

NASA-CR-193458

**EFFICIENT PARTITIONING AND ASSIGNMENT OF PROGRAMS
FOR MULTIPROCESSOR EXECUTION**

*11-5402
176753
p. 189*

Final Technical Report

Reporting Date: July 12, 1993

Project Summary

(NASA-CR-193458) EFFICIENT
PARTITIONING AND ASSIGNMENT ON
PROGRAMS FOR MULTIPROCESSOR
EXECUTION Final Technical Report, 8
Feb. 1989 - 1 Oct. 1992 (Toledo
Univ.) 189 p

N94-10942

Unclass

G3/61 0176753

Principal Investigator:

**Dr. Hilda M. Standley
Department of Computer Science and Engineering**

**The University of Toledo
Toledo, Ohio 43606**

NASA Lewis Research Center

Grant Number: NAG 3-975

February 8, 1989 - October 1, 1992

ABSTRACT

The general problem studied is that of segmenting or partitioning programs for distribution across a multiprocessor system. Efficient partitioning and the assignment of program elements are of great importance since the time consumed in this overhead activity may easily dominate the computation, effectively eliminating any gains made by the use of the parallelism. In this study, the partitioning of sequentially structured programs (written in FORTRAN) is evaluated. Heuristics, developed for similar applications are examined. Finally, a model for queueing networks with finite queues is developed which may be used to analyze multiprocessor system architectures with a shared memory approach to the problem of partitioning.

The properties of sequentially written programs form obstacles to large scale (at the procedure or subroutine level) parallelization. Data dependencies of even the minutest nature, reflecting the sequential development of the program, severely limit parallelism. The design of heuristic algorithms is tied to the experience gained in the parallel splitting.

Parallelism obtained through the physical separation of data has seen some success, especially at the data element level. Data parallelism on a grander scale requires models that accurately reflect the effects of blocking caused by finite queues. A model for the approximation of the performance of finite queueing networks is developed. This model makes use of the decomposition approach combined with the efficiency of product form solutions.

INTRODUCTION

Developments in hardware technology for parallel computers have far out-paced corresponding developments in software for these machines. Although parallel architectures have proliferated, the high-level language software developed for them has been for the most part architecture specific. Architectures vary in 1) the degree of the physical separation of non-similar program modules, 2) the amount of data parallelism, and 3) the topology of the network that links the pieces (program module and data). When a new architecture is developed, it usually dictates a new software paradigm which in turn calls for new high-level languages and, consequently, necessitates the rewriting of existing software solutions.

Of major concern is the massive amounts of existing software, written in traditional, sequential languages. The rewriting of this software represents an undertaking of monumental proportions. Being able to automatically convert this software to parallel forms to suit the new architectures would represent a major contribution to the effort to utilize parallel computers.

The "parallelizing" of existing sequential programs is not a novel idea. Much work has been done in intra-procedural analysis, typically examining array operations, affording a degree of fine-grained parallelism. Less understood is the potential for parallelism between procedures or subroutines. This work examines the latter, large-grained form of parallelism.

Partitioning a sequential program into parts, some of which may execute in parallel, does not lend

itself to analytic approaches that would produce an optimum result. Instead, heuristics (i.e. "rules of thumb") are employed with the aim of producing "good" results, although perhaps not optimum and not in all cases, using efficient decision algorithms. The heuristics may be products of applied "common sense" or the result of experience gained from observing the outcomes in different situations.

PARTITIONING SEQUENTIAL PROGRAMS FOR PARALLEL COMPUTATION

The utility of the data flow model of computation is studied in light of a general purpose model of parallel computation at the procedure level. Under data flow, all elements of a computation may execute in parallel with the exceptions indicated by data dependencies between procedures. The data dependencies may be denoted as edges on a data flow graph in which the nodes represent the procedures resulting from the partitioning of the program. Two types of parallelism are represented. The "spatial" parallelism, indicated by nodes that are not connected by data dependencies (or dependencies indicated by the transitive closure relationship of the data dependencies), and "temporal" parallelism, that exhibited by parallel computations of nodes that may be related by data dependencies but are permitted to be executing "at the same time" because they are operating on data at different stages in the data flow graph.

A major difficulty with converting an existing sequential program to the data flow format is determining which dependencies are valid and which may be ignored. Data that are used within a procedure represent valid dependencies. Data that are made available to a procedure (such as via subroutine parameters or common blocks), but not used, represent invalid (or nonexistent) dependencies. Data that is not needed may appear in parameter lists and in common blocks only for the programmer's convenience--it may have been easier to use a "copy" feature in an editor to reuse a line that held all of the necessary data--and more. Sorting out the valid dependencies from the invalid is a nontrivial problem.

The MLTCS3 - Supersonic Engine Fan Flutter Analysis FORTRAN code from NASA Lewis was analyzed for data dependencies existing between subroutines. The resulting data flow diagram is given in Figure 1. Spatial parallelism is not immediately evident, except between the routines, "mltz2" and "mlteta." The execution of these two routines does not significantly impact the entire program execution. Temporal parallelism is not apparent due to the iterative (looping) structure of the program. The potential contribution of some parallelism is determined to be completely outweighed by the overhead introduced by having to hierarchically structure the computation surrounding these two subroutines in order to expose the parallelism in the data flow model. While the data flow model may prove useful at the procedural level in computations offering specific structures, the problems encountered with MLTCS3 are not at all unique. This places the entire data flow approach in question when viewed as a model for inter-procedural computation in the general case. A model that requires much less structuring, such as an object-oriented model, may prove to provide a more natural transition from the sequential to the parallel.

HEURISTICS FOR PARTITIONING AND ASSIGNMENT

A study of the current literature found a number of heuristics used for different solution strategies. Heuristics included a variation on “branch and bound,” a predicted minimum, and selecting the local minimum [4]. A “First Fit Decreasing” heuristic [3] and heuristics involving “bottom-up clustering methods” and “top-down partitioning methods” [5], based upon graph decomposition techniques, are also used. Techniques that select local optima, also called Greedy Algorithms [7], are prevalent.

The development of heuristic-based algorithms must assume a model for partitioning and assignment. Until a suitable model is determined, the examination of heuristics will only produce a classification of heuristics. After a model is chosen, heuristics may then be developed.

AN EFFICIENT MODEL FOR CLOSED QUEUEING NETWORKS WITH FINITE QUEUES

Finite queueing techniques are an active and important area of research. Much of the interest in this area is generated by the need to model computer networks on all levels. Classical analysis methods assume the queue sizes to be infinite. Although an approximation, models built upon this assumption have performed well over a large class of systems. In many other situations, however, the effects of limited size (e.g. finite) queues strongly impact the performance of the system and modeling with only infinite queues is not sufficient.

Finite queue systems were first modeled by Markov chains. Using Markov chains, a state space is defined with each state describing the location of each customer in the network. The network is solved by solving the corresponding system of linear equations. Solutions involving such detailed techniques are impractical for large networks due to the large number of states and, consequently, the large number of computations.

Jackson [6] proved that for a special class of queueing networks with infinite queues, a network may be modelled analytically by use of a “product form” formula. This important result indicates that networks can be “solved” by simply multiplying together the steady-state probability density functions for the individual queues. Very efficient algorithms exist for networks that are product form. It is noted that networks with finite queues are not product form.

The method of network “decomposition,” described by the Decomposition Theorem [1], is an approach used to simplify the analysis of large networks. A network is decomposed into subnetworks and the subnetworks are analyzed individually. This is done by replacing a subnetwork by an equivalent server whose characteristics are such that the performance of the resulting network is the same as the original network. The method of decomposition was shown to produce exact results for product form networks [1, 2].

An approximation to a finite queueing network problem is presented as a dissertation, entitled “A Decomposition Approach to Finite Queueing Networks,” authored by Badie A. Taha and given as Appendix B. The approach presented decomposes a non-product form network into product form subnetworks. The specific network studied is a two-level finite queueing network, with a single server at the first level and m servers at the second level. The model for this network assumes that

the server at the first level views all servers at the second level as a single equivalent server with a variable service rate (the service rate is dependent upon the composite queue length), having a variable queue capacity. The notion of replacing a subnetwork by a single “flow-equivalent” server is justified by the Decomposition Theorem. Several finite, fixed queues may be replaced by one server with variable queue size based on the blocking and nonblocking possibilities of the server at level one.

The state space is grouped into subsets of states by collecting those states with the total number of customers, located at servers in the second level, less than or equal to some value, n . The probability of being in each subset of states is estimated and the performance measure (throughput) calculated for each subset using the product form solution.

The decomposition model is tested for a system consisting of four servers and three finite queues. Combinations of service times, finite queue sizes, routing probabilities, and the number of customers in the network form the bases for over 700 runs. The results give a relative error, when compared to an exact solution, of less than 1%. It is concluded that the decomposition approach and the use of the product form solution of infinite queue networks are feasible to apply to the analysis of finite queueing networks.

FUTURE RESEARCH DIRECTIONS

Other models are needed to provide an appropriate fit with the parallel execution of sequentially designed programs. The subroutine- and data-level partitioning will be evaluated with existing architectural designs for parallel computation. The finite queueing network model will be evaluated using more complex network interconnection structures. The outcome from such a model may determine near-optimal partitionings for sequential programs by suggesting heuristics tailored to the problem.

