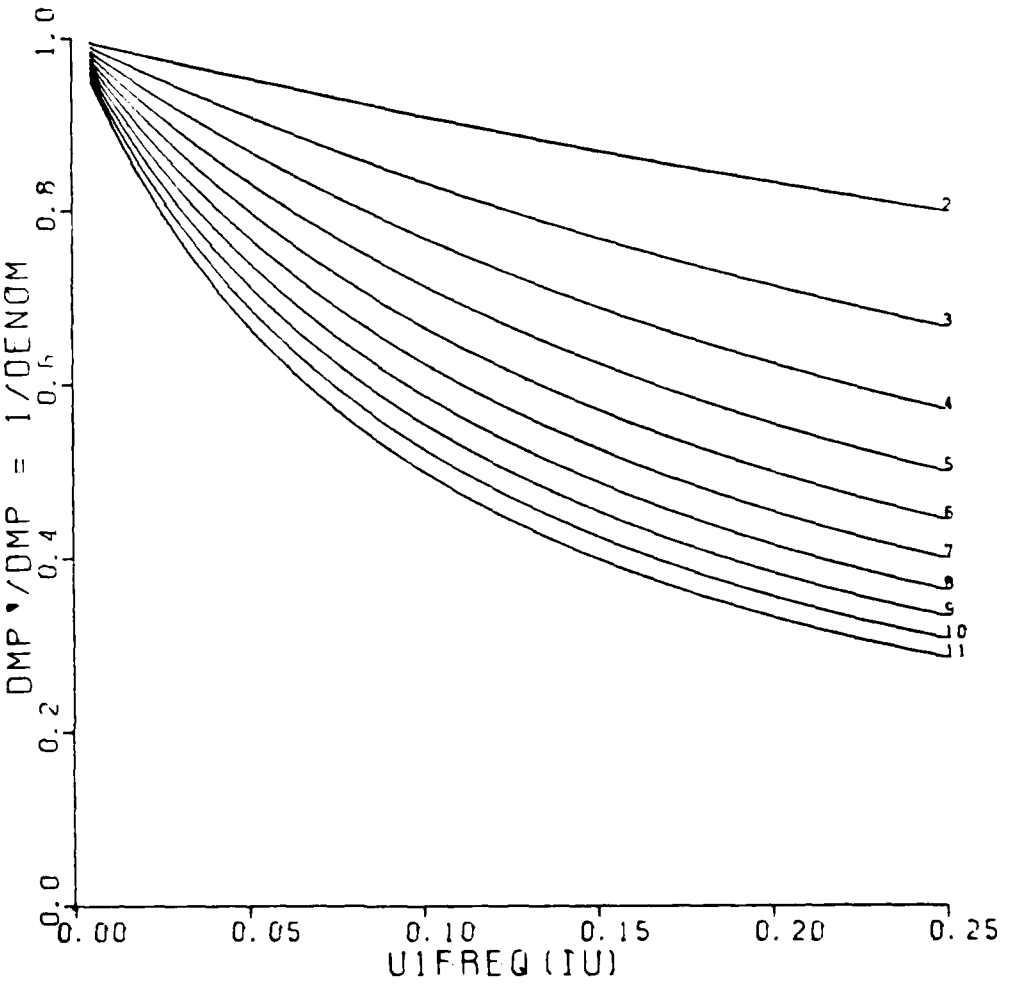


Fig. 4.6. Worst case DMP'/DMP.

For each experiment 11 levels of DMP were used from DMP=2 to DMP=12. This made a total of 66 different trials. CPU utilization was computed using the given MU's, the optimal branching probabilities, and assuming no storage constraints. The results of these experiments can be seen in Figures 4.7 through 4.12 where we have plotted a family of curves of DMP'/DMP vs UTIL'/UTIL, where UTIL' is computed using DMP'. For readability, only the even values of DMP are plotted. The number at the end of each curve corresponds to the DMP used to compute that curve. DMP'/DMP values are plotted from 1.0 (no DMP overhead) to 0.4 (60% DMP overhead loss). The range was chosen because 0.4 represents an extreme case. Beyond this value utilization loss is too great to be of practical interest. Table 4.1 contains the optimal utilization figures for each experiment using the full value of DMP.

		DMP					
MU'S	Avg. MU/LAM	2	4	6	8	10	12
Equal	.1	.142	.195	.222	.238	.248	.256
Equal	.3	.369	.527	.614	.666	.708	.736
Equal	.5	.529	.744	.854	.915	.951	.972
Unequal	.1	.202	.247	.273	.288	.297	.306
Unequal	.3	.498	.641	.725	.781	.819	.847
Unequal	.5	.671	.846	.928	.967	.985	.994

TABLE 4.1. Optimal utilization values for case studies.

Upon examining the curves, we find that they are quite similar. One general trend that can be noted is that the higher the DMP, the less sensitive utilization is to DMP loss. This is consistent with other studies of CPU utilization vs DMP, e. g., [Lips77]. The trend is that as DMP goes up, the curves tend to flatten.

Another noteworthy trend is that the cases with unequal I/O service rates (Figures 4.10, 4.11, and 4.12) are slightly less sensitive to DMP loss than the cases with equal I/O service rates (Figures 4.7, 4.8, and 4.9). This is probably because in the inequality case the faster I/O devices have P* values that are larger than the normalized ratios of the service rates. This means that on the average, the faster devices have more customers queued in front of them and may be said to have a higher "local" DMP. Thus it follows that the faster devices that dominate the system are less sensitive to a DMP decrease. This is due to the previously mentioned decreasing slope of the utilization curve for higher DMP values.

A significant factor to be noted from these experiments is that these cases are fairly robust with respect to utilization loss for the range considered. In fact, in all cases, the DMP'/DMP ratio is significantly less than the worst case UTIL'/UTIL ratio. This is encouraging if we intend to use the read-only model to approximate the read-write model.

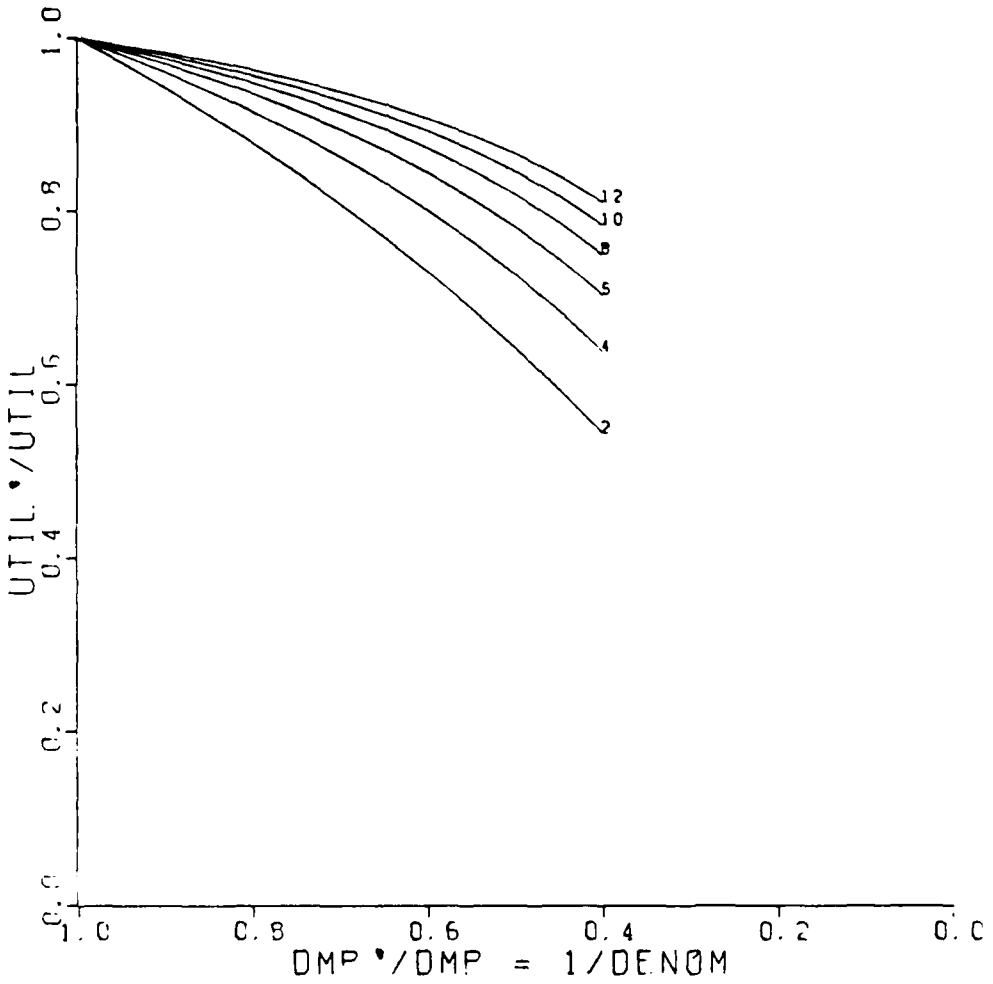


Fig. 4.7. Utilization loss due to overhead.
Experiment with MU's equal, Avg MU/LAMBDA = .1.

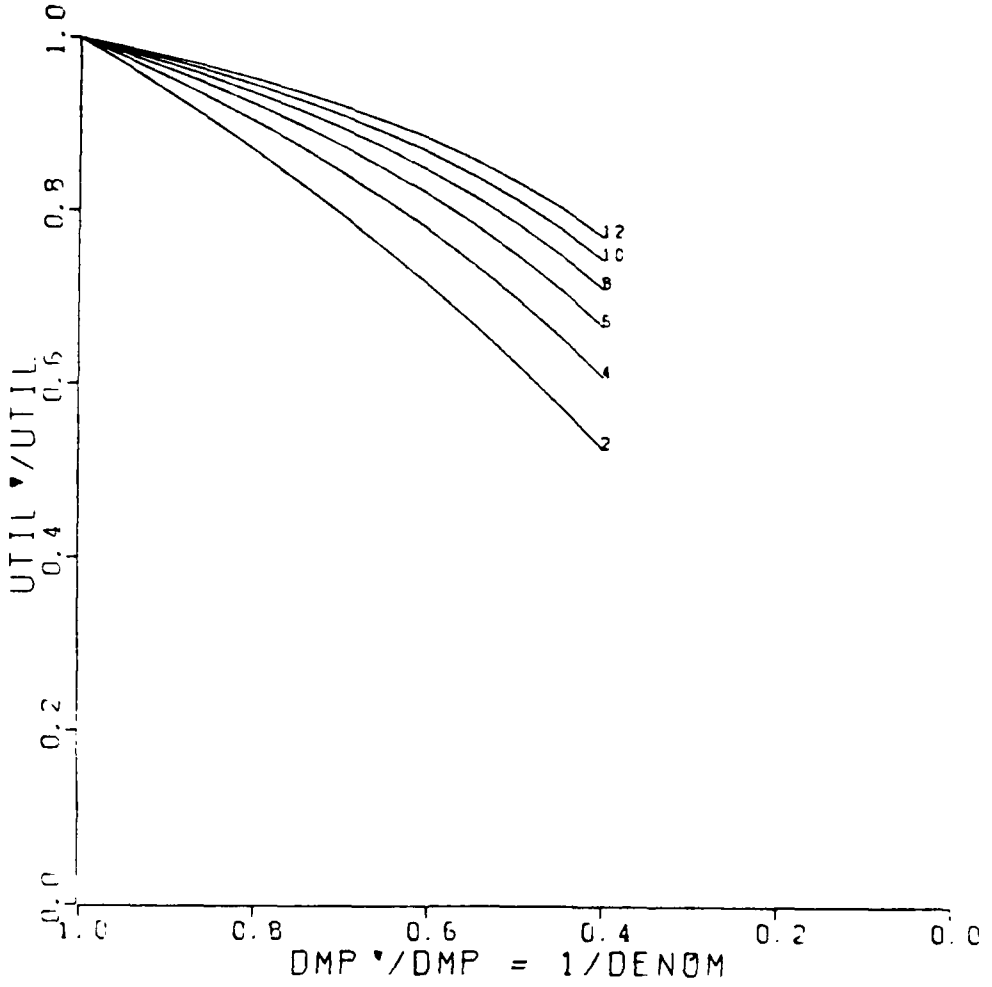


Fig. 4.8. Utilization loss due to overhead.
Experiment with MU's equal, Avg MU/LAMBDA = .3.

