

FILE COPY

4

IC-TR-89-48
Technical Report
1989



D-A208 809

RESOURCE CONTENTION MANAGEMENT IN PARALLEL SYSTEMS

University of Massachusetts

Insured by
Strategic Defense Initiative Office

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

Views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Strategic Defense Initiative Office or the U.S. Government.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

SDTICD
ELECTE
JUN 09 1989
H

89 6 09 073

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

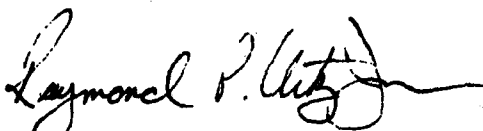
RADC-TR-89-48 has been reviewed and is approved for publication.

APPROVED:



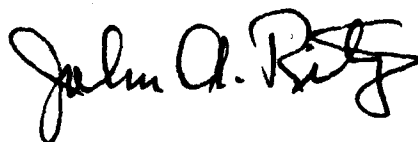
ALAN N. WILLIAMS, 1LT, USAF
Project Engineer

APPROVED:



RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command and Control

FOR THE COMMANDER:



JOHN A. RITZ
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COTC) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

RESOURCE CONTENTION MANAGEMENT IN PARALLEL SYSTEMS

Christos G. Cassandras
James F. Kurose
Don Towsley

Contractor: University of Massachusetts
Contract Number: F30602-81-C-0169
Effective Date of Contract: 10 June 1987
Contract Expiration Date: 31 December 1987
Short Title of Work: Resource Contention Management in
Parallel Systems
Period of Work Covered: Jun 87 - Dec 87

Principal Investigator: Christos G. Cassandras
Phone: (413) 545-1340

RADC Project Engineer: Alan N. Williams, 1Lt, USAF
Phone: (315) 330-2925

Approved for public release; distribution unlimited.

This research was supported by the Strategic Defense Initiative Office of the Department of Defense and was monitored by 1Lt Alan N. Williams, RADC (COTC), Griffiss AFB NY 13441-5700 under Contract F30602-81-C-0169.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188		
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS N/A			
2a. SECURITY CLASSIFICATION AUTHORITY N/A			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.			
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A						
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A			5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-89-48			
6a. NAME OF PERFORMING ORGANIZATION University of Massachusetts		6b. OFFICE SYMBOL (if applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COTC)			
6c. ADDRESS (City, State, and ZIP Code) Department of Electrical & Computer Engineering Amherst MA 01003			7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700			
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center		8b. OFFICE SYMBOL (if applicable) COTC	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-81-C-0169			
6c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700			10. SOURCE OF FUNDING NUMBERS			
			PROGRAM ELEMENT NO. 63223C	PROJECT NO. B413	TASK NO. 03	WORK UNIT ACCESSION NO. P3
11. TITLE (Include Security Classification) RESOURCE CONTENTION MANAGEMENT IN PARALLEL SYSTEMS						
12. PERSONAL AUTHOR(S) Christos G. Cassandras, James F. Kurose, Don Towsley						
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM Jun 87 TO Dec 87		14. DATE OF REPORT (Year, Month, Day) April 1989		
15. PAGE COUNT 156						
16. SUPPLEMENTARY NOTATION N/A						
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)			
FIELD	GROUP	SUB-GROUP	Resource Contention			
09	02		Parallel Processing			
			Load Balancing			
			Scheduling			
19. ABSTRACT (Continue on reverse if necessary and identify by block number) This research effort explored two issues: (1) The comparative study of simple load balancing algorithms for distributed real-time systems which showed that simple policies perform just as well as complex policies in a majority of the cases; (2) The second task was the development of on-line optimization procedures for load balancing algorithms and of task scheduling policies with real-time constraints.						
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED			
22a. NAME OF RESPONSIBLE INDIVIDUAL Alan N. Williams, 1Lt, USAF			22b. TELEPHONE (Include Area Code) (315) 330-2925		22c. OFFICE SYMBOL RADC (COTC)	

Table of Contents

1. INTRODUCTION.....	3
2. LOAD SHARING IN SOFT REAL-TIME DISTRIBUTED SYSTEMS.....	3
2.1. System Models and Protocols.	4
2.2. Overview of Comparative Study Results.	6
3. ADAPTIVE LOAD SHARING ALGORITHMS IN REAL-TIME DISTRIBUTED PROCESSING SYSTEMS.....	9
3.1. A Static Decentralized Adaptive Load Sharing Algorithm.....	10
3.1.1. Distributed Algorithm Description.	13
3.1.2. Marginal Loss Estimation.....	14
3.2. Simulation Results.....	17
3.3. Extension to Dynamic Load Sharing.....	29
3.4. Sensitivity Analysis for Distributed Processing Systems with Discrete Parameters.....	30
3.4.1. Dynamic Processor Scheduling.....	30
4. DESIGN AND ANALYSIS OF PARALLEL PROCESSING SYSTEMS.....	31
4.1. Multiprocessor Scheduling.....	32
4.2. Models of Parallel Systems.....	33
5. CONCLUSIONS.....	36
REFERENCES.....	37
APPENDIX A Load Sharing in Soft Real-Time Distributed Computer Systems	
APPENDIX B Augmented Chain Analysis of Markov and Semi-Markov Processes	
APPENDIX C A Comparison of the Processor Sharing and First Come First Serve Policies for Scheduling Fork-Join Jobs in Multiprocessors	
APPENDIX D Acyclic Fork-Join Queueing Networks	



For	
AI	☒
ed	☐
tion	☐
ion/	

Availability Codes	
Dist	Avail and/or Special
A-1	

1. INTRODUCTION.

This report overviews the results of our research over the duration of the project; detailed discussions of most results are included in the technical papers appearing as Appendices A-D. As originally proposed, resource contention problems were decomposed into *system-level* and *node-level*. At the system level, we have performed two main tasks: first, a comparative study of simple load-sharing schemes in distributed real-time systems, and second, development of actual algorithms to be implemented in practical systems. This work is reported in sections 2 and 3 respectively. At the node level, our objective has been to consider scheduling policies under real-time constraints. During the course of our work, it became apparent that nodes are often multiprocessors, where parallelism in task execution is possible. Thus, we have also focused on this issue, and obtained the results reported in section 4. Overall, our work has resolved many of the problems identified in the original proposal, as well as generated new ones. In some cases, there are useful extensions of our results which can be obtained in the future, given the framework created in this project.

Appendices A-D included in this report are technical papers which have already appeared or have been submitted to journals or to conference proceedings.

2. LOAD SHARING IN SOFT REAL-TIME DISTRIBUTED SYSTEMS.

A major focus of our research during the past contract year has been on high-level load sharing (LS) schemes for a class of distributed applications which are subject to *soft real-time constraints*. In such real-time systems, jobs generated at a node in the distributed system must complete execution within a specified amount of time after their initial arrival to the system; otherwise they are considered *lost*. Examples of systems exhibiting such soft real-time behavior include the general class of applications in which a process may spawn a number of subprocesses and then, after a fixed amount of time, must make a decision based on the results of the subprocesses which were able to execute (e.g., a distributed sensor system, in which multiple hypotheses are to be

generated and evaluated). A second application is in distributed industrial process control, where a failure to complete a computation within a specified time constraint (due to a momentary overload of work at a given node) may require the initiation of an expensive recovery procedure.

In these soft real-time systems, the primary performance metric is the maximization of the percentage of jobs completed within their specified time constraint. Our research on real-time LS algorithms has been based on the premise that *simple real-time LS policies may perform as well as their more complex counterparts*. It has been previously noted that for non-real-time systems, relatively *simple* decentralized policies may often provide effective load sharing in a distributed system [1]. These works have motivated our work during the past year, which establishes complementary results for the case of *real-time* systems, systems having performance requirements and evaluation metrics which differ significantly from those of non-real-time systems. We stress that, as in [1], the goal of the research reported in this section has not just been to propose any specific real-time load sharing algorithm nor to necessarily develop performance models for predicting the absolute performance of specific LS approaches, but rather to address the more fundamental question of the level of complexity required to implement effective load sharing, in this case in a distributed *real-time* environment.

2.1. System Models and Protocols

Our model of a distributed system consists of N nodes which are interconnected through a communication network; the network is assumed to be logically fully connected in that every node can communicate with every other node. A stream of jobs is submitted locally to node i . We assume that the nodes are heterogeneous in the sense that each node may have a different arrival rate of externally submitted jobs, but homogeneous in the sense that a job submitted at any node in the network can be processed at any other node in the network; this latter assumption can be easily relaxed.

We are interested in studying LS policies in a soft real time system, in which *a job is lost if it*

can not complete or begin execution (as the case may be) within a given time constraint. If the deadline cannot be met locally, a LS algorithm may be invoked to transfer the job to another node which can possibly meet the job's demands. We assume that a job cannot be transferred more than once in order to avoid the problem of "trashing" and assume that a constant delay, d , (representing communication and transfer processing delays) is required to transfer a job from one node to another. Thus, if a job first arrives at node i with an initial time constraint of $K1$ and is transferred to another node j for processing, its new time constraint at node j will be equal to $(K1-d)$.

Our research has examined two simple approaches:

1. quasi-dynamic load sharing QDLS

2. *probing*

which have been previously studied for non-real-time systems, and compares their real-time performance with that of the bounding cases of no load sharing and the theoretically optimum real-time LS algorithm.

A LS approach can be characterized by its transfer policy, and its location policy. The transfer policy determines a job should be transferred for remote execution. The location policy, determines where (i.e., at which remote node) a transferred job will be executed.

Both QDLS and probing have the same simple transfer policy:

Transfer policy (QDLS and probing):

A job is transferred from node i to a remote node if and only if the unfinished workload of the jobs currently at node i exceeds the time constraint for the job. A job will thus either queue for service at the node at which it initially arrives (in which case it will be guaranteed execution) or will be transferred to some remote node. We note that the transfer policy decision is made *dynamically*, based on the current state of the node. There are no previous analytic studies which have considered this transfer policy in a real-time environment.

The location policies of QDLS and probing are:

Location policy (QDLS):

If a job is to be transferred, a remote "target" node (to which the job is sent) is chosen *probabilistically* and *independent* of the current state of the remote nodes. Note that QDLS requires *no* non-local, dynamic state information. Although this location policy has been extensively studied for the non-real-time case, no previous analytic work has addressed this problem in a real-time environment.

Location policy (probing):

When a job is to be transferred a node *probes* some specified number of other system nodes (chosen at random) to determine if one of them can currently guarantee execution of this job, i.e., has an amount of unfinished work less than the time constraint of the job minus the transfer delay. A node may probe up to some maximum number, L_p , (the probe limit) of other nodes. If none of the probed nodes can execute the job, the job is lost. We note that probing may be considered a simplified form of *bidding* [2]. The probing policy studied here was first analytically examined in [1] (for non-real-time systems) and we follow their methodology when studying the system-level model (but not the node-level model) of probing.

2.2. Overview of Comparative Study Results.

In the course of our research, analytic performance models were developed to study the performance of the QDLS and LS approaches, as well as the case of no load sharing. The case of the theoretically optimum LS algorithm was examined through simulation. The details of the analysis are presented in Appendix A of this report.

Figure 1, which is discussed in additional detail in [3] (also Appendix A) shows representative performance results for the QDLS and probing real-time load sharing schemes and compares their performance with that of the *ideal* case of perfect-information load sharing and the case of *no load sharing* (NLS). In this case, jobs were required to *begin* execution within the specified time constraint or were otherwise lost. The performance results for the case in which jobs must complete execution within the time constraint showed similar behavior and are also discussed in

detail in [3].

The results are for a 20 node system in which the average job execution time was 1 job/second (exponentially distributed) and a network job transfer delay of $d = 0.2$ secs. The "ideal" case was modeled as an M/M/20 queueing system with a time constraint of K1. We note that in the M/M/20 system, jobs are scheduled to available processors using complete information about the system state and incur no transfer delay. Thus, the "ideal" performance bounds shown in the subsequent results are, in reality, unattainable. We also note that simulations were performed to validate our analysis. The simulations were performed without many of the assumptions required by the analysis; we note that the close correspondence between our simulation and analytic results indicate that reasonable modeling assumptions and approximations were made in the development of our analytic model of probing.

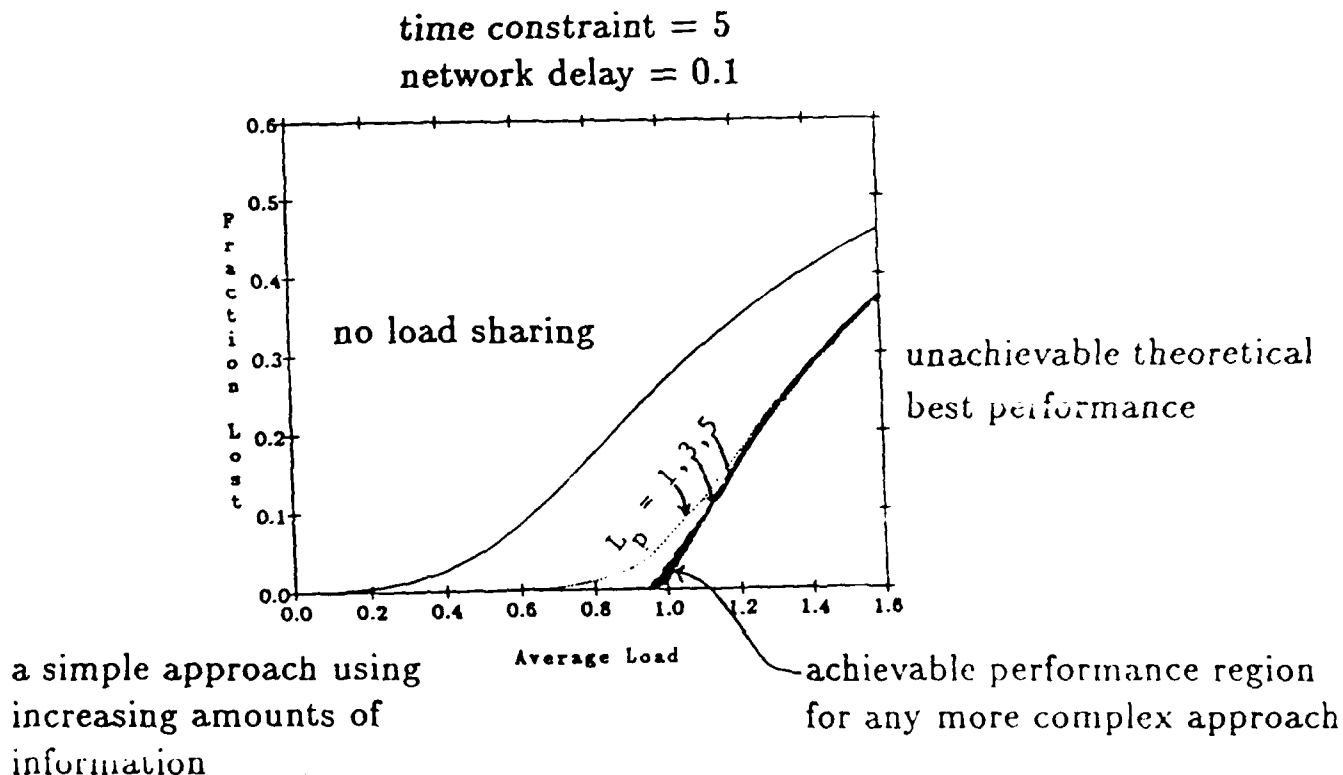


Figure 1: Performance of the Probing Policy for $L_p = 1, 3, 5$ under a symmetric load

The real-time performance of the QDLS and probing approaches is demonstrated in Fig. 1 for probe limits, $L_p = 1, 3, 5$; the case of $L_p=1$ corresponds to the QDLS policy. As expected the performance of probing approaches the ideal limit as L_p increases. Note, however, that a relatively small probing limit ($L_p=5$ when $K1=0.5$ (an *extremely tight* time constraint) and $L_p=3$ when $K1=5.0$), results in a real time performance extremely close to the unachievable upper bound. Also note that increasing the probing limit beyond a relatively small number can result at best in only a marginal performance improvement. *We may conclude then that since additional probing beyond some small probe limit incurs additional overhead, a relatively small probe limit would be sufficient in practice to implement effective real time load sharing.*

Perhaps more importantly, Fig. 1 provides a quantitative basis for addressing the question of determining the appropriate level of complexity for LS algorithms. We note that a more complex approach can *at best* achieve a performance level falling in the gap between our probing results and the theoretical optimum. For system parameters of practical interest (i.e., a system loading less than the physical capacity of the system and time constraints on the order of the service time of a job), this gap can be seen to be quite small. If the overhead we have not modeled is to be considered, the small performance difference between probing and a more complex approach, which requires additional communication and computational overhead, can only become smaller.

The most important conclusion then to be drawn from Fig. 1 and our additional results discussed in Appendix A is that for a relatively wide range of system parameters, the simple approaches studied perform significantly better than the case of no load sharing and often perform remarkably close to that of the theoretically optimum algorithm. Our conclusion thus complements previously-established results for LS in non-real-time systems [1]: very simple approaches, which use only a minimal amount of state information and have an extremely simple decision-making process (and hence are simple to implement) are often sufficient to provide effective load sharing in a distributed real-time computer system. A corollary then is that for all but the tightest of time constraints (e.g., values of the time constraint, $K1$, less than the average job service time), a more

sophisticated approach towards real-time load sharing can often result in only a small marginal performance improvement over the extremely simple load sharing algorithms.

3. ADAPTIVE LOAD SHARING ALGORITHMS IN REAL-TIME DISTRIBUTED PROCESSING SYSTEMS.

As in the previous section, our concern here is with distributed processing systems where jobs are constrained by real-time deadlines. Thus, the performance objective is to *minimize the fraction of jobs that are lost due to exceeding their deadline*. Our interest now, however, is in actually developing *adaptive* algorithms, which can be incorporated into the system itself, and perform load sharing on-line. First, let us identify three desired features for the practical applicability of such algorithms:

1. Load sharing schemes should be sufficiently simple so as to incur little overhead and communication costs.
2. Stochastic modeling assumptions regarding the nature of job arrival and service processes should be minimized or eliminated.
3. The algorithms should require little or no information about the parameters of the distributed processing systems (since these parameters may be hard to estimate in practice, as well as subject to changes).

As we shall describe below, the algorithms we have developed and investigated have been designed so as to satisfy these requirements.

Note that load sharing algorithms can be categorized in terms of their execution mode (*centralized* or *decentralized*), and information structure (*static* or *dynamic*). A static decentralized algorithm satisfies the simplicity requirement, since it can be executed at each node separately and with no instantaneous state information. One, however, should expect dynamic algorithms to perform better; hence, it is important to study the tradeoff between simplicity (no state information) and performance.

To address the second and third requirement above, our goal has been to exploit developments

in sample-path-based sensitivity analysis techniques within the context of our problem (e.g. [4]-[6]). In particular, we have used the Perturbation Analysis (PA) methodology, and have sought to obtain extensions and generalizations that are applicable.

In section 3.1 below, we present a static decentralized algorithm we have developed to solve the load sharing problem with real-time constraints. The approach is similar to the one used in problems with no real-time constraints (e.g. [7]). In section 3.1.1 we outline the algorithm, and in section 3.1.2, we describe the PA estimation procedure required in implementing this algorithm. In section 3.2 we include simulation results illustrating the performance of our algorithm. In section 3.3, we discuss extensions of the algorithm, including a dynamic version making use of state information. To actually develop such an algorithm, however, we have to derive extensions of our estimation procedures to accomodate discrete (integer-valued) parameters. We outline in section 3.4 the work we have done along those lines.

3.1. A Static Decentralized Adaptive Load Sharing Algorithm.

As in section 2.1, a distributed processing system with N processors is modeled as a network, with each node representing a processor. The flow of jobs arriving at node i is denoted by λ_i , $i=1,\dots,N$. The key idea of load sharing is to provide a control mechanism at each node i , so as to allocate the flow λ_i over all nodes. Thus, when a job is received at node i , two decisions are made:

1. Transfer decision: to keep the job at i or send it to some other node.
2. Location decision: to determine the node $j \neq i$ where the job should be sent (if the transfer decision is not to keep the job at i).

For simplicity, we assume that every node can communicate with every other node (however, this assumption can be easily relaxed). We will also initially assume that the communication delay in transferring a job is negligible (this assumption can be relaxed in the future).

Figure 2 shows the model of node i we will use. The responsibility of the "control" function is to split the flow λ_i into flows x_{ij} , $x_{ij} \geq 0$. Thus, jobs actually queued at node i for processing may

originate at any one of the locations indexed by $i=1, \dots, N$. In our model, there is a deadline τ_k associated with the k^{th} job for all $k=1, 2, \dots$. If w_k represents the waiting time of the k^{th} job, then the job is considered "lost" if $w_k > \tau_k$, i.e. it is rejected from the system and is not processed. The flow of lost jobs at node i is denoted by L_i , and will be referred to as the *loss rate* at i (in jobs/sec).

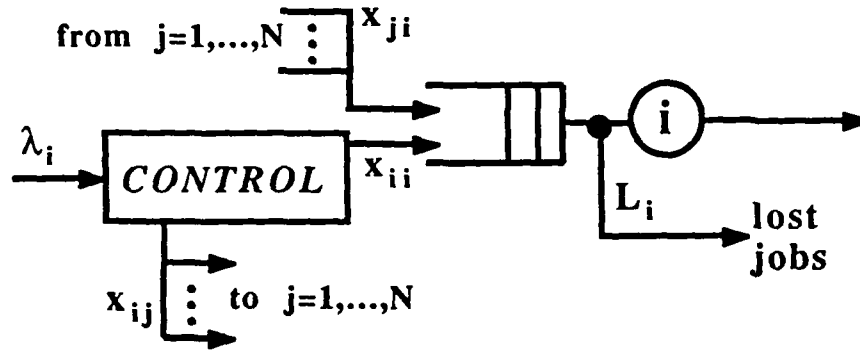


Figure 2: Node i Model

The objective of the load sharing algorithm is to determine the flows x_{ij} for all $i, j = 1, \dots, N$, so as to minimize the fraction of lost jobs in the overall system (i.e. the probability that a job violates its real-time constraint). Let P_L denote this fraction, and note that if a total of M jobs were observed, then the corresponding fraction $P_L(M)$ can be expressed as:

$$P_L(M) = \frac{1}{M} \sum_{i=1}^N M_i^L$$

where M_i^L is the number of lost jobs observed at node i . Equivalently, if we fix an observation interval to be of length T , then we can write:

$$P_L(T) = \frac{1}{M/T} \sum_{i=1}^N \frac{M_i^L}{T}$$

By letting $T \rightarrow \infty$, (M/T) becomes the total job flow into the system, denoted by f , and (M_i^L/T) becomes the loss rate at node i , defined above as L_i . Thus, we get:

$$P_L = \frac{1}{f} \sum_{i=1}^N L_i \quad - (1)$$

and since f is the fixed total system load, the algorithm must *minimize the sum of loss rates over all processors*. Thus, assuming the load at each node is given as f_i , the problem can be stated as follows:

determine $x_{11}, x_{12}, \dots, x_{ij}, \dots, x_{NN}$ so as to minimize:

$$\sum_{i=1}^N L_i(x_{1i}, \dots, x_{Ni})$$

s.t. $x_{ij} \geq 0$ for all $i, j = 1, \dots, N$ – (2a)

$$\sum_{j=1}^N x_{ij} = f_i \text{ for all } i = 1, \dots, N$$

– (2b)

Using standard results from optimization theory, one can show that the necessary conditions for solving this problem are the following:

$$\frac{\partial L_j}{\partial x_{ij}} = \psi_i \quad \text{if } x_{ij} > 0$$

– (3a)

$$\frac{\partial L_j}{\partial f_i} \geq \psi_i \quad \text{if } x_{ij} = 0$$

– (3b)

for all $i, j=1, \dots, N$, where ψ_i is some constant to be determined. The derivative $\partial L_j / \partial x_{ij}$ represents the *sensitivity* of the loss rate at node j with respect to a change in the flow x_{ij} . This is also referred to as the *marginal or incremental* loss rate at node j with respect to jobs coming from i . From node i 's point of view, the interpretation of these conditions is as follows: the job flows x_{ij} (allocated by i) must be set so that all marginal loss rates are equal, provided $x_{ij} > 0$; if $x_{ij} = 0$, then the corresponding marginal loss rate must be higher (i.e. node j is a particularly bad node). An algorithmic implementation can now be easily derived, whereby each node gradually adjusts its job flow allocations until conditions (3a), (3b) are satisfied. The crucial information required for the execution of such an algorithm consists of the marginal loss rates above.