(DATA FLOW DIAGRAM)

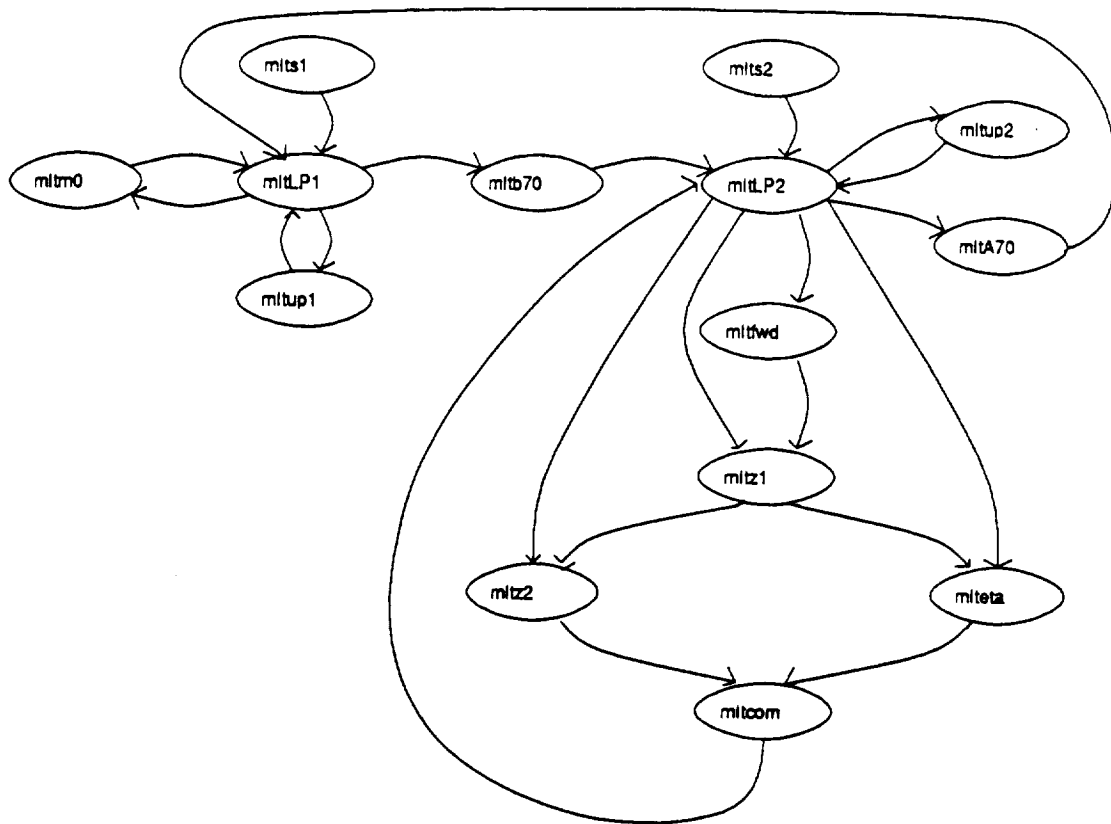


Figure 1: MLTCS3 - Supersonic Engine Fan Flutter Analysis (Data Flow Diagram)

BIBLIOGRAPHY

- [1] Chandy, K. M., Herzog, U., and Woo, L., "Parametric Analysis of Queueing Networks," *IBM J. Res. Dev.*, Vol. 19, January 1975, pp. 36-42.
- [2] Chandy, K. M., Herzog, U., and Woo, L., "Approximate Methods for Analyzing Queueing Network Models of Computing Systems," *Comp. Surveys*, Vol. 10, No. 3, Sept. 1978, pp. 281-317.
- [3] Dimitriadis, S., Karplus, W. J., "Scheduling the Solution of Ordinary Differential Equations on Multiprocessor Computers, Multiprocessors and Array Processors, 1988, The Society for Computer Simulation, pp. 58-66.
- [4] Dutta, A., Koehler, G., Whinston, A., "On Optimal Allocation in a Distributed Processing Environment," *Management Science*, Vol. 28, No. 8, August 1982, pp. 839-853.
- [5] Huff, S. L., "Preliminary Design for Complex Software Systems Using Graph Decomposition," *Proceedings of the International Conference on Cybernetics and Society*, IEEE 1980, pp. 479-484.
- [6] Jackson, J. R., "Jobshop-Like Queueing Systems," *Management Science*, Vol. 10, No. 1, October 1963, pp. 131-142.
- [7] Lo, V. M., "Heuristic Algorithm for Task Assignment in Distributed Systems," *IEEE Transactions on Computers*, Vol, 37, No. 11, November 1988, pp. 1384-1397.

APPENDIX A

“Queueing Network Models for Hierarchical Shared Memory Multiprocessor System”

by H. M. Standley and Badie A. Taha

presented at the

**Twenty-Second Annual Modeling and Simulation Conference, Pittsburgh, Pennsylvania, May,
1991, pp. 1203-1210.**

QUEUEING NETWORK MODELS FOR HIERARCHICAL SHARED MEMORY MULTIPROCESSOR SYSTEM*

H.M. Standley and B.A. Taha
Computer Science And Engineering Department
The University Of Toledo
Toledo, Ohio 43606

ABSTRACT

A multiprocessor shared memory organization, offering a recursively defined hierarchical structure, is modeled using a closed queueing network with finite queues. Memory modules of differing service characteristics are located at various levels in the hierarchy. One approximate solution of closed queueing networks with blocking can be obtained by forming an equivalent product form network with multiple chains of customers. Approximate solutions by decomposition to a flow equivalent service model that utilizes the recursive structure of the network is outlined.

Key words: Queueing networks, performance evaluation, memory interference, blocking, finite buffers.

INTRODUCTION

Shared memory multiprocessor computing systems have come to play a major role in both application and research areas of computer science. The parallelism offered by multiprocessor systems appears to represent the most promising way of obtaining the high-performance computing needed in many application fields. Characteristics such as fault tolerance, flexibility, functional upgrading, and cost effectiveness have spurred development of multiprocessor systems and a variety of multiprocessor architectures with different design alternatives have been proposed and implemented. The development of methodologies and tools for the prediction of the performance of multiprocessor architectures is growing in importance as architectural designs proliferate.

A multiprocessor system consists of a set of master modules (processors) and a set of slave units (memories and I/O modules) linked together by an interconnection structure. Master units are the system elements allowed to issue access requests to the interconnection structure for the transfer of data. Slave units receive access requests from master units and can accept and honor them according to a given service discipline such as first come first served (FCFS). Slaves units may have limited size buffers for waiting requests. The actual nature of the information transfer can be either a write request (write operation) or a read request (read operation). Performance may be degraded due to conflicts (interference) which occur whenever two or more processors attempt to access the same memory module simultaneously. Researchers involved with the design of large-scale shared memory multiprocessors [8, 10, 19] have found that although the performance of such systems is influenced in complex ways by many parameters, memory access time (including time spent waiting for access due to conflicts) is the major design factor influencing performance in such systems.

* Supported in part by NASA Lewis Research Center - Grant Number: NAG 3-975

The shared memory organization in multiprocessor architectures is evolving toward a complex hierarchy of storage media consisting of a small amount of fast memory (registers) followed by increasingly larger amounts of slower memory which may include one or more levels of cache, main memory, extended store, disks, and mass store. The focus of this paper is the problem of memory interference in shared memory multiprocessor systems in which the memory modules are organized in a hierarchy as in Figure 1. The objective is to develop analytic models of the effect of interference in a shared hierarchical memory multiprocessor system with blocking. The blocking mechanism, due to limited size waiting buffers (queues) at each of the memory modules in the network, provides strong motivation for the development of a solution to closed queuing networks with finite capacity buffers (closed queuing networks incorporate a constant number of circulating customers).

PREVIOUS WORK

In this section, the most general queuing networks used in modeling computer systems are reviewed. Several queuing networks characterized by the structure of the queue interconnection and by distribution of the arrival and service processes have been analyzed in the literature. Product form queuing networks are the most interesting networks due the fact that they are found to be solvable with relatively simple mathematical tools. This is because the solution of the network (probability distribution of customers over the network) is easily obtained by analyzing each station in the network in isolation and then combined them in a product form expression. This solution is then used to calculate all performance measurements. Product form queuing networks and solution of blocking networks are discussed in the this sections.

Jackson [12] showed that the product form solution [28] holds for networks in which customers are allowed to visit any station and cycles between stations are allowed, i.e., in which a customer is allowed to visit several times the same station before leaving the network. These networks are known as Jackson networks. They are analyzed by recognizing that the time behavior of the state of the network (distribution of customers over stations) is a Markov chain and that the global balance equations can be written directly to balance the flow in and out of any network state. A particular case of Jackson networks is represented by networks in which no arrivals from the outside world are allowed (closed networks), so that the number of customers in the network, N , is kept constant. Gordon and Newell [9] analyze this class of networks, showing that the solution for the steady state distribution of customers in the network is also of product form type. The balance equations in this case are simpler than the equations of Jackson's, since the arrivals from the outside world and departures toward the outside world are not allowed. The development and analysis of one of the most general queuing networks was due to the combined efforts of Baskett, Chandy, Muntz, and Palacios [2]. The result is known as the BCMP theorem, bearing their initials. This class of networks contains an arbitrarily but finite number M of service stations. There is an arbitrarily but finite number R of different classes of customers. Customers may travel through the network and change class according to transition probabilities. Thus a customer of class r who completes service at station i may next require service at station j in class s with probability $p_{ir,js}$. Service stations may have different service time distributions and may depend on the number of customers at the station. Multiple servers at each station and different service disciplines for managing the queues are allowed.

All solution described above assume infinite queues [6,13,20]. In real life systems, the storage space is always finite. Hence a more realistic model of such systems requires modeling queues with finite capacity. An important feature of queuing networks with finite queues is that the flow of customers through a server may be momentarily stopped when another node in the network reaches its capacity. That is a called *blocking*. This type of network is difficult to analyze since the steady state queue length distribution has not been shown to have a product form solution as in networks without blocking. A comprehensive survey on queuing networks with blocking can be found in [1,15,17,18,26].

THE MODEL

Consider a multiprocessor system in which processors generate memory access requests to memory modules that are organized in a hierarchy. A processor generates requests to a memory module at the first level through the crossbar interconnection structure that gives to any processor the capability of accessing any memory module at any time without interfering with other processors, provided that the memory module is free. No contention for the use of the crossbar switch exists in this case. A processor attempting to access a memory module can be either served immediately or delayed in a waiting buffer due to another processor accessing the same memory module. Requests that are satisfied are terminated and the requesting processor resumes its internal processing. Requests that are not satisfied proceed to the next higher level in the hierarchy of memory modules unless blocking occurs due to the size of buffers at that level. It is assumed that the higher the level, the greater its access time and the greater its storage capacity. Despite the fact that the motivation for this research is the multiprocessor network with shared memory hierarchy connected as a tree, the models considered in this research assume an arbitrarily connected, finite buffer, closed queuing network.

A request to a memory module is processed in two steps: (1) checking the availability of the requested data and (2) servicing the memory access if the data are available, leading to the model of the two single server service stations, m_a and m_b , as in Figure 2. A controller server, m_a , has a single queue with limited capacity, and tests the availability of the requested data. Requests that require service from the same memory module (if the data are available) are directed to a second server with probability α . With probability $1 - \alpha$ the request is directed to a memory module at the next higher level in the hierarchy. The second server, m_b , also has a finite queue of waiting requests. Server m_a continues to serve other requests as long as the queue of the second server, m_b , is not full, or if the request needs to be routed to the next higher level in the hierarchy and the queue of the server at that level is not full. Otherwise, if server m_b is requested and its queue is full, or if the request needs to be routed to the next higher level in the hierarchy and the queue of the server at that level is full, server m_a provides no service (i.e. is blocked) to new requests until the server m_b or a server at the next higher level has completed its servicing and a place in the corresponding queue becomes available. This type of blocking is referred to as *transfer blocking* (TB) [1] or *blocking before service* (BBS) [15].