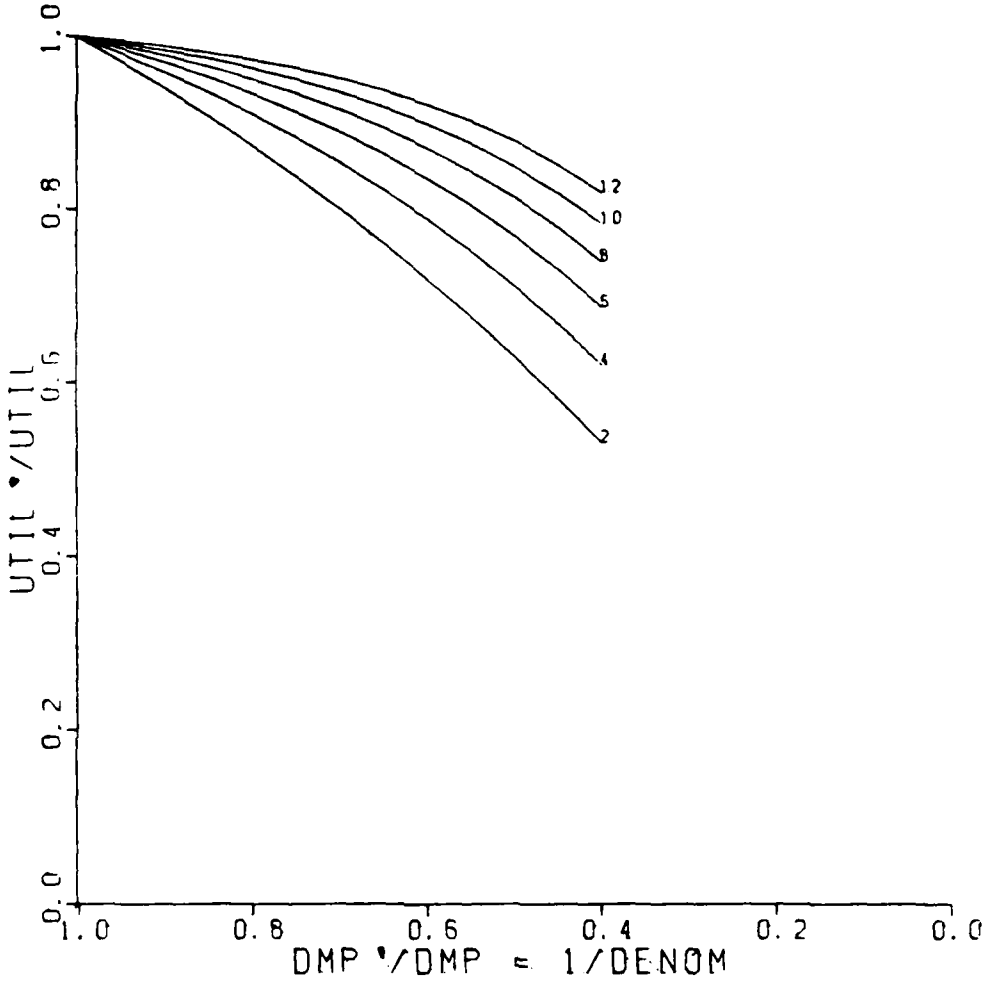


Fig. 4.9. Utilization loss due to overhead.
Experiment with MU's equal, Avg MU/LAMBDA = .5.

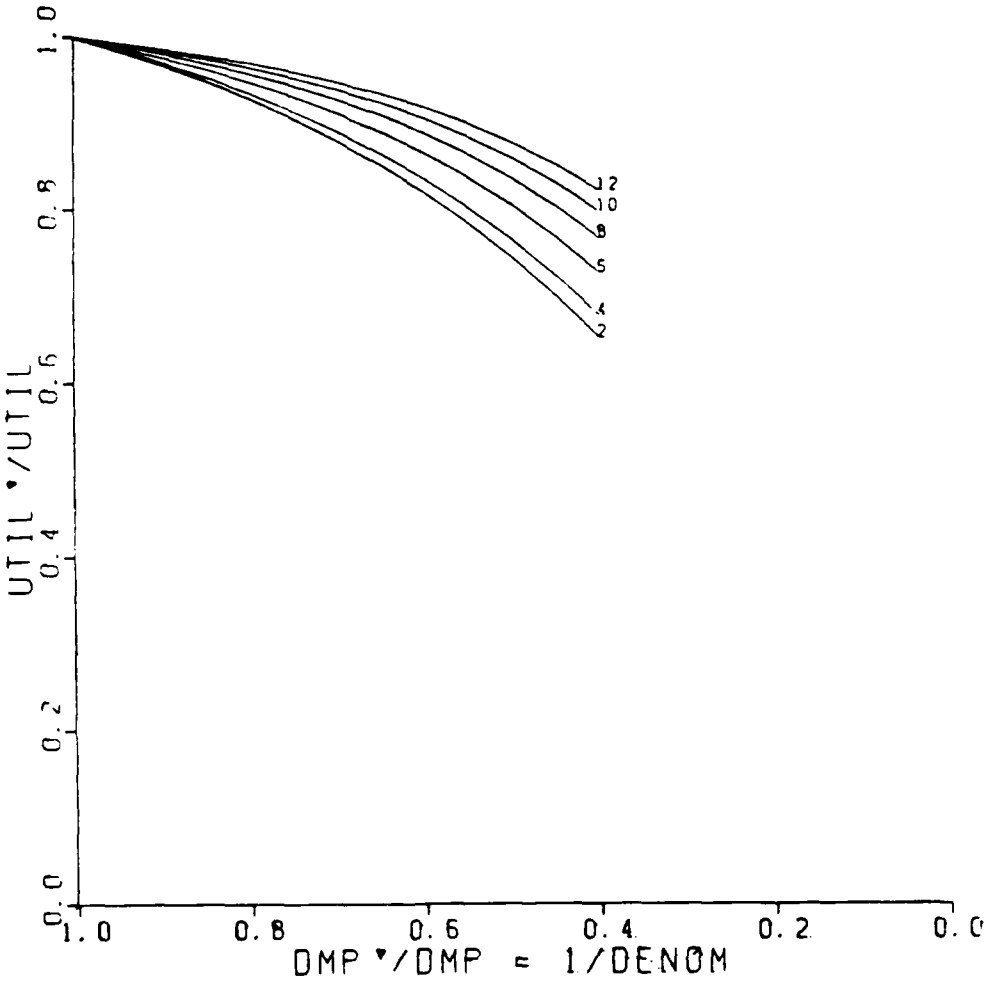


Fig. 4.10. Utilization loss due to overhead.
Experiment with MU's unequal, Avg Mu/LAMBDA = .1.

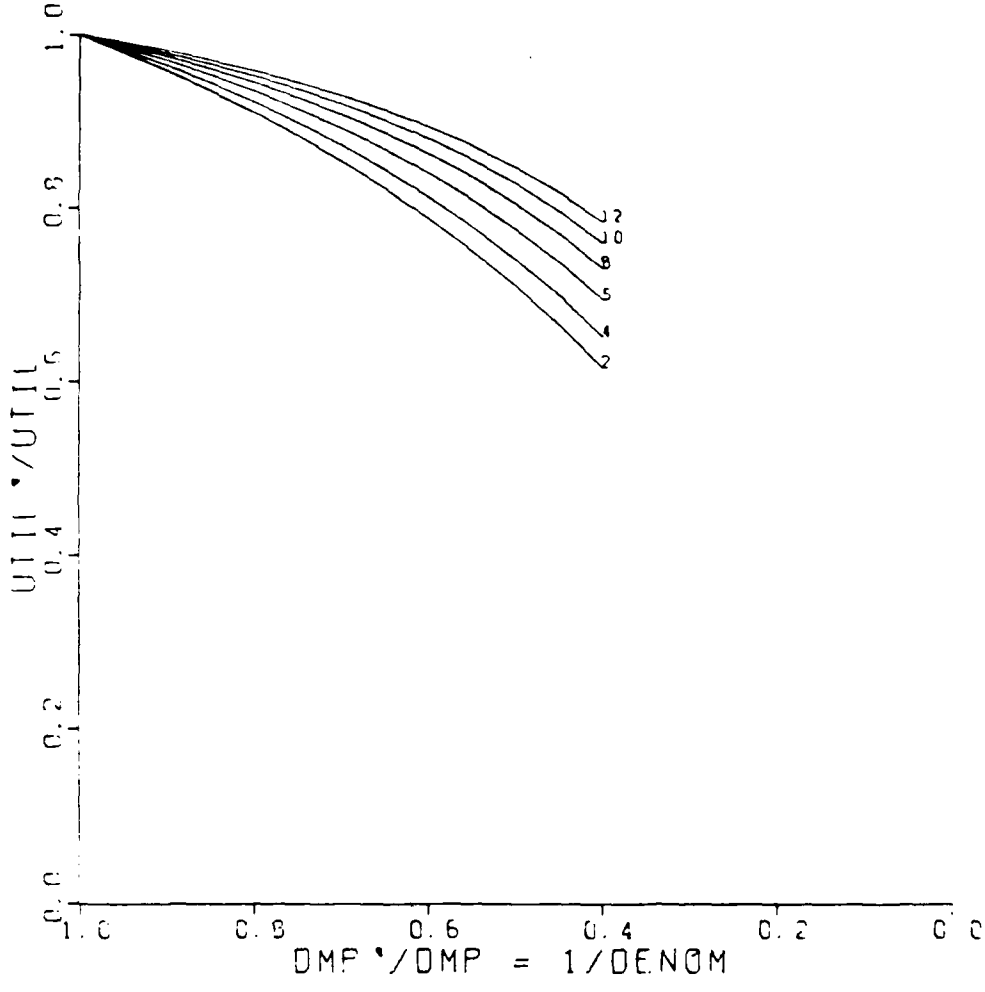


Fig. 4.11. Utilization loss due to overhead.
Experiment with MU's unequal, Avg MU/LAMBDA = .3.

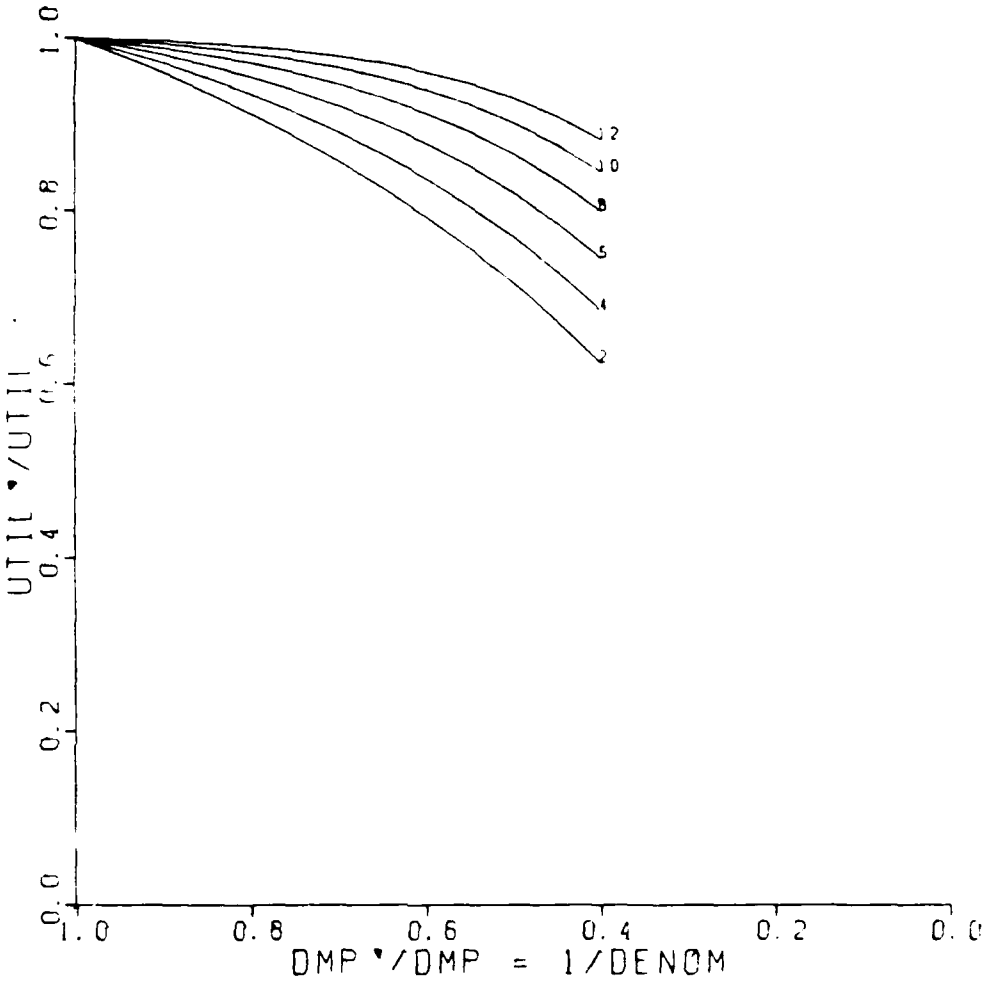


Fig. 4.12. Utilization loss due to overhead.
Experiment with MU's unequal, Avg MU/LAMBDA = .5.