Before describing the distributed algorithm, note that the number of marginal loss rate estimates needed is actually only N . To see this, let:

$$\beta_j = \sum_{i=1}^N x_{ij}, \quad j = 1, \dots, N$$

and observe that:

$$\frac{\partial L_j}{\partial x_{ij}} = \frac{\partial L_j}{\partial \beta_j} \frac{\partial \beta_j}{\partial x_{ij}} = \frac{\partial L_j}{\partial \beta_j}, \quad j = 1, \dots, N$$

which represents the sensitivity of the loss rate at node j with respect to the total incoming job flow β_j . This simplification holds as long as L_j is a function of β_j and not the N individual flows x_{ij} , i.e. as long as these flows are indistinguishable at node j .

3.1.1. Distributed Algorithm Description.

First, note that rather than controlling x_{ij} at node i , we can define *routing variables*:

$$\phi_{ij} = \frac{x_{ij}}{f_i}, \quad 0 \leq \phi_{ij} \leq 1$$

representing the fraction of job flows that node i allocates to j , $j=1, \dots, N$.

Next, let us define the following useful quantities:

$$a = \min_{j=1, \dots, N} \left\{ \frac{\partial L_j}{\partial \beta_j} \right\} \quad - (4)$$

$$a_j = \left[\frac{\partial L_j}{\partial \beta_j} - a \right] \quad - (5)$$

$$\Delta_{ij} = \min \left\{ \frac{x_{ij}}{f_i}, \eta \frac{a_j}{f_i} \right\} \quad - (6)$$

where:

- a is the minimum marginal loss rate over all nodes. The corresponding node is denoted by $k_{\min}$, and stands for the "best" node under the current load allocation, i.e. the best candidate for sending additional jobs to (since this node's loss rate will increase the least).
- a_j is the difference between the marginal loss rate at node j and the "best" marginal loss rate. Note that if $k_{\min} = j$, then $a_j = 0$.
- Δ_{ij} represents the adjustment to be made to ϕ_{ij} , based on current information. The quantity

η is called the *step size* of the algorithm, and regulates the amount of adjustment to be made at each iteration.

The precise algorithm execution is the following:

1. Initialize routing variables ϕ_{ij}^0 , $i, j = 1, \dots, N$.
2. Wait for some *observation period* T_m , $m = 1, 2, \dots$
3. Iteration m , $m = 1, 2, \dots$:
 - 3.1. Each node j estimates: $\frac{\partial L_j}{\partial \beta_j}$, $j = 1, \dots, N$ (see next section).
 - 3.2. Each node j sends $\frac{\partial L_j}{\partial \beta_j}$ to every other node $i \neq j$.
 - 3.3. Each node i determines a , $k_{\min}$, a_{ij} , and Δ_{ij} (defined above).
 - 3.4. Each node i updates its routing variables ϕ_{ij}^m , $j = 1, \dots, N$:

$$\phi_{ij}^{m+1} = \begin{cases} \phi_{ij}^m - \Delta_{ij}, & j \neq k_{\min} \\ \phi_{ij}^m + \sum_{k \neq k_{\min}} \Delta_{ik}, & j = k_{\min} \end{cases} \quad - (7)$$

4. Repeat steps 2 and 3.

The issue that remains is the estimation of the marginal loss at each node in step 3.1.

3.1.2. Marginal Loss Estimation.

Returning to the node model in Fig.2, we now address the question: how can we estimate the sensitivity of L_i with respect to the total incoming flow β_i ? We will restrict ourselves here to one approach which allows this process to be done on-line, based on the Perturbation Analysis (PA) methodology.

Consider a node in isolation, and let arriving jobs be indexed by $k=1, 2, \dots$. Let a_k denote the arrival time of the k^{th} job, d_k its departure time, and w_k its waiting time in the queue. Furthermore, suppose each job is assigned a processing time denoted by π_k for the k^{th} job. In our model, if w_k

$> \tau_k$ the job is lost (τ_k is the deadline for the job). Thus, if S_k is the actual service time the system provides this job, we can write:

$$S_k = \begin{cases} \tau_k, & \text{if } w_k \leq \tau_k \\ 0, & \text{otherwise} \end{cases}$$

It is easy to see that the departure time satisfies the following recursive equation:

$$d_k = \max \{d_{k-1}, a_k\} + S_k, \quad k=1,2,\dots \quad - (8)$$

Similarly, since $d_k = w_k + S_k$, the waiting time satisfies:

$$w_k = \max \{w_{k-1} + S_{k-1} - A_k, 0\}, \quad k=1,2,\dots \quad - (9)$$

where A_k is the k^{th} interarrival time defined by $A_k = a_k - a_{k-1}$. Note that $w_k = 0$ whenever the k^{th} job terminates an idle period at the processor. Defining:

$$I_k = a_k - d_{k-1} = A_k - w_{k-1} - S_{k-1} \quad - (10)$$

note that the duration of an idle period is given by I_k provided $I_k > 0$ in (10).

Now let the incoming flow β be perturbed by some amount $\delta\beta$. Equivalently, the mean interarrival time of jobs $\sigma=1/\beta$ is perturbed by an amount $\delta\sigma$. Thus, all interarrival times are perturbed by some amount δA_k which is easily obtained from $\delta\sigma$ depending on the job interarrival time distribution. This causes perturbations δw_k in the waiting times w_k . These perturbations, in turn, may affect the service times S_k . To understand this perturbation process in more detail, note that in a perturbed stochastic realization, (9) becomes:

$$w'_k = \max \{w'_{k-1} + S'_{k-1} - A'_k, 0\}, \quad k=1,2,\dots \quad - (11)$$

where $w'_k = w_k + \delta w_k$, $S'_k = S_k + \delta S_k$, and $A'_k = A_k + \delta A_k$. Similar to (10) we can also define I'_k as follows:

$$I'_k = A_k + \delta A_k - w_{k-1} - \delta w_{k-1} - S_{k-1} - \delta S_{k-1} \quad - (12)$$

Our goal now is to determine recursive expressions for δw_k and δS_k combining the equations above. As long as these expressions depend only on quantities which are directly observable while

the system is in operation, one can always predict on-line the effect of a flow perturbation.

Following some algebraic manipulations, we get:

$$\delta w_k = \begin{cases} 0 & \text{if } I_k > 0, I'_k > 0 \\ \delta w_{k-1} + \delta S_{k-1} - (I_k + \delta A_k) & \text{if } I_k > 0, I'_k \leq 0 \\ \delta w_{k-1} + \delta S_{k-1} - \delta A_k & \text{if } I_k \leq 0, I'_k > 0 \\ I_k & \text{if } I_k \leq 0, I'_k \leq 0 \end{cases} \quad - (13)$$

$$\delta S_k = \begin{cases} \pi_k & \text{if } w_k > \tau_k, \delta w_k \leq \tau_k - w_k \\ -\pi_k & \text{if } w_k \leq \tau_k, \delta w_k \leq \tau_k - w_k \\ 0 & \text{otherwise} \end{cases} \quad - (14)$$

where it is important to observe that δw_k and δS_k are evaluated based on known information: δw_k and δS_k are the iteration variables (initialized to 0); τ_k and π_k are given for every job; I_k is obtained from (10); δA_k is computed based on the flow perturbation of interest and the interarrival time distribution; and I'_k is obtained from (12) using known values.

From a computational standpoint, the procedure for obtaining δw_k involves only simple arithmetic and comparisons, as shown in (13), (14). In addition, it is assumed that all arrival time information is stored with the job (time stamping). The only additional burden is the need to save prior service time information δS_{k-1} in evaluating δw_k .

Of course, our ultimate objective in the distributed load sharing algorithm is to estimate derivatives of the form $\partial L / \partial \beta$, where L is the loss rate at the standalone processor model we have considered here. Given δw_k , however, this is a relatively simple task. Let M_k^L be the number of lost jobs after k jobs have been served (either actually processed or rejected), and let δM_k^L the perturbed value due to $\delta \beta$. We can now evaluate δM_k^L on-line as follows:

$$\delta M_{k+1}^L = \begin{cases} \delta M_k^L - 1 & \text{if } w_k > \tau_k, \delta w_k \leq \tau_k - w_k \\ \delta M_k^L + 1 & \text{if } w_k \leq \tau_k, \delta w_k \leq \tau_k - w_k \\ 0 & \text{otherwise} \end{cases} \quad - (15)$$

Finally, an estimate of the derivative $\partial L / \partial \beta$ is given by:

$$\left(\frac{1}{d_k}\right) \frac{\delta M_k^L}{\delta \beta}$$

where d_k defines the length of the observation period on which the estimate is based, $\delta \beta$ is sufficiently small, and δM_k^L is obtained from (15), using (13) and (14) to evaluate δw_k .

3.2. Simulation Results.

In this section, we present results from the algorithm implementation on simulated distributed processing systems. Our objectives here are:

- to demonstrate the convergence of this load sharing algorithm, both for simple models (where analytical solutions may be found), and more complex ones (for which analytical solutions are not available).
- to demonstrate the adaptive nature of the algorithm, where flows are automatically adjusted in response to drastic changes in the system's operating conditions.
- to study the effect of the two parameters affecting the performance of this algorithm: the step size η , and the observation period length T , which defines the points where flow adjustments are made.

In the results that follow, we denote by A_i and S_i the arrival and service process characteristics at node i . We also denote by C_i the deadline (waiting time constraint) distribution. For instance, $A_2: \text{EXP}(1.0)$ indicates that the interarrival times at node 2 are exponentially distributed with mean 1.0; $C_1: \text{CO}(2.0)$ indicates that all jobs submitted to node 2 have a constant deadline fixed at 2.0 sec.

In the first few cases studied, we have considered a four-node system and examined the following cases.

Case 1: $A_1, A_2, A_3, A_4: \text{EXP}(1.0)$
 $S_1: \text{EXP}(1.0), S_2, S_3, S_4: \text{EXP}(4.0)$
 $C_1, C_2, C_3, C_4: \text{CO}(2.0)$

In Fig. 3, for a fixed observation period of length T , defined by 30,000 jobs per iteration, we

show how the performance (in terms of fraction of lost jobs) improves as a function time and convergence is attained, provided the step size η is sufficiently small. Note the instability resulting at $\eta=5.0 \times 10^{-1}$. In Fig. 4, we study the effect of observation period length for a fixed step size $\eta=5.0 \times 10^{-2}$. For the case of 1,000 jobs per iteration only, performance initially tends to the optimum very rapidly; subsequently, however, the loss fraction experiences oscillations due to the high variance of the marginal loss estimates.

Case 2: A_1, A_2, A_3, A_4 : EXP(1.0)
 S_1 : EXP(1.0), S_2, S_3, S_4 : EXP(4.0)
 C_1, C_2, C_3, C_4 : UN[1.5,2.5]

The only change here is in the deadline distribution: deadlines are now drawn from a uniform distribution in [1.5,2.5] (mean deadline is the same). Results are shown in Fig. 5, with $\eta=5.0 \times 10^{-2}$ and 20,000 jobs per iteration. Note that the loss fraction converges around 0.60 as before.

Case 3: A_1, A_2, A_3, A_4 : EXP(1.0)
 S_1 : UN[0.5,1.5], S_2, S_3, S_4 : UN[3.5,4.5]
 C_1, C_2, C_3, C_4 : CO(2.0)

In Fig. 6, we show algorithm convergence with $\eta=5.0 \times 10^{-2}$ and 20,000 jobs per iteration. In this case, however, node service times are bounded through uniform distributions. It is still possible to obtain analytical expressions for loss rates in this system; however, our aim here is simply to demonstrate the validity of our estimation procedure, which does not require any change compared to the previous models.

Case 4: A_1, A_2, A_3, A_4 : UN[0.5,1.5]
 S_1 : UN[0.5,1.5], S_2, S_3, S_4 : UN[3.5,4.5]
 C_1, C_2, C_3, C_4 : CO(2.0)

This is similar to Case 3, except for the interarrival time processes, which are also uniform. No analytical expressions are available for such node models. Results with $\eta=5.0 \times 10^{-2}$ and 20,000 jobs per iteration are shown in Fig. 7.

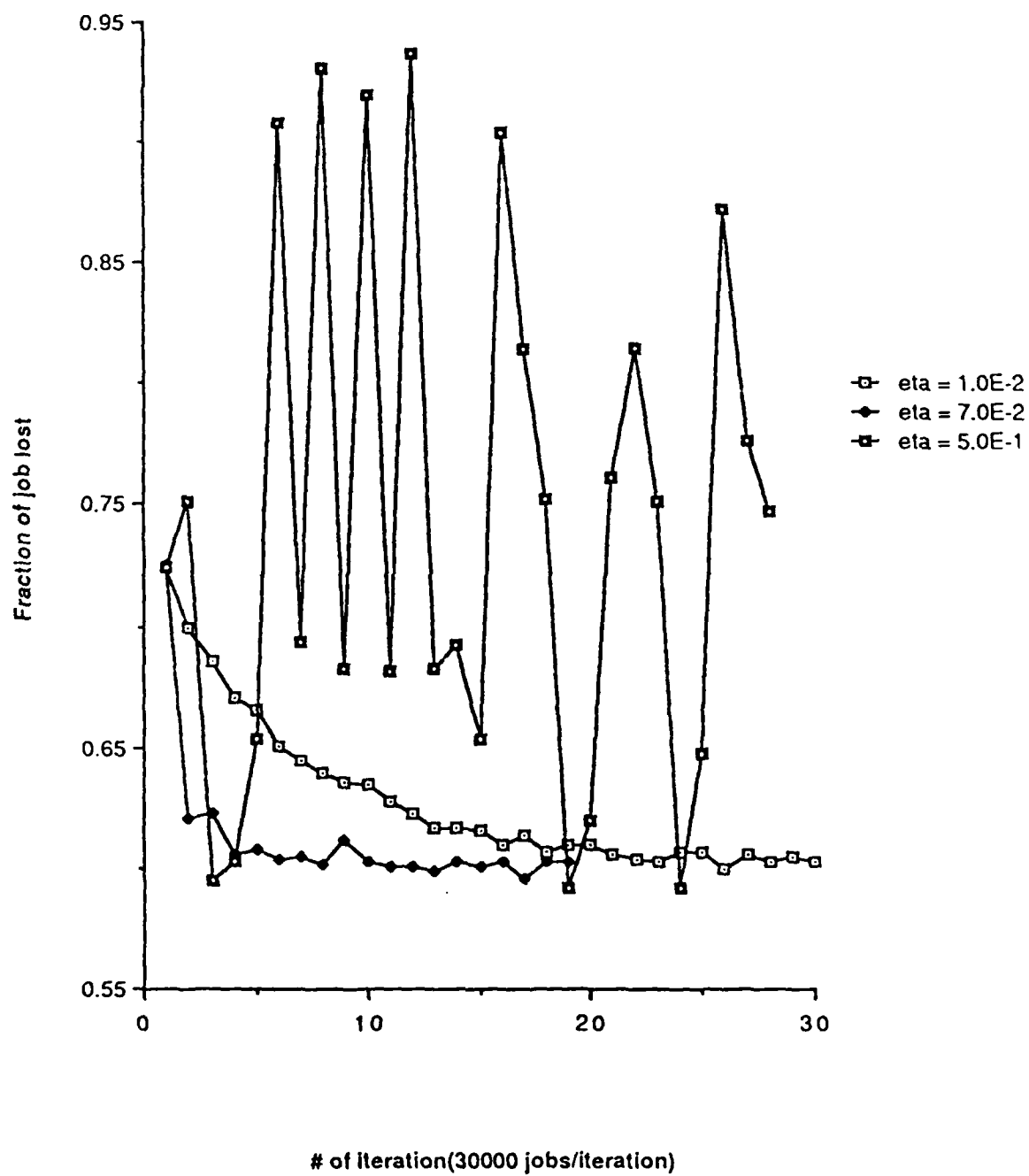


Figure 3: Effect of Step Size on Load Sharing Algorithm (4 nodes, Case 1)

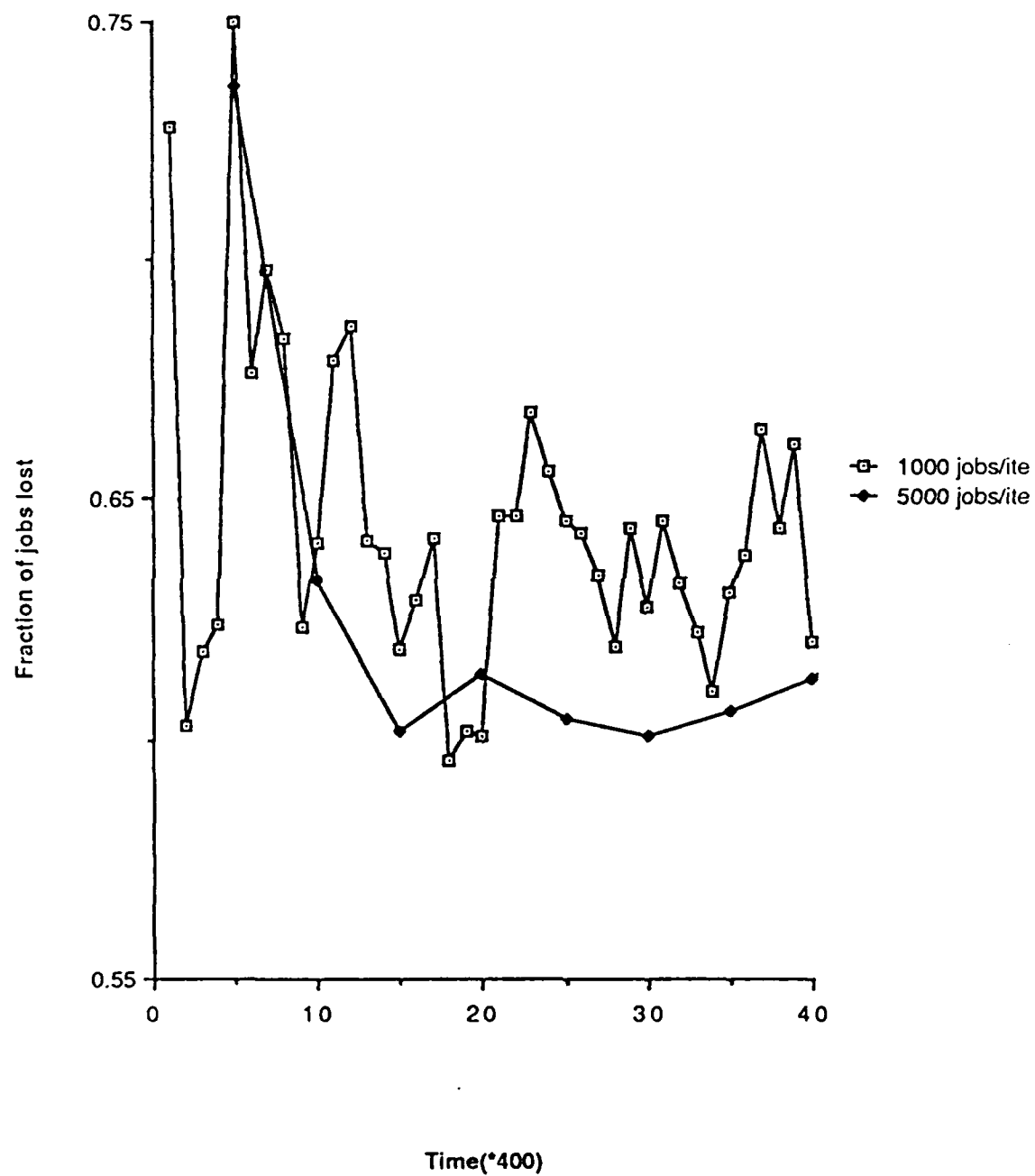


Figure 4: Effect of Observation Period Length on Load Sharing Algorithm (4 nodes, Case 1)

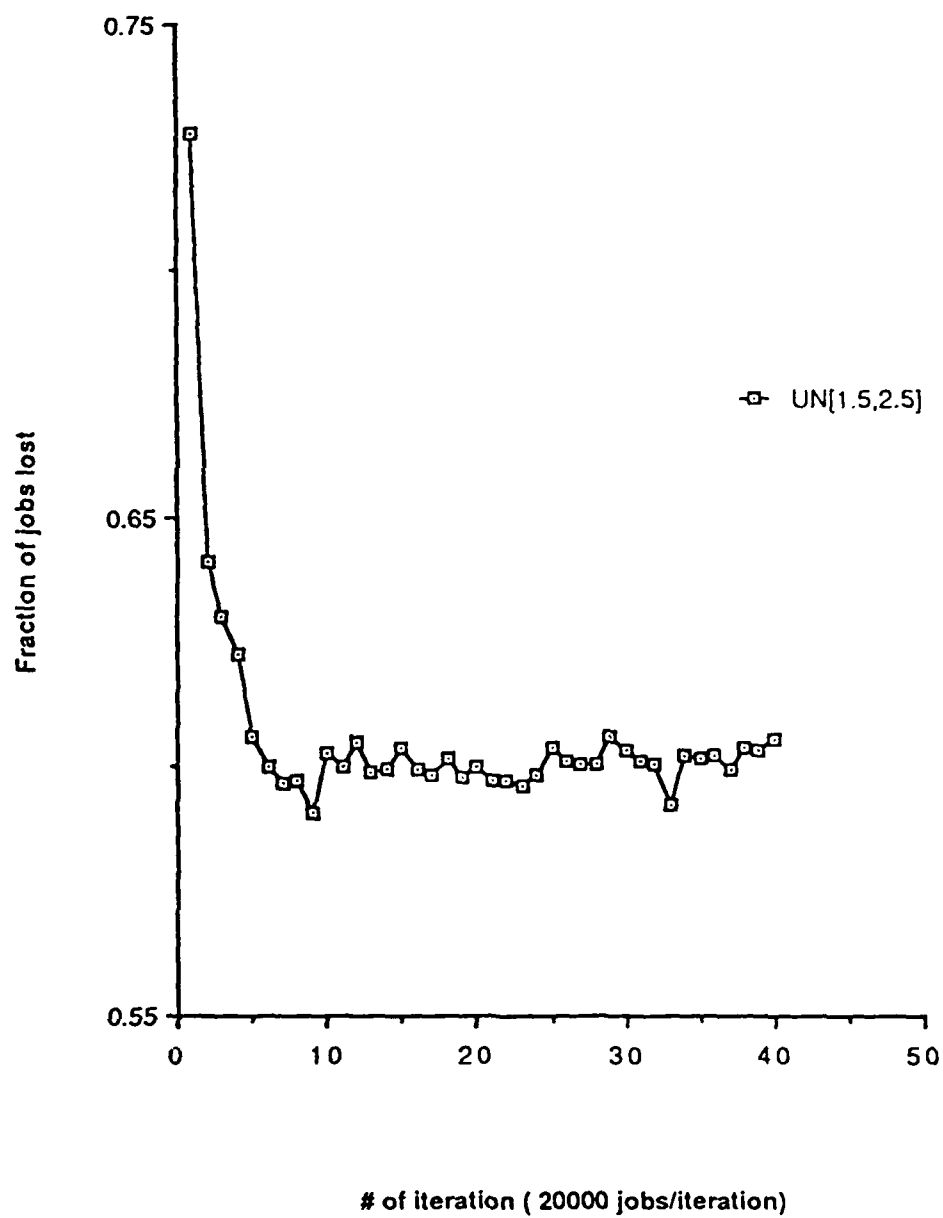


Figure 5: Uniformly Distributed Waiting Time Constraint (4 nodes, Case 2)