The multiprocessor system with N processors and M memory modules (Figure 1) is then modeled as a closed queuing network consisting of $2 \times M$ single server finite capacity service stations that represent memories, and one service station having N servers (processors). No queuing is allowed (or required) at the processors' station. Memory requests generated by the processors' station represent customers circulating in the network. A closed queuing network with limited size queues for the case of N processors and seven memory modules in three levels ($N, 7, 3$) is provided in Figure 3.

ANALYSIS OF BLOCKING NETWORKS BY CYCLES

The network K is defined in terms of the following parameters: There are $M + 1$ service centers, R classes of customers, N_r customers of class r and T routing cycles. We assume that $R = T$. Memory requests (customers) proceed through the network K according to a transition matrix with elements $p_{m'r'}$, the probability that a request of class r completing service at server m will next go to server m' and change its class membership to r' . All cycles are closed, the number of requests is held constant at N_r . All servers have FCFS service discipline and queue-dependent service rate. Each request has an associated work demand which is assumed to be drawn from an exponential distribution. All classes have the same distribution. Each server has a limited capacity for waiting requests, C_i (i.e. finite queues).

Consider a special case of the closed queuing network with the parameters $R = 1$ and $T = 1$ under TB blocking. For $1 \leq N \leq \min C_i$, $i = 0, 1, \dots, M$, there is no blocking and the network has a product form queue length distribution (Gordon and Newell [9]). Onvural and Perros [16] showed that when

$N = \min (C_i, i = 1, 2, \dots, M) + 1$, the network has product form queue length distribution.

The same result can be applied in the case of closed cycles in the same network with multiple classes of requests. A closed cycle in the network is defined as a directed path that starts and ends at the same node. Since all processors are identical it is possible to divide the network into cycles each cycle represents a set of servers visited by a class of requests. Requests are allowed to change class type to facilitate the fact that a processor could generate requests to any memory module. By limiting the number of requests in each class N_i to the minimum station capacity in the corresponding cycle, the resulting network will still have a product form solution. In a tree structured network a cycle can be identified starting with a service node at level one and ending with the service node at the last level. The number of stations in each cycle will be the same as the number of levels in the network. Two cycles can share the same station. All approximation methods that can be used in multiple closed chains for networks with infinite buffers can be used to approximate the solution of the closed network defined in this paper.

ANALYSIS USING DECOMPOSITION

Consider the network defined above without considering multiple routing cycles or different classes of customers, $R = T = 1$. One approach toward the development of approximation algorithms is decomposing the network into individual queues or subnetworks and analyzing them in isolation. Exact decomposition of queuing networks requires state dependent arrival and service rates [15]. For closed queuing networks, the dependencies of the parameters of a queue in isolation is very strong due to the fixed population of customers. This section outlines two approximation procedures based on a recursive approach of decomposing the network into subnetworks that can be analyzed in isolation.

The main objective of the analysis is to reduce entire network into an equivalent network of two stations -each having a single load dependent server- by calculating the flow equivalent service model of servers 1, 2, ..., $2M$. The basic approach is to replace a subnetwork of queues by a single ("composite") queue which is *flow-equivalent* to the subnetwork, i.e., the customer flow through the composite queue is equal to the customer flow through the subnetwork. This can be done repeatedly, replacing subnetworks (including those with composite queues) by flow-equivalents until the solution to the resulting network is attainable. Next, we outline two approximate analysis procedure for closed queuing networks with blocking. The flow equivalent service rate model will be used at different levels in the network.

Procedure DECOMP 1

The analysis by decomposition is developed according to the following principles.

- 1) Find an equivalent network with infinite queue capacities. This can be done by finding another network, K^* , with the same structure but different number of customers such that the throughput of K^* approximates that of K [1].
- 2) Decompose the queuing network from step 1 into subsystems of infinite queue capacities. The tree structured queuing network is simply decomposed into subnetworks of three nodes starting at the lower level (level 1).
- 3) Analyze of the subsystems in isolation. Calculate the flow equivalent service model for each subnetwork of three nodes.
- 4) Replace each subsystem by its composite load-dependent (flow equivalent service) server.
- 5) Repeat until all servers in the hierarchy are replaced by one composite server.

Procedure DECOMP 2

A variation of step 1 above is to reconfigure the network by adjusting the transition probabilities in the network based on approximating the probability of a server at level l being blocked by a server at level $l+1$. The blocking probability can be viewed as the probability of a customer staying in the same place (blocked server) and the service time for the blocked server will be changed to the service time of the blocking server at level $l+1$ as long as it is blocked [26]. The result will be another network, K^* , with the same structures but different transition probabilities.

The blocking probabilities are obtained as follows: Let $\mathbf{n} = (n_0, n_1, \dots, n_M)$ be a state vector representing the number of customers at each station, and $p(\mathbf{n})$ be the probability that the network is in state \mathbf{n} . Calculate $p(\mathbf{n})$ using a product form solution algorithm [9] for queuing networks without capacity limits. Now, if there is a transition from server i at level l to server j at level $l+1$, $p_{ij} \neq 0$, then the probability that server i is blocked is equal to the probability that server j is full. This is equal to the sum of the probabilities that the number of customers at server j is greater than or equal to its capacity or that it too is blocked.

Adjusting the transition probabilities and calculating the blocking probabilities are performed for all levels as follows:

- 1) Calculate $p(\mathbf{n})$ for the whole network disregarding the queue capacities.
- 2) Calculate the probability that the server at level L is full.
- 3) Adjust the transition probabilities for the servers at level $L-1$ based on the blocking probabilities as a result of the server at level L is full.
- 4) Repeat steps 1, step 2 for level $L-1$, and step 3 for level $L-2$

This is to be performed iteratively until the transition probabilities at level 1 are adjusted. Once the transition probabilities are adjusted the resulting network is non blocking and the queue capacities are considered infinite. Steps 2 through 5 from *procedure DECON*E are then carried out.

CONCLUSION

Models of a shared memory multiprocessor system are outlined in which multiple processors compete for access to a shared memory organized as a hierarchical network. Memory modules at different levels in the hierarchy possess different operating characteristics such as the amount of storage and access time. Requests for memory access are specified by variable distribution and queued in finite queues at each memory module.

The classical queuing network model (Jackson 1963, Gordon and Newell 1967) is adapted to accommodate the hierarchical network with finite queues. Analysis procedures based on reconfiguring the blocking network are outlined. A mean value analysis by cycles is being tested and compared to simulation runs and to the results of Suri and Diehl [26] and Akyildiz [1] on closed queuing networks with blocking.

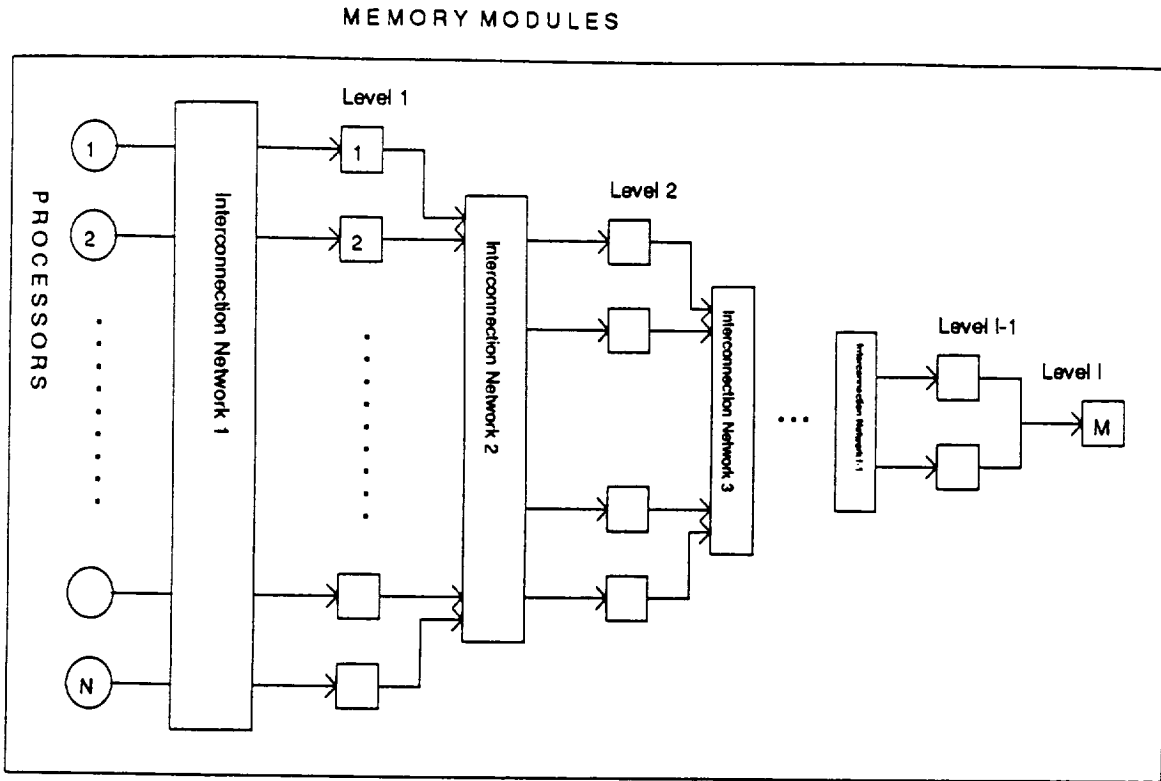


Figure 1.