4.6.1.B. Access Frequency Error

In Section 4.2 we established that file access frequencies are assignment dependent. If we use the read-only model to approximate the read-write model, we ignore this fact. Therefore, we introduce the possiblity that our solution does not actually minimize the differences between the P*'s and the assigned frequencies (TOTFREQ's). Here we will derive some bounds on the error. As in the case of DMP error, we will see that the worst case bounds are not very useful. Thus, we will again provide graphs showing the worst error that is introduced for some more likely cases.

Again, recall that the read-only model furnishes a solution with at most NDEVS-1 extra file copies. As before, the worst case error for access frequency computation occurs when we have NDEVS-1 extra copies of file i_U. We can see this by refering to equations 4.1, 4.2, and 4.3. Note that DENOM determines the difference between the single copy frequency and any multiple copy frequency. DENOM is maximized when the largest value of UlFREQ(i) (i. e., UlFREQ(i_U)) is replicated the maximum number of times (NDEVS times).

If we wish to compute the percentage of error, UnFREQ(i)/UlFREQ(i) or QnFREQ(i)/QlFREQ(i), we must compute the worst case value of DENOM. For convenience we recall equation 4.3,

$$\text{DENOM} = \sum_{i=1}^{\text{NFILES}} \text{QlFREQ}(i) + \sum_{i=1}^{\text{NFILES}} \sum_{j=1}^{\text{NDEVS}} \text{ASMT}(i,j) * \text{UlFREQ}(i).$$

With NDEVS copies of i_U we have

$$\sum_{i=1}^{\text{NFILES}} \text{ASMT}(i,j) = 1 \text{ for } i \neq i_U$$

and

$$\sum_{i=1}^{\text{NFILES}} \text{ASMT}(i_U,j) = \text{NDEVS}.$$

Since $\sum_{i=1}^{\text{NFILES}} (\text{QlFREQ}(i) + \text{UlFREQ}(i)) = 1$

we may give the worst case DENOM value as

$$1 + (\text{NDEVS}-1) * \text{UlFREQ}(i_U).$$

Thus we have

QnFREQ(i) = QlFREQ(i)/DENOM', and

UnFREQ(i) = UlFREQ(i)/DENOM'

where

DENOM' = 1 + (NDEVS-1) * UlFREQ(i_U).

If we wish to compute QnFREQ(i)/QlFREQ(i) or UnFREQ(i)/UlFREQ(i) for this worst case we can see that

QnFREQ(i)/QlFREQ(i) = (QlFREQ(i)/(1+(NDEVS-1)*UlFREQ(i_U)) /
QlFREQ(i)

= 1 / (1+(NDEVS-1)*UlFREQ(i_U)).

The computation for UnFREQ(i)/UlFREQ(i) is, of course, exactly the same. Note that all this expression equals 1/DENOM'.

The above value turns out to be equal to DMP'/DMP for the worst case. This result is derived as follows.

$$
\begin{aligned}
DMP'/DMP &= (DMP-OHEAD)/DMP \\
&= DMP - ((NDEVS-1)*U1FREQ(i_U) * DMP / \\
&\qquad (1 + (NDEVS-1) * U1FREQ(i_U)) / DMP \\
&= 1-(NDEVS-1)*U1FREQ(i_U)/(1+(NDEVS-1)*U1FREQ(i_U)) \\
&= (1+(NDEVS-1)*U1FREQ(i_U)-(NDEVS-1)*U1FREQ(i_U)) / \\
&\qquad (1+(NDEVS-1)*U1FREQ(i_U)) \\
&= 1/(1 + (NDEVS-1) * U1FREQ(i_U))
\end{aligned}
$$

Thus we may use Figure 4.6 to obtain either the DMP'/DMP worst case error, the UnFREQ(i)/U1FREQ(i) worst case error, or the QnFREQ(i)/Q1FREQ(i) worst case error. We also note that the limits that apply to overhead obviously apply here.

While the frequency errors may be of some interest, again we are ultimately interested in how this affects the CPU utilization. CPU utilization is not affected by the redistribution of individual frequencies but rather by the TOTFREQ's assigned to the devices. We may easily derive a lower bound on the problem given $U1FREQ(i_U) = 0$. In this case we have no updates and the model reduces to the read-only case and the answer is exact. Given that file i_U is replicated NDEVS times we may construct an upper bound on the redistribution of the TOTFREQ's. Let $ASMTFR(i_U,j)$ approach 0 for all devices but one, say J, and for J, let $ASMTFR(i_U,J)$ approach 1. It is important that $ASMTFR(i_U,j)$ does not equal 0 for any device so that each device j

contains a copy of i_U. Likewise, let P*(J) approach 1, obviously P*(j) approaches 0 for all other $j \neq J$. Furthermore, let UlFREQ(i_U) approach 1. Obviously, all other UlFREQ(i) and all QlFREQ(i) approach 0. Thus, for the limiting case we approach

UnFREQ(i_U) = 1/(1+(NDEVS-1)*1) = 1/NDEVS.

Since all devices carry a copy of i_U we approach

TOTFREQ(j) = 1/NDEVS for all j,

in the limiting case.

Again we note that the values for the extreme cases are not as useful as some more likely cases. We conducted the same experiments as we did while assessing the effect of DMP loss on utilization. This time we assume no DMP loss in order to independently assess the impact of TOTFREQ redistribution on utilization. For any given redistribution, the worst case will result if the access error (TOTFREQ(j)-TOTFREQ(j)/DENOM) is shifted from the faster devices to the slowest device. Note that it is entirely possible that the net effect of access redistribution may be considerably less than this. In fact there may be no net redistribution, and the only utilization loss factor is the overhead. However, for these experiments, we assume worst case redistribution. The results are summarized in Figures 4.13-4.18, where we have again plotted UTIL'/UTIL vs 1/DENOM' (the access error). We have used the same range and scale for ease of comparison,

and, again, the numbers at the ends of the curves are DMP values. The optimal utilization values remain the same and may be read from Table 4.1.

One general trend to notice is that the lower the DMP, the less sensitive the utilization is to access shifting. This is the reverse of the DMP situation shown in Figures 4.7 - 4.12. With a small number of customers, the utilization is low to begin with (see Table 4.1). A redistribution factor applied to a small number of customers is naturally less significant than the same redistribution factor applied to a larger number of customers.

Next, note that utilization is less sensitive to access redistribution in the cases where the MU's are equal than in the cases where they are unequal. Since all service rates are the same in the uniform case, shifting more access to the "slowest" device has little impact over a large range of cases. However, when the service rates are unequal by a large enough amount, as in the nonuniform case, the system is fairly sensitive to frequency redistribution. Here UTIL'/UTIL drops off more rapidly than 1/DENOM'. Recall that this was not the case for the DMP loss effect.

4.6.1.C. Combined error effects

Here we show the results of the same basic experiments, but this time we include both of the error generating factors, overhead and redistribution. The results are plotted in Figures 4.19-4.24.

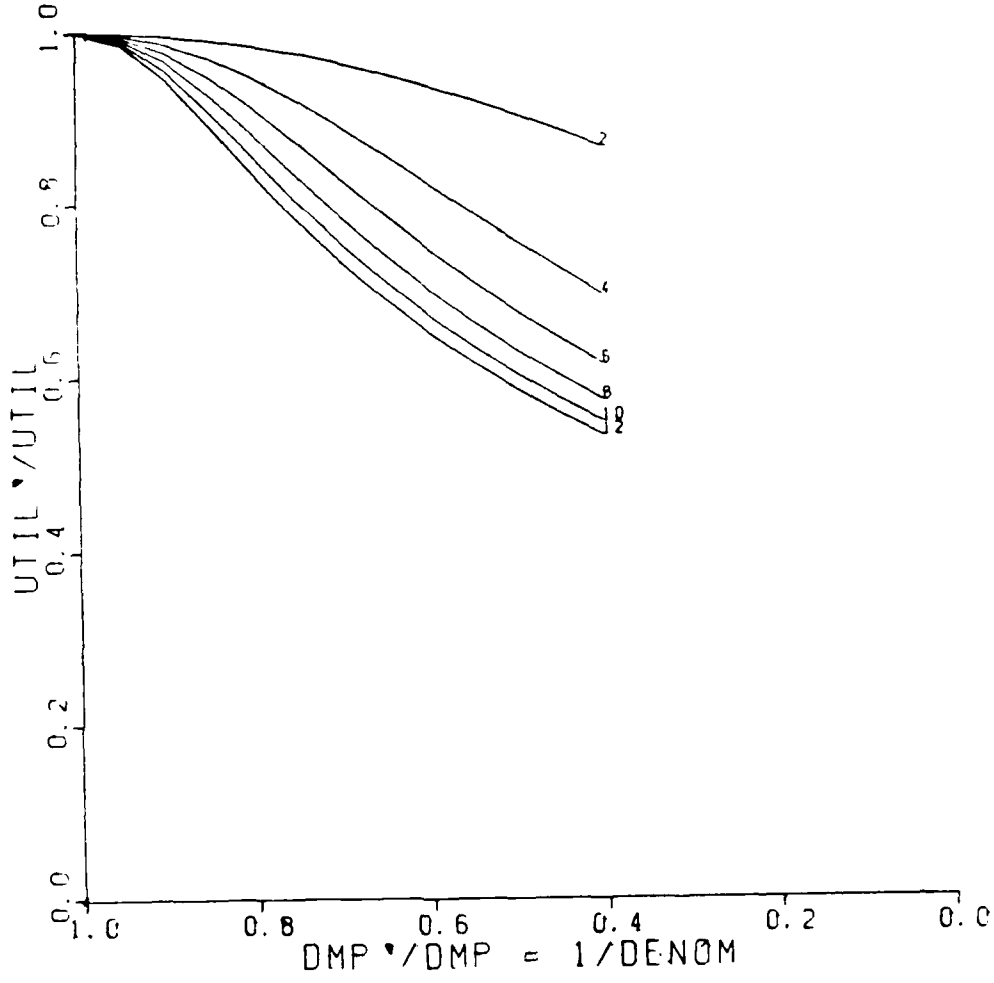


Fig. 4.13. Utilization loss due to access redistribution Experiment with MU's equal, Avg MU/LAMBDA = .1.