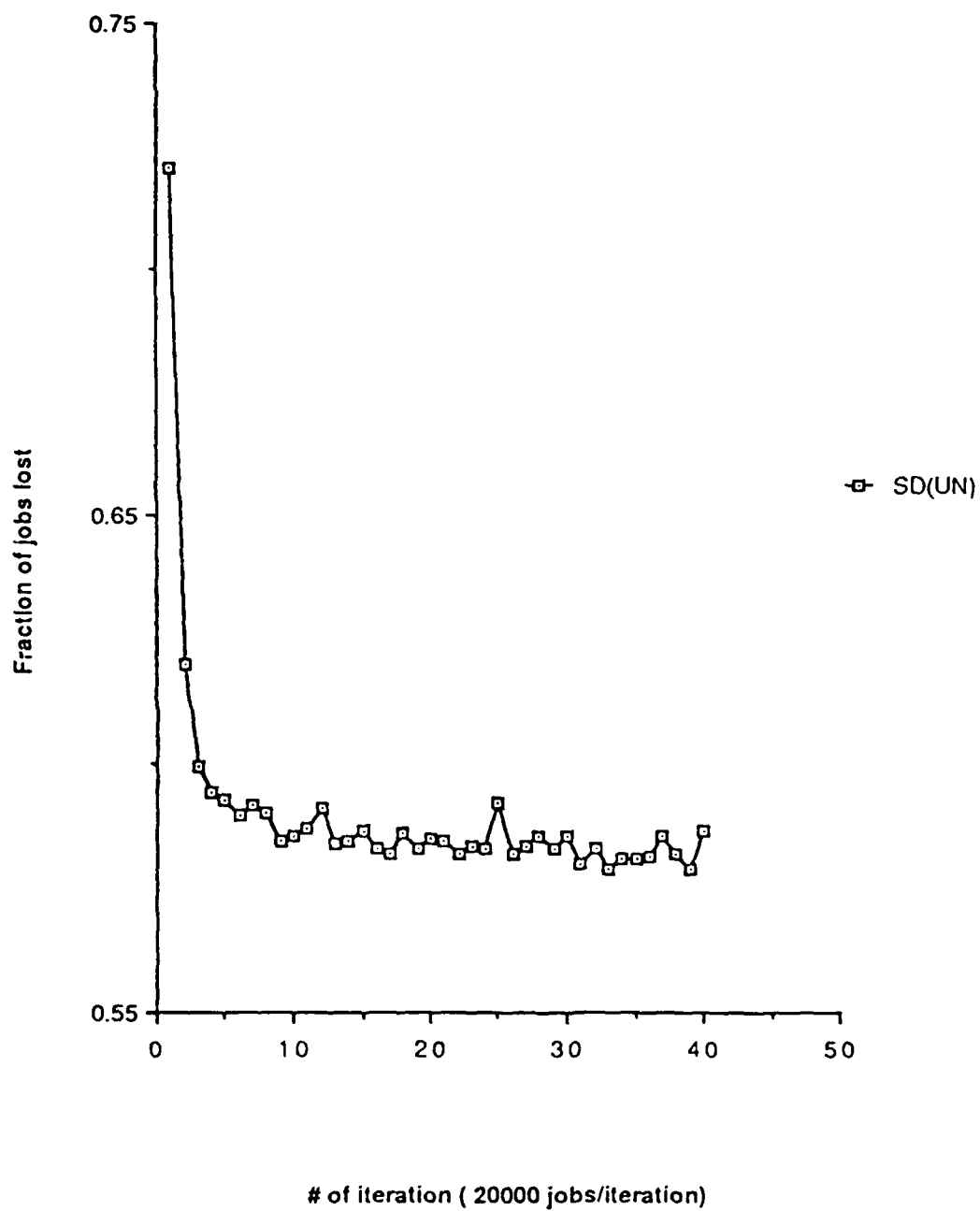


Figure 6: *Uniformly Distributed Service Times (4 nodes, Case 3)*

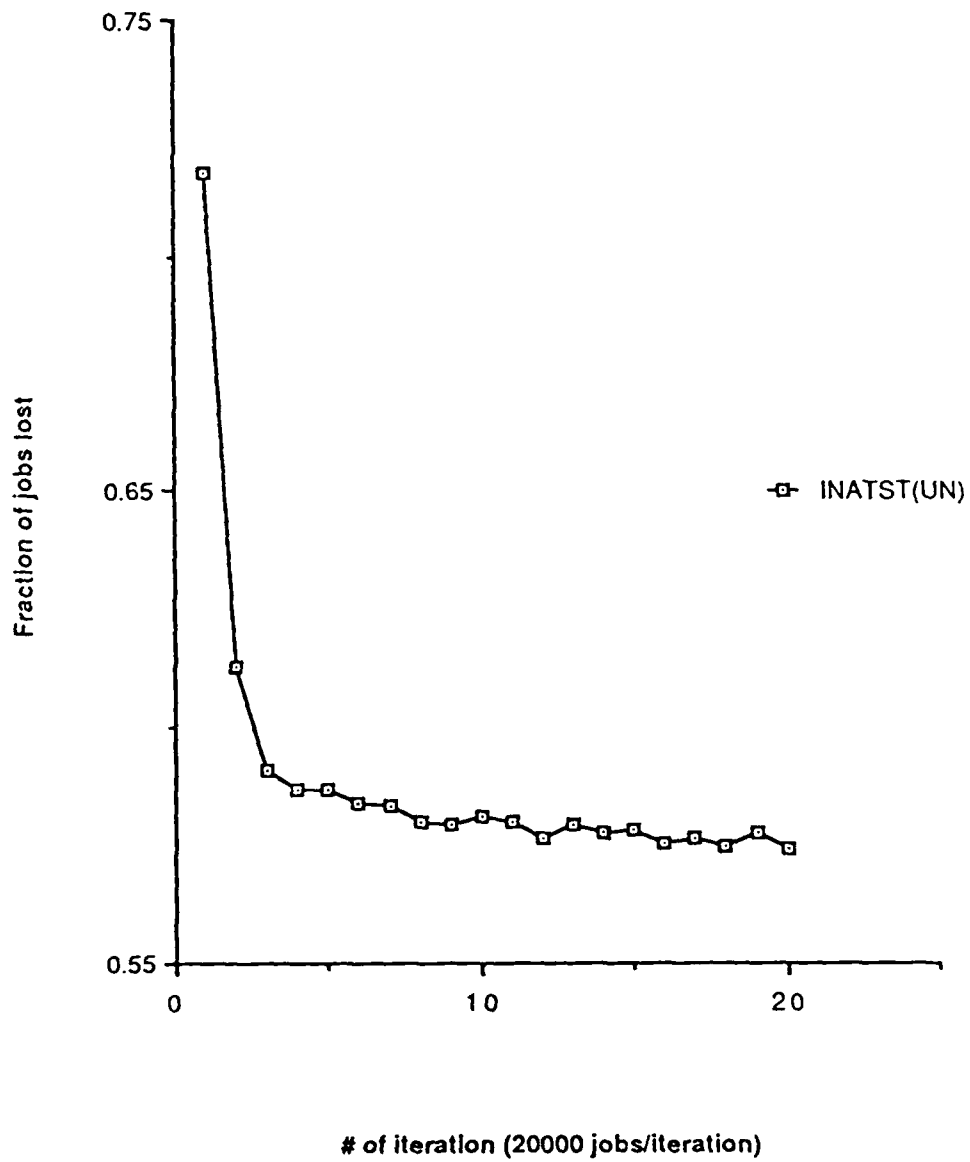


Figure 7: Uniformly Distributed Intrarrival and Service Times (4 nodes, Case 4)

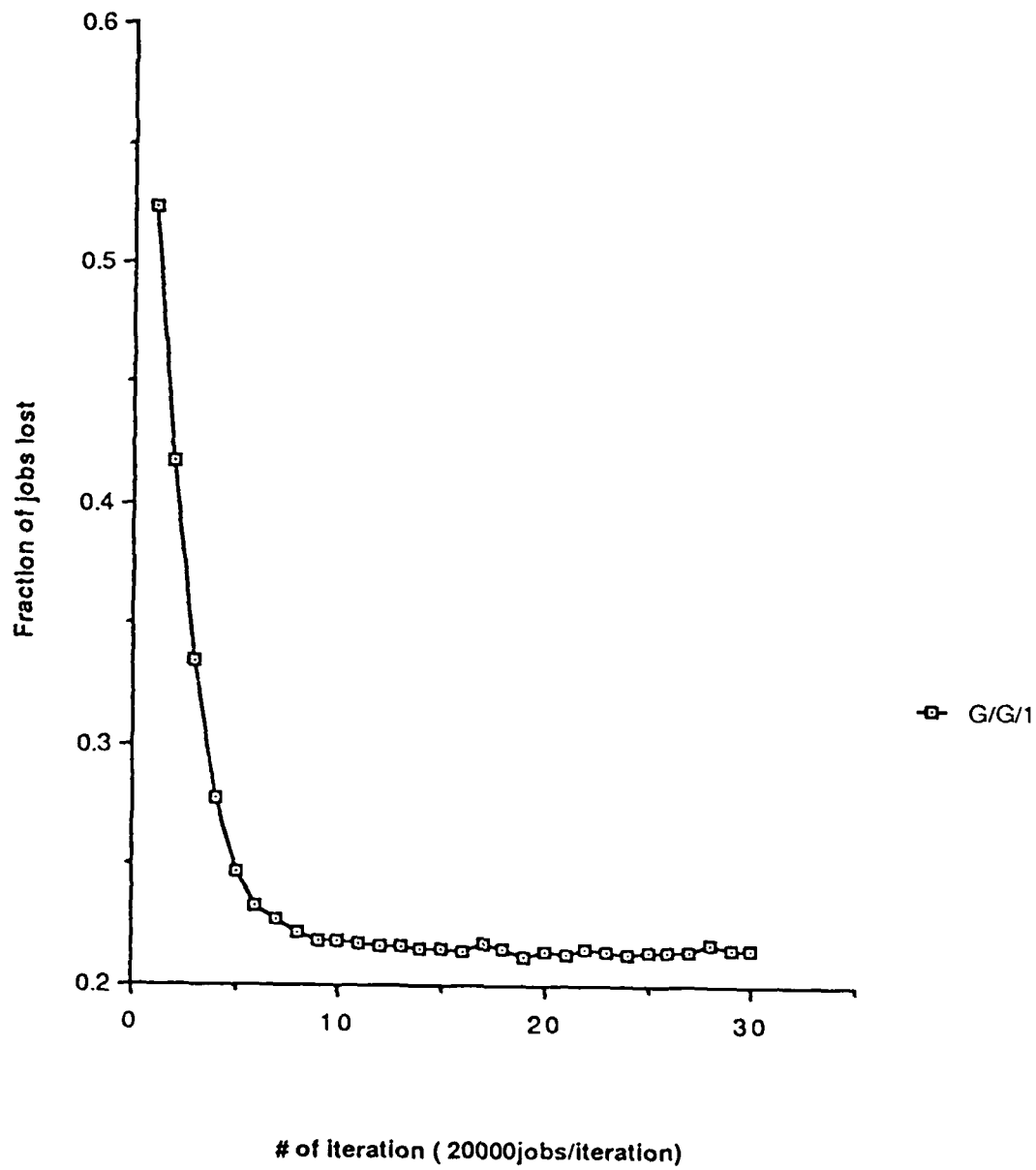


Figure 8: *Generally Distributed Intrarrival and Service Times (4 nodes, Case 5)*

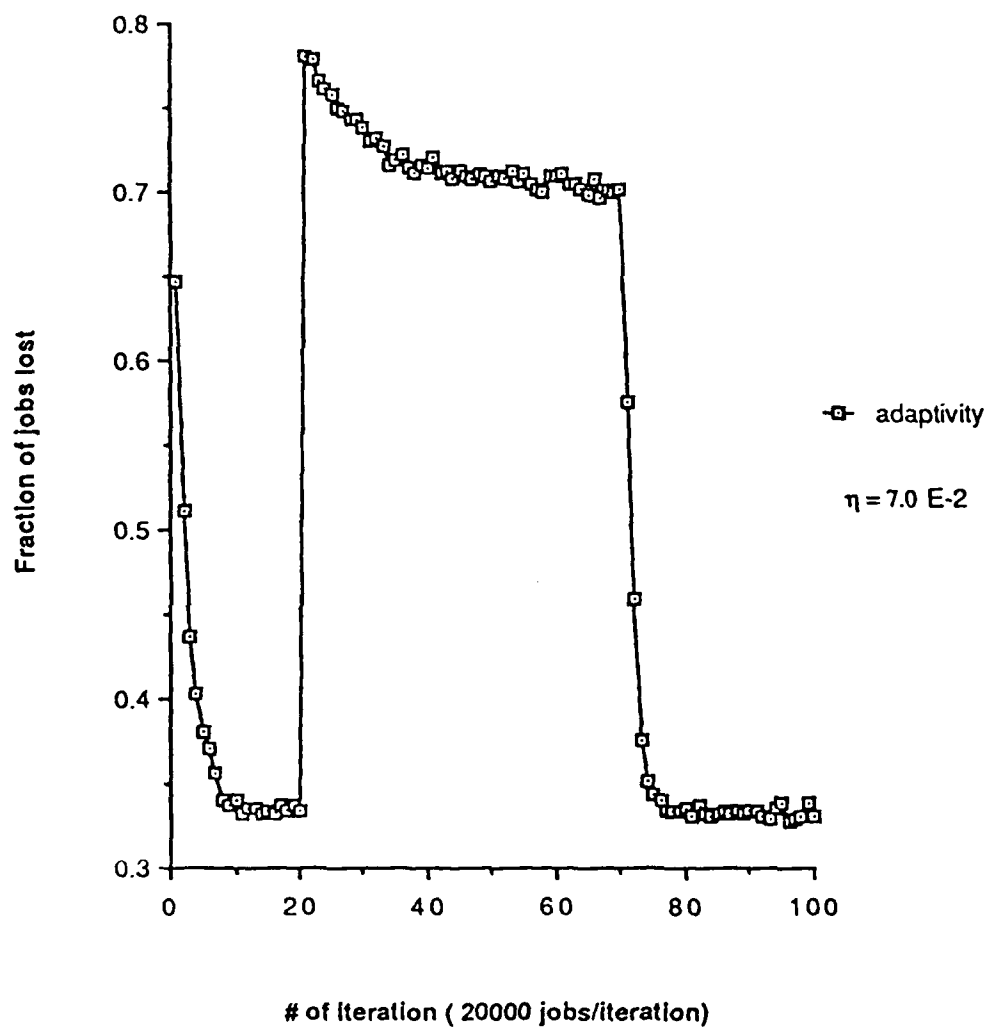


Figure 9: Adaptivity of Load Sharing Algorithm (4 nodes, Case 6)

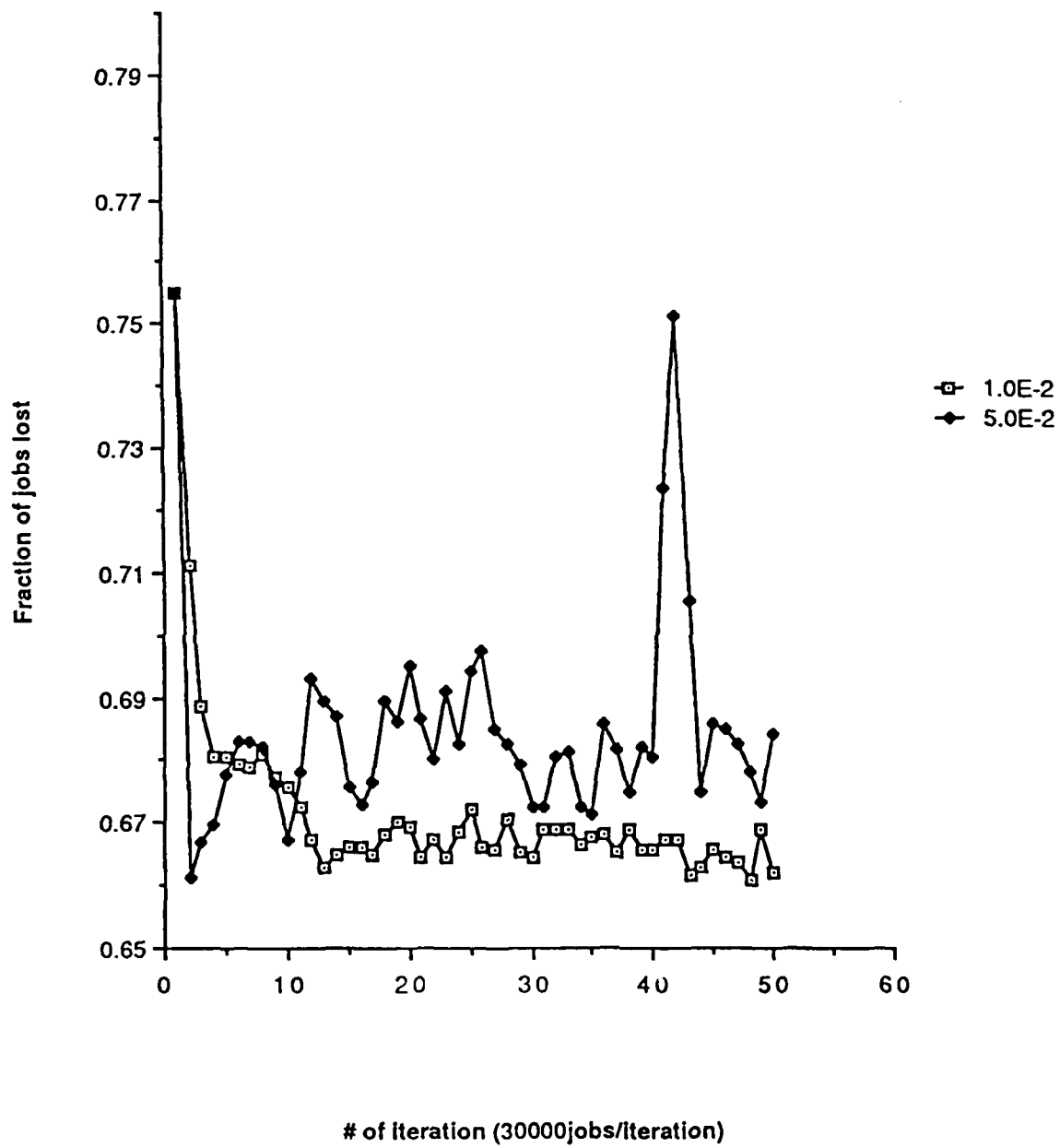


Figure 10: *Effect of Step Size on Load Sharing Algorithm (7 nodes)*

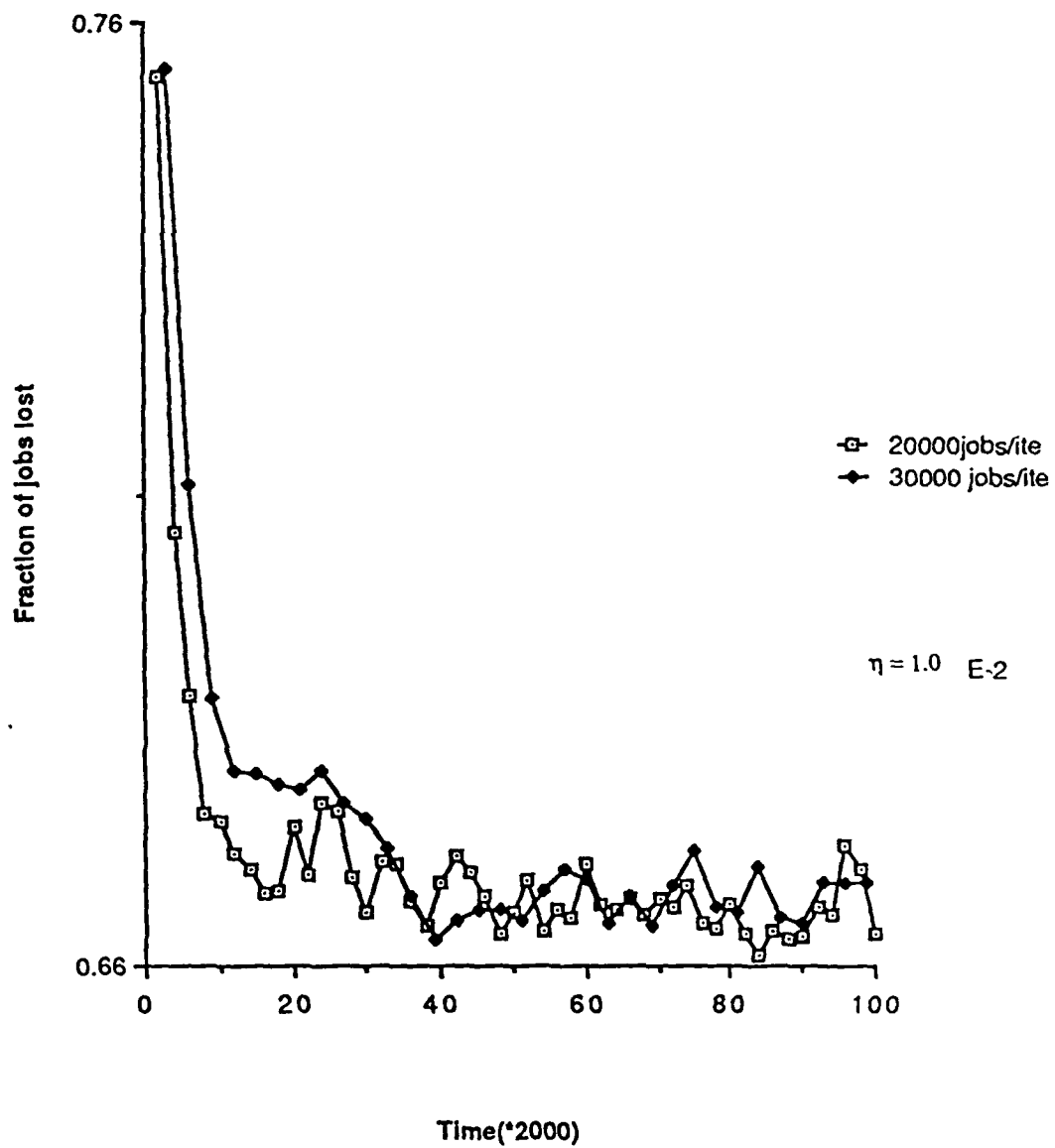


Figure 11: Effect of Observation Period Length on Load Sharing Algorithm (7 nodes)

Case 5: A_1 : U.[0.5,1.5], A_2 : EXP(4.0), A_3 : CO(5.0), A_4 : DP(3.0,0.8; 8.0,0.2)
 S_1 : UN[0.5,1.5], S_2, S_3, S_4 : UN[3.5,4.5]
 C_1, C_2, C_3, C_4 : CO(2.0)

In this case, arrival processes are quite different at each node. At node 4, we have a discrete probability distribution, i.e. the interarrival time is 3.0 with probability 0.8 and 8.0 with probability 0.2. Once again, we show convergence in Fig. 8, with $\eta=5.0 \times 10^{-2}$ and 20,000 jobs per iteration.

Case 6: A_1 : EXP(1.0), A_2 : UN[3.5,4.5], A_3 : UN[7.5,8.5], A_4 : EXP(3.0)
 S_1 : EXP(1.0), S_2 : EXP(4.0), S_3 : EXP(6.0), S_4 : EXP(8.0)
 C_1, C_2, C_3, C_4 : CO(2.0)

Our purpose here is to demonstrate the adaptive properties of this algorithm in a general system with different arrival processes and inhomogeneous processors (node 4 is 8 times faster than node 1).

In Fig. 9 we show the behavior of the algorithm when node 1 experiences a degradation of a factor of 20 (i.e. the mean service time becomes 20.0 after the 20th iteration). As expected, the fraction of jobs lost immediately increases (from about 0.33 to about 0.78). The load sharing is then gradually adjusted to a new optimal allocation with a loss fraction of about 0.70. Finally, the initial service rate of node 1 is restored, and load sharing gradually returns to an allocation yielding a loss fraction of about 0.33.

In the remaining cases examined, we have considered a seven-node system. As shown in Figures 10-11, the basic properties of the algorithm are unaffected by an increase in the size of the system. The main effect is in the selection of appropriate step size and observation periods, which become more constrained. In Fig. 10, note that the algorithm tends to become unstable even if $\eta=5.0 \times 10^{-2}$, a value which was sufficiently small in the four-node model.

3.3. Extensions to Dynamic Load Sharing.

For the static distributed load sharing algorithm considered above, it is clear that the choice of step size and observation period length parameters is critical in guaranteeing fast and reasonably smooth convergence. There are several simple enhancements one can immediately notice, which remain to be done. Specifically, there is no reason that these two parameters should remain fixed throughout the algorithm execution; it is reasonable to start out with large step size and short observation periods, which can provide fast initial improvement. Subsequently, these parameters can be adjusted to avoid instability and to gradually approach optimal performance.

Another issue that remains to be addressed is that of the effect of communication delays in transferring jobs through a network. When such delays are not negligible, we can no longer replace the individual marginal losses $\partial L_j / \partial x_{ij}$ by the single derivative $\partial L_j / \partial \beta_j$, where β_j is the total flow into node j . In other words, rather than a single class of jobs, node j must now distinguish between N classes (depending on the source of the job), or at least two classes: local and remote.

The next interesting task is that of extending load sharing to include instantaneous state information, such as the *job backlog* (queue length) x_i at node i . One can then investigate *threshold-based* load sharing schemes, which operate as follows:

- whenever a job is submitted to node i , check the queue length x_i and compare it to some specified threshold T_i (to be determined).
- if $x_i < T_i$, then keep the job at node i .
- if $x_i \geq T_i$, then send the job to some other node, using routing variables ϕ_{ij} , $j \neq i$.