A Multiprocessor with Hierarchical Shared Memory

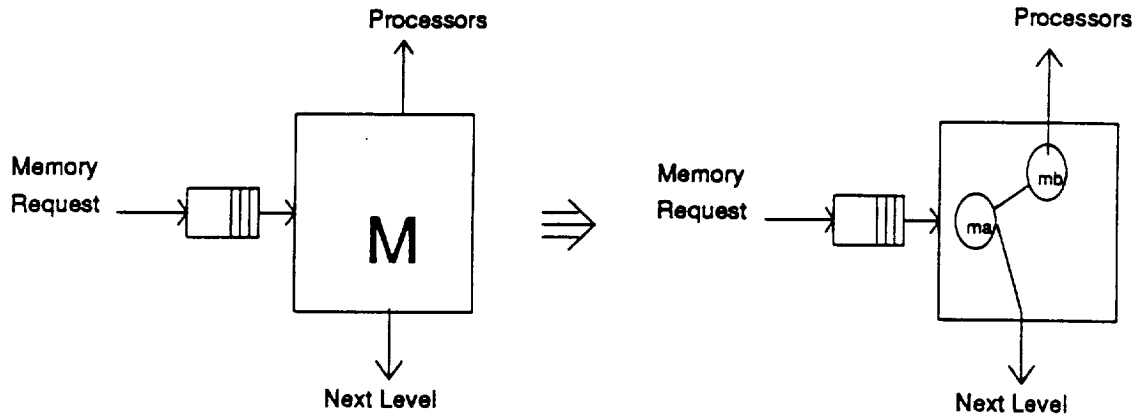
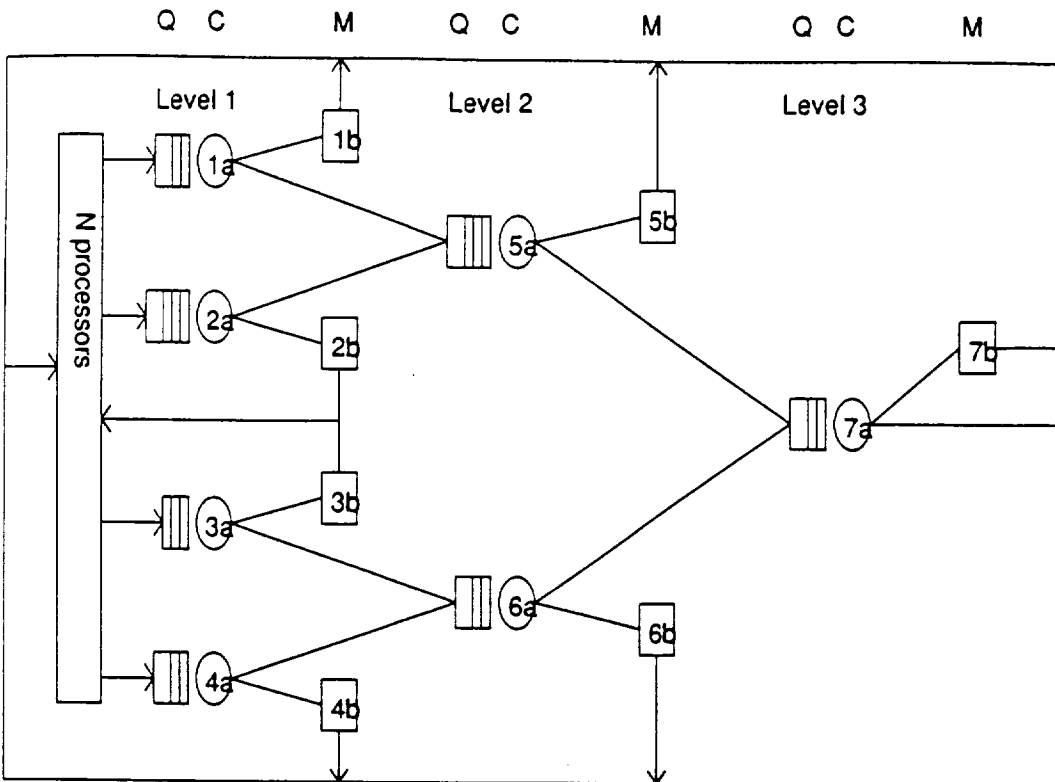


Figure 2

Modeling One Memory Module with two Servers



Q: Queues, C: Controles, M: memories (N Processors & M=7 Memories)

Figure 3.

Queuing Model of a Multiprocessor System (N,7,3)

REFERENCES

- [1] Akyildiz I. F., "Product Form Approximations for Queuing Networks with Multiple Servers and Blocking," IEEE Trans. Comp., Vol. 38, 1989, pp. 99-115.
- [2] Basket F., Chandy K. M., Muntz R. R., and Palacios F. G., "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," J. ACM, Vol. 22, No. 2, April 1975, pp. 248-260.
- [3] Baskett F., and Smith A. J., "Interference in Computer Systems with Interleaved Memory," Comm. ACM, Vol. 19", (June, 1976), pp. 327-334.
- [4] Bhandarkar D. P., "Analysis of Memory Interference in Multiprocessors," IEEE Trans. Comp., Vol. C-24, (Sept., 1975), pp. 897-908.
- [5] Bhandarkar D. P., and Fuller S. H., "Markov Chain Models for Analyzing Memory Interference in Multiprocessor Computer Systems," Proc. 1st Annual. Symp. on Comp. Arch., Univ. of Florida, Gainesville, FL, (Dec., 1973).
- [6] Buzen J. P., "Computational Algorithms for Closed Queuing Networks with Exponential Servers," Comm. ACM, Vol. 16, No. 9, September 1973, pp. 527-531.

- [7] Fukuda A., "Equilibrium Point Analysis of Memory Interference in Multiprocessor System," IEEE Trans. Comp., Vol. 37, No. 5, (May, 1988), pp. 585-593.
- [8] Gajski D., Kuck D., Lawrie D., and Sameh A., "CEDAR - A Large Scale Multiprocessor," Proc. of the ICPP, August 1983, pp. 524-529.
- [9] Gordon W. J., and Newell G.F., "Closed Queueing System with Exponential Servers," Oper. Res., Vol. 15, No. 2, April 1967, pp. 245-255.
- [10] Gottlieb A., Grishman R., Kruskal C., McAuliffe K., Rudolph L., and Snir M., "The NYU Ultra-computer - Designing a MIMD, Shared Memory Parallel Machine," IEEE Trans. Comp., Vol. C-32, (Feb., 1983), pp. 175-189.
- [11] Hoogendoorn C. H., "A General Model for Memory Interference in Multiprocessors," IEEE Trans. Comp., Vol. C-26, (Oct., 1977), pp. 998-1005.
- [12] Jackson J. R., "Jobshop-Like Queueing Systems," Management Science, Vol. 10, No. 1, October 1963, pp. 131-142.
- [13] Lemoine A. J., "Networks of Queues- A Survey of Equilibrium Analysis," Management Science, Vol. 24, No. 4, December 1977, pp. 464-481.
- [14] Mudge T. N., and Makrucki B. A., "Probabilistic Analysis of a Crossbar-Switch," Proc. 9th Symp. on Comp. Arch., Austin, Texas, April 1982.
- [15] Onvural R. O., "Survey of Closed Queueing Networks with Blocking," ACM Comp. Surveys, Vol. 22, No. 2, June 1990, pp. 83-121.
- [16] Onvural R. O., and Perros H. G., "Throughput Analysis in Cyclic Queueing Networks with Blocking," IEEE Trans. Soft. Eng., SE-15, No. 6, 1989, pp. 800-808.
- [18] Perros H. G., "Queueing Networks with Blocking: a bibliography," Perf. Eval. Rev., Vol. 12, No. 2, 1984, pp. 8-12. Perfo
- [18] Perros H. G., and Altiok T., "Approximate Analysis of Open Networks of Queues with Blocking: Tandem Configuration," IEEE Trans. on Soft. Eng., Vol. SE-12, March 1986, pp. 450-462.
- [19] Pfister G. F., Brantley W. C., George D. A., Harvey S. L., Kleinfelder W. J., McAuliffe K. P., Melton E. A., Norton V. A., and Weiss J., "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture," Proc. of the ICPP, Aug. 1985.
- [20] Reiser M., and Lavenberg S. S., "Mean-value Analysis of Closed Multichain Networks," J. ACM, Vol. 27, April 1980, pp. 313-322.
- [21] Rau B. R., "Interleaved Memory Bandwidth in a Model for Multiprocessor Computer System," IEEE Trans. Comp., Vol. C-28, (Sept., 1979), pp. 678-681.
- [22] Sethi A. S., and Deo N., "Interference in Multiprocessor Systems with Localized Memory Access Probabilities," IEEE Trans. Comp., Vol. C-28, (Feb., 1979), pp. 157-163.
- [23] Skinner C., and Asher J., "Effect on Storage Contention on System Performance," IBM Syst. J., Vol. 8, (1969), pp. 319-333.
- [24] Smilauer B., "General Model for Memory Interference in Multiprocessor and Mean Value Analysis," IEEE Trans. Comp., Vol. C-34, (Aug., 1985), pp. 744-751.
- [25] Strecker W. D., "Analysis of Instruction Execution Rate in Certain Computer Structures," Ph.D Dissertation, Carnegie-Mellon University, Pitsburg, Pennsylvania, 1970.
- [26] Suri R., and Diehl G., "A Variable Buffer Size Model and its Use in Analyzing Closed Queueing Networks with Blocking," Management Science, Vol. 32, pp. 206-225, February 1986.
- [27] Taha B. A., and Standley H. M., "A General Model for Memory Interference in a Multiprocessor System with Memory Hierarchy," Proc. of the ICPP, August 1989, Vol. I, pp. 225-232.
- [28] Takahashi Y., Miyahart H., and Hasegawa T., "An Approximation Method for Open Restricted Queueing Networks," Oper. Res., Vol. 28, May-June 1980, pp. 594-602.
- [29] Yen D. W. L., Patel J. H., and Davidson E. S., "Memory Interference in Synchronous Multiprocessor Systems," IEEE Trans. Comp., Vol. C-31, Nov., 1982, pp. 1116-1121.

APPENDIX B

“A Decomposition Approach to Finite Queueing Networks”

by

Badie A. Taha

The University of Toledo

March 1993

THE UNIVERSITY OF TOLEDO

COLLEGE OF ENGINEERING

DECEMBER 8, 1992

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY
SUPERVISION BY Badie Ahmad Taha
ENTITLED A Decomposition Approach to Finite Queueing Networks
BE ACCEPTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF Doctor of Philosophy in Engineering Science

THESIS ADVISOR DR. Hilda M. Standley

CHAIRMAN OF DISCIPLINE DR. Roger J. McNichols

Recommendation concurred:

R. J. McNichols
R. J. McNichols
Donald J. King
Hilda M. Standley

Committee

on

Final Examination

A Dissertation

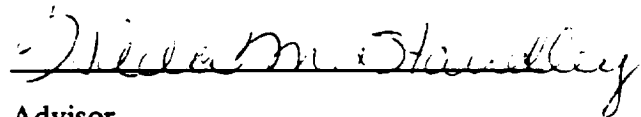
entitled

A Decomposition Approach to Finite Queueing Networks

by

Badie Ahmad Taha

as partial fulfillment of the requirements of
the Doctor of Philosophy Degree in
Engineering Science



Advisor

Dean of the Graduate School

The University of Toledo

March 1993

ACKNOWLEDGMENTS

I would like to extend my great appreciation and sincere gratitude to my advisor Dr. Hilda M. Standley for her valuable guidance, supervision, kindness, and generous assistance throughout the various stages of this research project.