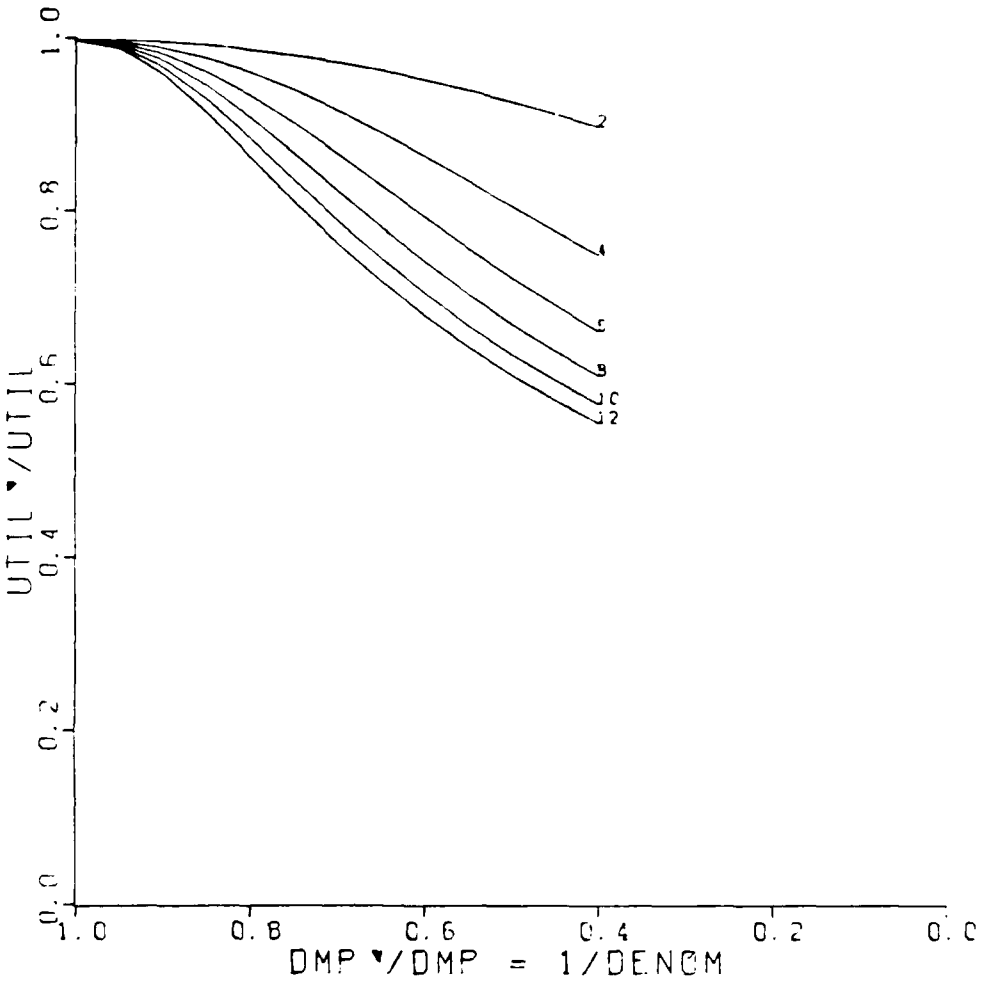


Fig. 4.14. Utilization loss due to access redistribution Experiment with MU's equal, Avg MU/LAMBDA = .3.

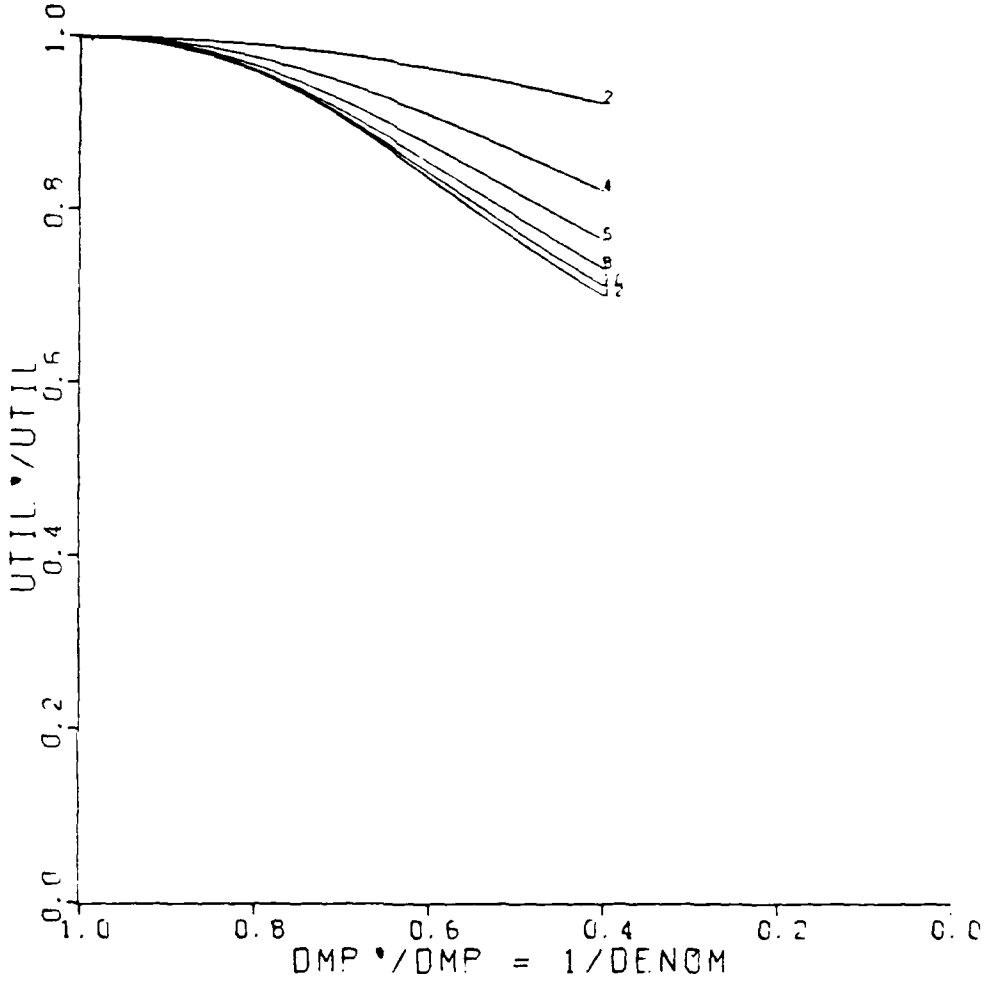


Fig. 4.15. Utilization loss due to access redistribution Experiment with MU's equal, Avg MU/LAMBDA = .5.

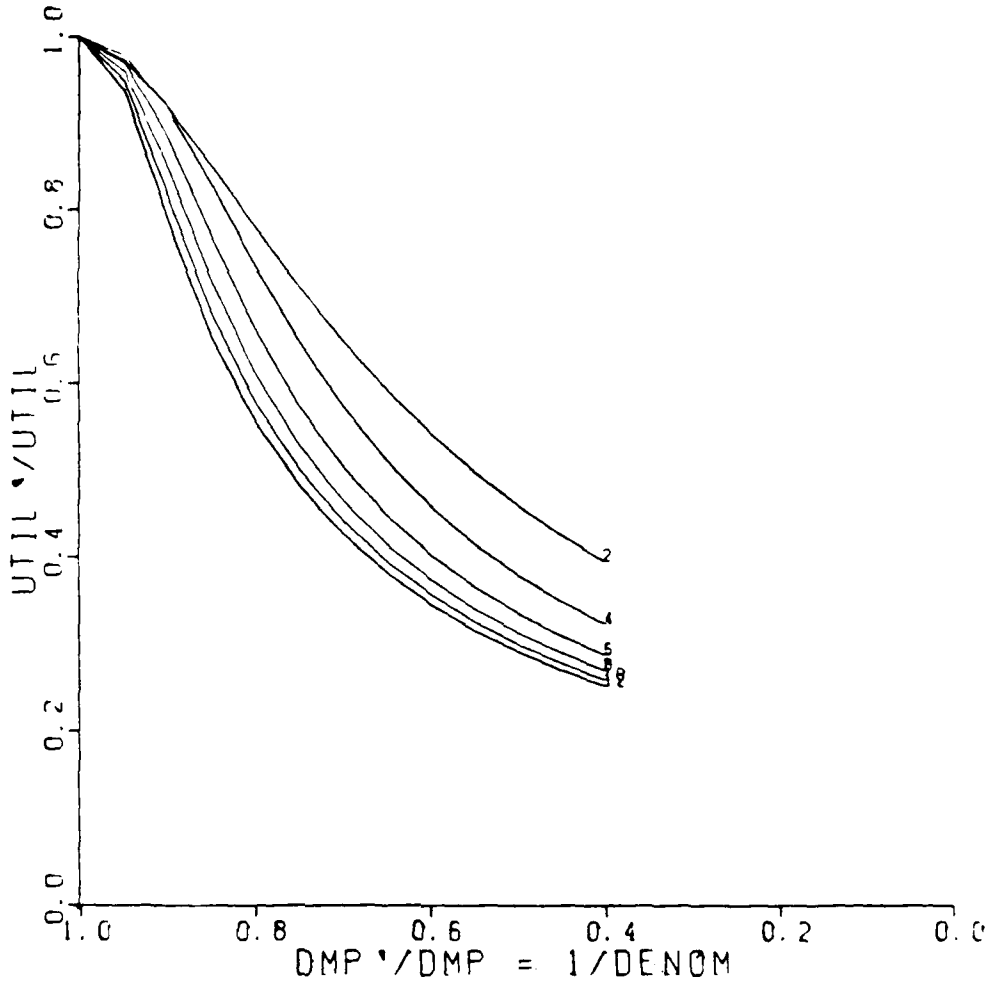


Fig. 4.16. Utilization loss due to access redistribution Experiment with MU's unequal, Avg MU/LAMBDA = .1.

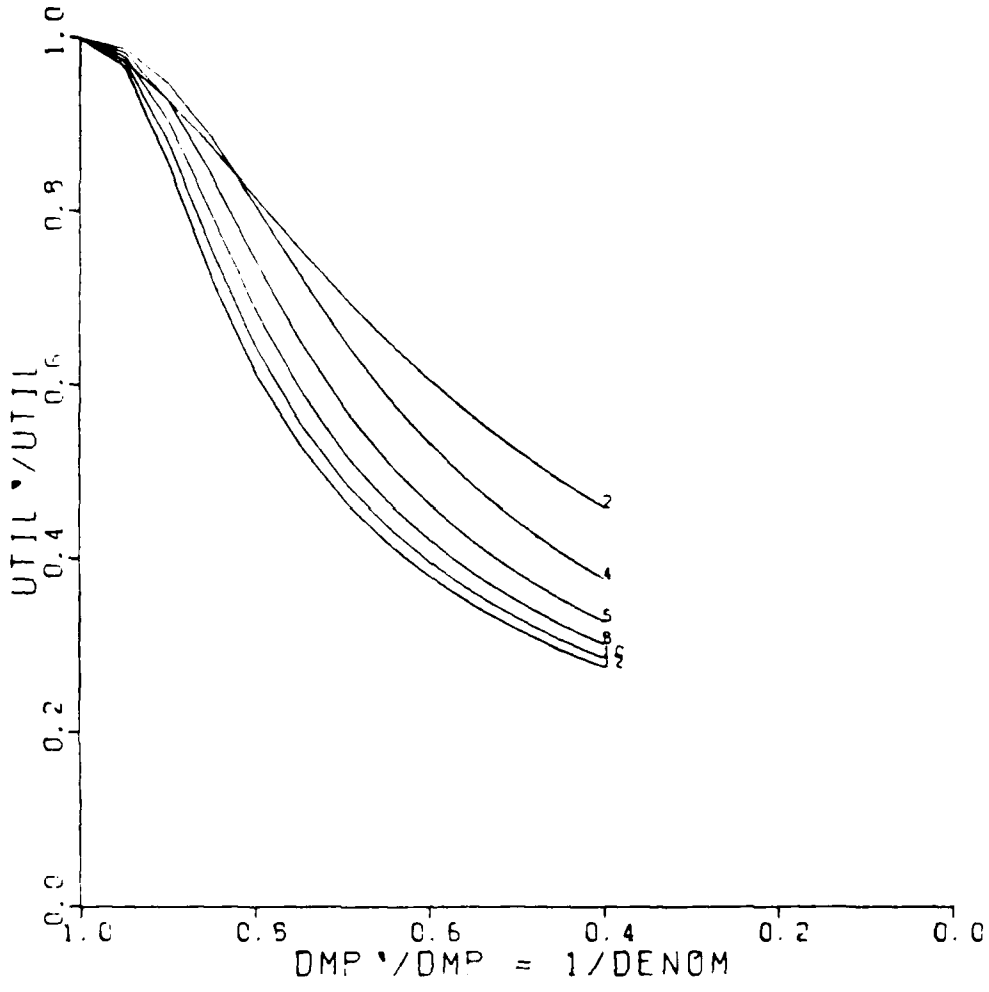


Fig. 4.17. Utilization loss due to access redistribution Experiment with MU's unequal, Avg MU/LAMBDA = .3.