Thus, the problem here involves both adjusting the thresholds and the routing variables. Note that T_i is integer-valued, hence standard gradient estimation techniques (including PA) are not applicable. It is often the case that such systems are characterized by discrete (integer-valued) parameters. Part of our work has therefore focused on investigating how to obtain sensitivity estimates in this case. This work is outlined in section 3.4, and described in more detail in [8] (also Appendix B).

3.4. Sensitivity Analysis for Distributed Processing Systems with Discrete Parameters.

The problem we have addressed is the following. Suppose a system is characterized by several discrete parameters (such as the thresholds defined above). Selecting the optimal values of these parameters can drastically affect the performance of the system. Furthermore, these parameters can be used to automatically adjust the system to changing operating conditions (e.g. processor failures, sudden traffic increases). However, to be able to make such adjustment requires knowledge of the *performance sensitivity* with respect to the parameters. This information is generally very hard to obtain, since the functional relationship between performance measures and parameters is not available.

The main idea we have investigated is that of modeling systems of interest through *augmented* Markov or semi-Markov chain models. We have developed a general framework for obtaining the types of sensitivities mentioned above, and have verified its validity for some simple cases (see Appendix B). This approach is still based on direct observation of a system in operation, and requires little overhead. It remains to use this approach in order to implement a dynamic load-sharing scheme as described in the previous section.

Another area where this approach appears to be promising is that of *scheduling* different types of jobs at a processor. This is a complex problem of significant practical interest, since it is often the case that jobs are classified in terms of priority, real-time constraints, execution length, or other characteristics. We present a brief overview of the problem in the next section..

3.4.1. Dynamic Processor Scheduling.

As shown in Fig. 12, the scheduling problem involves selecting the next job to be processed from a collection of K queues, each representing a different class. In the simple case where the process rate of class k is μ_k , and a measure of priority is represented by the waiting cost per unit time c_k , it

can be shown that the policy minimizing the mean job delay is a simple *static* one: always process a job from the class with the highest $(\mu_k c_k)$ value [9]. If, however, real-time constraints are present, queue capacities are limited, or other complications are introduced, a *dynamic* scheduling policy is expected to provide better performance. For some simple cases, we have early results showing that threshold based policies are in fact optimal.

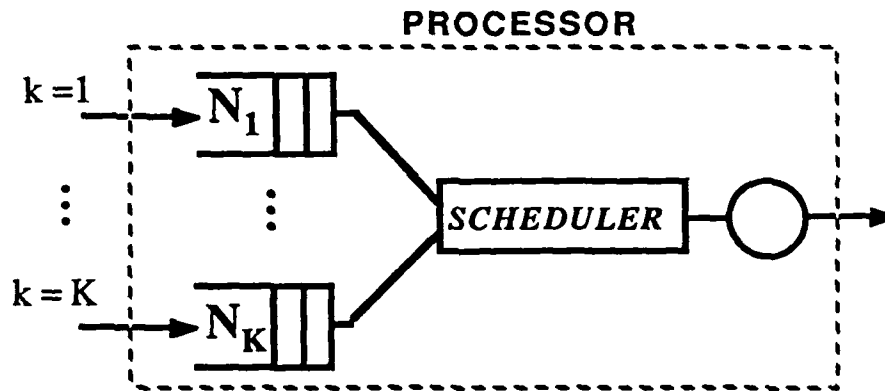


Figure 12: The Processor Scheduling Problem

The policy we have formulated to be analyzed using the augmented chain approach described in Appendix B is the following. Suppose job classes have been prioritized so that the highest priority jobs are in queue 1, and so on. Our objective here may be to minimize a combination of average delays and loss fractions for jobs with deadlines. Then, for the case where $K=2$, consider:

- if $N_1 > T_1$, process class 1
- if $N_1 \leq T_1$, then: if $N_2 > T_2$, serve class 2, otherwise serve class 1.

In this scheme, the processor only serves class 2 jobs if queue 1 is sufficiently low and queue 2 sufficiently high. As in other threshold-based policies, the question is that of determining the optimal values of T_1 , T_2 . This problem remains to be solved, and comparisons with other scheduling policies remain to be made.

4. DESIGN AND ANALYSIS OF PARALLEL PROCESSING SYSTEMS.

In this section we report on two problems related to parallel processing systems that we studied.

In the first problem we studied the behavior of two different scheduling policies on a multiprogrammed multiprocessor that executes parallel programs. In the second problem, we developed mathematical models for a class of parallel systems that can be modeled as *acyclic fork-join* queueing networks. We report on each of these in the remainder of this section.

4.1. Multiprocessor scheduling

We studied the performance of a first come first serve (FCFS) and processor sharing (PS) policies for scheduling parallel programs on a multiprogrammed multiprocessor. Specifically, we developed analytic models that predict the behavior of PS when used to schedule *fork/join* jobs onto a multiprocessor and compared its performance to FCFS. Here a fork/join job consists of a number of tasks that can be executed independently of each other. The job is not considered to be complete until the last task completes. The fork/join job is the simplest nontrivial example of a parallel job.

We developed an analytic model that provides tight bounds on the expected response time of a fork/join job under the assumptions that jobs arrive to the multiprocessor according to a Poisson process and that task service times are independent and identically distributed exponential random variables. Details of the analysis can be found in [10] (also Appendix C). We study two PS disciplines, one called *task scheduling* processor sharing, the other *job scheduling* processor sharing. The first policy schedules tasks independently of each other, thus allowing parallel execution, whereas the second policy schedules entire jobs to individual processors. The second policy does not allow parallel execution of a job. We find that task scheduling *does not always outperform job scheduling*. Specifically, job scheduling always performs better when the processor utilizations are high. This is because at high utilizations there is little advantage to parallel execution of a single job. On the other hand, task scheduling gives preference to jobs with many tasks over jobs with few tasks unlike job scheduling which gives equal preference to all jobs. Consequently, small jobs complete more quickly at high utilizations under job scheduling.

We also compare processor sharing with FCFS. We find that FCFS outperforms processor sharing for a large class of workloads. We also compare the performance of processor sharing and FCFS for systems with two classes of jobs. We find that the system performs poorly when the processors are partitioned between the classes as compared to a system that shares the processors amongst all jobs.

There remain many unanswered questions. These include: What are the effects of priorities on the behavior of different classes of jobs? What are the effects of real time constraints? How should job and task scheduling be integrated together to achieve the best features of each policy?

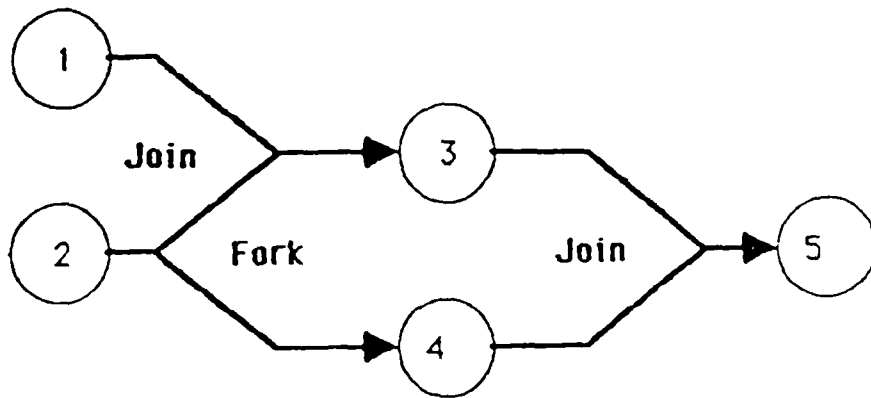
4.2. Models of Parallel Systems.

We studied a class of *acyclic fork-join queueing networks* (AFJQN's) that arise in the performance analysis of parallel processing applications. We obtained the maximum throughputs and developed upper and lower bounds on the response times of jobs that execute in these systems. We describe what an AFJQN is and the results of our analysis in the remainder of this section.

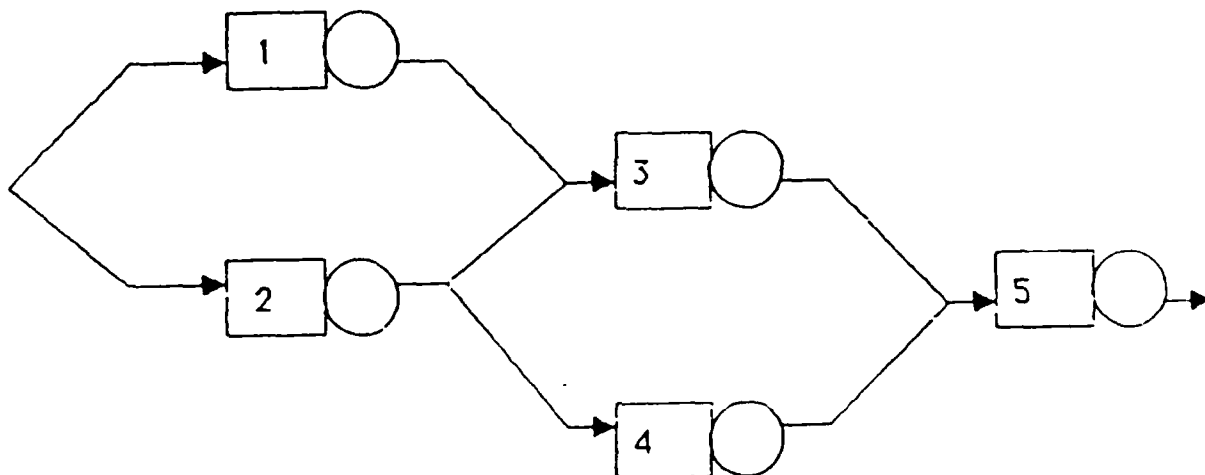
AFJQN's arise naturally in parallel processing applications. Many parallel programs are decomposed into tasks, each of which can execute on a separate processor. The division of the parallel program into tasks can be described by a directed graph where the nodes of the graph correspond to tasks and directed edges represent the precedence relations between tasks. In many cases the underlying graph is acyclic and the program is implemented with the use of *fork* and *join* constructs. Briefly, a fork exists at each point in a parallel program that one or more tasks can be initialized simultaneously ⁽¹⁾. A join occurs whenever a task is allowed to begin execution following the completion of one or more tasks. Forks and joins are reflected in the underlying computation graph in the following manner. A task that has one or more outgoing edges corresponds to a fork. A task with one or more incoming edges corresponds to a join. These are

(1) Strictly speaking a fork implies that {two or more} tasks are started. However, our definition simplifies the notation required for the analysis

exemplified in the **parbegin** and **parend** constructs available in parallel programming languages such as Concurrent Pascal [11], Concurrent Sequential Processes (CSP) [12], and Ada [13].



(a)



(b)

(a) A program

(b) The associated
Fork join
Queueing network

Figure 13: (a) A parallel program. (b) Associated AFJQN.

Consider a multiple processor where each task of a specific program is mapped onto a separate processor. The execution of a single program request can be described as follows: (i) Upon completion of a marked task, tokens associated with the program are routed to each processor handling the tasks that follow the marked task in the underlying computation graph; (ii) Once a processor has received tokens from all tasks that precede a marked task in the computation graph, this processor is allowed to execute it. Let this multiprocessor be required to service a stream of requests corresponding to different instances of that program and assume each processor executes its tasks in the same order that program requests arrive to the system. We have described, in brief, an AFJQN. Figure 13a illustrates a hypothetical parallel program using forks and joins, and Fig. 13b illustrates the corresponding fork-join queueing network.

This class of queueing networks has not, in general, been solved. In our work, we have obtained expressions for the maximum throughput in job requests per unit time that can be processed for an arbitrary computation graph where the number of processors is at least as large as the number of tasks and for very general assumptions on the job request process and service time requirements of all of the tasks. In addition, we have obtained upper and lower bounds on the expected program execution time through the use of stochastic ordering principles (see [14]). We have shown, for example, that decreased (increased) variability in the time between job requests results in a decrease (increase) in the job execution time. Consequently we can numerically obtain bounds by assuming that the times between job arrivals are constant. In addition, we have shown that if we assume that the times required to traverse each path between the source and the destination in the AFJQN, then we obtain a pessimistic bound on the average response time by taking the average of the {maximum} of the times over all paths between source and destination. Details of the analysis can be found in [15] (also Appendix D).

A number of tasks remain to be done and a number of interesting questions remain to be answered. For example, we have not developed a software system to actually calculate bounds on the mean program execution time. In addition, there are numerous other parallel processing

architectures to be considered such as one where tasks are not mapped to a processors but, rather, a processor is allowed to execute any task that is ready for execution. The work reported above does not address some of the issues raised in real-time systems. For example, what is the probability that a job will miss a deadline?

5. CONCLUSIONS.

We have addressed both system-level and node-level issues in distributed systems. At the system level, we have considered load sharing for jobs with real-time constraints, and determined that simple policies can provide performance very near the ideal optimum. We have also derived and tested load sharing algorithms which can be implemented under general conditions, requiring no specific modeling assumptions or knowledge of system parameters. At the node level, we have formulated a task scheduling problem, and have investigated some parallelism issues for the case of multiprocessor nodes. We have determined that the advantages of parallelism are dependent on several factors, and that a simple FCFS approach is occasionally preferable.

In the development of adaptive load sharing algorithms, we have limited ourselves to the static case. We have, however, obtained in the course of our work a general framework for on-line marginal loss estimation to be used for extensions to the dynamic case. This is the subject of future work. Furthermore, an issue to be addressed is that of the interaction between the system level and node control in the presence of real-time constraints. The task scheduling problem itself also remains to be addressed in detail; our results to-date have generated a suitable framework for accomplishing this in the near future. Finally, our work on parallelism issues has given rise to a number of problems, such as the effect of priorities, and the question of effectively combining job and task scheduling to achieve the best features of each.

REFERENCES

- [1] D. Eager, E. Lazowska, and J. Zahorjan, "Dynamic Sharing in Homogeneous Distributed Systems", *IEEE Trans. Software Eng.*, SE-12, 5, pp. 662-675, 1986.
- [2] J. Stankovic, K. Ramamritham, and S. Cheng, "Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems", *IEEE Trans. Comput.*, C-34, 12, pp. 1130-1144, 1986.
- [3] R. Chipalkatti, and J.F. Kurose, "Load Sharing in Soft Real-Time Distributed Computer Systems", unpublished manuscript, 1987.
- [4] C.G. Cassandras, and S.G. Strickland, "Perturbation Analytic Methodologies for Design and Optimization of Communication Networks", to appear in *IEEE J. on Selected Areas in Commun.*, 1988.
- [5] M.I. Reiman, and A. Weiss, "Sensitivity Analysis via Likelihood Ratios", *Proc. 1986 Winter Simulation Conf.*, pp. 285-289, 1986.
- [6] R.Y. Rubinstein, "The Score Function Approach for Sensitivity Analysis of Computer Simulation Models", to appear in *Mathematics and Computers in Simulation*, 1987.
- [7] R.G. Gallager, "A Minimum Delay Routing Algorithm Using Distributed Computation", *IEEE Trans. on Communications*, COM-25, 1, pp. 73-85, 1977.
- [8] S.G. Strickland, and C.G. Cassandras, "Augmented Chain Analysis of Markov and Semi-Markov Processes", *Proc. 25th Allerton Conf.*, 1987.
- [9] J. Baras, A. Dorsey, and A. Makowski, "Two Competing Queues with Linear Costs and Geometric Service Requirements: The μ c-rule is Often Optimal", *Adv. Appl. Prob.*, 17, 186-209, 1985.
- [10] D. Towsley, J.A. Stankovic, and C.G. Rommel, "A Comparison of the Processor Sharing and First Come First Serve Policies for Scheduling Fork-Join Jobs in Multiprocessors", to be presented at *Symp. of High Performance Computer Systems*, Dec. 1987.
- [11] P. Brinch Hansen, "The Programming Language Concurrent Pascal", *IEEE Trans. Software Eng.*, SE-1, pp. 199-207, 1975.
- [12] C.A.R. Hoare, "Communicating Sequential Processes", Prentice-Hall International, London, 1978.
- [13] I.C. Pyle, "The Ada Programming Language", Prentice-Hall International, London, 1981.
- [14] D. Stoyan, "Comparison Methods for Queues and Other Stochastic Models", J. Wiley and Sons, NY, 1984.
- [15] F. Baccelli, W.A. Massey, and D. Towsley, "Acyclic Fork-Join Queueing Networks", subm. to *J. of ACM*, 1987.

APPENDIX A

Load Sharing in Soft Real-Time Distributed Computer Systems¹

RENU CHIPALKATTI²

JAMES F. KUROSE

*Department of Computer and Information Science
University of Massachusetts
Amherst, Mass. 01003*

Abstract

In soft real-time distributed computer systems, a job submitted at a node in the network must complete or begin execution within a specified *time constraint*, otherwise it is considered *lost*. When a single node occasionally experiences an overload of jobs, it may still be possible to execute some of the otherwise lost jobs by invoking a *load sharing algorithm* to distribute the local overload to other system nodes. We examine several relatively simple approaches to load sharing and show that these *simple* real-time load sharing algorithms may often perform as well as their more complex counterparts. Approximate analytic performance models are developed and validated through simulation. The performance results suggest that, over a relatively wide range of system parameters, the performance of these simple approaches are substantially better than the case of no load sharing and often close to that of a theoretically optimum algorithm.

¹This work supported in part by the National Science Foundation under contract DMC-8504793 and RADC under contract F302602-81-C-0169

²The author is now with GTE Laboratories, Waltham MA

1 Introduction

A primary motivating factor behind the development of distributed computer systems has been the need to efficiently utilize the resources available within the distributed environment. In this paper, we consider the case of sharing the *computational* resources of the system nodes. This can be done by transferring jobs which are submitted to heavily loaded nodes to more lightly loaded nodes. This process of sharing the workload over the entire system is generally known as *load balancing* or *load sharing* (LS). Although a cost (e.g., a time delay) is typically incurred by transferring a job from one node to another, the performance of a distributed computer system can generally be improved by an effective load sharing policy [20].

In this paper, we study *soft real-time systems*. Real time tasks can essentially be classified into two: (1) tasks which must *begin* execution within a specified amount of time after their initial arrival to the system and (2) tasks which must *complete* execution within a fixed amount of time after their initial arrival to the system. The first set of jobs are characterized by a bounded queueing time whereas the second is characterized by a bounded waiting time. For both types of jobs, those failing to meet their deadline are considered *lost*. *One important purpose of load sharing in a real-time system then is to minimize this percentage of jobs lost.* Examples of systems exhibiting such soft real-time behaviour include applications in distributed systems for industrial process control [23], autonomous manufacturing [3], and air traffic control [9]. In these applications, results of a computation are typically needed in order to perform some control function at a given point in time. Failure of a job to meet its deadline may then require the initiation of a recovery procedure, which can be very costly from a performance standpoint [9].

It has been previously noted that for non-real-time systems, relatively *simple* decentralized policies may often provide effective load sharing in a distributed system [21] [7]. These works, in particular the analytic work in [7], motivate our present work which establishes complementary results for the case of *real-time* systems, systems having performance requirements and evaluation metrics which differ significantly from those of non-real-time systems. We stress that, as in [7], our goal here is not to propose any specific real-time load sharing algorithm nor to necessarily develop performance models for predicting the absolute performance of specific LS approaches, but rather to address the more fundamental question of the level of complexity required to implement effective load sharing, in this case in a distributed *real-time* environment.

In this paper, we adopt an analytic approach towards evaluating various approaches towards real-time load sharing. In section 2, we review previous work in the area of real-time load sharing and then, in section 3, describe the distributed system model used in this paper. Our analysis for tasks with bounded queueing time begins in section 4, and we adopt the general methodology [7] (also [25]) of first developing a model for a single node in isolation and then combining these node-level models

into a single system level-model. In section 4.1, we first develop a model of job loss from a generic system node in isolation. Because we are interested in studying *real-time* performance, our model of a node is necessarily different from that traditionally adopted in load balancing studies for non-real-time systems. In particular, rather than adopting a Markov chain model based on the number of jobs queued for execution at a node, we characterize a node's state by its amount of "unfinished work" and derive a set of integro-differential equations governing the evolution of unfinished work at a node. We must additionally carefully distinguish between locally-arriving jobs and transferred jobs, since the latter arrive with tighter time constraints due to the transfer delay incurred.

In section 4.2, we then compose instances of this generic node model to create a system-level model for the entire distributed system. Central to this composition is the assumption (first introduced in [7], and also used in [25]) of independence among the states of different nodes, an assumption we later validate for our system under study through simulation. We then use this system-level model to quantitatively study the real-time performance of two simple approaches towards real-time load sharing. In both of the approaches studied, a job whose deadline can not be met locally may be transferred to a remote node for possible execution. In the first approach, termed "quasi-dynamic load sharing" (QDLS), a job which can not meet its deadline locally is sent to a probabilistically-chosen remote node. This job will then be either successfully executed or lost at the remote node. We note that the policy of probabilistically selecting a remote node for execution has been extensively studied for the *non-real-time* case [15,19,21,24,7]; the policy of transferring jobs when real-time constraint can not be met locally, however, has not been examined in any previous studies. The second approach studied is the *probing* approach examined in [7] for the case of non-real-time systems. In this approach, a node may probe some limited number of other system nodes and then transfer a job if one of these nodes can execute the job within its deadline. If none of the probed nodes can do so, the job is then lost. Finally, we compare the QDLS and probing policies to the bounding cases of no load sharing and the theoretically optimum LS algorithm. A similar study is performed for jobs with bounded waiting time in section 5. However we shall restrict ourselves only to the simple probing policy as the model becomes sufficiently complex.

We will see that for a relatively wide range of system parameters, the simple approaches studied perform significantly better than the case of no load sharing and often perform remarkably close to that of the theoretically optimum algorithm. Our conclusion thus complements previously-established results for LS in non-real-time systems [7]: very simple approaches, which use only a minimal amount of state information and have an extremely simple decision-making process (and hence are simple to implement) are often sufficient to provide effective load sharing in a distributed real-time computer system.

2 Previous Work on Real-Time Load Sharing

We can classify previous efforts in the area of load sharing in real-time systems into two classes: those that adopt the multiprocessor model and those that adopt the distributed system model. In the multiprocessor model, jobs arrive at an omniscient centralized controller which matches (schedules) the jobs to the processors. Typically, the set of jobs arrival times, timing constraints and execution times are known *a priori* to the centralized scheduler. In the distributed system model (adopted in this paper), jobs may arrive to any node in the system and a node has no *a priori* information about future arrival times of jobs nor about the state of the other nodes in the network.

The work of Muntz and Coffman [18] and Leinbaugh [12] adopts the multiprocessor model. These efforts are directed towards determining a minimum system configuration which can support the specified job load for a given process to processor scheduling policy. Real time multiprocessor scheduling has also been examined in [17], in which a graph model is used to represent timing constraints among a set of periodic tasks. In [2], an approximate algorithm is presented for optimally scheduling n periodic tasks on m processors. The real time scheduling problem for multiprocessors was also considered in [16], although the performance metrics adopted in [16] (essentially, an equal average load at each node) are perhaps more applicable in a non-real-time environment.

There have been relatively few previous efforts adopting the distributed system model of real time load sharing, and it is clear that work in this area has just recently begun. For example, the explicit purpose of [13] is the study of real time scheduling in a *uniprocessor* environment as a precursor to examining similar issues in a distributed environment. In [26] [22], a *specific* load sharing scheme for real time systems is proposed and its performance examined through simulation. The load sharing policy introduced in [26] [22] is based on the use of *focused addressing* and *bidding* and is meant for distributed systems in which real time periodic jobs are given preference over other real time jobs. In this approach, a node which can not guarantee the execution of a job within the specified time constraint permits other nodes to bid for the execution of the job and at the same time may transfer the job to that node (or set of nodes) which are estimated to be most likely to be able to successfully execute the job. Although this sophisticated algorithm was shown to perform quite well, it is closely tied to the notion of periodic tasks. Also, the authors do not consider the performance of the bidding scheme relative to all but the simplest of other possible approaches. In this paper we demonstrate that, in fact, simple approaches may perform as well as the more sophisticated approaches over a wide range of system parameters.