I would also like to extend my thanks and appreciation to my research committee members: Dr. Donald Ewing for his encouragement in starting the graduate program at the University of Toledo, Dr. Roger J. McNichols who gave useful advice and under whom I had the good fortune of learning "Queueing Theory", and Dr. Kenneth Wolf for his insightful comments and support.

My special thanks and appreciation go to the Computer Science and Engineering Department under the leadership of both Dr. Donald Ewing and Dr. Hilda M. Standley for providing me with the financial support by means of a research assistantship and giving me the opportunity to teach several courses during my graduate study.

I can find no words to express my deepest gratitude and love for my family and friends, for without their encouragement and good-wishes this achievement would have been impossible.

DEDICATION

To:

my father, mother, brothers and sisters,

who loved and supported me

my wife ZUHUR,

who helped and encouraged me

my children, HANEEN, OMAR, RUBA, MOHAMAD, and AHMAD

who survived remarkably well without my continuous attention

I dedicate this dissertation

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	i
DEDICATION	ii
TABLE OF CONTENTS	iii
LIST OF FIGURES	vi
LIST OF TABLES	ix
ABSTRACT	x
Chapter	
1. INTRODUCTION	1
1.1 Main Objective	4
1.2 Review of Product Form Networks	5
1.3 Exact and Approximate Solution by Decomposition	13
2. BACKGROUND ON FINITE QUEUE NETWORKS	18
2.1 Definitions of Blocking	18
2.1.1 Transfer Blocking	19
2.1.2 Service Blocking	21
2.2 Complexity of Finite Queue Networks	24
2.2.1 A Non Product Form Networks	24
2.2.2 Dependency Between Servers and Decomposition	27
2.3 Previous Work	29
2.3.1 Exact Analysis	29
2.3.2 Approximate Analysis	32

2.4 Other Approaches to Modeling Finite Queue Networks	33
2.4.1 State Dependent Routing Networks	33
2.4.2 Reversible Routing Networks	34
2.5 Queue Equivalency and Decomposition	35
3. ANALYSIS OF FINITE QUEUE NETWORKS BY DECOMPOSITION	39
3.1 Two-Level Finite Queue Networks	39
3.1.1 Network Definition	43
3.1.2 Performance Measurements	40
3.2 Decomposition by Grouping of States	49
3.3 Probabilities of the Queue Capacities	53
3.3.1 Identification of Subsets of States	56
3.3.2 Ratios of the Group Probabilities π_n	57
3.3.3 Generalization to Multiple Servers at Level 2	60
3.4 Computation Procedure of Performance Measurements	63
3.5 A Complete Example	67
4. MODEL EVALUATION	74
4.1 Comprehensive Test	74
4.2 Implementation Issues	79
4.2.1 Ease of Implementation	79
4.2.2 General Use Of The Method	79
4.3 Summary	82
5. CONCLUSIONS & FUTURE RESEARCH	133
5.1 Introduction	133
5.2 Summary and Contribution	134
5.3 Conclusion	134

5.4 Future Research	134
5.4.1 Non-Exponential Service time Distribution	135
5.4.2 Multiple Class Networks	135
APPENDIXES	137
APPENDIX A Modeling Concepts In Queueing Networks	137
APPENDIX B Product Form Algorithm I	140
APPENDIX C Product Form Algorithm II	141
APPENDIX D General Decomposition	143
APPENDIX E An Induction Proof	154
BIBLIOGRAPHY	157

LIST OF FIGURES

Chapter 1	1
Figure 1.1 Two-Level Finite Queueing Network	4
Figure 1.2a Closed Product Form Queueing Network	7
Figure 1.2b State Transition Diagram	7
Figure 1.3a Queueing Network Model	8
Figure 1.3b Infinitesimal Generator	9
Figure 1.4a Model of Timesharing System	17
Figure 1.4b Model of Computer System	17
Figure 1.4c Overall Model of Terminals and System	17
 Chapter 2		 18
Figure 2.1a Two-Server Finite Queueing Network	20
Figure 2.1b State Transition Diagram	20
Figure 2.2 State Transition Diagram	22
Figure 2.3b State Space	24
Figure 2.3a Three-Server Finite Queueing Network	26
Figure 2.3c A Portion of the Transition Diagram	26
Figure 2.4a Three-Server Finite Network	28
Figure 2.4b Network of Servers 2 and 3	28
Figure 2.4c Network with Aggregated Server	28
Figure 2.5a Two-Server Finite Network	31
Figure 2.5b Equivalent Infinite Network	31
Figure 2.6 Two-Level Finite Network	37
Figure 2.7 Flow-Equivalent Queue-Equivalent Model	38
 Chapter 3		 39
Figure 3.1a Two-Level Finite Network	41

Figure 3.1b Equivalent Network of Two Servers	42
Figure 3.2a Two-Level Finite Network	43
Figure 3.2b State Space	44
Figure 3.3a Two-Level Finite Network	49
Figure 3.3b State Space	49
Figure 3.3c State for Subset 2	51
Figure 3.3d State for Subset 3	52
Figure 3.3e State for Subset 4	45
Figure 3.4 State Space Diagram with 4 Customers	57
Figure 3.5a Infinite Network (level 2)	58
Figure 3.5b State Space	58
Figure 3.6a Finite Network (level 2)	60
Figure 3.6b State Space	60
Figure 3.7a States Associated with Subset $i-1$ (blocked states)	61
Figure 3.7b States Associated with Subset i (unblocked states)	61
Figure 3.8a Infinite Network (level 2)	61
Figure 3.8a States Associated with Subset $i-1$	61
Figure 3.8b States Associated with Subset i	62
Figure 3.8c States Not Associated with any Subset	62
Figure 3.9a Network Example	67
Figure 3.9b State Space	68
Chapter 4	74
Figure 4.1 Test Example with ($M=4$)	76
Figure 4.2 Asymmetric Queues	77
Figure 4.3 Four Servers at Level 2	78
Figure 4.4 Finite & Infinite Queues	81

APPENDIX D	137
Figure D.1 Queueing Network Model	141
Figure D.2 State Space Diagram & State Probabilities	145
Figure D.3 Queueing Network Model (server 1 shorted)	146
Figure D.4 Equivalent Queueing Network	147
Figure D.5 State Space Diagram & State Probabilities	153
Figure D.6 State Probabilities & Normalized State Probabilities	153
Figure D.7 State Probabilities of the Aggregated Servers	153

LIST OF TABLES

Chapter 4	74
Tables 1-12 Test Model with ($N=4, c_2=1, c_3=1, c_4=1$)	83
Tables 13-24 Test Model with ($N=4, c_2=2, c_3=2, c_4=2$)	95
Tables 25-36 test Model with ($N=4, c_2=1, c_3=2, c_4=1$)	107
Tables 37-48 Test Model ($N=4, c_2=1, c_3=2, c_4=3$)	119
Tables 49-50 Test Model with ($N=8, c_2=6, c_3=1, c_4=2$)	131
Tables 51-52 Test Model ($N=4, 6, 8, c_2=1, c_3=1, c_4=1, c_5=1$)	132

An Abstract of
A DECOMPOSITION APPROACH
TO FINITE QUEUEING NETWORKS

Badie Ahmad Taha

Submitted in partial fulfillment
of the requirements of the
Doctor of Philosophy Degree in Engineering Science

The University of Toledo
March 1993

This research proposes to combine the standard decomposition-aggregation approach and product form solution methods to solve queueing networks with blocking due to finite queues. This involves developing a method called the flow-equivalent, queue-equivalent server. This server is both the service rate equivalent as well as the queue size equivalent for a subnetwork. It is shown how this server forms the equivalent for a subnetwork, how to solve a network that contains a variable queue size and how to generate the parameters for the server. This approach differs from others in that it uses the aggregation product form solution methods.

The equivalent server is used to analyze networks of two levels where level one consists of one server and level two consists of more than one server. The server at level one can be blocked by any server at level two due to queue capacities. Comprehensive test results are included.

CHAPTER 1

INTRODUCTION

Queueing models are useful tools for analyzing systems in which conflicts develop when several entities try simultaneously to access the same resource. The behavior of many physical systems is characterized by the presence of several congestion points, corresponding to the sharing of different types of resources. In these cases it is difficult to represent the complex behavior of the system with a single queue model. A more appropriate model is the queueing network, i.e., a network of interconnected queues that, in general, behave in an interdependent manner [20, 26, 30, 40].

Queueing network analysis techniques have been applied to telephone networks, manufacturing systems, computer systems, and communication networks [3, 5, 11, 14, 23, 31, 38]. Early analytical results were obtained for restricted systems. The restrictions have been relaxed to include general service times, closed and mixed systems, scheduling disciplines other than first-in first-out (such as last-come first-served or service based on priority), variable rate servers, and multiple classes. Some solutions are exact [10, 13, 24]; others provide approximate answers [6, 39]. The classical analysis of queueing networks assumes the queue sizes to be infinite and the resulting models have worked well for a large class of systems. For certain systems, however, the effect of a finite queue size can be considerable [32]. In queueing networks with finite queues, if the queue to which a customer attempts to go is full, the customer is forced to remain at the current server and the server is not allowed to begin service for

another customer (it is blocked) until a space is available at the destination server's queue.

The amount of research effort directed toward analyzing the effects of finite queues in queueing networks reflects the importance of this topic [1, 2, 7, 28, 33-37, 43-46]. A number of different network models, modeling different real systems, have been studied. The quality of the analysis is judged by a number of factors: the behavior of the model compared to the real network, the extent to which results may be generalized, and the complexity of the computation. The researcher modeling a real system must consider all factors to find a suitable model, in terms of size, accuracy, and efficiency.

This research attempts to develop an approximation that combines two of the more powerful techniques of queueing network theory, namely the product form solution and decomposition methods, extending the result to a class of finite queueing networks. Decomposition is a technique in which a network is broken into a number of smaller and simpler networks. The smaller networks are then solved separately and efficiently using a product form solution. Each subnetwork is replaced by a flow-equivalent server whose characteristics are such that the performance of the resulting network is the same as the original network. The results of the analysis by decomposition and flow-equivalence have been shown to be exact for product form networks [9, 10, 12, 13]. The class of product form networks, however, does not include networks with finite queues [15].