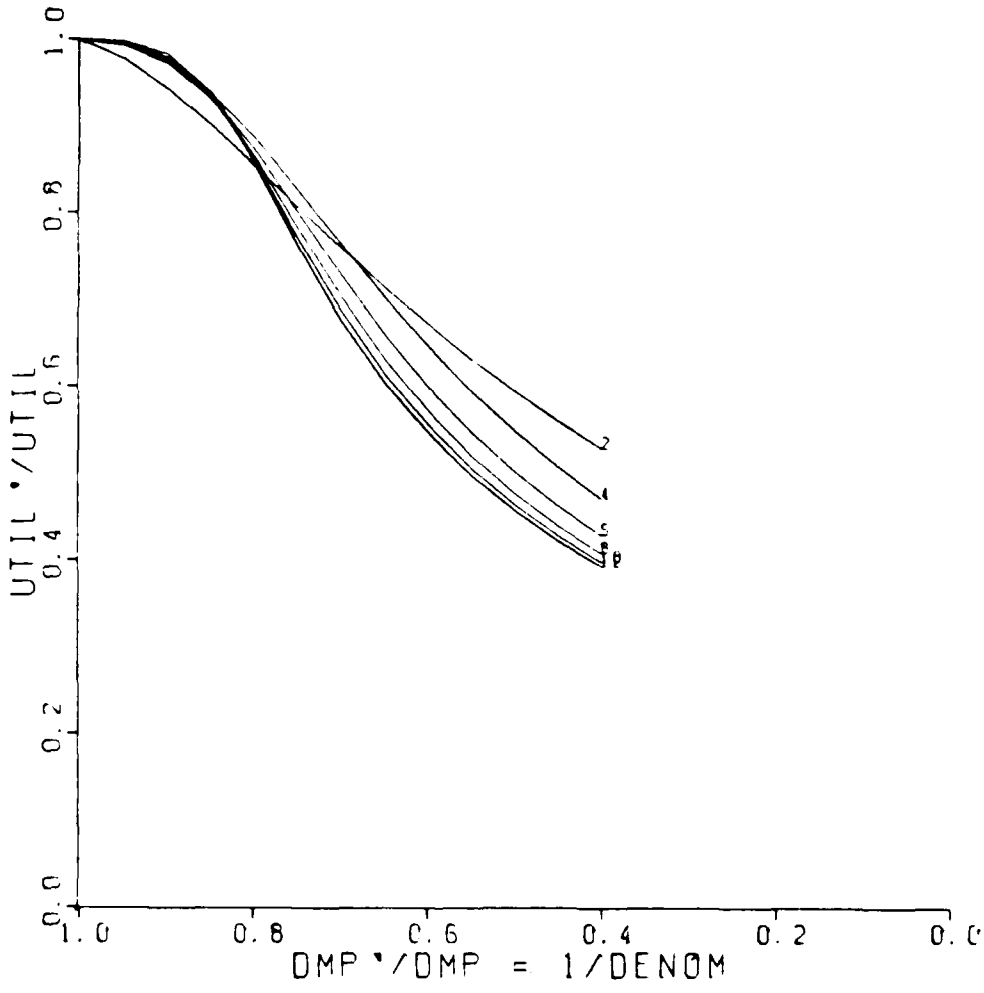


Fig. 4.18. Utilization loss due to access redistribution Experiment with MU's unequal, Avg MU/LAMBDA = .5.

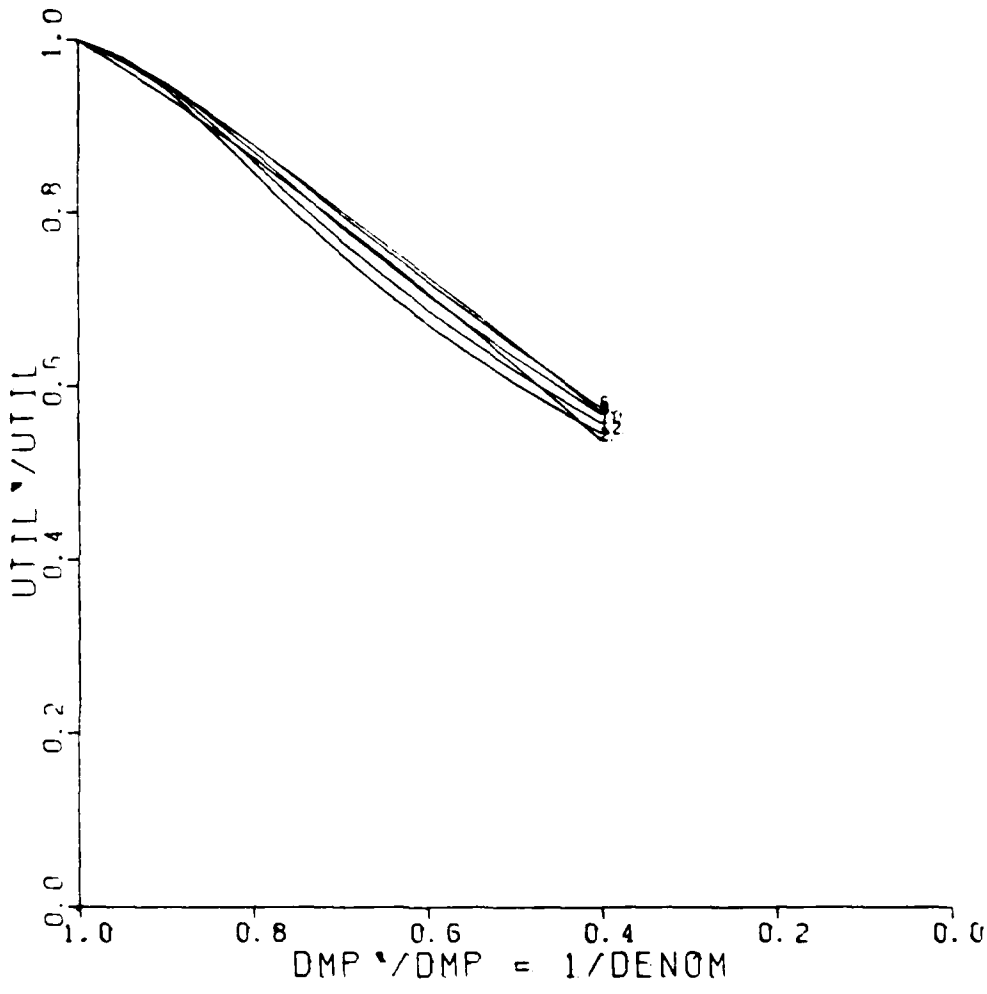


Fig. 4.19. Utilization loss due to both error factors. Experiment with MU's equal, Avg MU/LAMBDA = .1.

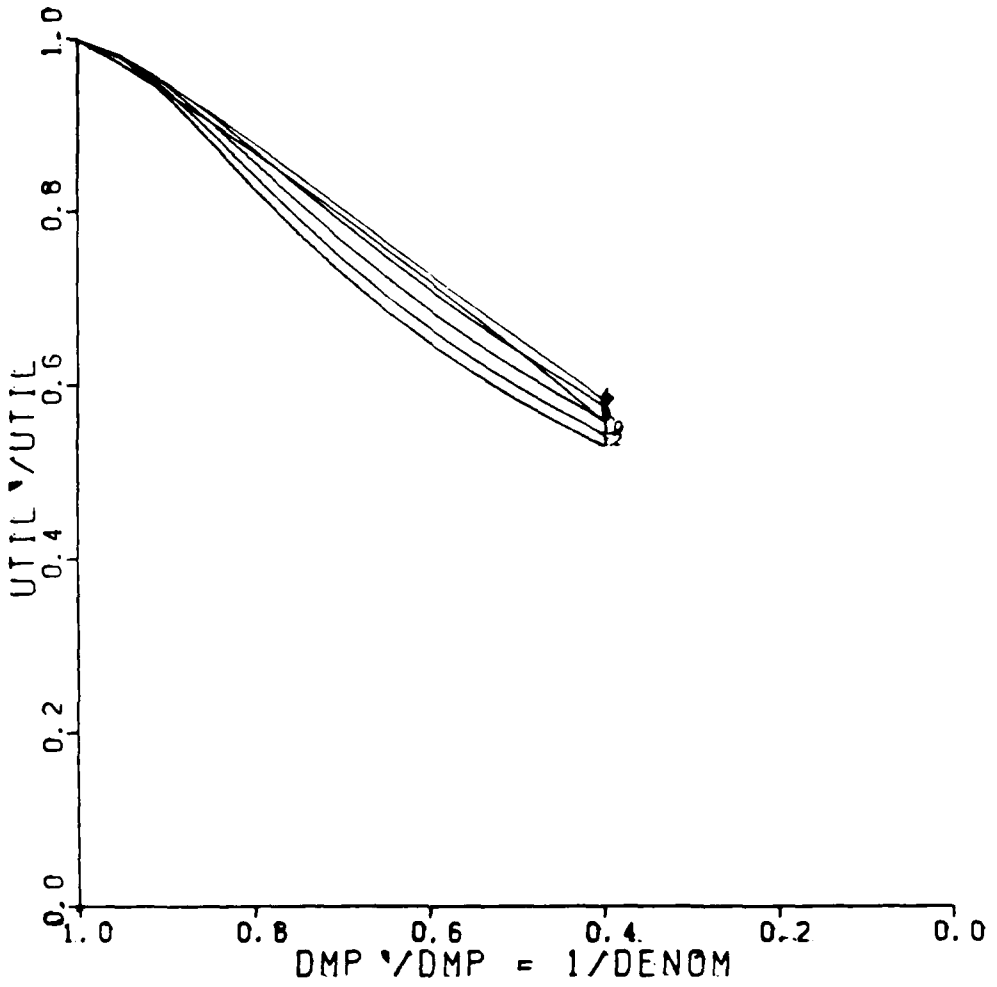


Fig. 4.20. Utilization loss due to both error factors. Experiment with MU's equal, Avg MU/LAMBDA = .3.