3 The Model of the Distributed System

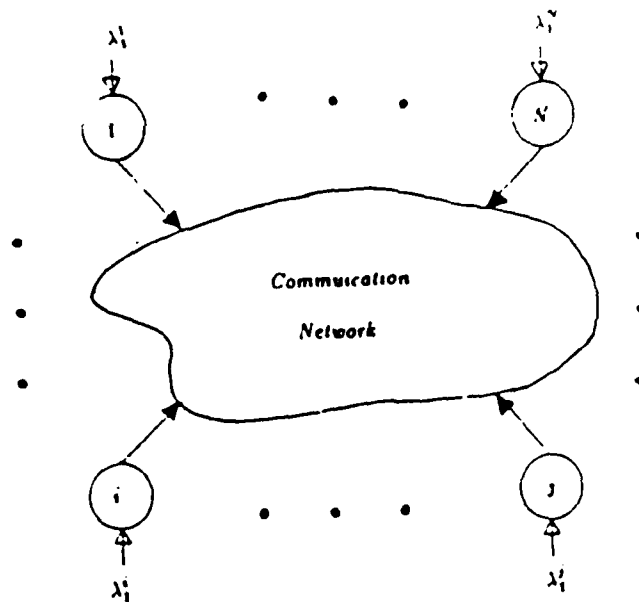


Figure 1: Model of a distributed system

Our model of the distributed system is shown in figure 1. The system consists of N nodes which are interconnected through a communications network; the network is assumed to be logically fully connected in that every node can communicate with every other node. A stream of jobs is submitted locally to node i . Unless stated otherwise, we will assume that the nodes are heterogeneous in the sense that each node may have a different arrival rate of externally submitted jobs, but homogeneous in the sense that a job submitted at any node in the network can be processed at any other node in the network; this latter assumption can be easily relaxed.

We are interested in studying LS policies in a soft real time system, in which a job is lost if it can not complete or begin execution, (as the case may be) within a given time constraint. If the deadline cannot be met locally, a LS algorithm may be invoked to transfer the job to another node which can possibly meet the jobs demands. We will assume that a job cannot be transferred more than once in order to avoid the problem of "trashing" and assume that a constant delay, d , (representing communication and transfer processing delays) is required to transfer a job from one node to another. Thus, if a job first arrives at node i with an initial time constraint of $K1$ and is transferred to another node j for processing, its new time constraint at node j , which we will denote $K2$, will be equal to $K1 - d$.

4 LS for Real Time Tasks with Bounded Queueing Time .

As mentioned earlier, real time task with bounded waiting time tasks are time constrained such that a job must begin execution within $K1$ time units of its initial arrival. For the above mentioned jobs we will examine two simple approaches:

- *quasi-dynamic load sharing (QDLS)* [15,19,21,24,7].
- *probing* [7].

which have been previously studied for non-real-time systems and compare their real-time performance with that of the bounding cases of no load sharing and the theoretically optimum real-time LS algorithm.

As discussed in [7], an LS approach can be characterized by its *transfer* policy, and its *location* policy. The *transfer* policy determines *when* a job should be transferred for remote execution. The *location* policy, determines *where* (i.e., at which remote node) a transferred job will be executed.

Both approaches examined have the same simple transfer policy:

- **Transfer policy (QDLS and probing):** A job is transferred from node i to a remote node if and only if the unfinished workload of the jobs currently at node i exceeds the time constraint for the job. A job will thus either queue for service at the node at which it initially arrives (in which case it will be guaranteed execution) or will be transferred to some remote node. We note that the transfer policy decision is made *dynamically*, based on the current state of the node. We are not aware of any previous analytic studies which have considered this transfer policy in a real-time environment.

The location policies of QDLS and probing are:

- **Location policy (QDLS):** If a job is to be transferred, a remote "target" node (to which the job is sent) is chosen *probabilistically* and *independent* of the current state of the remote nodes. Note that QDLS requires *no* non-local, dynamic state information. Although this location policy has been extensively studied for the non-real-time case [15,19,21,24,7], we are not aware of previous analytic work addressing this problem in a real-time environment.
- **Location policy (probing):** When a job is to be transferred a node *probes* some specified number of other system nodes (chosen at random) to determine if one of them can currently guarantee execution of this job, i.e., has an amount of unfinished work less than the time constraint of the job minus the transfer delay. A node may probe up to some maximum number, L_p , (the probe limit) of other nodes. If none of the probed nodes can execute the job, the job is lost. We note that probing may be considered a simplified form of *bidding* [22]. The probing policy studied here was first analytically examined in [7] (for non-real-time systems) and we follow their methodology when studying the system-level model (but not the node-level model) of probing.

In the analytic performance models developed in the following sections, we will ignore several aspects of LS approaches which, in practice, may influence their performance. Specifically, we will ignore the

processor overhead required to transfer jobs as well as the overhead and time delays required to probe a set of nodes. We will also assume in our *analytic model* of probing (but not in our simulation model used for validation), that a remote node which responds positively to a probing message will always be able to execute the transferred job, even though that node's workload may change between the time it sends a positive response and the time a transferred job arrives. Our reason for ignoring these details is that as in [7], *rather than analyzing the absolute performance of a specific LS algorithm, we are instead interested in analyzing the relative performance of a set of LS approaches as a function of their complexity.* In particular, we are interested in examining possible performance differences between simple probing, a more sophisticated approach towards LS and a theoretically optimum LS algorithm. Ignoring the overhead effects, a more complex approach can *at best* achieve a performance level falling between probing and the theoretical optimum. If this gap is small (as we find is often the case), the performance of probing and any more complicated approach are necessarily close. When overhead is considered, the small performance difference between probing and a more complex approach, which requires additional communication and computational overhead, can only become smaller. Thus, our abstract models do provide the basis for a meaningful comparison of the relative performance of real time load sharing strategies. We also note that when the effects of overhead that we have not modeled are negligible (as our simulation results demonstrate can be the case), our analytic models also provide a means for assessing the absolute real time performance of an LS approach as well.

4.1 Performance Models of the QDLS and Probing LS Algorithms

In this section we develop analytic models in order to quantitatively assess the real time performance of the QDLS and probing LS policies. As a first step, in section 4.1 we develop a performance model for predicting the steady state job loss from the "generic node" shown in figure 2, *without reference to any specific LS policy.* This model is then used in section 4.2.1 to predict the real time performance of a system in which no load sharing (NLS) occurs. Then, adopting the methodology introduced in [7,25] (with modifications to permit us to examine QDLS in a heterogeneous system), the generic model of a node in isolation is then extended in sections 4.2.2 and 4.2.3 to provide a *system-level* model for studying the performance of QDLS and probing.

4.2 Job Loss from a Generic Node

Figure 2 shows our "generic" model of an individual system node. It consists of an upper queue and a lower queue, connected to a single server, representing the computational resource at a node. A job arriving to the lower queue with an execution time of x , must complete execution within $K1 + x$ time units; a job entering the upper queue must complete within $K2 + x$. Equivalently, a job arriving to the lower (upper) queue must *begin* service within an amount of time, $K1$ ($K2$) after its arrival;

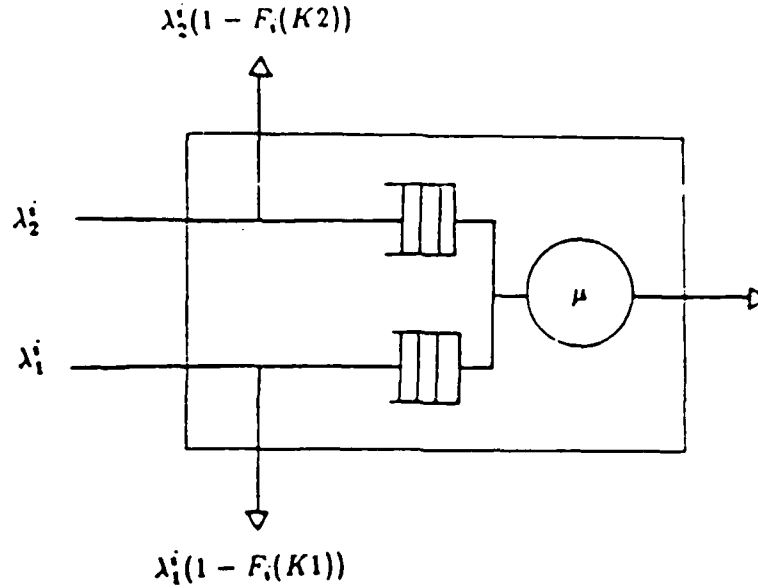


Figure 2: A generic node in isolation

otherwise it will be lost.

The server has a mean service rate of μ jobs/sec and the service policy is FCFS across all the jobs belonging to both the queues; we note that in the case that the difference between K_1 and K_2 is small, (as will often be the case when we use this model in a LS context), FCFS closely approximates a "shortest deadline first" scheduling policy. We assume that the arrival of jobs at the lower and upper queues can be modeled by Poisson processes with mean rates λ_1 and λ_2 , respectively.

The problem of queues with impatient customers has been well-studied in the field of operations research. Gavish *et al.* [8], study an FCFS M/M/1 system where arriving customers are admitted only if their waiting times plus service times do not exceed some fixed amount. Baccelli *et al.* [1], study a single-server system in which a customer is lost when its waiting time exceeds a random threshold. They derive equations for several configurations of arrival rates, service time distributions and patience thresholds (time constraints). As we will be only interested in determining the probability of customer loss, we may adopt a simpler approach than these efforts.

Thus, let $F(w, t)$ denote the probability that at time t , the unfinished work in both queues is less than or equal to w . Without loss of generality we also assume that $K_1 > K_2$. If $B(x)$ denotes the PDF of the service time distribution, then following the approaches in [8] [10], we can derive the following time-evolution equations for $F(w, t + \Delta t)$. We consider three different regions.

In the region $0 < w \leq K_2$ we have,

$$F(w, t + \Delta t) = (1 - \lambda_1 \Delta t)(1 - \lambda_2 \Delta t)F(w + \Delta t, t) \\ + \lambda_1 \Delta t(1 - \lambda_2 \Delta t) \int_0^w B(w - u) d_u F(u, t)$$

$$+ \lambda_2 \Delta t (1 - \lambda_1 \Delta t) \int_0^w B(w - u) d_u F(u, t)$$

The expression on the left hand side gives the probability that the unfinished work in the queue at time $t + \Delta t$ is less than w . This condition can be realized in several ways. First, no jobs may have arrived in the interval $[t, t + \Delta t]$. In this case, there must have been an amount of work less than $w + \Delta t$ at time t . The second term on the right hand side of the above equation denotes the probability that exactly one arrival (in time Δt) at the lower queue brings new unfinished work to the queue such that the unfinished work at $t + \Delta t$ is less than w . Similarly, the third term denotes the probability of exactly one arrival at the upper queue such that the new unfinished work at $t + \Delta t$ is less than w .

In the region $K2 < w \leq K1$ we have:

$$\begin{aligned} F(w, t + \Delta t) = & (1 - \lambda_1 \Delta t)(1 - \lambda_2 \Delta t) F(w + \Delta t, t) \\ & + \lambda_1 \Delta t (1 - \lambda_2 \Delta t) \int_0^w B(w - u) d_u F(u, t) \\ & + \lambda_2 \Delta t (1 - \lambda_1 \Delta t) \left(\begin{aligned} & F(K2, t) \int_0^{K2} B(w - u) \frac{d_u F(u, t)}{F(K2, t)} \\ & + (1 - F(K2, t)) \int_{K2}^{w + \Delta t} \frac{d_u F(u, t)}{1 - F(K2, t)} \end{aligned} \right) \end{aligned}$$

The first two terms of the above equation are similar to those described above. The third term is for the case of an arrival at the upper queue in the interval $[t, t + \Delta t]$. With probability $F(K2, t)$, the job joins the queue. In this case, the new work brought in by the job plus the unfinished work at time t must be less than w . Note that the probability of this latter event must be conditioned on the event of the unfinished work at time t being less than $K2$. Similarly, with probability $1 - F(K2, t)$ the arriving job finds an amount of work greater than $K2$, and hence does not join the queue. In this case the unfinished work at time t must have been less than $w + \Delta t$. This probability must also be conditioned, this time on the fact that an arriving job did not join the queue.

Rearranging the terms in the above two equations and taking limits as $\Delta t \rightarrow 0$, we get:

$$\begin{aligned} \frac{\partial F(w, t)}{\partial t} - \frac{\partial F(w, t)}{\partial w} &= -(\lambda_1 + \lambda_2) F(w, t) + (\lambda_1 + \lambda_2) \left\{ \int_0^w B(w - u) d_u F(u, t) \right\} \quad 0 < w \leq K2 \\ \frac{\partial F(w, t)}{\partial t} - \frac{\partial F(w, t)}{\partial w} &= -\lambda_1 \left\{ F(w, t) - \int_0^w B(w - u) d_u F(u, t) \right\} \\ &\quad - \lambda_2 \left\{ F(K2, t) - \int_0^{K2} B(w - u) d_u F(u, t) \right\} \quad K2 < w \leq K1 \end{aligned}$$

and taking limits as $t \rightarrow \infty$, we obtain the following steady state equations:

$$\frac{dF(w)}{dw} = (\lambda_1 + \lambda_2)F(w) - (\lambda_1 + \lambda_2) \left\{ \int_0^w B(w-u) d_u F(u) \right\} \quad 0 < w \leq K2 \quad (1)$$

$$\begin{aligned} \frac{dF(w)}{dw} = & \lambda_1 \left\{ F(w) - \int_0^w B(w-u) d_u F(u) \right\} + \\ & \lambda_2 \left\{ F(K2) - \int_0^{K2} B(w-u) d_u F(u) \right\} \quad K2 < w \leq K1 \end{aligned} \quad (2)$$

In order to provide a check of the above equations, we show in Appendix A that equation 2 can be independently derived in a different manner using level crossing arguments [4]. In the case that job execution times are exponentially distributed with a mean of $\frac{1}{\mu}$, the solution to equation 1 is given by:

$$\begin{aligned} F(w) &= F(0^+) \left\{ \frac{\mu}{\mu - \lambda_1 - \lambda_2} - \frac{\lambda_1 + \lambda_2}{\mu - \lambda_1 - \lambda_2} e^{-w(\mu - \lambda_1 - \lambda_2)} \right\} \quad 0 < w \leq K2 \\ F(w) &= F(K2) + \frac{\lambda_1 + \lambda_2}{\mu - \lambda_1} F(0^+) e^{\lambda_2 K2} \{ e^{-(\mu - \lambda_1)K2} - e^{-(\mu - \lambda_1)w} \} \quad K2 < w \leq K1 \end{aligned} \quad (3)$$

At this point, we could now proceed in a similar fashion to derive an expression for $F(w)$ in the region $K1 < w$. However, if we are only interested in computing the fraction of jobs lost, we can derive a simpler third equation by considering *flow conservation* across the boundaries shown in figure 2. The total flow into the node consists of the sum of λ_1 and λ_2 , while the total flow out of the node consists of a departure stream from the server and the two loss streams, one from each of the queues. Hence,

$$\lambda_1 + \lambda_2 = \{1 - F(0^+)\}\mu + \{1 - F(K2)\}\lambda_2 + \{1 - F(K1)\}\lambda_1 \quad (4)$$

We can now solve the set of simultaneous equations 3 through 4, to numerically obtain $F(K1)$, $F(K2)$ and $F(0^+)$.

4.3 Incorporating a Generic Node Model into a Distributed System Model

We now incorporate our model of a generic node in isolation into a system-level model in order to study the performance of no load sharing, QDLS, and probing. In each of our models, each system node will be represented by the generic node model of figure 2. The arrival rates to the lower and upper queues at node i will be denoted λ_1^i and λ_2^i , respectively, and $F_i(w)$ will denote the PDF of the unfinished work at node i ; note that although $F_i(w)$ is a function of λ_1^i , λ_2^i , $K1$, and $K2$, we have not indicated this functional dependence directly.

When incorporating our model of a generic node into a system-level model, we let the arrivals to the lower queue at node i , λ_1^i , represent the "external" arrivals of jobs to node i ; these "external"

arrivals, with an initial time constraint (until execution begins) of $K1$, represent jobs which are first submitted to the system at node i and are inputs to our model, specifying the load on the distributed system. With probability $1 - F_i(K1)$, an externally arriving job at node i will not be able to meet its time constraint locally (i.e., at node i). In this case, the job will either be lost or will be sent to another node for possible execution, depending on the load sharing policy employed.

The arrivals to the upper queue at node i , λ_2^i , represent the arrival of "internally transmitted" jobs to node i , i.e., the arrival of jobs which have been transferred to node i from other nodes in the system, and thus depend on the LS scheme used. The time constraint (until execution begins) for these internally transmitted jobs is $K2$. Recall that d is the fixed delay associated with a job transfer and thus $K2 = K1 - d$. Since a job can be transferred at most once, a job which arrives to the upper queue and cannot meet its deadline is unconditionally lost.

4.3.1 Job Loss with No Load Sharing (NLS)

With no load sharing, no jobs are transferred between nodes and hence $\lambda_2^i = 0$ for all nodes i . In this case, we can solve equations 3 and 4 for $F_i(0^+)$, $F_i(K1)$ and $F_i(K2)$ for a given λ_1^i and compute the system-wide loss by:

$$\text{loss rate}^{NLS} = \sum_{i=1}^N \lambda_1^i (1 - F_i(K1)) \quad (5)$$

4.3.2 Job Loss with Quasidynamic Load Sharing (QDLS)

Recall that in our QDLS approach, when an externally arriving job arrives at a node and can not finish execution within the time constraint, $K1 + x$, it is transferred to a probabilistically-chosen remote node for possible execution. In our system-level model of QDLS then, all jobs exiting before joining the lower queue in figure 2 are transferred to another node for possible remote execution. Let λ_1^{ij} denote the job transfer rate from node i to node j and let λ_1^{ii} represent the externally arriving jobs that are executed locally. Given the QDLS scheme and given that an externally arriving job is transferred if and only if it can not be executed locally, we have the following flow constraints:

- $\lambda_1^i (1 - F_i(K1)) = \sum_{j=1, j \neq i}^N \lambda_1^{ij}$
- $\lambda_1^i F_i(K1) = \lambda_1^{ii}$
- $\lambda_2^j = \sum_{i=1, i \neq j}^N \lambda_1^{ji}$, for all nodes j .

Given a set of flows satisfying the above constraints, the system-wide job loss under QDLS can be easily computed. Since all job loss at node i can only occur in the upper queue, we can first solve equations 3 and 4 for $F_i(0^+)$, $F_i(K1)$ and $F_i(K2)$ and then compute the system-wide loss under QDLS by:

$$\begin{aligned} \text{loss rate }^{QDLS} &= \sum_{i=1}^N \lambda_i^1 (1.0 - F_i(K2)) \\ &= \sum_{i=1}^N (\sum_{k=1 \neq i}^N \lambda_k^1 (1.0 - F_i(K2))) \end{aligned} \quad (6)$$

Clearly then, the system-wide rate at which jobs are lost depends on the values of $\{\lambda_i^1\}$ (both directly as shown above in equation 6 and indirectly through the dependence of $F_i(K2)$ at node i on $\{\lambda_i^1\}$ and $\{\lambda_i^2\}$.) Thus, we are interested in determining the values of $\{\lambda_i^1\}$ which minimize equation 6 subject to the flow constraints; this can be accomplished using any constrained optimization procedure, including the procedure described in [11].

Finally, we note that unlike [7], we have not assumed a system of homogeneous nodes; this necessitated the use of an optimization procedure. We have, however, adopted two assumptions introduced in [7] in deriving equation 6. First, we have assumed that the individual $F_i()$'s are independent; that is, a job's probability of being executed within its deadline at one node is independent of the state of the other system nodes. A second assumption is that the arrival process at each of the upper queues, which is formed by the superposition of the overflow processes of the other system nodes, is Poisson. We note that these assumptions become asymptotically correct as the number of system nodes gets very large [7,25] or as the ratio, λ_i^2/λ_i^1 becomes very small. We also note that as shown in section 5, for N equal to 20 nodes, our simulation studies yield performance results which are extremely close to those predicted by the analysis, thus corroborating the appropriateness of our analytic approximations.

4.3.3 Job Loss with Probing

For the case of probing, we follow [7] directly and obtain analytic results for the homogeneous case in which λ_i^1 is identical for all nodes, i . As a consequence the steady state probabilities $F_i(K2)$, $F_i(K1)$ etc. will also be identical for all nodes. Since a job is lost only if it can not be executed locally within $K1$ time units and some L_p (the probe limit) other nodes are probed at random, each of which is then found to have a current load of unfinished work greater than $K2$, we have:

$$\text{loss rate }^{probing}_i = \lambda_i^1 (1 - F(K1))(1 - F(K2))^{L_p} \quad (7)$$

Note that we cannot yet solve equations 3 and 4 for $F(K2)$, and thereby compute the loss using 7, for the λ_i^2 which result from the probing policy are still unknown. Again considering the homogeneity of the system, we note that the steady state transfer rates out of all nodes must be identical; similarly, the

rate at which jobs are transferred into the nodes must also be equal. This then implies that the steady state flow of jobs out of any given node must equal the steady state flow of internally transmitted jobs that are accepted and successfully executed at this node. We thus have the following flow constraint for all nodes:

$$\lambda_1(1 - F(K1)) - \lambda_1(1 - F(K1))(1 - F(K2))^{L_1} = \lambda_2 F(K2) \quad (8)$$

where we have dropped the i superscripts and subscripts since the system is homogeneous. Equations 3, 4, and 8 provide four equations in four unknowns ($F(K1)$, $F(K2)$, $F(0^+)$ and λ_2). We can now solve this set of simultaneous equations for $F(K1)$ and $F(K2)$ and directly compute the loss using equation 7.

4.4 Numerical Results

In this section we present representative performance results for the QDLS and probing real time load sharing schemes and compare their performance with that of the ideal case of perfect-information load sharing and the case of no load sharing (NLS). We consider a 20 node system (the same size considered in [7]), in which $\mu = 1$ job/second and a delay of $d = 0.2$, i.e., the transfer delay is 20% of the job execution time.

We model the "ideal" case as an $M/M/20$ queueing system with a time constraint of $K1$ and have obtained the $M/M/20$ performance results through simulation. We note that in the $M/M/20$ system, jobs are scheduled to available processors using complete information about the system state and incur no transfer delay. Thus, the "ideal" performance bounds shown in the subsequent results are, in reality, unattainable. This will be evident in our performance results, where for large values of $K1$ and heavy external arrival rates, the QDLS and the probing curves approach limiting values which are slightly above and to the left (i.e., poorer performance) than the upper bound predicted by our "ideal" case of the $M/M/20$ queue.

Figure 3 shows the fractions of jobs lost as a function of the average system arrival rate, $\sum_{i=1}^N \lambda_i^i / N$ under the QDLS policy. Performance results are presented for different values of the initial time constraint, $K1$ (0.5 sec. and 5.0 sec.) and transfer delays of 0.1 and 0.2 time units (10% and 20% of the average job execution time). The system load was asymmetric, with half the nodes having an average arrival rate of $\frac{\sum_{i=1}^N \lambda_i^i}{2N}$ while the remaining half had an average arrival rate of $\frac{3 \sum_{i=1}^N \lambda_i^i}{2N}$. We should also note that in order to test our numerical, optimization, and simulation procedures, we first studied QDLS in a completely symmetric system (not shown). As expected, the optimum $\{\lambda_i^{ij} | i \neq j\}$ were found to be equal, as were the optimum $\{\lambda_i^{ii}\}$.

Several properties of the QDLS policy are evident from Figure 3. First, note that even for the most stringent timing conditions ($K = 0.5$ and $d = 0.2$) QDLS performs significantly better than no load

sharing. We additionally note that as the asymmetry in the arrival rates increases, QDLS performs increasingly better than no load sharing. While these results might not seem surprising at first, we note that QDLS is perhaps the simplest of all possible real time LS approaches and makes use of no non-local dynamic state information. We also note that for a time constraint of 5.0 (i.e., where all jobs must begin execution within 5 times their average execution time), the performance of this simplest of all LS policies approaches that of the "ideal" case. Moreover, this performance difference is particularly small in the system load regions of practical interest, in which the arrival rate of jobs to the system is less than 70% of the physical capacity of the system. Figure 3 also indicates that the ideal curves show a knee at an average load of 1.0; above this point the job arrival rate exceeds the system's capacity and thus some jobs will necessarily be lost. We also note that lost jobs are not executed; if these jobs were to be executed, higher losses would result since these lost jobs would place additional demands on the service capacity of the nodes.

Finally, note that we have also plotted simulation results (point values shown as filled squares) in Figure 3 for $K1 = 0.5$. These simulation results were obtained *without* making the independence assumptions and the Poisson approximation for λ_2^i , required by the analysis. In the simulation, the optimized λ_1^{ij} 's from the analytic model were used to determine the probabilities with which a transferred job from node i was sent to some remote node, j . A transferred job arrived at its destination d time units later with a new time constraint of $(K1 - d)$. Note that the close correspondence between the simulation and analytic results corroborates our earlier belief that the approximations introduced were indeed justifiable. In the case that d was chosen to be a smaller value (e.g., $d = 0.1$, not shown here), the simulation and analytic results were found to match even more closely.