A method for the analysis of finite queueing networks is proposed based upon decomposition and a queueing structure in which the queue size varies over time. This structure is derived from one server's view of the rest of the network. The method developed in this research combines the efficiency of the product form solutions with

the generality of decomposition to attack different types of finite queueing networks. The main idea in the development is the formation of one server's view of the rest of the network and the development of the tools needed to model this view.

Discussion of finite queueing networks with blocking is presented in Chapter 2. The discussion covers the reasons for the complexity involved in analyzing queueing networks with finite queues. Previous approaches to modeling finite queueing systems are reviewed. Chapter 3 presents the major contribution of this work an approximation method for the analysis of finite queueing networks based upon decomposition. This method allows us to combine the approaches of product form networks and decomposition. "*ProductForm*" [11] queueing networks are associated with many developments in general networks due to the fact that they are solvable with relatively simple mathematical tools. This is because the solution of the network is obtained by analyzing each service station in the network in isolation and then combining the results in a product form expression. The resulting method that is developed combines the speed of the product form solution with the generality of decomposition to solve finite queueing network problems. Comprehensive examples are included and the analytical results are compared with exact solution.

Model evaluation and some of the implementation issues are discussed in chapter 4. Chapter 5 includes a summary of the work done and an outline of areas where the analysis may be extended.

The main objective of this research is stated next followed by an overview of product form networks and decomposition. Some modeling concepts in queueing systems are reviewed in APPENDIX A.

1.1 Main Objective

Consider a closed queueing network in which a fixed number of customers are circulating between multiple limited-queue-size servers, i.e. the number of customers allowed at any service station is limited (Figure 1.1). This number includes the queue size and the customer in service. This research assumes that when the queue at server j is full and a customer finishes service at server i and needs to be routed next to server j , the customer at server i cannot leave that server. Server i is said to be *blocked*. The problem here is that a condition at server j (queue is full) has shut down service at server i , so that the requirements (i.e., independence) of the simple analysis no longer hold. It is this nonclassical feature that complicates the analysis of limited queue networks. This research presents an approximate solution to finite queueing networks which allows utilization of existing simple solutions for infinite queueing networks.

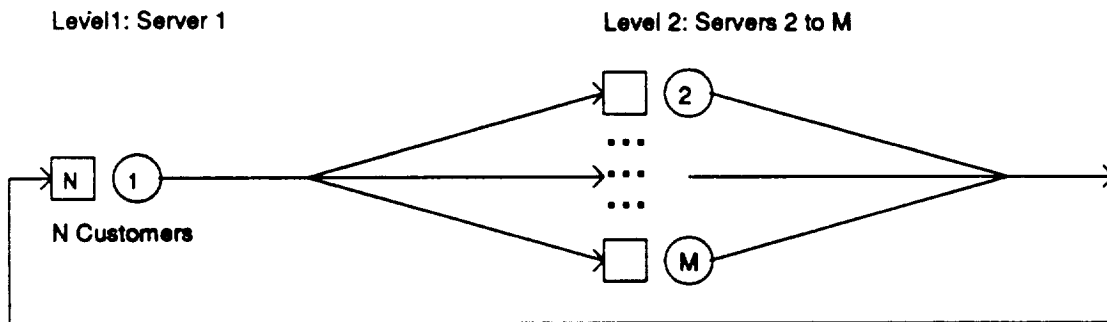


Figure 1.1: Two Level Finite Queueing Network

1.2 Review of Product Form Queueing Networks

A queueing network is formed by connecting a set of queues and servers and allowing customers to depart from one server to another server. The network parameters include:

N the number of customers in the network.

M the number of service stations.

μ_i the mean service rate at service station i in terms of completions per unit time,
 $s_i = \frac{1}{\mu_i}$ is the mean service time.

\mathbf{n} (n_1, n_2, \dots, n_M) the state of the network (n_i customers at service station i).

$P(\mathbf{n})$ equilibrium state probability distribution (probability that the network is in state \mathbf{n}).

\mathbf{P} routing matrix; p_{ij} is the probability that a customer visits server j after service at the i^{th} server (fraction of customers leaving service station i and going to service station j).

r_i the mean number of visits made by a customer to service station i and computed by

$$r_i = \sum_{j=1}^M r_j p_{ji} \quad \text{for } i = 1, 2, \dots, M \quad (1.1)$$

ρ_i the utilization of station i . The utilization of a service station is defined as the proportion of time the server is busy, or, equivalently, as the average number of customers in service there. The utilization is obtained by solving the balance equations:

$$\mu_i \rho_i = \sum_{j=1}^M \mu_j \rho_j p_{ji}, \text{ for } i = 1, 2, \dots, M \quad (1.2)$$

Figures 1.2a and 1.2b show a relatively small closed network and the corresponding state space, respectively. The state of the system is represented by a vector (n_1, n_2, n_3) where n_i is the number of customers at server i . With a set of vectors representing the states of the system, the state transition diagram is formed by adding arcs between the states which represent the feasible changes in the state of the system due to an arrival of a customer to a queue or completion of service for a customer. The arcs are labeled with the rate at which the transitions occur (i.e the service rate multiplied by the routing probability). As an example, in Figure 1.2b, with the completion of service for a customer at server 1 and the customer joining the queue at server 2, the system experiences a transition represented by the arc from state $(4,0,0)$ to the state $(3,1,0)$ with rate $\mu_1 p_{12}$. Service times are assumed to be exponentially distributed random variables.

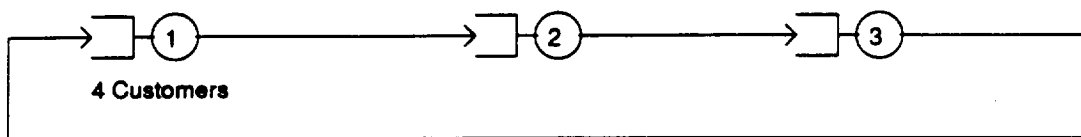


Figure 1.2a: Closed Product Form Queueing Network

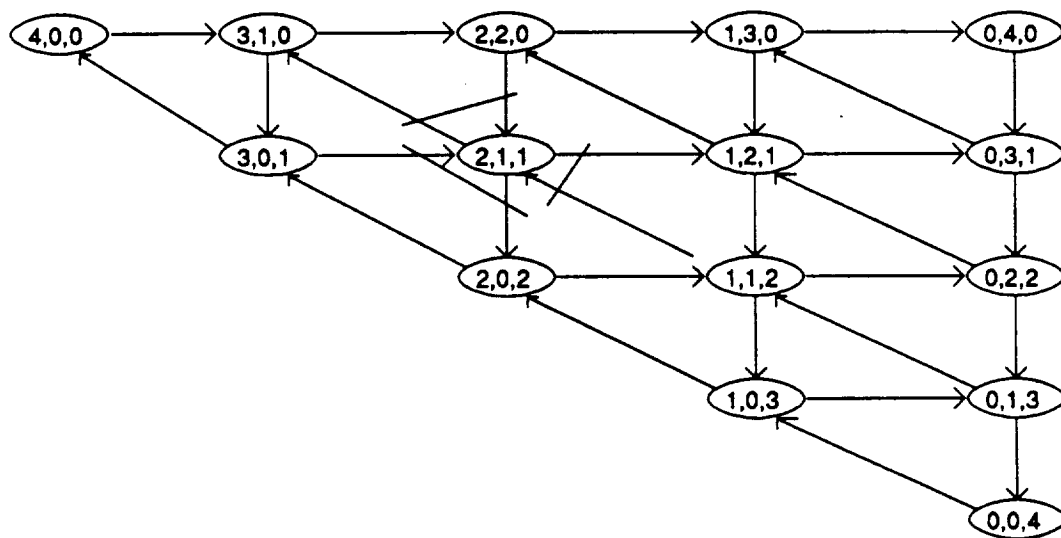


Figure 1.2b: State Transition Diagram

When a network is in equilibrium, the steady-state solution must satisfy the *global balance property* which states that the total flow out of a state equals the total flow into the state. The global balance equations for closed networks [31] are given by

$$\sum_{i=1}^M \mu_i (1 - p_{ii}) P(\mathbf{n}) = \sum_{j=1}^M \sum_{i=1}^M \mu_i p_{ij} P(\mathbf{n})_{i \rightarrow j}, \quad \text{for all } \mathbf{n}. \quad (1.3)$$

where $(\mathbf{n})_{i \rightarrow j} = (n_1, n_2, \dots, n_i + 1, n_j - 1, \dots, n_M)$ (the number of customers at station i is increased by 1 and the number of customers at station j is decreased by 1, due to a customer finishing service at station j and moving to station i).

These global balance equations hold for all Markov chains in equilibrium. One way to generate these equations is to form the transitive matrix π (or the infinitesimal generator) [20]. Each element in π , π_{ij} , is equal to the transition rate from state i to state j . This is equal to the service rate of the source server multiplied by the probability of the transition to the destination server. As an example, the infinitesimal generator for the network in Figure 1.3a, with 2 customers, is given in Figure 1.3b.

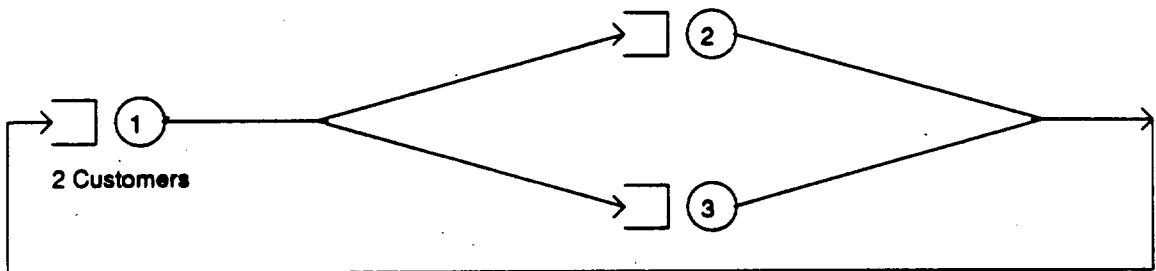


Figure 1.3a: Queueing Network Model

n	(2,0,0)	(1,1,0)	(0,2,0)	(1,0,1)	(0,1,1)	(0,0,2)
(2,0,0)	$-(\mu_1 p_{12} + \mu_1 p_{13})$	$\mu_1 p_{12}$		$\mu_1 p_{13}$		
(1,1,0)	μ_2	$-(\mu_2 + \mu_1 p_{12} + \mu_1 p_{13})$				
(0,2,0)		μ_2	$-\mu_2$			
(1,0,1)	μ_3			$-(\mu_3 + \mu_1 p_{13})$		$\mu_1 p_{13}$
(0,1,1)		μ_3		μ_2	$-(\mu_2 + \mu_3)$	
(0,0,2)				μ_3		$-\mu_3$