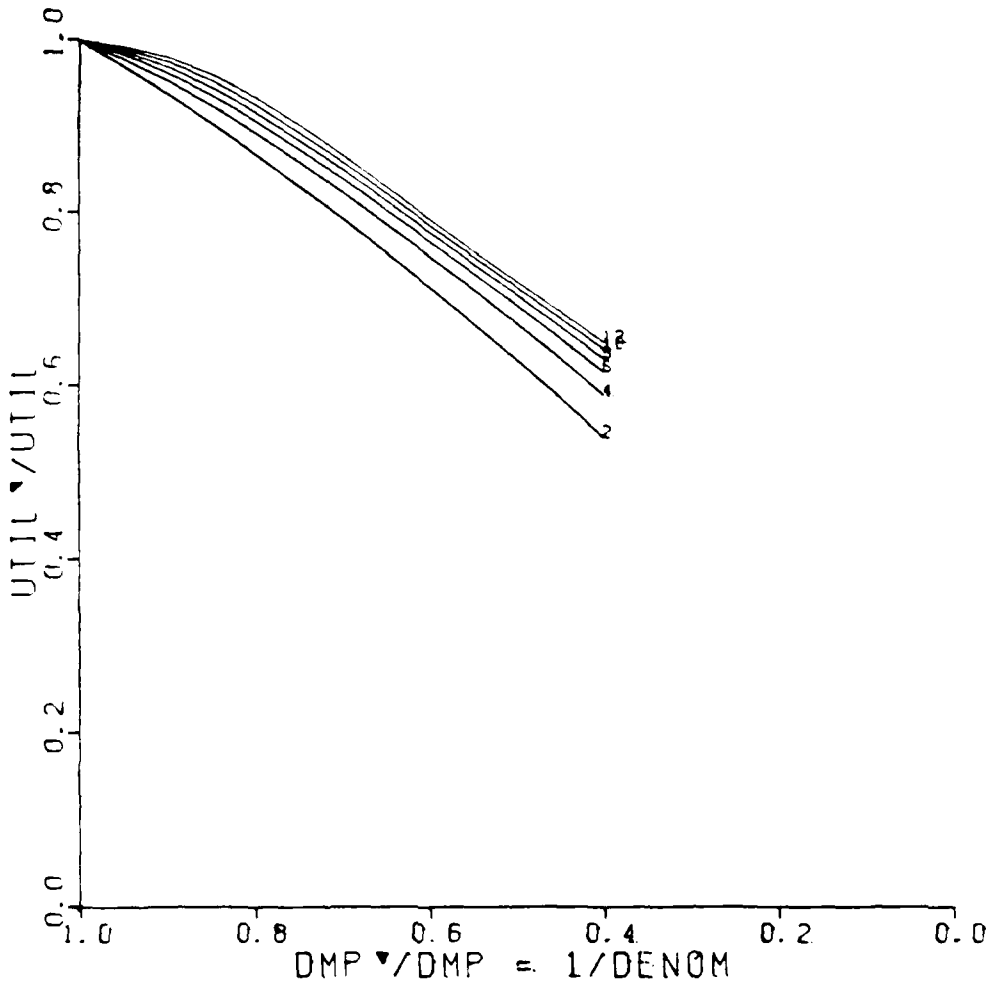


Fig. 4.21. Utilization loss due to both error factors. Experiment with MU's equal, Avg MU/LAMBDA = .5.

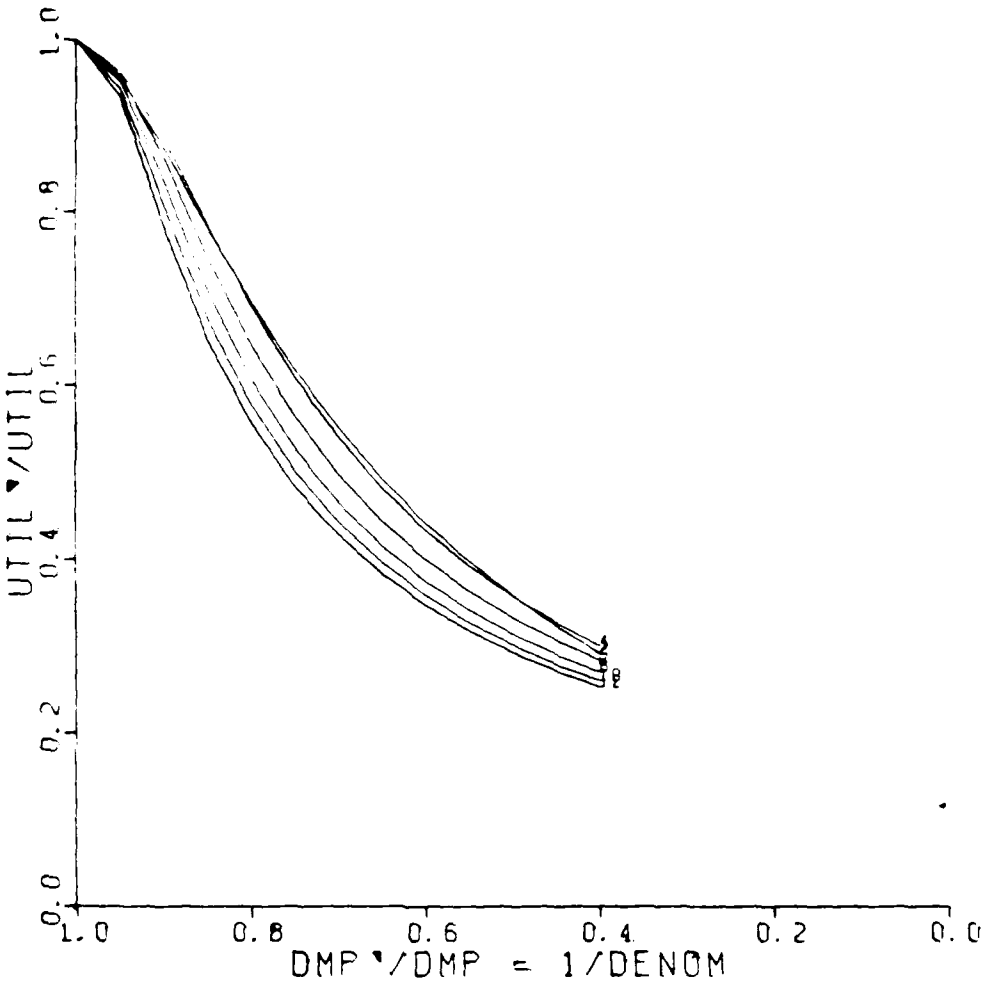


Fig. 4.22. Utilization loss due to both error factors. Experiment with MU's unequal, Avg Mu/LAMBDA = .1.

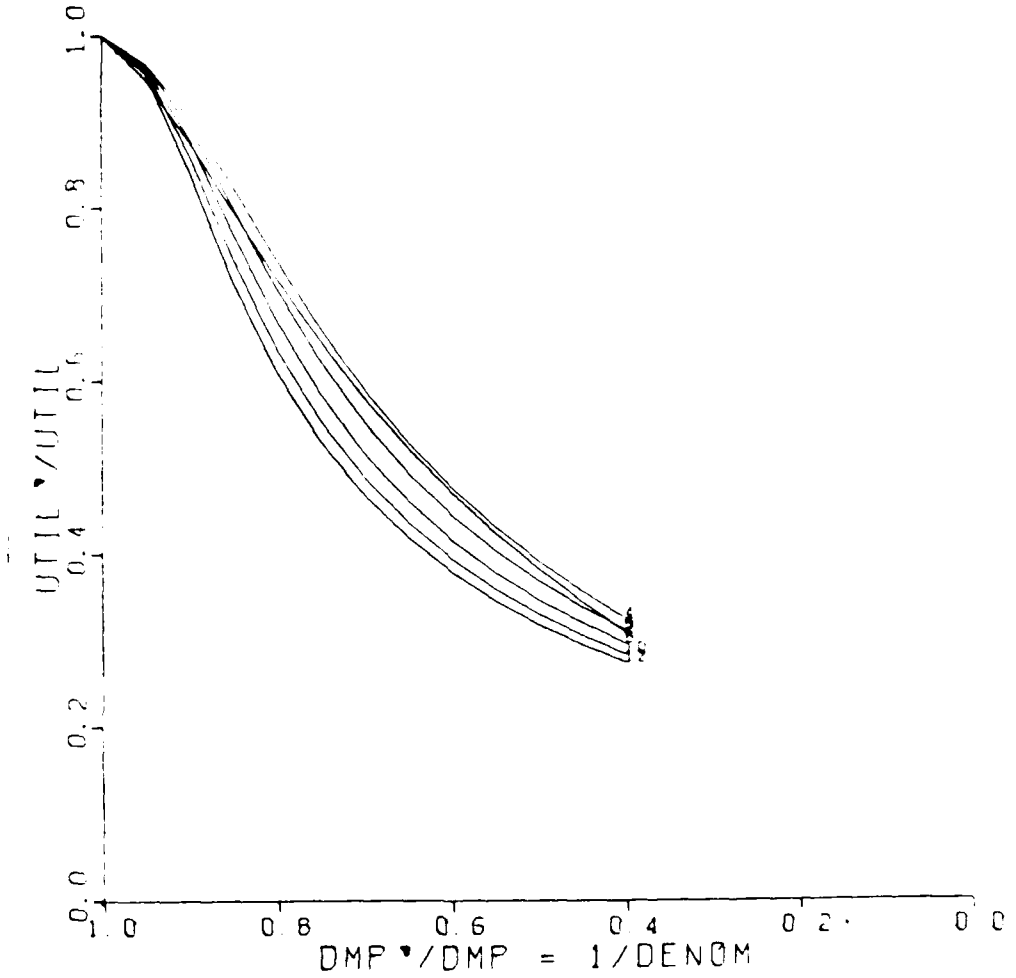


Fig. 4.23. Utilization loss due to both error factors. Experiment with MU's unequal, Avg MU/LAMBDA = .3.

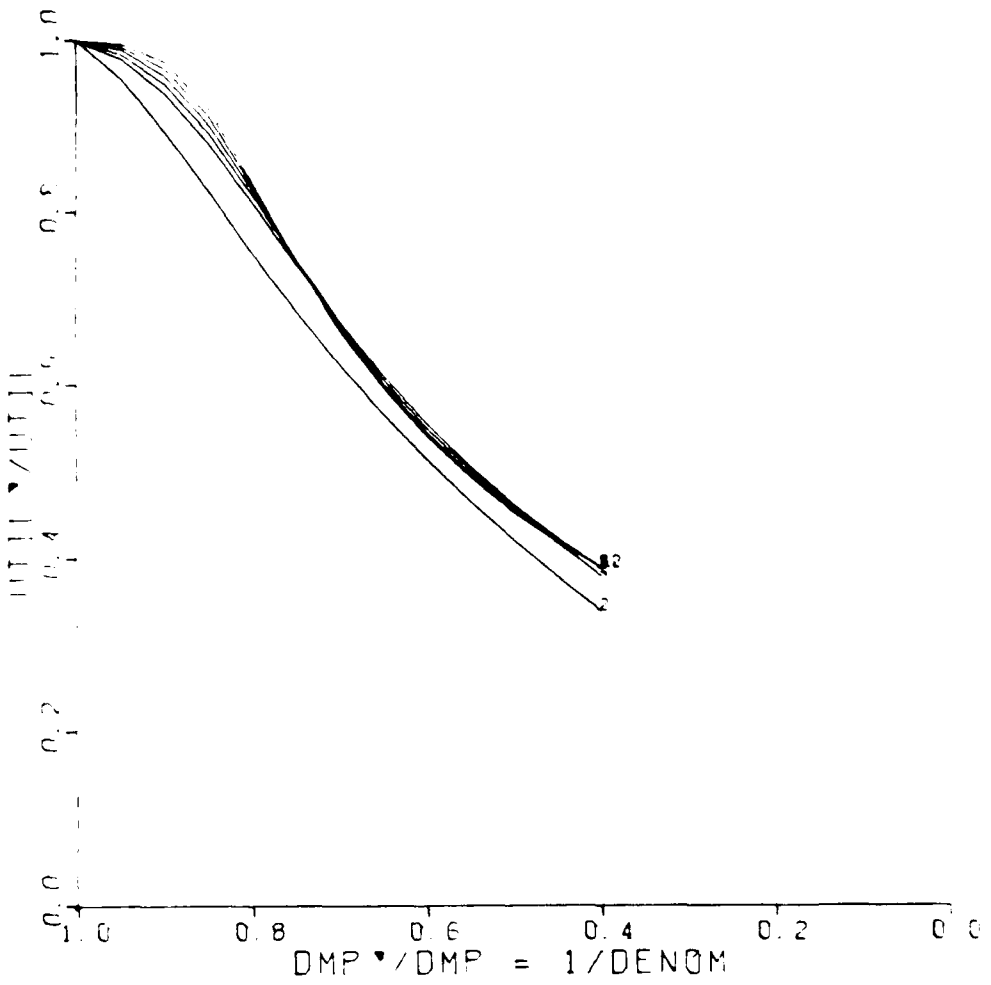


Fig. 4.24. Utilization loss due to both error factors. Experiment with MU's equal, Avg MU/LAMBDA = .5.

We note that for the case where the MU's are equal, UTIL'/UTIL decreases less rapidly than 1/DENOM'. For values of 1/DENOM as small as 0.8, we may obtain a worst case utilization decrease of 10-15%. This is encouraging. However, for the case where the MU's are unequal, utilization drops of more rapidly. It is obvious that the relative values of the MU's is a critical factor in assessing the worst case accuracy of the heuristic.

Another interesting trend is that the results tend to be independent of the DMP values. This is indicated by the tight clustering of the curves. If this holds in general, this may help simplify future heuristics.