The real time performance of the probing approach is demonstrated in figure 4 for probe limits, $L_p = 1, 3, 5$. As expected the performance of probing approaches the ideal limit as L_p increases. Note, however, that a relatively small probing limit ($L_p = 5$ when $K1 = 0.5$ (an *extremely tight* time constraint) and $L_p = 3$ when $K1 = 5.0$), results in a real time performance extremely close to the unachievable upper bound. Also note that increasing the probing limit beyond a relatively small number can result at best in only a marginal performance improvement; this results from the fact that the probability that a job (which is to be transferred out) is accepted by the m^{th} probed node, is given by $F(K2)[1 - F(K2)]^{m-1}$, and this probability decreases rapidly with increasing m . We may conclude then that since additional probing beyond some small probe limit incurs additional overhead, a relatively small probe limit would be sufficient in practice to implement effective real time load sharing.

Once again, simulations were performed to validate our analysis. The results (plotted as filled squares) in Figure 4 are shown for $K1 = 0.5$ and $L_p = 3$. As with the QDLS simulations, the simulations were performed without assuming independence among the states of the system nodes or a Poisson

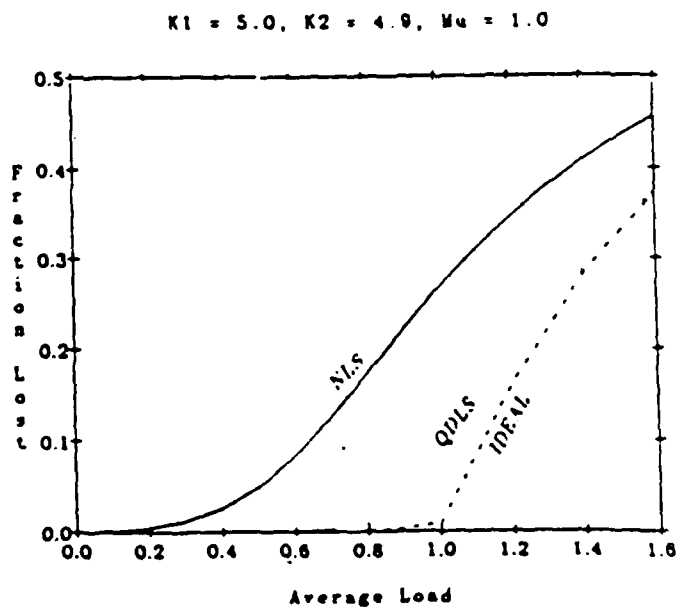
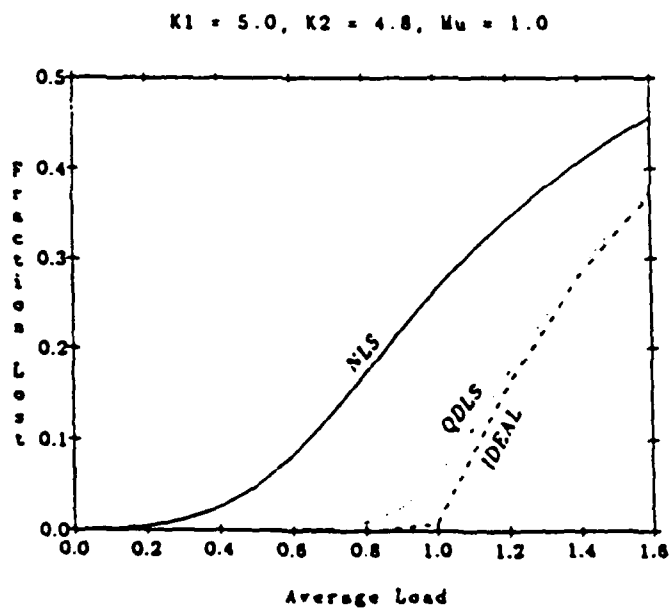
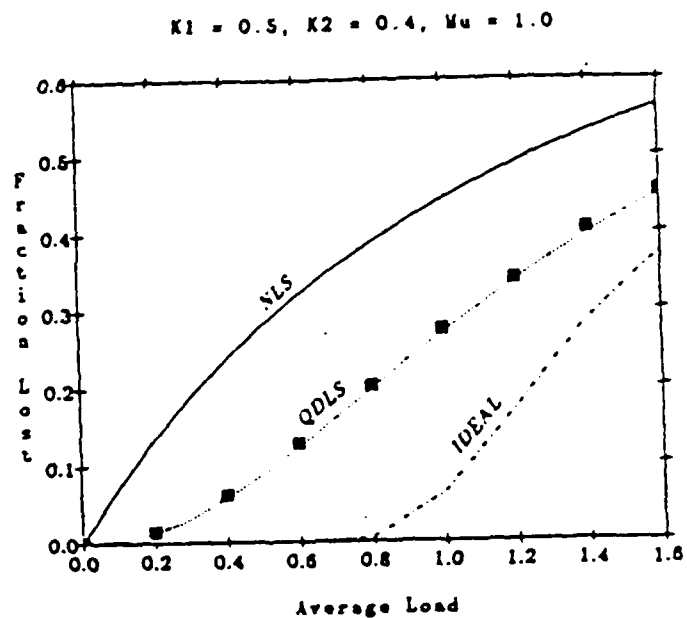
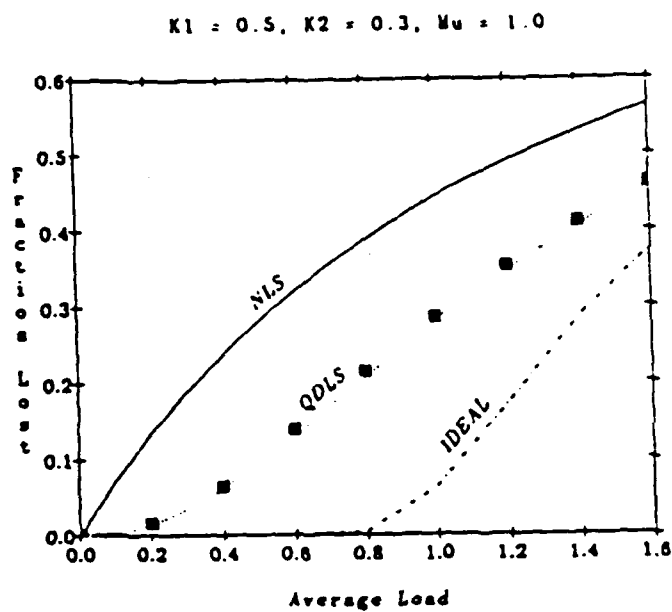


Figure 3: Performance of QDLS policy for a 20 node system with asymmetric loads

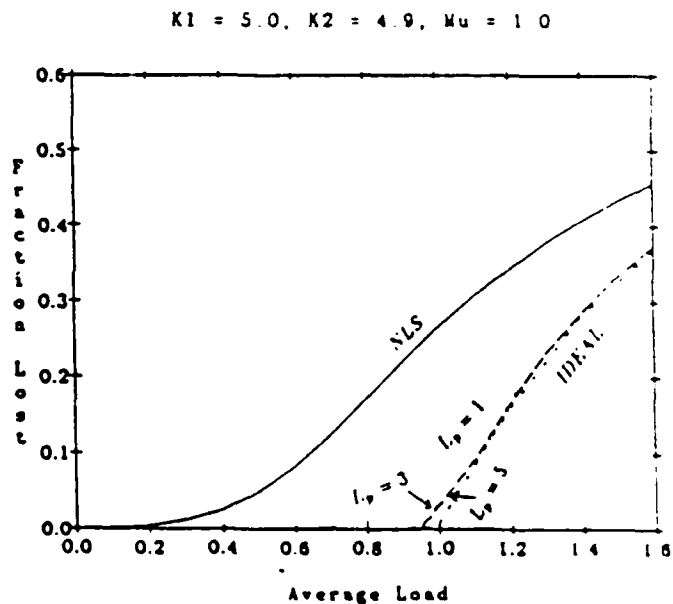
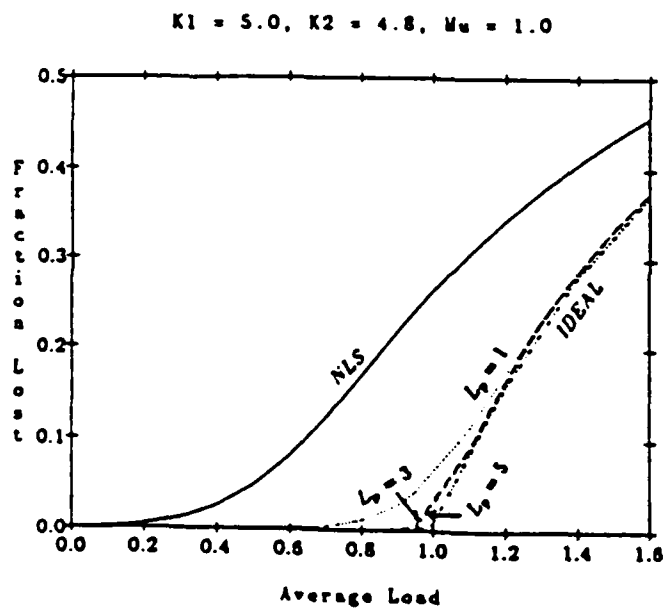
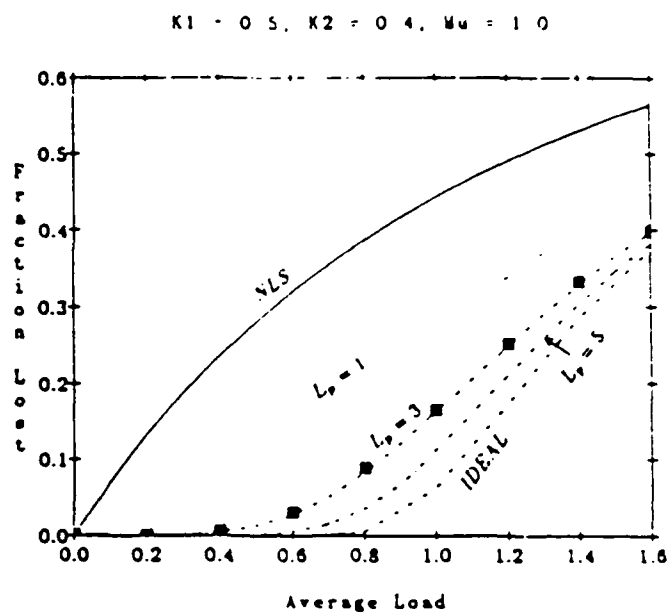
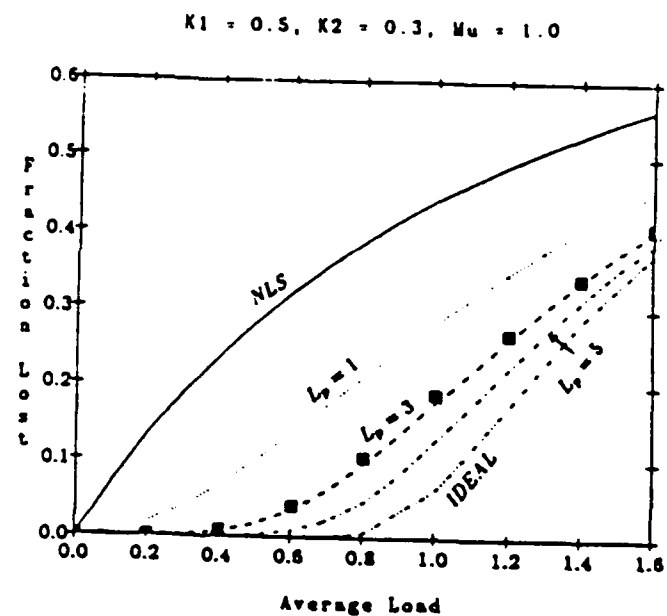


Figure 4: Performance of the Probing Policy for $L_p = 1, 3, 5$ under a symmetric load.

arrival rate at the upper queue. We also note that in our simulations, if a node responded positively to a probe and its workload had increased beyond $K2$ within the d time units required to transfer a job, the transferred job was simply lost, as would be the case in a real system. Again, we note that the close correspondence between our simulation and analytic results indicate that reasonable modeling assumptions and approximations were made in the development of our analytic model of probing.

Figure 5 indicates the dependence of the fraction of jobs lost on the time constraint, $K1$. Performance results are shown for two different values of a symmetric load (an average of 0.4 and 1.2 external arrivals/time unit per station), and a fixed transfer delay of 0.1. For $\lambda = 0.4$ the curve for the Ideal case lies along the x- axis. As expected, as $K1$ increases, the performance of QDLS and probing with $L_p = 3$ approach the upper performance bound. More importantly, note that the performance with $L_p = 3$ is close to the ideal limiting performance values for even very stringent time constraints.

In summary, Figures 3 through 5 provide a quantitative basis for addressing the question of determining the appropriate level of complexity for LS algorithms. We note that a more complex approach can *at best* achieve a performance level falling in the gap between our probing results and the theoretical optimum. For system parameters of practical interest (i.e., a system loading less than the physical capacity of the system and time constraints on the order of the service time of a job), this gap has been shown to be quite small. If the overhead we have not modeled is to be considered, the small performance difference between probing and a more complex approach, which requires additional communication and computational overhead, can only become smaller.

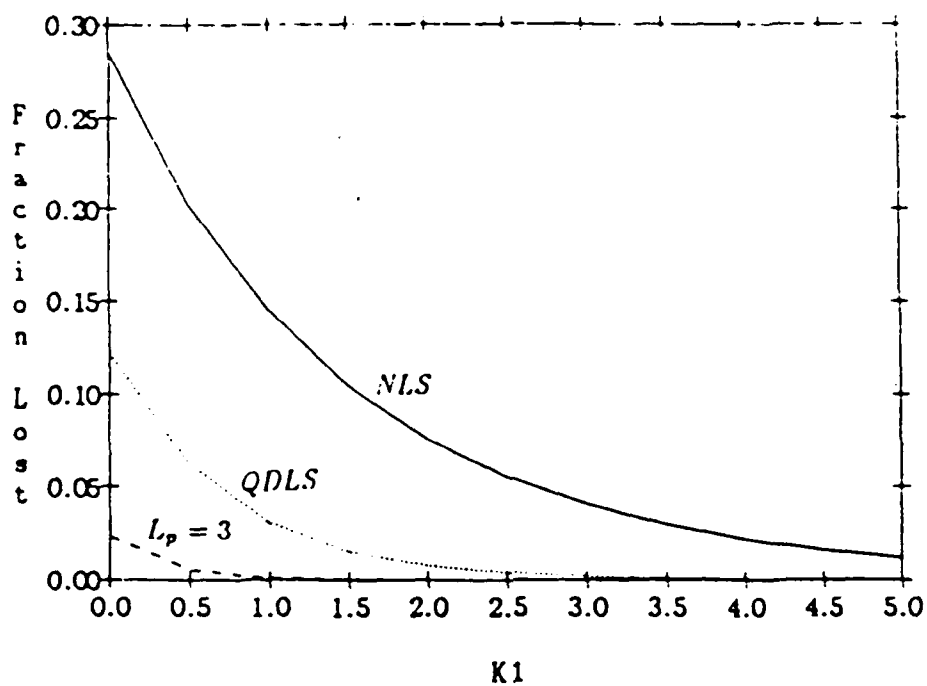
5 Real-Time Tasks With Bounded Waiting Time

In this section we focus on a second model for real-time jobs, one in which a job must *complete* execution within a fixed time after its initial arrival into the system. A job arriving at a node can only be serviced locally if the time it could spend in the queue plus its service requirement is less than the deadline, otherwise it is considered lost. As stated earlier, we assume that a job initially arrives at a node with a fixed time constraint $K1$, a FCFS scheduling policy at each node and a constant network delay d associated with the job transferring procedure. Hence transferred jobs must complete execution at the destination node in $K1 - d = K2$ time units.

For this system, the transfer and location policies are modified to:

- **Transfer Policy :** A job is to be transferred from a node i if and only if the unfinished work at node i plus the service requirement of the job exceeds its time constraint $K1$ and its service time is less than $K2$.
- **Location Policy :** The location policy is the same as in the previous section. For QDLS, a node probabilistically selects a "target node" and the job is transferred to this node. For probing,

$\Lambda = 0.4, K_2 = K_1 - 0.1, \mu = 1.0$



$\Lambda = 1.2, K_2 = K_1 - 0.1, \mu = 1.0$

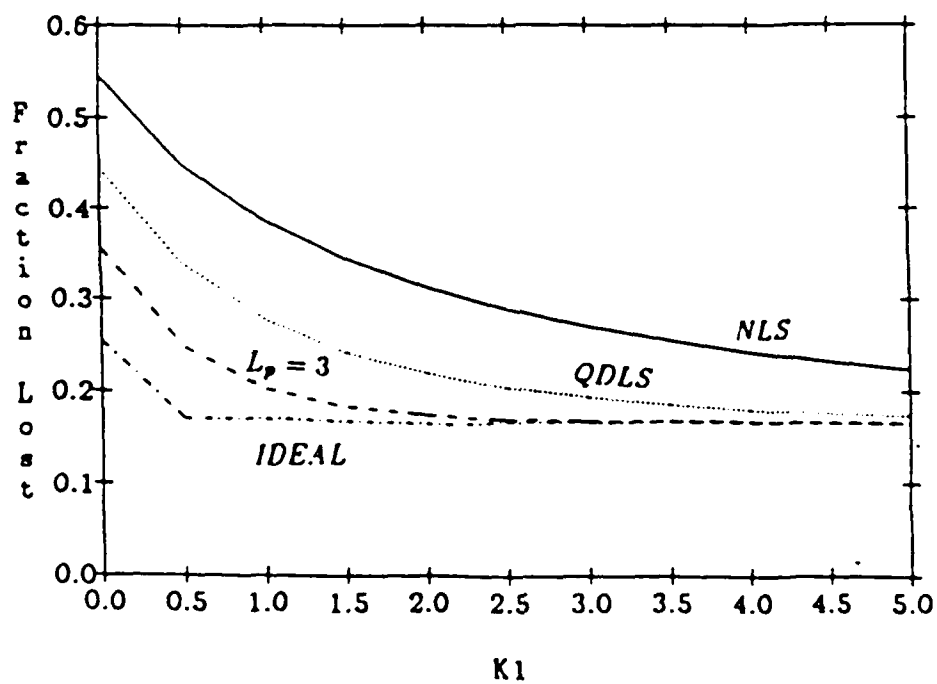


Figure 5: Convergence of various LS approaches to Ideal for large K_1 .

a node probes a fixed number of nodes in the system to determine whether any of them can guarantee the execution of the job, i.e if the unfinished work at the destination node plus the service requirement of the job is less than $K2$. The job is transferred to the first node which responds positively.

Note that for this system, the basis on which a job is denied service depends not only on the current state of the node but also on the service time requirements of the job. Clearly, tasks requiring a service time greater than $K1$ cannot be executed locally. Furthermore, of the tasks which cannot be processed locally, those requiring a service time greater than $K2$ will not be accepted by any node. Therefore, transferring these tasks is futile. This distinguishes the transfer policy used in this system from the system we had considered in the previous section, in which an attempt was made to transfer each job, which could not be executed locally, regardless of its service time. As we will see, this sufficiently complicates the model so that obtaining a closed form expression for the unfinished work becomes a difficult task.

In the model developed below, we again ignore the processor overhead required transfer the job and the time delay involved in probing. Hence the actual performance realized would in fact be higher than the computed values. The reason for this omission is that we are again interested only in the relative performance of the system rather than its absolute performance. Also, the model does not account for the fact that the state of the node, to which a job is transferred, may undergo a change during the time required for the transfer. However, our simulations do account for any changes which may occur in the system during the time d .

5.1 Performance Models for an Individual Node in Isolation

Figure 6 shows the model of an individual node in the system. Externally arriving jobs (with rate λ_1) comprise the lower job arrival stream. Of these jobs, only those whose deadline can be met by the node are allowed to join the lower queue (hence all jobs in the lower queue have a time constraint equal to $K1$). The externally arriving jobs which are not accepted into the lower queue are either lost or transferred out to other nodes. In the QDLS scheme the upper job arrival stream (with arrival rate λ_2) represents the transferred jobs, whereas in the Probing Policy the upper stream represents probe arrivals. A probe which is accepted then, translates into an arrival to the upper queue. Hence all jobs in the upper queue have a time constraint of $K2$. We will assume that arrivals to the lower and upper queue follow a Poisson distribution with rate λ_1 and λ_2 respectively.

The bounded waiting for jobs having a single fixed deadline has been extensively examined. For $GI/G/1$ systems, Loris-Tegheim [14] obtained the generating functions of the Laplace-Steiltjes transforms of the distribution of the waiting time, for cases where the random variables give rise to a rational transform function. Cohen [5] (Model II) also used transform techniques to study the equivalent prob-

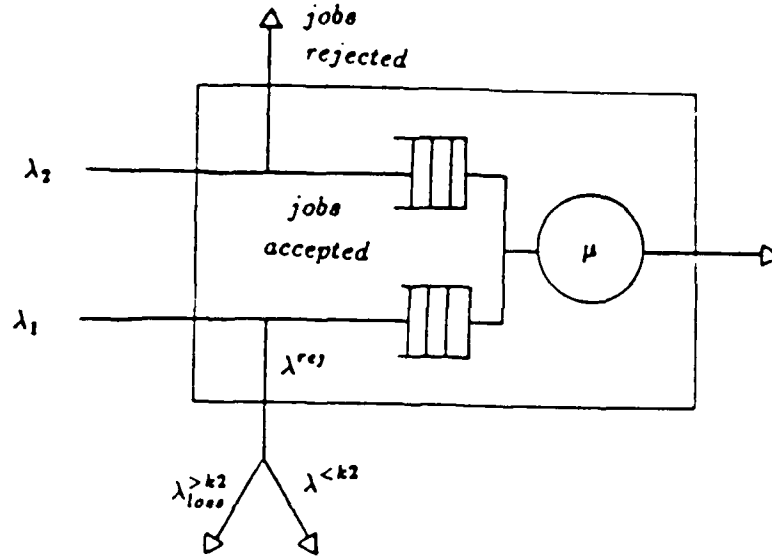


Figure 6: A generic node in isolation

lem for $M/M/1$ and $M/D/1$ queueing systems. Gavish et. al. [8] derived analytical expressions for the virtual waiting time distribution and the loss incurred by the system, for an $M/M/1$ system with an FCFS service discipline. Their method is simpler than the techniques used by Loris-Tegheim [14] and Cohen [5]. We thus choose to extend the results in [8] to incorporate the second time constraint needed in our model.

As in the previous section, following the approach of [8] [10], we can derive the time evolution expression for the distribution of the unfinished work, $F(w, t + \Delta t)$ in two different regions. Let $B(x)$ denote the PDF of the service time distribution of the externally arriving jobs. Note that the decision of whether or not a job can be locally processed depends not only the current unfinished work at the node but also on its service time requirements. Furthermore, of the jobs which cannot receive service locally, only those with service time less than K_2 may possibly be successfully executed as a result of the LS policies. Thus the service time distribution of *transferred jobs* will no longer be $B(x)$. Let $G(x)$ denote the PDF of the service time distribution of these transferred jobs.

In the region $0 < w \leq K_2$ we have,

$$\begin{aligned}
 F(w, t + \Delta t) = & (1 - \lambda_1 \Delta t)(1 - \lambda_2 \Delta t) F(w + \Delta t, t) \\
 & + \lambda_1 \Delta t (1 - \lambda_2 \Delta t) \left\{ \int_0^{w+\Delta t} (1 - B(K_1 - u)) d_u F(u, t) F(w + \Delta t, t) \right. \\
 & \quad \left. + \int_0^w B(w - u) d_u F(u, t) \right\} \\
 & + \lambda_2 \Delta t (1 - \lambda_1 \Delta t) \left\{ \int_0^{w+\Delta t} (1 - G(K_2 - u)) d_u F(u, t) F(w + \Delta t, t) \right. \\
 & \quad \left. + \int_0^w G(w - u) d_u F(u, t) \right\} \quad (9)
 \end{aligned}$$

The probability that the unfinished work in the queue at time $t + \Delta t$ is less than w can evolve from the state of the queue at time t in the following ways. The first term on the right hand denotes the probability of the event that no jobs arrive in the interval $[t, t + \Delta t]$, in which case, there must have been an amount of work less than $w + \Delta t$ at time t . The second term is a composite of two terms and arises in the event that exactly one arrival (in time Δt) occurs at the lower queue. In this case: (a) if the unfinished work at time t plus the work brought in exceed $K1$, the job is rejected and no additional work is added to the queue; (b) if the unfinished work at time t plus the work brought in is less than $K1$, the job joins the queue and brings additional unfinished work to the queue such that the unfinished work at $t + \Delta t$ is less than w . The third term is similar to the second one, except that it holds in the event that an arrival occurs at the upper queue and no arrival occurs at the lower queue in the interval $t + \Delta t$.