Figure 1.3b: Infinitesimal Generator π

Now the global balance equations are simply $\pi^t \mathbf{P} = 0$, with \mathbf{P} the vector of state probabilities. The system is solved for \mathbf{P} with the *boundary condition* that the sum of the state probabilities is equal to 1, $\sum_{\text{all states}} P(\mathbf{n}) = 1$. For the network in Figure 1.2a (state space Figure 1.2b), if $P(\mathbf{n})$ is the probability the state of the network is (n_1, n_2, n_3) , then the global balance equations are

$$\begin{aligned}
 (\mu_1 (1-p_{11}) + \mu_2 (1-p_{22}) + \mu_3 (1-p_{33})) * P(\mathbf{n}) = & \mu_1 p_{11} P(n_1, n_2, n_3) + \\
 & \mu_2 p_{21} P(n_1 - 1, n_2 + 1, n_3) + \\
 & \mu_3 p_{31} P(n_1 - 1, n_2, n_3 + 1) \\
 & \mu_1 p_{12} P(n_1 + 1, n_2 - 1, n_3) + \\
 & \mu_2 p_{22} P(n_1, n_2, n_3) + \\
 & \mu_3 p_{32} P(n_1, n_2 - 1, n_3 + 1) + \\
 & \mu_1 p_{13} P(n_1 + 1, n_2, n_3 - 1) + \\
 & \mu_2 p_{23} P(n_1, n_2 + 1, n_3 - 1) + \\
 & \mu_3 p_{33} P(n_1, n_2, n_3)
 \end{aligned}$$

For State (2,1,1), $p_{ij} = 0$ (including $i = j$) except p_{12} , p_{23} , and p_{31} , and the global balance equation

$$\begin{aligned}
 (\mu_1 + \mu_2 + \mu_3) P(2,1,1) = & \mu_1 p_{12} P(3,0,1) + \\
 & \mu_2 p_{23} P(2,2,0) + \\
 & \mu_3 p_{31} P(1,1,2) +
 \end{aligned}$$

The network leaves the state (2,1,1) at the rate $(\mu_1 + \mu_2 + \mu_3) P(2,1,1)$. The network arrives at the state (2,1,1) from the state (3,0,1) at the rate $\mu_1 p_{12} P(3,0,1)$, from state (2,2,0) at the rate $\mu_2 p_{23} P(2,2,0)$, and from state (1,1,2) at the rate $\mu_3 p_{31} P(1,1,2)$. The total rate of arrivals equals the total rate of departures when the network is at equilibrium. The network is defined to be at equilibrium when the

equality in Equation 1.3 occurs [31].

Jackson [24] proved that, for a special class of queueing networks, the steady-state probability density function for the Markov system corresponding to the network is the product of the steady-state probability density functions for the individual queues. This important result indicates that networks can be solved by simply multiplying together the results for each queue. Systems in this class satisfy the *local balance property* which states that: the rate at which the network arrives at a given state n due to an arrival at a certain server's queue equals the rate at which the network leaves the state due to a departure from that server queue. In other words, *local balance* is satisfied when the number of transitions into any state is equal to the number of transition out of that state [31]. The local balance property and product form networks are equivalent in the sense that the product form applies to any network exhibiting the local balance property and vice versa [31].

As an example of the local balance property, the rate at which the network arrives at state $(2,1,1)$ (Figure 1.2b) due to an arrival at server 1, $\mu_3 p_{31} P(1,1,2)$, must be balanced by the rate at which the network leaves the state $(2,1,1)$ due to a departure from server 1, $\mu_1 p_{12} P(2,1,1)$. In this case the balance equation is $\mu_3 p_{31} P(1,1,2) = \mu_1 p_{12} P(2,1,1)$.

The other local balance equations associated with state $(2,1,1)$ are

$$\mu_1 p_{12} P(3,0,1) = \mu_2 p_{23} P(2,1,1)$$

$$\mu_2 p_{23} P(2,2,0) = \mu_3 p_{31} P(2,1,1)$$

Closed networks with a total of N customers and M servers, which satisfy local balance equations have the property that the probabilities of the states (state probability distributions) can be easily calculated [32]. The following equation giving the

probability, $P(\mathbf{n})$, that the network is in some state, \mathbf{n} , is called the "product form" [12, 13, 19].

$$P(n_1, n_2, \dots, n_M) = \frac{1}{G(N, M)} \prod_{i=1}^M (\rho_i)^{n_i} \quad (1.4)$$

where

$$G(N, M) = \sum_{\mathbf{n}} \prod_{i=1}^M (\rho_i)^{n_i} \quad (1.5)$$

$G(N, M)$ is a normalization constant required to make P a discrete probability distribution function and ρ_i is the utilization of the i^{th} queue in the network. A variety of closed, open, and mixed networks are shown to be of the product form [3].

Product form solutions are easily used to find the probabilities of the states. The problem is that the number of states grows exponentially with the size of the network. In general, the number of states in a closed queueing network with N customers and M service stations is equal to $\frac{(M+N-1)!}{(N-1)! M!}$ [31], this is the same as counting the number of ways of putting N identical objects into M different bins. This makes the calculation of $G(N, M)$ and the state probabilities of a network of any size computationally prohibitive. This is because the calculation of the normalization constant runs through all possible combinations of states with N customers in the network (see algorithm I in APPENDIX B). Buzen [9] derived an iterative algorithm to calculate the normalization constant without running through the possible states for a closed network with N customers and M service stations (see algorithm II in APPENDIX C). Other performance measures such as throughput and utilization, could also be calculated very efficiently using this result. Buzen's discovery and the discovery that many multiple class, load dependent networks are also product form led to the extensive use and study of product form networks. These networks have the property that they model a

great number of real systems and can be efficiently solved. They form the vast majority of exactly solvable queueing networks.

To summarize, any Markovian network can be solved, at least in principle, by solving the global balance equations. This involves solving a linear system of T equations, where T is the number of states, which grows exponentially with the size of the network. On the other hand, the local balance property specifies a class of product form networks that can be solved very efficiently. This precipitates interest in approximating nonproduct form queueing networks using product form models. One method for doing this is described next.

1.3 Exact and Approximate Solution by Decomposition

In order to analyze large networks, we can decompose the network into subnetworks and analyze them individually [11, 15, 17]. Each subnetwork is then replaced by an equivalent server whose characteristics are such that the performance of the resulting network is the same as the original network. An important feature of decomposition is that the parameters of the aggregated server (equivalent server) depend only on the parameters of the servers being aggregated (servers in the corresponding subnetwork) and are independent of other servers in the network [20, 31]. Through an analogy to Norton's Theorem as applied to electrical networks [10], the parameters for the aggregated server are determined by ignoring (shorting out) the rest of the network and calculating the throughput of the subnetwork for different populations. The calculated throughput when there are n customers in the subnetwork (shorted network) is used as the variable rate server when n customers are queued at the aggregated server. This variable rate server is represented in a queueing network (Figure 1.4c) by adding an arrow through the server.

The results of the analysis by decomposition and aggregation were shown to be exact for product form networks [10, 11]. Decomposition is extended to multiple classes of product form networks in [29]. The conditions and errors involved in the decomposition of general queueing networks are discussed in [15]. In the case of general networks, the condition for replacing a subnetwork with an equivalent server is that the departure rate from the subnetwork depends only upon the number of customers queued there. If the subnetwork comes to a probabilistic equilibrium quickly with respect to arrivals to and departures from the subnetwork, then approximate decomposition tends to work well [15]. This occurs if customers leaving servers in the subnetwork are very likely routed back to other servers in the subnetwork and less likely routed to servers outside the subnetwork. In decomposing nonproduct form networks, some error will be generated because the flow equivalent servers will not model exactly the interaction between the individual portions of the network, thus the analysis will be approximate.

Decomposition of queueing networks allows us to model parts of a network without considering the entire network by replacing the part modeled by a single load dependent server. An example of this is the computer system modeled in Figure 1.4a. The model consists of users and the computer system which involves terminals, the CPU and the disks. Users pause while thinking at the terminals, and then enter requests to the system. The requests require use of the CPU and disks. The speed at which the request is completed is dependent upon the speed of the CPU, the access speed of the disks, the number of programs in process and the amounts of memory in the system. After the request is satisfied, the user thinks again before entering the next request. Each of the phases in the real system is modeled by a corresponding event in the queueing model. As a queueing system, this process can be seen as customers moving from the delay server 1 (terminal stage) to server 2 (CPU) and then to the

disks (servers 3 and 4). The request is satisfied by the customer moving between and getting service at the CPU and the disks. When the request is completed, i.e. has left a disk server, the customer returns to the delay server.

The overall system in Figure 1.4a can be analyzed by decomposing it into two subnetworks, the terminals as one subnetwork, and the rest of the system as the second subnetwork. The subnetwork Figure 1.4b is solved for the different population levels, n , either analytically or by simulation, to determine the corresponding throughput, $\lambda(n)$. We can now replace the servers representing the computer system (CPU and disks) with a single composite server. The rate at which requests are completed at the single composite server is dependent on the number of requests queued for it. Since we have the rate at which requests are completed in the computer system (the throughput $\lambda(n)$ for Figure 1.4b), as a function of the number of requests queued, we have the parameters for the load dependent server: $\mu(n) = \lambda(n)$, $n = 1, 2, \dots, N$. The flow equivalent server of the CPU and disks allows us to find the overall performance measures by solving the much simpler product form network in Figure 1.4c.

The power of the decomposition method allows us to find approximate solutions of nonproduct form networks [1]. Figure 1.4a is a product form network and we need not use decomposition to solve it, due to the fact that it satisfies the local balance property which was outlined in the previous section. Suppose we now impose a multiprogramming limit L on the number of processes allowed to be executing in the system. This means that if L customers are in the box labeled *system*, another user request would be denied entry into the box until one of the others completed. This is a form of blocking and destroys the local balance property rendering the product form solution invalid. This multiprogramming limit L can be approximated by the Figure 1.4c model if we set $\mu(n) = \lambda(n)$, $n = 1, 2, \dots, L$, and $\mu(n) = \lambda(L)$, $n = L+1, \dots, N$. This

represents the fact that the throughput of the computer system cannot exceed $\lambda(L)$. So by solving the two product form networks, Figure 1.4b and Figure 1.4c, we obtain an approximate solution to a nonproduct form system, Figure 1.4a, with multiprogramming limit L . Thus the power and importance of the decomposition approach is illustrated: the solution of computational nonproduct form networks is done by product form subnetworks.