Since these results are computed on worst case assumptions, it may be possible to obtain reasonable answers to the read-write problem by solving the equivalent read-only problem. The cases presented here provide both encouraging and discouraging results. It is obvious that P*-matching is one critical factor that becomes more critical as the values of the MU's differ. Before we accept or condemn this method, we should consider how the method seems to perform compared to worst case conditions. Therefore, we apply this heuristic to the cases studied in Chapter III. We discuss these results in the next section.

4.6.2. Case studies

The 12 experiments that were conducted in Chapter III were reformulated as read-write problems using the file data in Figure 4.25. Note that we simply divided FREQ(i) equally between QlFREQ(i) and UlFREQ(i) for all i. All QLEN(i) = ULEN(i) = MLEN(i), so the data reduces to the same read-only models.

Some comments are appropriate regarding the severity of these test cases. Note that the relation of the service rates is approximately such that MU(1) = 1.6*MU(2) = 3.2*MU(3). This approaches the situation of the previous section that was shown to be the most sensitive to frequency redistribution. Also note that the value of $UlFREQ(i_U)$ = .115. From Figure 4.6 we see that the worst case 1/DENOM value for 3 I/O devices is about 0.81. Since the average MU to LAMBDA ratio is about 0.2, these cases are somewhat similar to the cases illustrated in Figures 4.22 and 4.23. These cases show worst case utilization to be in the vicinity of 60% of optimal. It should now be apparent that there is ample room for the heuristic to go wrong and that the "deck" is not stacked in it's favor.

The results of these experiments are summarized in Table 4.2 where we have reproduced much of Table 3.2. Note that for 5 cases, the read-only answers were exactly the same as the read-write answers. In these cases the storage constraints became active and there were no replicated

files. In the other 7 cases, the average ratio of read-write utilization to read-only utilization was approximately .92. Including all 12 cases the average ratio was approximately .95. This would seem to indicate that the actual utilization loss does not really approach the worst case loss at all. Another appropriate comparison is compare the read-write results against the single copy algorithms.

NFILES = 12

NO	QIFREQ	UIFREQ	QLENGTH	ULENGTH
1	0.115	0.115	120.000	120.000
2	0.060	0.060	110.000	110.000
3	0.050	0.050	100.000	100.000
4	0.045	0.045	90.000	90.000
5	0.040	0.040	80.000	80.000
6	0.040	0.040	70.000	70.000
7	0.035	0.035	60.000	60.000
8	0.030	0.030	50.000	50.000
9	0.025	0.025	40.000	40.000
10	0.025	0.025	30.000	30.000
11	0.020	0.020	20.000	20.000
12	0.015	0.015	10.00	10.000

Fig. 4.25. Read-write equivalent for Figs. 3.9-3.11.

Abbreviations:
ID: Experiment identification code
RW: Read-write heuristic
1/D: Value of 1/DENOM
IND: Industrial algorithm
SS: Split storage algorithm
SA: Integral storage, split access algorithm.

Table entries are CPU utilizations.

ID	RW	1/D	IND	SS	SA
VF0	.3999	1	.3594	.3283	.3999
VF2	.4577	1	.3078	.4506	.4577
VT0	.5257	1	.4155	.4008	.5257
VT2	.5888	.9434	.3958	.6058	.6176
MF0	.4569	1	.4345	.3943	.4569
MF2	.4662	.9524	.4196	.4893	.4885
MT0	.5978	1	.6096	.5262	.5978
MT2	.6368	.9434	.5042	.6708	.6695
NF0	.4557	.8975	.5106	.5021	.5065
NF2	.4689	.8602	.5264	.5254	.5261
NT0	.4868	.8517	.5986	.5712	.5712
NT2	.6233	.8586	.6825	.6727	.6727

TABLE 4.2. Performance of read-write heuristic.

In the cases where the read-only answer equals the read-write answer, the RW utilization figures compare very favorably against the IND and SS figures. In the cases where the storage constraints are somewhat active (cases VT2, MF2, MT2), the multiple copy read-write answers also

compare very favorably to the industrial assignments, and are only slightly less than the split storage assignments. Finally, in the cases where the storage constraints are not active, the read-write utilization figures are consistently lower than those of the other methods. We should also note that these least accurate answers also coincided with the largest 1/DENOM values. However, these values were above the worst case 1/DENOM of 1/1.23 = .813.

The most important point to notice is that failing to assign files so as to match the P*'s is potentially much more critical than incurring some DMP overhead loss. We approximate QnFREQ(i) and UnFREQ(i) by QlFREQ(i) and UlFREQ(i) respectively. It is likely that better estimators for these values will yield better results. In summary, we note that there is evidence that the read-only model can produce some reasonable approximations to the read-write model. However, we must be cautious about a result and examine how closely the assignment approaches the worst case situation. In these cases the accuracy is only fair. In defense of the heuristic we must note that to compute an exact solution to the actual problem would be prohibitively costly (if not impossible) for all but very small problems. This is nothing new and typifies the standard trade-off between cost and accuracy that is frequently encountered in many engineering situations.

4.7. Chapter summary

In this chapter we extended the model of Chapter III to include read-write files. We developed a method to formulate read-write files by introducing assignment dependent access frequencies. We also introduced a way to represent the overhead associated with multiple copy updates. We presented a model to solve for read-write files when no replication is allowed. We also presented several formulations of the problem allowing file replication. To solve these models, we proposed the heuristic use of the read-only model. Experimental evaluation of this method indicates that we can expect fair to good results when compared against single copy assignment algorithms. The results can probably be improved with better approximations for the QnFREQ(i) and UnFREQ(i) variables.

CHAPTER V

CONCLUSION

5.1. Summary of Results

This work extended previous FAP research in two major areas. Using a performance optimization approach (measured by CPU throughput), we presented a method to obtain a file assignment allowing replicated copies of read-only files. We then extended the method to account for read-write files in both the replicated and non-replicated cases.

In the read-only model discussion, we presented the concept of replicating files and splitting access to the files as a method to improve CPU utilization. The problem was formulated as a mixed integer linear goal programming problem. We proved that we could obtain alternate optimal solutions that required no more than a proven amount of storage. The problem was then solved heuristically with a method that was shown to take polynomial time in the worst case except for the solution of a linear programming problem. Case studies were used to provide evidence that the heuristic can be very accurate.

The basic ideas were then extended to allow read-write files. This extension led to the concepts of "useful" DMP, to represent the additional overhead, and the assignment

dependence of query and update frequencies. We demonstrated that the case without replication of files reduced to the read-only problem. We then presented several formulations for the case allowing replication. We used the read-only solution technique to heuristically solve for the read-write problem. Worst case bounds were established for the approximation. Additionally, we bounded the accuracy for some less severe cases. Finally, case studies were presented to give evidence that the heuristic could be useful in many instances.

5.2. Future Research

The ideas presented in this work are by no means the final words on the subject. Many extensions are suggested by challenging some of the basic assumptions. For example, the assumption of exponentially distributed service times is standard, but not usually true. Some models could be developed that are independent of this assumption. Likewise, there are many different queue disciplines other than first-come-first-served. Another assumption that could be changed is that utilization is the correct performance measure. Maximum utilization appears reasonable in the context of this work, but a case can be made that response time may be a more appropriate metric in many other situations.

While we have provided a basis for formulating the read-write problem, we have obviously not developed the ultimate solution technique. Consider if we choose to use response time as our performance measure. Then perhaps more file copies will be necessary, and we know that the accuracy of our heuristic would be more subject to question. One thing to pursue in this light would be to obtain good estimators for UnFREQ(i) and QnFREQ(i).

One obvious extension is to consider a topology other than the star topology. With the recent emphasis on computer networks, it would be useful to study many topologies such as rings or trees. Ultimately, we would like to be able to model general topologies.

Finally, there is an obvious need for validation of this model and others like it. Analytic models are created far more often than they are tested and evaluated on actual systems. Since improved performance on real systems is the true goal, it is critical to validate models such as the ones presented here to be able to assess their ultimate usefulness.

File assignment techniques have proved their usefulness in the past. Given the current emphasis on computer networking, there is every indication that improved techniques are needed more than ever. The file assignment problem is a rich problem with many important unanswered questions. We believe that this is a fruitful area for

research that needs much attention and will receive much attention in the future.

APPENDIX A.

COMPARISON OF FILE ASSIGNMENT MODELS

The following table is taken from [Dowd78]. It is a useful summary of many of the critical features of the 13 FAP formulations surveyed in [Dowd78].

	Model												
Features considered	a	b	c	d	e	f	g	h	i	j	k	l	m
Multiple files				x	x	x	x	x	x	x	x	x	x
File copies	x	x	x			x	x					x	
Query & update traffic	x	x	x			x	x					x	
Storage costs	x	x	x	x		x	x						
Linear objective function	x	x	x	x	x								
Queuing delays						x	x	x	x	x	x	x	x
Throughput or time measure				x	x	x	x	x	x	x	x	x	x
Solution bounds						x	x	x	x	x	x	x	x
Capacity					x	x				x	x	x	x
Dynamic routing			x										x
File-program dependence			x										
Integral file assignment	x		x			x	x	x		x	x	x	x
File-node dependence			x		x								
General topologies	x	x	x				x						x
Optimal file assignment	x		x	x	x	x			x				
Tractable heuristic		x					x			x	x	x	x
File node availability						x	x						

Model	Reference	Model	Reference
a	[Case72]	h	[Hugh73]
b	[Chan76]	i	[Buze74a]
c	[Morg77]	j	[Fost76]
d	[Rama70]	k	[Fost77]
e	[Aror73]	l	[Jone78]
f	[Chu73]	m	[Dowd77]
g	[Mahm76]		

APPENDIX B.

COMPUTATION OF THE NORMALIZATION CONSTANT, G

Gordon and Newell [Gord67] derived an expression for the steady state distribution for the number of customers at any service facility (node) in a general queuing network. For the central server model, let node 0 represent the CPU. Then, let Pr(c(0),c(1),...,c(NDEVS)) represent the steady state probability of c(j) customers being at the j-th node, and note that

$$\sum_{j=0}^{NDEVS} c(j) = DMP.$$

For the central server model, Gordon and Newell's result becomes

$$Pr(c(0),c(1),...,c(NDEVS)) = (1 / G(DMP)) * \prod_{j=0}^{NDEVS} x(j)^{c(j)}$$

where

x(1) = 1,

x(j) = LAMBDA * P(j) / MU(j), j=1,...,NDEVS,

and where

LAMBDA is the CPU burst rate,

MU(j) is the service rate of I/O device j, and

P(j) is the branching probability from the CPU to device j.

G(DMP) is a normalization constant to insure that the steady state probabilities sum to one. Dowdy [Dowd77] demonstrated that G(DMP) may be computed for real values of DMP using a method that is computationally dependent only on NDEVS. By his method:

$$G(DMP) = \sum_{j=1}^{NDEVS} \left[\frac{x(j)^{(DMP+1)} - 1}{x(j) - 1} \right] * \left[\frac{x(j)^{(NDEVS-1)}}{\prod_{\substack{k=1 \\ k \neq j}}^{NDEVS} (x(j) - x(k))} \right] .$$

CPU utilization is then computed as

UTILIZATION = G(DMP - 1) / G(DMP).

REFERENCES

[Abra73] Abramson, N. and Kuo, F. F. Computer Communication Networks, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

[Aho74] Aho, A. V., Hopcroft, J. E., and Ullman, J. D. The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974.

[Ames77a] Ames, J. E. Dynamic file assignment in a distributed database system. Ph. D. dissertation, Dept. of Computer Science, Duke University, Durham, NC, 1977.

[Ames77b] Ames, J. E., and Foster, D. V. Dynamic file assignment in a star network. Proceeding Computer Networks Symposium, Gaithersburg, MD, 1977.

[Ande75] Anderson, G. A., and Jensen, E. D. Computer interconnection structures: taxonomy, characteristics, and examples. Computing Surveys 7,4 (Dec 1975), 197-213.

[Aror73] Arora, S. R. and Gallo, A. Optimization of static loading and sizing of multilevel memory systems, JACM 20, 2 (April 1973), 307-319.