In the region $K2 < w < K1$ we have,

$$\begin{aligned}
 F(w, t + \Delta t) = & (1 - \lambda_1 \Delta t)(1 - \lambda_2 \Delta t)F(w + \Delta t, t) \\
 & + \lambda_1 \Delta t(1 - \lambda_2 \Delta t) \left\{ \int_0^{w+\Delta t} (1 - B(K1 - u)) d_u F(u, t) F(w + \Delta t, t) \right. \\
 & \quad \left. + \int_0^w B(w - u) d_u F(u, t) \right\} \\
 & + \lambda_2 \Delta t(1 - \lambda_1 \Delta t)F(w + \Delta t, t)
 \end{aligned} \tag{10}$$

Equation 10 is similar to Equation 9. Note that since the unfinished work is $w > K2$ at time $t + \Delta t$, a probe/job arriving at the node in the interval Δt , will be declined. Hence, an arrival at upper queue adds no work to the node

Expanding $F(w + \Delta t, t)$ with respect to the first variable and taking the limits as $\Delta t \rightarrow 0$ we get

$$\begin{aligned}
 \frac{\partial F(w, t)}{\partial t} - \frac{\partial F(w, t)}{\partial w} = & -\lambda_1 \int_0^w \{B(K1 - u) - B(w - u)\} d_u F(u, t) \\
 & - \lambda_2 \int_0^w \{G(K2 - u) - G(w - u)\} d_u F(u, t) \quad 0 < w \leq K2 \\
 \frac{\partial F(w, t)}{\partial t} - \frac{\partial F(w, t)}{\partial w} = & -\lambda_1 \int_0^w \{B(K1 - u) - B(w - u)\} d_u F(u, t) \quad K2 < w \leq K1
 \end{aligned}$$

and taking limits as $t \rightarrow \infty$, we obtain the following steady state equations:

$$\begin{aligned}
 \frac{dF(w)}{dw} = & \lambda_1 \int_0^w \{B(K1 - u) - B(w - u)\} d_u F(u) \\
 & + \lambda_2 \int_0^w \{G(K2 - u) - G(w - u)\} d_u F(u) \quad 0 < w \leq K2
 \end{aligned} \tag{11}$$

$$\frac{dF(w)}{dw} = \lambda_1 \int_0^w \{B(K1 - u) - B(w - u)\} d_u F(u) \quad K2 < w \leq K1 \tag{12}$$

Equations 11 and 12 can also be directly obtained using level crossing arguments [4] (see Appendix A).

By definition the maximum unfinished work at a node must be less than $K1$. Thus the normalization condition becomes,

$$\int_0^{K1} f(u) du = 1 \quad (13)$$

So far we have not obtained an explicit expression for the PDF, $G(x)$. In order to do so, we consider the various job streams shown in Figure 6. A job is denied service locally if the unfinished work in the queue plus the service time of the job exceeds $K1$. Hence the probability that a job cannot be processed locally is given by,

$$P^{rej} = \int_0^{K1} [1 - B(K1 - u)] d_u(F(u),$$

and the rate at which jobs are rejected from the lower stream of jobs is thus,

$$\lambda^{rej} = \lambda_1 P^{rej}.$$

Let $J(x)$ be the service time distribution of the jobs in the λ^{rej} stream. Then,

$$\begin{aligned} J(w) &= \frac{1}{P^{rej}} \int_0^w b(u) \int_{K1-u}^{K1} d_x F(x) du \\ &= \frac{1}{P^{rej}} \int_0^w b(u) [1 - F(K1 - u)] d_u(F(u)) \end{aligned}$$

Of these jobs, only those with service time less than $K2$ may be transferred out (represented by the stream $\lambda^{<K2}$) in Figure 6). Thus,

$$\begin{aligned} G(w) &= \frac{\int_0^w b(u) [1 - F(K1 - u)] du}{\int_0^{K2} b(u) [1 - F(K1 - u)] du} \\ &= \frac{J(w)}{J(K2)} \end{aligned} \quad (14)$$

Using the expression for $G(w)$ given in 14, in Equations 11 and 12, we observe that the pdf, $f(w)$ is a function of an integral containing $F(x)$ itself. This adds sufficient complexity to the equations 11 and 12 that we resort to numerical techniques to solve for $F(w)$.

Before proceeding to compute the losses incurred in the system, we note that since we are considering a homogenous system (in which each node experiences identical external job arrival rates), the long term time averages of the unfinished work at each node will be the same. We therefore do not attach a superscript i with the variables which denote the node. As in section 4, in the homogenous case the system wide losses can again be easily computed by simply observing a single node in isolation.

5.2 Job Loss with no Load Sharing

With no load sharing, $\lambda_2 = 0$, and the problem reduces to a bounded waiting time problem for an $M/M/1$ queue. Equations 11 and 12 become identical to those derived in [8]. Analytical expressions for $f(w)$ have been obtained in [8]. The loss rate experienced by a single node can be easily computed from,

$$\text{Loss rate}^{NLS} = \lambda_1 P^{rej} \quad (15)$$

5.3 Job Loss with QDLS

For a homogenous system the QDLS policy reduces to the Probing Policy with $L_p = 1$. Therefore, we discuss the losses incurred by a single node in the following section. We note that for a heterogenous system, extensive numerical computations are required to solve both, equations 11 and 12 as well as to optimize the losses. We have thus chosen to study the simpler homogenous system.

5.4 Job Loss with Probing

When Probing Policy is used, locally rejected jobs with service time greater than $K2$ are simply lost. The loss rate at any node due to the *large service times* (the stream $\lambda_{loss}^{>K2}$ in Figure 6) is given by,

$$\begin{aligned} \lambda_{loss}^{>K2} &= \lambda^{rej} \int_{K2}^{\infty} du J(u) \\ &= \lambda_1 (1 - B(K1) + \int_{K2}^{K1} b(u) [1 - F(K1 - u)] du) \end{aligned}$$

For the remaining jobs in the stream λ^{rej} (with service time less than $K2$ and denoted by the stream $\lambda^{<K2}$ in Figure 6), probes are sent to L_p other nodes. A job is lost only if all nodes which are probed cannot meet the deadline of the job. Thus the loss rate at a node due to *congestion* at other nodes is,

$$\lambda_{loss}^c = (\lambda^{rej} - \lambda^{>K2}) \left(\int_0^{K1} [1 - G(K2 - x)] d_u F(u) \right)^{L_p}.$$

The loss rate at a single node can be determined from,

$$\text{Loss rate}^{probing} = \lambda_{loss}^c + \lambda_{loss}^{>K2} \quad (16)$$

Equations 11 and 12 can be numerically solved for a given set of parameters $\lambda_1, \lambda_2, K1, K2$ and $F(0^+)$ (the impulse function of the unfinished work at 0) (see Appendix B). However, $F(0^+)$ and λ_2 are still unknown. We now exploit the homogeneity of the system (see [7]) to obtain the flow constraint at a

single node. As described in section 4.3.2 at steady state, the flow rate at which jobs are internally transferred out of a node must equal the flow rate at which probes are accepted. Thus,

$$\lambda_2 \int_0^{K2} G(K2 - x) d_u F(u) = \lambda_1^{rej} - \lambda_{loss}^{>k2} - \lambda_{loss}^c \quad (17)$$

Equations 13 and 17 can now be solved to determine the unknowns $F(0^+)$ and λ_2 .

5.5 Numerical Results

In this section we compare the losses of the probing policy for real-time tasks with bounded waiting time with that of no LS and perfect LS for a system of 20 nodes. We assume that the service time distribution of the jobs that arrive externally is exponentially distributed and that $\mu = 1$ jobs/second.

To solve for $F(0^+)$ and λ_2 in equations 13 and 17, standard IMSL packages were used. However, to numerically perform the integration required in equations 13 and 17, the value of the function at discrete points are required. These values were determined from the equations 11 and 12. Note that the left hand side of these equations can be expressed as $f(w)$ (the pdf of the unfinished work), which transforms the integro-differential equations to integral equations. The integral equations were solved using the method of substitution and the integration was numerically performed using the trapezoidal rule.

One technique to determine the value of the function $f(w)$ at discrete points, for a given set of $F(0^+)$ and λ_2 would be to explicitly substitute the expression of the PDF $G(x)$ using equation 14 in equation 11. Thus $f(w)$ becomes a function of the integral containing $F(w)$. However, for incorrect values of $F(0^+)$ and λ_2 , initially provided by IMSL, the solution of $f(w)$ (using the the method of substitution to solve 11) fails to convergence.

A second approach (used here) would be to iteratively iteratively determine the value of $G(x)$ and use the given set of $G(x)$ at each step, to solve the integral equation 11 (see Appendix B for details). Partial success was obtained and convergence still posed a considerable problem, particularly for large values of $K1$ and λ_1 . For these values, the losses due to failure in probe attempts becomes significant and the contribution of $\lambda_{loss}^{>c}$ to the system wide loss rate is larger than that of $\lambda_{loss}^{>k2}$. In fact for these cases simulations can be performed in a shorter amount of time. Where possible we have provided numerical values for the losses. For tight time constraints, e.g $K1 = 1.5$ convergence can be achieved relatively fast. These results are plotted in Figure 7. For a larger time constraint, $K1 = 3.0$ (see Figure 8) convergence was slower. For example, $L_p = 5$ and $\lambda_1 > 1.4$ convergence of our numerical technique posed a great problem. Nevertheless, for higher values of $K1$, we were able to obtain convergence for values of $\lambda_1 \leq 0.7$ (see Figure 9). It should be pointed out that our technique thus does provides useful results for most values of the practical range of system parameters.

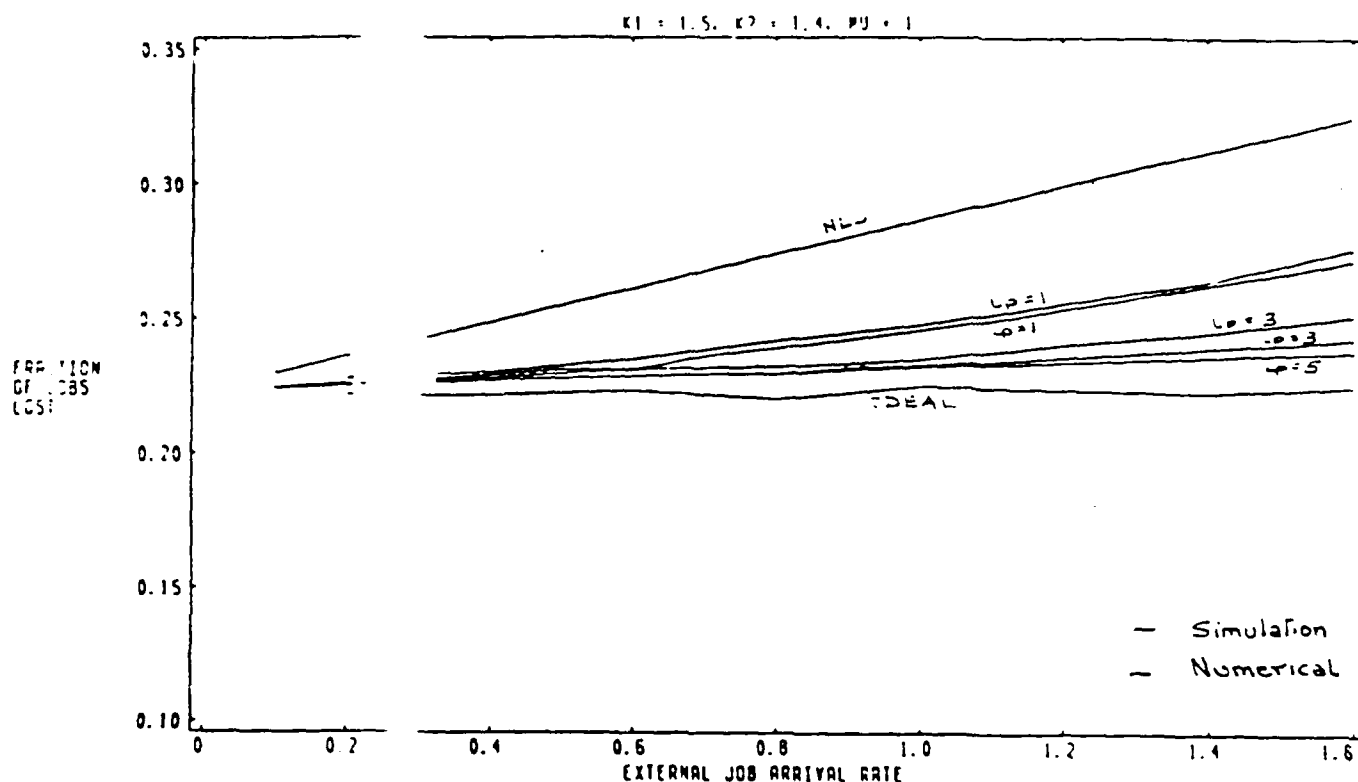


Figure 7: Job loss for Probing for tasks with bounded queue time for $K1 = 1.5$

For $L_p = 0$, the probing policy reduces to NLS for which analytical solutions of the function $f(w)$ is available in [8]. The losses computed from equation 15, for $L_p = 0$ were identical to those obtained in [8].

As in the previous section the "Perfect LS" system is modelled by an $M/M/20$ queueing system, with time constraint, $K1$. This is identical to the system where the jobs are scheduled among the node processors using complete state information. In the ideal case this information is known at no cost and incur no transfer delays. Thus the performance of the "ideal" LS policy provides an unattainable lower bound.

Figure 7 plots the fraction of jobs lost for an extremely tight time constraint, $K1 = 1.5$ time units, and transfer delay of 0.1 time units. Note that since $\mu = 1$, 22.3% of the jobs have service times greater than 1.5 and hence a minimum of 22.3 % jobs are always lost. The graph clearly shows the vast improvement in the performance of the system, even for $L_p = 1$. For low arrival rates, $\lambda_1 < 0.8$ (which is 80% of the practical range interest of λ_1), $L_p = 1$ achieves close to ideal performance. The simulations results are plotted for $L_p = 1$ and $L_p = 3$. These were obtained without imposing the poisson assumption on the arrival rates of the probes (λ_2). The close match between the simulation results and the numerical results thus justifies this assumption. In the simulations we have also accounted for the fact that the state of the system may undergo a change during the period when a node first responds

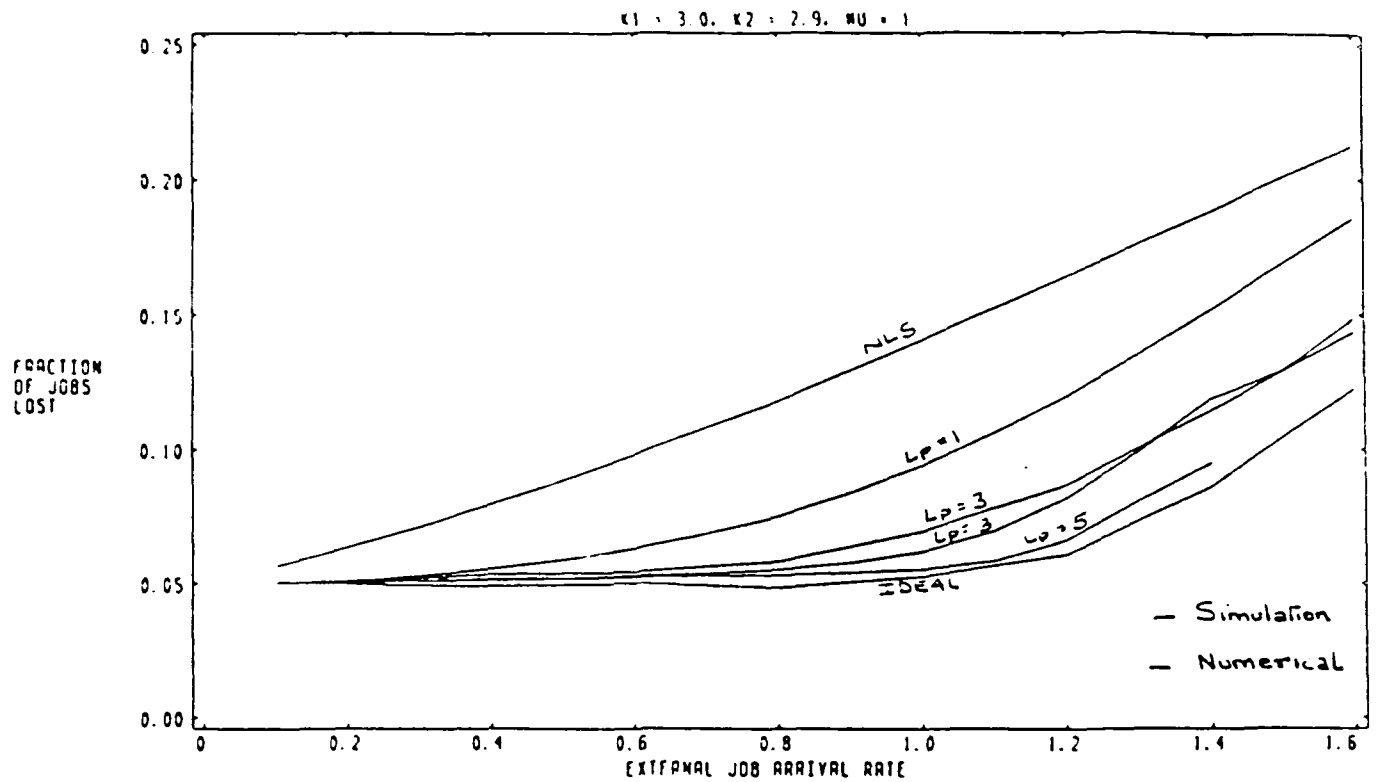


Figure 8: Job loss for Probing for tasks with bounded queue time for $K1 = 3.0$

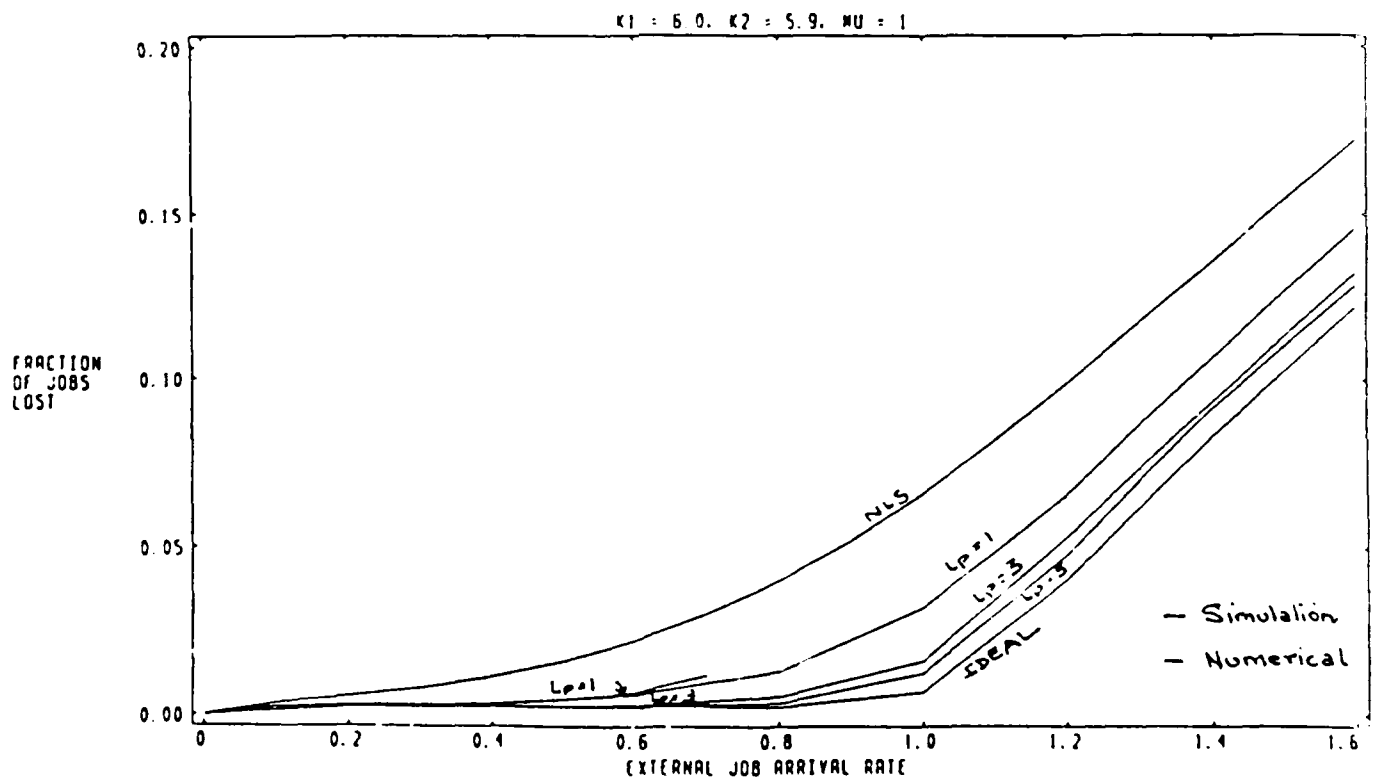


Figure 9: Job loss for Probing for tasks with bounded waiting time for $K1 = 6.0$

positively to a probe and the arrival of the job.

Figure 8 shows the performance of the LS policies for $K1 = 3.0$. For this value of the time constraint, only 4.5 % of the jobs are lost due to the large service times (> 3.0). Accordingly, load sharing plays an important role in preventing losses even at low values of λ_1 . For $\lambda_1 < 0.8$ we see that $L_p = 3$ gives rise to losses that are close to the ideal. Note that the probe limit required is larger, than for $K1 = 5$. This is due to the fact that a larger percentage of the jobs which cannot be locally processed are now eligible for transfer. Also since the time constraint is larger more external jobs can be accepted by a node thus increasing its unfinished work load. Values obtained via simulations are plotted for $L_p = 3$.

Figure 9 plots the loss incurred by the system for a much more relaxed time constraint, $K1 = 6.0$. As mentioned earlier, we were able to obtain results for values of $\lambda_1 > 0.7$. Surprisingly, we see that with $L_p = 3$ the losses incurred are close to ideal for $\lambda_1 < 0.8$.

A common trend observed in all the performance curves is that the probing policy with probe limit of one achieves a great reduction in the losses over the NLS policy for a wide range of λ_1 . Further improvement in the performance is obtained by using $L_p = 3$ and we see that this value of the probe limit reduces the losses significantly over those obtained with $L_p = 1$, particularly for larger values of λ_1 . But note that this reduction is less. By comparing the graphs for $L_p = 3$ and $L_p = 5$ particularly for $K1 = 6.0$ we note that choosing higher values of L_p provides only a marginal improvement. This observation only serves to strengthen our claim that relatively small probe limits are adequate.

Comparing the losses experienced by real-time system with the two classes of tasks in Figures 3, 7 and 9, we observe that the losses are significantly smaller (for equal arrival rates and comparable time constraints) in for the system in which jobs must complete execution within the fixed time limit. This is true since all jobs with large service times are filtered out, allowing for a larger number of shorter jobs to be processed locally. The reverse holds for tasks with bounded queue time. A large job greatly increases the unfinished work in the system, thus preventing all jobs which arrive during its residency in system from executing locally. As such, the average waiting time of bounded-queueing-time jobs, with time constraint $K1$, will be larger for than that of bounded-waiting-time jobs, with time constraint, $K1 + \mu$.

6 Conclusions

In this paper, we have examined the relative performance of several different decentralized approaches towards load sharing in order to address the question of determining the appropriate level of complexity for load sharing algorithms in a distributed real-time environment. Queueing theoretic models were developed to quantitatively assess the performance of two relatively simple approaches towards load sharing as well as the bounding case of no load sharing. The "ideal" case of load sharing with perfect information and no transfer delays was studied through simulation. The assumptions and

approximations made in our analysis were validated through simulation.