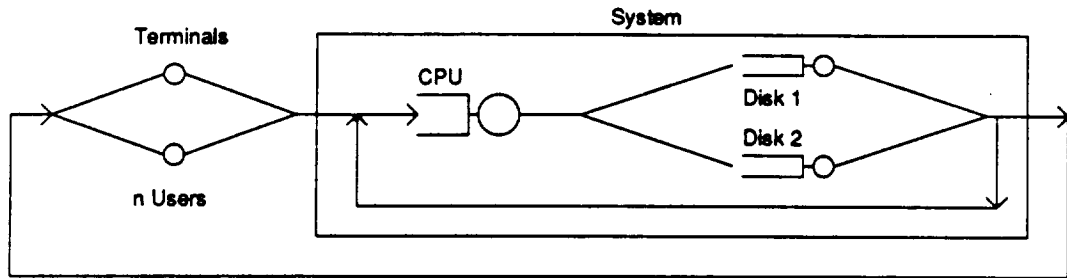


Figure 1.4a: Model of Timesharing System

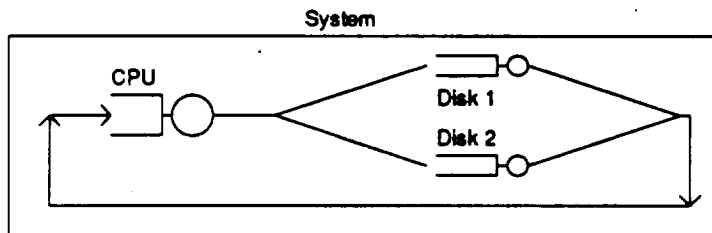


Figure 1.4b: Model of Computer System

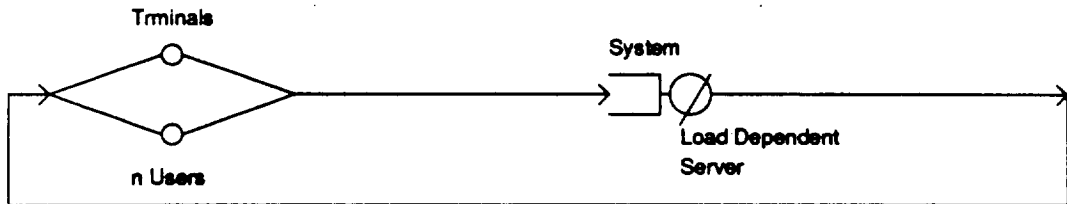


Figure 1.4c: Overall Model of Terminals and System

CHAPTER 2

BACKGROUND ON FINITE QUEUEING NETWORKS

Within the broad description of finite queueing networks there are a number of ways that blocking occurs. In this chapter finite queueing networks with blocking are discussed including a review of the usual mechanisms in which blocking in finite queueing networks occurs. The difficulties in the analysis of such networks are outlined and previous work on modeling finite queue networks and other related approaches are discussed.

2.1 Definition of Blocking

In a finite queue network, the number of customers allowed at any service station is limited. This number includes the queue size and the customer in service. In this research it is assumed that when the queue at server j is full, and a customer finishes service at server i and needs to be routed next to server j , the customer at server i cannot leave that server and server i is said to be *blocked*. Also, the customer at server i is said to be *blocked*, since it has to wait for a space at server j . The problem here is that a condition at server j (queue is full) has shut down service at server i , so that the requirements of classical queueing analysis (assumption of infinite queues which simplify the analysis) no longer hold. It is this nonclassical feature that complicates the analysis of finite queue networks.

Two types of blocking are discussed in the literature. These are *transfer* and *service* blocking.

2.1.1 Transfer Blocking

Transfer blocking [32] occurs when a customer finishes service at server i and is denied transfer (prohibited from being transferred) to another server, j , because the destination queue is full. The customer is forced to remain at the blocked server, i , until it may enter the destination queue at server j . Server i remains blocked for this period of time and cannot serve any other customer waiting in the queue. The customer is moved to the destination queue as soon as space becomes available.

An example of this type of system is the two server tandem queueing network in Figure 2.1a. Figure 2.1b shows the state space of the network. The state with the superscript refers to the server where service has completed but the customer is blocked and cannot be transferred to the destination server. The value of the superscript denotes the destination server (to which server the blocked customer will go next). The state $(2^2, 2)$ refers to the state where the queue at server 2 is full and a customer has completed service at server 1 and is transfer blocked. A large number of real systems with different characteristics are modeled using this type of network as in the case of manufacturing systems [42].

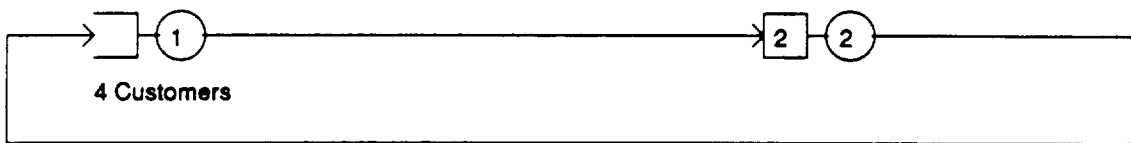


Figure 2.1a: Two-Server Finite Queueing Network



Figure 2.1b: State Transition Diagram

2.1.2 Service Blocking

Service prohibited blocking [32] occurs when a customer is denied service because the queue to which it will be routed next is full. This means that blocking prohibits serving the customer. In the transfer blocking case, blocking occurs after the service is completed and before the transfer, but in service blocking case blocking occurs before service starts. An example is the state space in Figure 2.2 for network 2.1a. In this case the state space is lacking the state with the superscripts which corresponds to the point when the transfer of a customer is blocked.

Computer and communication networks are modeled largely using these two types of blocking networks [1]. In transfer blocking, servers work on customers until blocking occurs. In service blocking, the lack of space in the destination queue must force the server to shut down before serving the next customer. When considering a two server one-finite queueing network (Figures 2.1a and 2.1b), the difference between the transfer blocked and service blocked definitions appears to be artificial. The throughput in a two server queueing network using the transfer blocking definition is equal to the throughput of an equivalent queueing network (has the same number of servers and same rate matrix) having a finite queue (queue 2) that is larger by one and applying the service blocked definition (Figures 2.2). But the difference can be shown to be real when larger networks with finite queues are considered [32].

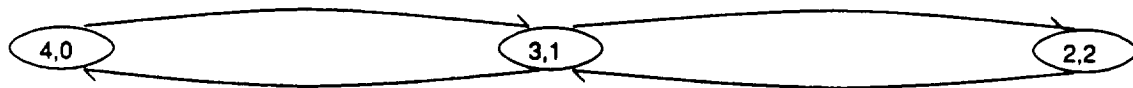


Figure 2.2: State Transition Diagram

2.2 Complexity of Finite Queue Networks

Even with the existence of the product form solution and the decomposition methods, the analysis of finite queue networks is still difficult because of the structural problems in these networks that prevent these methods from accurately modeling them. The basic problem in analyzing this type of network is two part: first the network is not product form, and secondly the servers are coupled together in a very fundamental way. These problems are discussed in detail in the following sections.

2.2.1 A Non-Product Form Network

Finite queue networks are not product form networks since in general they do not satisfy the local balance property which states that the rate of flow into a state caused by a customer arriving at a certain queue for service is equal to the rate of flow out of this state caused by a customer leaving that queue [31,32]. As an example consider the state space Figure 2.3a of the network in Figure 2.3b. A portion of the state transition diagram is given in Figure 2.3c. Since there are 2 transitions into $(2^2, 2^3, 2)$, and only one transition from it, the local balance property does not hold [31].

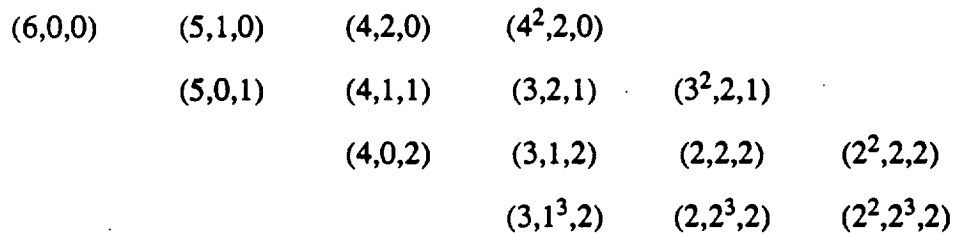


Figure 2.3a: State Space Diagram (Transfer Blocking Definition)

The transition to the state $(2^2, 2^3, 2)$ from the state $(2^2, 2, 2)$ (arc labeled a) is balanced

by the transition from $(2^2, 2^3, 2)$ to $(3^2, 2, 1)$ (arc labeled b) (i.e., the customer completing at server 2 is balanced by a customer completing at server 3). However the transition from $(2, 2^3, 2)$ to $(2^2, 2^3, 2)$ (arc labeled c) is not balanced by another transition (i.e., the completion at server 1 cannot be balanced by a completion at server 2 since server 2 is blocked). The network, Figure 2.3b, cannot have product form solution since it does not satisfy the local balance property [32].

2.2.2 Dependency Between Servers and Decomposition

The dependency between servers makes the job of finding a valid decomposition very difficult. To see this let us try to apply the standard decomposition approach on a three-server two-finite queue network in Figure 2.4a. The usual decomposition method suggests that both servers 2 and 3 in Figure 2.4a be decomposed to form a single load dependent server. The service rates of this load dependent server are calculated so as to model the flow rate when different number of customers are waiting for servers 2 and 3. The model in Figure 2.4b is used to calculate the service rates for the composite server in Figure 2.4c. The service rate of the composite server when there are n customers in network 2.4c, $\mu(n)$, is set equal to the throughput of the network 2.4b when there are n customers, $X(n)$.

The problem with this decomposition process is that the infinite queue at the composite server in 2.4c does not model the real system since it does not model the blocking effect of servers 2 and 3 on server 1. Characteristics of the queueing network, such as the number of servers, the size of the queues, the routing matrix, and others, have an effect on the dependency between servers. When analyzing large systems, decomposition in some form must be used [2,8,18,43] and standard decomposition ([10,15]) will not work correctly. The problem is that the regular decomposition method destroys the direct connection between servers and replaces it with a connection between subnetworks and aggregated servers.

In order to handle the complexity of finite queueing networks, this research attempts to model them using decomposition to represent the effect finite queues with blocking has upon servers. Existing decomposition methods are not able to do this. This research proposes a decomposition method in chapter 3 that produces parameters that model the effect of finite queues. Existing methods are discussed in section 2.3.