[Aspl76] Asplund, C. L. ASQ User's Manual, Department of Computer Science Tech. Report 78-4, Vanderbilt University, Nashville, TN, 1976.

[Bard78] Bard, Y. The VM/370 performance monitor. Comuting Surveys 10, 3 (Sept 1978), 333-342.

[Bond76] Bondy, J. A., and Murty, U. S. R. Graph Theory with Applications. American Elsevier, New York, NY, 1976.

[Brin73] Brinch-Hansen, P. Operating System Principles. Prentice-Hall, Englewood Cliffs. New Jersey, 1973.

[Buze71a] Buzen, J. P. Analysis of system bottlenecks using a queueing network model, Proceedings of ACM Sigops Workshop on System Performance Evaluation, Cambridge, Mass. April 1971.

[Buze71b] Buzen, J. P. Optimizing the degree of multiprogramming in demand paging systems, Proceedings IEEE Computer Science Conference, Sept 1971, 139-140.

[Buze71c] Buzen, J. P. Queuing network models of multiprogramming, Ph. D. Th., Div. of Engineering and Applied Physics, Harvard University, Cambridge, MA, May 1971.

[Buze73] Buzen, J. P. Computational algorithms for closed queueing networks with exponential servers, CACM 16, 9 (Sept 1973), 527-531.

[Buze74a] Buzen, J. P. and Chen, P. P. -S. Optimal load balancing in memory hierarchies, Information Processing 74.

[Buze74b] Buzen, J. P. and Goldberg, P. S. Guidelines for the use of infinite source queueing models in the analysis of computer system performance, AFIPS Vol 43 (1974), 371-374.

[Buze75] Buzen, J. P. Cost effective analytic tools for computer performance evaluation, Proc. IEEE COMPCON 1975, IEEE, New York, New York, 293-296.

[Buze78] Buzen, J. P. A queueing network model of MVS, Computing Surveys 10, 3 (Sept 1978), 319-33[illegible].

[Case72] Casey, R. G. Allocation of copies of a file in an information network, Proc AFIPS 1972 SJCC 40, 617-625.

[Case73] Casey, R. G. Design of tree networks for distributed data, Proc AFIPS 1973 NCC 42, 341-348.

[Chan76] Chandy, K. M. and Hewes, J. E. File allocation in distributed systems, Proc. Internat. Symp. on Computer Perf. Modeling, Measurement, and Eval., Cambridge, Mass, March, 1976,10-13.

[Chan78] Chandy, K. M., and Sauer, C. H. Approximate methods for analyzing queueing network models of computer systems, Computing Surveys 10, 3 (Sept 1978), 281-317.

[Chen75] Chen, P. P. -S. Queueing network model of interactive computing systems, IEEE Proceedings 63, 6 (June 1975),954-947.

[Chu69] Chu, W. W. Optimal file allocation in a multiple computer system, IEEE Transactions on Computers C-18,10 (Oct 1969),885-889.

[Chu73] Chu, W. W. Optimal file allocation in a computer network, In [Abra73], 82-94.

[Denn78] Denning, P. J., and Buzen, J. P. The opertional analysis of queueing network models, Computing Surveys 10, 3 (Sept 1978), 225-261.

[Depp76] Deppe, M. E., and Fry, J. P. Distributed data bases; a summary of research. Computer Networks 1,2 (Sept 1976).

[Dowd77] Dowdy, L. W. Optimal branching probabilities and their relationship to computer network file distribution, Ph. D. Th., Department of Computer Science, Duke University, Durham, NC, 1977.

[Dowd78] Dowdy, L.W. and Foster, D.V. The file assignment problem in perspective, Department of Computer Science Tech. Report 78-3, Vanderbilt University, Nashville, TN, 1978.

[Eswa74] Eswaran, K. P. Placement of Records in a File and File Allocation in a Computer Network, IFIP Conf. Proc.,Stockholm, Sweden (Aug 1974),304-307.

[Ferr78] Ferrari, D. Computer Systems Performance, Prentice-Hall, Englewood Cliffs, NJ, 1978.

[Fish78] Fisher, M. L. and Hochbaum, D. S. Data base location in computer networks. TR 78-10-09, Wharton School, University of Pennsylvania, Philadelphia, PA, October 1978.

[Fost74] Foster, D. V. File assignment in memory hierarchies, Ph. D. Thesis, University of Texas, Austin, Texas, August 1974.

[Fost76] Foster, D. V. and Browne, J. C. File assignment in memory hierarchies, Proc. Modeling and Perf. Eval. of Computer Systems, North-Holland Publishing Co., Amsterdam, (Oct 1976), 119-127. Also, Computer Science Tech. Report CS-1976-11, Duke University, Durham, NC, 1976.

[Fost77] Foster, D. V., Dowdy, L. W., and Ames, J. E. File assignment in a star network, Forthcoming in CACM. Systems and Information Science Tech. Rep. 77-3, Vanderbilt University, Nashville, TN, 1977.

[Gall76] Gallie, T. M., and Ramm, D. Computer Science/I: an introduction to structured programming, Kendall/Hunt, Dubuque, IA, 1976.

[Giam76] Giammo, T. Validation of a computer performance model of the exponential queueing network family, Acta Informatica 7, 2 (1976), 123-136.

[Gord67] Gordon, W. J. and Newell, G. P. Closed queuing systems with exponential servers. Operations Research 15,2 (April 1967), 254-265.

[Grah78] Graham, G. S., ed. Queueing network models of computer system performance, special issue, Computing Surveys 10,3 (Sept 78).

[Grap77] Grapa, E. and Belford, G. G. Some theorems to aid in solving the file assignment problem. CACM 20,11 (Nov 1977), 878-882.

[Hugh73] Hughes, P. H. and Moe, G. A structural approach to computer performance. Proc. AFIPS 1973 SJCC, 109-120.

[Horo76] Horowits, E., and Sahni, S. Fundamentals of Data Structures. Computer Science Press, Woodland Hills, CA, 1976.

[Igni76] Ignizio, J. P. Goal Programming and Extensions. Lexington Books, D. C. Heath and Co., 1976.

[Jack57] Jackson, J. R. Networks of waiting lines. Operations Research 5 (1957), 518-521.

[Jone78] Jones, L. G., Foster, D. V., and Krolak, P. D. A goal programming formulation to the file assignment problem. Department of Computer Science Technical Report 78-5, Vanderbilt University, Nashville, TN, 1978.

[Jone79a] Jones, L. G., and Foster, D. V. A heuristic solution to the file assignment problem. Proceedings IEEE SOUTHEASTCON 79, Roanoke, VA, April, 1979.

[Jone79b] Jones, L. G., and Foster, D. V. Toward a multiple copy file assignment model for files in a computer system. Proceedings Southeast Regional ACM Conf., Orlando, FL, April, 1979.

[Kane74] Kaneko, T. Optimal task switching policy for a multilevel storage system, IBM J. Res. Dev, July 1974, 310-315.

[Kane75] Kaneko, T. Optimization of a storage hierarchy under multiprogramming environment, IBM Research report RC 5222, Jan 1975.

[Klei75] Kleinrock, L. Queueing Systems, Volume 1: Theory. John Wiley and Sons, New York, New York, 1975.

[Klei76] Kleinrock, L. Queueing Systems, Volume 2: Computer Applications. John Wiley and Sons, New York, New York, 1976.

[Lee78] Lee, E. Y. S. Distributed data base design methodology. TRW Defense and Space Systems Group Tech. Report 30451-78-025, Redondo Beach, CA, 1978.

[Levi75] Levin, K. D. and Morgan, H. L. Optimizing distributed data bases- a framework for research. Proc AFIPS 1975 NCC 44, 473-478.

[Lips77] Lipsky, L. and Church, J. D. Applications of a queuing network model for a computer system. Computing Surveys 9, 3 (September 1977), 205-221.

[Mahm76] Mahmoud, S. and Riordon, J. S. Optimal allocation of resources in distributed information networks. ACM Transactions on Database Systems 1, 1 (Mar 1976), 66-78.

[Mill65] Miller, I., and Freund, J. E. Probability and Statistics for Engineers, Prentice-Hall, Englewood Cliffs, NJ, 1965.

[Moor72] Moore, F. R. Computation model of a closed queueing network with exponential servers. IBM J. Res. Dev. 16,€ (Nov 1972), 567-572.

[Morg77] Morgan, H. L. and Levin, K. D. Optimal program and data locations in computer networks. CACM 20,5 (May 1977), 315-322.

[Munt78] Muntz, R. R. Queueing networks: A critique of the state of the art and directions for the future. Computing Surveys 10, 3 (Sept 1978), 353-359.

[Peeb78] Peebles, R., and Manning, E. System architecture for distributed data management, Computer 11, 1 (Jan 1978), 40-47.

[Pric74] Price, T. G. Probability models of multiprogrammed computer systems. Ph. D. Th., Dept. of Electrical Engineering, Stanford University, Palo Alto, CA, Dec. 1974.

[Rama70] Ramamoorthy, C. V. and Chandy, K. M. Optimization of memory hierarchies in multiprogrammed systems. JACM 17,3 (July 1970), 426-445.

[Rose73] Rose, L. L., and Gotterer, M. H. A theory of dynamic file management in a multilevel store,International Journal of Computer and Information Sciences,Dec 1973.

[Rose75a] Rose, L. L., and Gotterer, M. H. File evaluation in auxiliary storage, International Journal of Computer and Informations Sciences, Sep 1975.

[Rose75b] Rose, L. L., and Gotterer, M. H. An analysis of file movement under dynamic file management strategies, BIT 15, 3 (1975),304-313.

[Rose78] Rose, C. A. A measurement procedure for queueing network models of computer systems. Computing Surveys 10, 3 (Sept 1978), 263-280.

[Schw78] Schwetman, H. D. Hybrid simulation models of computer systems. CACM 21,9 (Sept 1977), 718-723.

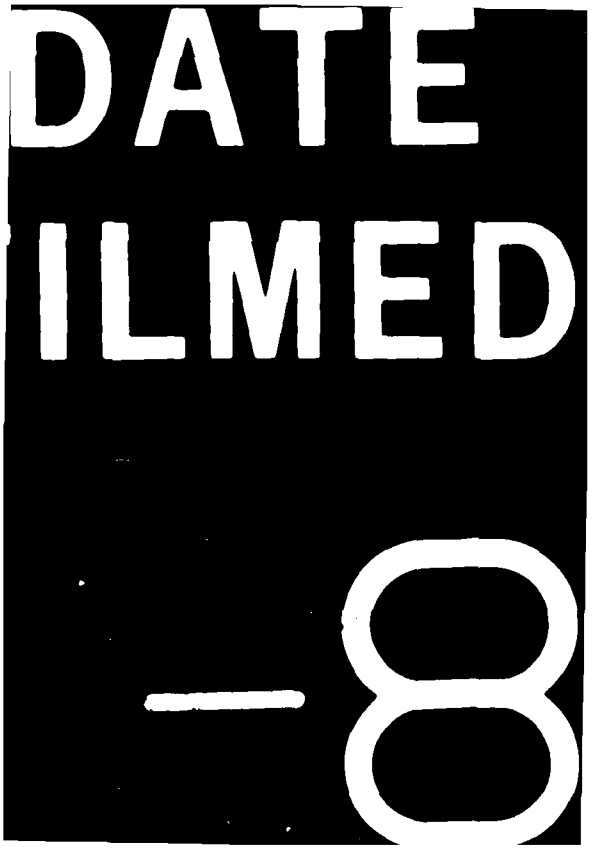
[Svob76] Svobodova, L. Computer Performance Measurement and Evaluation Methods: Analysis and Applications, Elsevier North-Holland, New York, New York, 1976.

[Taha71] Taha, H. A. Operations Research - an Introduction, The Macmillan Co., New York, New York, 1971.

[Taha76] Taha, H. A. Integer Programming - Theory, Application, and Computations, Academic Press, New York, New York, 1976.

[Wagn75] Wagner, H. M. Principles of Operations Research. Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

[Wong78] Wong, J. W. Queueing network modeling of computer communication networks. Computing Surveys 10, 3 (Sept 1978), 343-351.


DATE
ILMED
-8