A major conclusion of this study of real-time LS approaches is complementary to that previously established for non-real-time systems [21,7]: *simple* approaches, which use a minimum amount of global state information and involve very simple decision mechanisms, can often achieve a performance level close to that of a theoretically optimum real-time load sharing algorithm. A corollary then is that for all but the tightest of time constraints (e.g., values of the time constraint, $K1$, less than the average job service time), a more sophisticated approach towards real-time load sharing can often result in only a small marginal performance improvement over the extremely simple load sharing algorithms. In particular, it was shown that a simple probing approach using a small probe limit, performed close to optimal over a wide range of arrival rates and for all but the most stringent time constraints.

We believe that future work in this area may be directed towards extending and generalizing the results presented in this paper. In particular, it is of interest to develop performance models for systems in which arriving jobs may have a deadline drawn from additional deadline distributions and again assess the appropriate level of complexity for load sharing algorithms in these cases. It would also be of interest to consider the *local* scheduling of jobs to processors in these cases.

REFERENCES

- [1] F. Baccelli, P. Boyer, G. Hebuterne, "Single-Server Queues with Impatient Customers", *Adv. Appl. Probability*, Vol. 16, pp. 887-905, 1984.
- [2] J. Bannister and K. Trivedi, "Task Allocation in Fault Tolerant Distributed Systems", *Acta Informatica*, Vol. 20, pp. 261-281, 1983.
- [3] D. Bourne and M. Fox, "Autonomous Manufacturing: Automating the Job Shop", *IEEE Computer*, Vol. 17, No. 9, pp 76-89, Sept. 1984.
- [4] P. Brill and M. Posner, "Level Crossing in Point Processes Applied to Queues: the Single Server Case", *Operations Research*, Vol. 25, No. 6, pp. 662-674, July-Aug. 1977.
- [5] J.W Cohen, "Single Server Queue with Uniformly Bounded Virtual Waiting Time," *J. Appl. Prob.* Vol. 3, pp. 265-284, 1969.
- [6] B. Doshi and H. Heffes, "Overload Performance of Several Processor Queueing Disciplines for the M/M/1 Queue", *IEEE Trans. Communications*, Vol. COM-34, No. 6, pp. 538-546, June 1986.
- [7] D. Eager, E. Lazowska, and J. Zahorjan, "Dynamic Sharing in Homogenous Distributed Systems", *IEEE Trans. of Software Eng.*, Vol. SE-12, No. 5, pp. 662-675, May 1986.
- [8] Bezalel Gavish, Paul Schweitzer, "The Markovian Queue with Bounded Waiting Time", *Management Science*, Vol. 23, No. 12, pp. 1349-1357, August 1977
- [9] V. Hunt and G. Kloster, "The FAA's Advanced Automation System: Strategies for Future Air Traffic Control Systems", *IEEE Computer*, Vol. 20, No. 2, pp. 19-33, Feb. 1987.
- [10] L. Kleinrock, *Queueing Systems: Volume 1: Theory*, Wiley Interscience, New York, 1975.
- [11] J.F.Kurose, S.Singh, "A Distributed Algorithm for Optimum Static Load Balancing in Distributed Computer Systems", *IEEE 1986 Infocom Conference Proceedings*, pp. 458-468, (Miami, April 1986).
- [12] D. W. Leinbaum and M. Yamini, "Guaranteed Response in a Hard Real-Time Environment", *IEEE Trans. on Software Eng.* , Vol. SE-6, No. 1, pp. (Jan. 1980).
- [13] C. Locke, H. Tokuda and E. Jensen, "A Time-Driven Scheduling Model for Real-Time Operating Systems", CMU Tech. Report (Archons Technical Report) May 1985.
- [14] J. Loris-Teghem, "The Waiting Time Distribution in a Generalized Queueing System with Uniformly Bounded Sojourn Times," *J. Appl. Prob.* Vol. 9, pp. 642-649, 1972.

- [15] P. McGregor and R. Boorstyn, "Optimal Load Sharing in a Computer Network", *Proc. ICC '75*, pp. 41-14 - 41-19, 1975.
- [16] P. Ma, E. Lee and M. Tsuchiya, "A Task Allocation Model for Distributed Computing Systems", *IEEE Trans. Computers*, Vol. C-31, No. 1, pp. 41-48, Jan. 1982.
- [17] A. Mok, "Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment", MIT Tech. Report MIT/LCS/TR-297, May 1983.
- [18] R. Muntz and E. G. Coffman, "Preemptive Scheduling of Real-Time Tasks on Multiprocessor Systems", *JACM*, Vol. 17, No. 2, April 1970.
- [19] E. DeSouza e Silva and M. Gerla, "Load Balancing in Distributed Systems with Multiple Classes and Site Constraints", *Performance '84*, pp. 17-33, 1984.
- [20] J. Stankovic, "A Perspective on Distributed Computing", *IEEE Trans. Computers*, Vol. C-33, No. 12, pp. 1102-1115, Dec., 1984.
- [21] J. Stankovic, "Simulations of Three Adaptive, Decentralized Controlled Job Scheduling Algorithms", *Computer Networks*, Vol. 8, pp. 199-217, Aug. 1984.
- [22] J. Stankovic, K. Ramamritham, S. Cheng, "Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems", *IEEE Trans. Computers*, Vol. C-34, No. 12, pp. 1130-1144, Dec., 1985.
- [23] H. Steusloff, "Advanced Real Time Languages for Distributed Industrial Process Control", *IEEE Computer*, Vol. 17, No. 2, pp. 48-59, Feb., 1984.
- [24] A. Tantawi and D. Towsley, "Optimal Static Load Balancing in Distributed Computer Systems", *Journal of ACM*, Vol. 32, No. 2, pp 445-465, April 1985.
- [25] K.J. Lee and D. Towsley, "A Comparison of Priority-Based Decentralized Load Balancing Policies", *Performance '86*, pp. 70-78, May 1986.
- [26] W. Zhao and K. Ramamritham, "Distributed Scheduling Using Bidding and Focused Addressing", *IEEE Real-Time Symposium*, Dec. 1985.

Appendix A : Derivation of Equation 2, 11, 12 Using Level Crossing [4] [6]

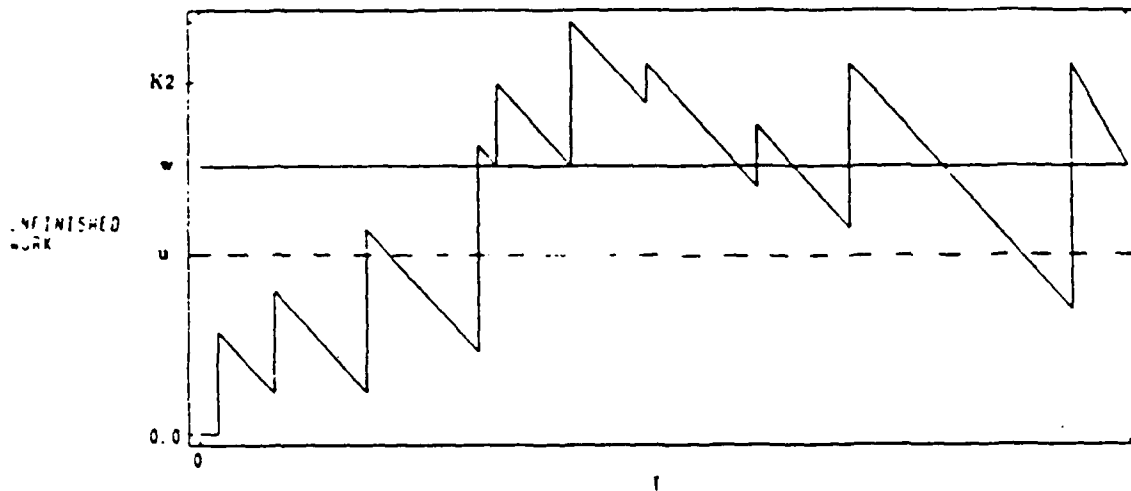


Figure 10: Level Crossing in the region $w \leq K2$

As shown in figure 10, if we plot the unfinished work in the generic queue in figure 2 as a function of time, we obtain a "sawtooth" line, where the vertical jumps represent increments of work brought to the queue by an arriving customer and the slope of the decreasing sections of the line is -1. The point at which an increasing or decreasing section of the sawtooth line intersects a horizontal line of height w is referred to as an "upcrossing" or "downcrossing" at w , respectively.

A major result of level crossing states that the density function, $f(w)$, of the "virtual waiting time" (i.e., the total unfinished work in the queueing system) is equal to the rate at which downcrossings cross a line of constant height, w , and that for ergodic systems, the rate of downcrossings equals the rate of upcrossings through this line [4] [6].

Tasks with Bounded Queueing Time:

In order to determine $f(w)$ in the region $w \leq K2$, we note that an upcrossing occurs at w when an arrival to the generic queue finds some amount, u ($u \leq w$), of unfinished work in the queue upon its arrival and itself joins the queue and brings in an amount of work greater than $w - u$. If $B(x)$ is the PDF of the service time demands of an arriving customer, then the probability that the amount of work brought in by an arriving customer is greater than $w - u$ is simply $1 - B(w - u)$, and the rate of upcrossings at w in our generic queue is given by

$$\text{rate of upcrossings} = (\lambda_1 + \lambda_2) \int_0^w (1 - B(w - u)) f(u) du$$

Equating the rate of upcrossings to $f(w)$, the rate of down crossings immediately gives equation 1 in the region $w \leq K2$.

In order to determine $f(w)$ in the region $K2 < w \leq K1$, we separately consider upcrossings due to arrivals at the lower and upper queues in figure 2. Following an identical argument as above, the rate of upcrossings at w due to arrivals at the lower queue is given by:

$$\text{rate of upcrossings due to arrivals at lower queue} = \lambda_1 \int_0^w (1 - B(w - u)) f(u) du$$

Note that a job arriving at the upper queue will only join the queue if it finds an amount of unfinished work, u , less than $K2$. The rate of upcrossings due to arrivals at the upper queue is thus given by:

$$\text{rate of upcrossings due to arrivals at upper queue} = \lambda_2 \int_0^{K2} (1 - B(w - u)) f(u) du$$

Equating the rate of upcrossings and $f(w)$, the rate of down crossings at w , immediately yields equation 2.

Tasks with Bounded Waiting Time:

Following the above arguments, for $0 < w \leq K2$, a job arriving at the lower queue will give rise to an upcrossing from some level u ($< w$), iff the amount of work brought in by the job exceeds $w - u$, but is less $K1 - u$ (otherwise the job's deadline cannot be met). Since $B(x)$ is the PDF of jobs arriving at the lower queue, the probability that an upcrossing occurs is, $[B(K1 - u) - B(w - u)] f(u) du$. Similarly, for a job arriving in the upper queue the probability that an upcrossing occurs from the level u is, $[G(K2 - u) - G(w - u)] f(u) du$. Therefore, the total rate of upcrossing is given by,

$$\begin{aligned} \text{rate of upcrossings due to an arrival} &= \lambda_1 \int_0^w \{B(K1 - u) - B(w - u)\} d_u F(u) \\ &+ \lambda_2 \int_0^w \{G(K2 - u) - G(w - u)\} d_u F(u) \end{aligned}$$

Equating the rate of down crossing with that of upcrossing yields Equation 11.

In the second region, $K2 < w \leq K1$, an upcrossing from any level u ($< w$) will occur only when a job arrives at the lower queue. Hence,

$$\text{rate of upcrossings due to an arrival} = \lambda_1 \int_0^w \{B(K1 - u) - B(w - u)\} d_u F(u)$$

Equation 12 is obtained by equating the rate of upcrossing and downcrossing.

Appendix B : Solution of Equation 3 and 11 and 12

Tasks with Bounded Queueing Time:

For the purpose of clarity let $F_1(w)$ denote the function $F(w)$ in the region $0 < w \leq K_2$ and $F_2(w)$ denote the function $F(w)$ in the region $K_2 < w \leq K_1$. Consider equation 2 in the region $0 < w \leq K_2$:

$$-\frac{dF_1(w)}{dw} = (\lambda_1 + \lambda_2) \left\{ \int_0^w B(w-u) d_u F_1(u) - F_1(w) \right\}$$

Define $f_1(w) = \frac{dF_1(w)}{dw}$. Let F_1^* denote the Laplace Transform of $f_1(w)$. Taking the Laplace transform and rearranging the terms we get [10],

$$F_1^*(s) = \frac{sF_1(0^+)}{s + (\lambda_1 + \lambda_2)(B^*(s) - 1)} \quad (18)$$

Assuming service times are exponentially distributed, we have $B^*(s) = \mu/(\mu + s)$. Thus,

$$F_1^*(s) = \frac{sF_1(0^+)}{s + (\lambda_1 + \lambda_2)(\mu/(\mu + s) - 1)}$$

On taking the inverse Laplace transform, we obtain an expression for $f_1(w)$. Then $F_1(w)$ is simply given by:

$$F_1(w) = \int_0^w f_1(u) du$$

The solution to equation 2 in the region $K_2 < w \leq K_1$ is obtained in a similar manner. We can rewrite the expression as:

$$\begin{aligned} -\frac{dF_2(w)}{dw} = & (\lambda_1 + \lambda_2) \left\{ \int_0^{K_2} B(w-u) d_u F_1(u) - F_1(K_2) \right\} \\ & + \lambda_1 \left\{ \int_{K_2}^w B(w-u) d_u F_2(u) - \int_{K_2}^w d_u F_2(u) \right\} \end{aligned}$$

Define $g(w) = U(w - K_2)f_2(w)$, where $f_2(w)$ is the density function in the region defined by, $K_2 < w \leq K_1$ and $U(w - K_2)$ is a step function. Let $G^*(s)$ be the Laplace transform of $g(w)$. Then for $B^*(s) = \mu/(\mu + s)$ we get,

$$G^*(s) = \frac{(\lambda_1 + \lambda_2)C e^{-\mu K_2} e^{-s K_2}}{s + \mu - \lambda_1}$$

where

$$C = F(0^+) e^{(\lambda_1 + \lambda_2) K_2}$$

On inverting the above transform we obtain an expression for $f_2(w)$, in the region $w > k_2$. The desired result for $F_2(w)$ can now be easily derived since

$$F_2(w) = \int_0^{K_2} f_1(u) du + \int_{K_2}^w f_2(u) du$$

Note that although we have derived expressions for an exponentially distributed service time, the above technique can be used to solve a general class of service time distribution.

Tasks with Bounded Waiting Time:

Separating the value of $F(w)$ at zero (denoted by $F(0^+)$), Equations 11 and 12 can be rewritten as:

$$\begin{aligned} f(w) &= \lambda_1 \{B(K_1) - B(w)\} F(0^+) + \lambda_2 \{G(K_2) - G(w)\} F(0^+) \\ &\quad + \lambda_1 \int_{0^+}^w \{B(K_1 - u) - B(w - u)\} f(u) du \\ &\quad + \lambda_2 \int_{0^+}^w \{G(K_2 - u) - G(w - u)\} f(u) du \quad 0 < w \leq K_2 \\ f(w) &= \lambda_1 \{B(K_1) - B(w)\} F(0^+) \\ &\quad + \int_{0^+}^w \{B(K_1 - u) - B(w - u)\} d_u F(u) \quad K_2 < w \leq K_1 \end{aligned}$$

Similarly, equation 13 takes the form,

$$F(0^+) \left(1 + \int_{0^+}^{K_1} f(u) du \right) = 1$$

Hence, with the knowledge of the values of the function in the interval $(0, K_1]$, the two unknown $F(0^+)$ & λ_2 can be computed from the equations 13 and 17.

The explicit form of the function $G(x)$ was not used due to the poor convergence obtained while solving for $F(0^+)$ & λ_2 . A second level of iterations was introduced to compute $G(x)$. Our algorithm was as follows:

1. Initialise $G(x) = B(x) \quad 0 \leq x \leq K_1$
2. WHILE NOT DONE
 - For the given value of $G(x)$, compute the value of $F(x)$ for $x \in [0, K_1]$, using the method of substitution to solve the integral equations 11 and 12.
 - Determine $F(0^+)$ & λ_2 (using IMSL routines) by solving equations 13 and 17.
 - Compute the fraction of jobs lost using equation 16.

- Compute the new $G(x)$ from equation 14.

3. DONE = TRUE when three successive iterations give losses within 0.1% of each other.

APPENDIX B

STEPHEN G. STRICKLAND and CHRISTOS G. CASSANDRAS
 Department of Electrical and Computer Engineering
 University of Massachusetts, Amherst, MA 01003

ABSTRACT

We present a new method for estimating performance sensitivities, with respect to parameters of interest, of Markov and (some) semi-Markov processes from information contained in a single nominal sample realization. Given a nominal parameter value, we use a perturbation of the parameter to define a perturbed system. We then construct what we call an *augmented chain* which in effect allows us to construct perturbed realizations from nominal ones, and hence compute sensitivities of any quantities measurable on these realizations. We show that the "observability" problem encountered in our earlier work can be overcome through an appropriate transformation of the augmented chain and present some experimental results.

1 INTRODUCTION

In performance optimization problems involving discrete event dynamic systems, analytical expressions of the performance measure (in terms of controllable parameters) generally do not exist. Gradient-based optimization methods typically employed in such cases require the sensitivity (or partial derivative) of the performance with respect to the parameter(s) over which the optimization is being performed. Traditionally, one resorts to simulation and uses a finite difference estimator to compute these sensitivities; this requires two simulation runs for each parameter. Recently, several new approaches have been developed, which extract information from an observed nominal state trajectory of the system in question, and directly estimate the sensitivity of the performance measure with respect to a parameter of interest. These sample path based techniques include the Likelihood Ratio Method [8],[3] and Perturbation Analysis [4],[5]-[7]. In general, these methods realize considerable computational savings as compared to the two-simulation approach since they require only a single simulation run. More importantly, since they involve only observed data, they may also be used in on-line or real-time control schemes.

This paper considers new sample path based method which has significant advantages in many situations. One advantage of this approach is that nominal system parameters need not be known. Further, the method can be applied to discrete (integer-valued) parameters (e.g. buffer capacities, routing thresholds, customer class sizes) for which the performance measures are necessarily discontinuous.

The method presumes the existence of two Markov chains whose structure is known. We assume both represent the same underlying system of interest, but differ in the value of some parameter; thus, we refer to them as the *nominal* and *perturbed* chains. Observation of a sample realization of the nominal system allows direct measurement of its sample performance. If we can at the same time estimate the sample performance for a perturbed realization, then we can immediately compute a finite difference sensitivity estimate. We accomplish this by using the event-driven nature of the underlying system to construct an *augmented chain* related to the nominal and perturbed chains in the following two ways: (1) The augmented chain is *stochastically similar* to the perturbed chain in that the stationary state probabilities of the perturbed chain are obtainable as the probabilities of appropriately defined aggregate states in the augmented chain. (2) The augmented chain is *observable* with respect to the nominal chain in that we can estimate the augmented chain state probabilities using information contained in a single observed nominal realization.

In previous work [1] we developed a method for constructing an augmented chain which was always stochastically similar to both the nominal and the perturbed chains; however, it was not always observable with respect to the nominal chain. We proposed a solution to this problem which involved generating additional "artificial" events to supplement those observed in the nominal sample path. This requires, however, that we know or estimate the rate parameter associated with these events. In this paper, using a more direct construction, we formalize the notion of observability and show that under some general conditions an *observable* augmented chain can always be constructed through a transformation of the initial augmented chain. We consider extensions to semi-Markov processes and present some experimental results.

¹This work is supported in part by the National Science Foundation under grant ECS-8504676 and by the Rome Air Development Center under contract F302602-81-C-0169

2 DIRECT SYNTHESIS OF AUGMENTED CHAINS

In this section we extend our previous results [1] by imposing an *event structure* on the Markov chains under consideration. This allows us to obtain an augmented chain directly, as well as accommodate a more general class of Markov chains. In addition, it results in a more compact representation.

We assume that the Markov chains considered meet the following conditions:

(A1) There is a finite set of events; each transition defined in a chain is associated with a unique event. Moreover, all transitions associated with a given event have the same rate (i.e. the transition rate is a function of the event type alone).

(A2) For each state, there is at most one outgoing transition corresponding to each event.

Remark : We can accommodate state dependent transition rates by expanding the number of event types.

2.1 Definitions and Notation

Consider a discrete event dynamic system, represented by $\{S, E, D\}$, where S is a state space, E is a set of events which cause all possible state transitions, and D is a transition function, $D : S \times E \rightarrow S$.

Let $E^f(s)$ denote the set of events which can occur when the system is in state s ; we will refer to this as the *feasible set* of events at state s . Then, given $s \in S$ and some $e \in E$, we define $D(s, e)$ as

$$D(s, e) = \begin{cases} \text{destination state when } e \text{ occurs in state } s & \text{if } e \in E^f(s) \\ 0 & \text{if } e \notin E^f(s) \end{cases}$$

If an event $e \notin E^f(s)$ occurs at state s , the event is effectively ignored and the state is unchanged. This is to be distinguished from the case of a "self-loop" transition, where $D(s, e) = s$ for some event $e \in E^f(s)$. Note that $D(s, e)$ also defines a *destination matrix* describing the system.

A Markov chain α is obtained from the discrete event system definition above by requiring that the events of each type constitute Poisson processes, and by providing an intensity function $F : E \rightarrow \mathbb{R}$, characterizing each of these processes. Thus, we may write $\alpha = \{S, E, D, F\}$.

Note that we can easily obtain the infinitesimal generator of α , Q , from D and F as

$$Q(s_i, s_j) = \begin{cases} \sum_e \delta(D(s_i, e), s_j) F(e) & s_i \neq s_j \\ -\sum_{k \neq i} Q(s_i, s_k) & s_i = s_j \end{cases} \quad (1)$$

where $s_i, s_j \in S$, and we effectively sum the rates of all possible transitions from s_i to s_j caused by events in $E^f(s_i)$ ($\delta(\cdot, \cdot)$ is the indicator function: $\delta(x, y) = 1$ if $x = y$ and 0 otherwise).

2.2 Construction and Properties of Reduced Augmented Chains

Given the preceding notation, consider two Markov chains $\alpha_1 = \{S_1, E_1, D_1, F_1\}$ and $\alpha_2 = \{S_2, E_2, D_2, F_2\}$. For convenience, we make the following additional assumption:

(A3) α_1 and α_2 are finite, irreducible, and ergodic chains; hence, they have unique stationary state probability vectors π_1, π_2 , determined by $Q_i \pi_i = 0$, $i = 1, 2$.

We then define the Reduced Augmented Chain (RAC) (the term "reduced" is used to distinguish this augmented chain from the maximal augmented chain (MAC) defined in [1]) corresponding to α_1 and α_2 , as a Markov chain

$$\alpha_R = \{S_R, E_R, D_R, F_R\}$$

where

$$S_R = S_1 \times S_2$$

$$E_R = E_1 \cup E_2$$

and D_R is defined for each element $(s_i, s_j) \in S_R$ with $s_i \in S_1$ and $s_j \in S_2$ by

$$D_R[(s_i, s_j), c] = \begin{cases} (D_1(s_i, c), D_2(s_j, c)) & \text{if } D_1(s_i, c) \neq 0, D_2(s_j, c) \neq 0 \\ (D_1(s_i, c), s_j) & \text{if } D_1(s_i, c) \neq 0, D_2(s_j, c) = 0 \\ (s_i, D_2(s_j, c)) & \text{if } D_1(s_i, c) = 0, D_2(s_j, c) \neq 0 \\ 0 & \text{if } D_1(s_i, c) = 0, D_2(s_j, c) = 0 \end{cases} \quad (2)$$

and finally, F_R is given by

$$F_R(c) = \begin{cases} F_1(c) & \text{if } c \in E_1 \\ F_2(c) & \text{if } c \in E_2, c \notin E_1 \end{cases} \quad (3)$$

It is shown in [2] that this definition results in a RAC identical to that obtained in our earlier work [1]. A key feature of α_R is that it generally contains transient states, which can be removed since we are interested in the stationary state probabilities. A simple algorithm for constructing directly the irreducible, ergodic sub-chain of the RAC (i.e. the RAC with transient states removed), given an initial state, is also given in [2]. Hereafter, when referring to the RAC, we will assume that all transient states have been removed; thus, α_R refers to the irreducible set of states containing a given initial state.

An example of a RAC construction is shown in Figure 1, where we show the RAC which results from a nominal chain representing a homogeneous M/M/1/1 queueing system, with transition rates λ (for arrival events, a) and μ (for departure events, d), and a perturbed chain representing an M/M/1/2 system with the same transition rates. The elements S, E, D , and F of our representation are shown along with the corresponding state diagrams. Note that the construction includes two transient states, which are ignored.

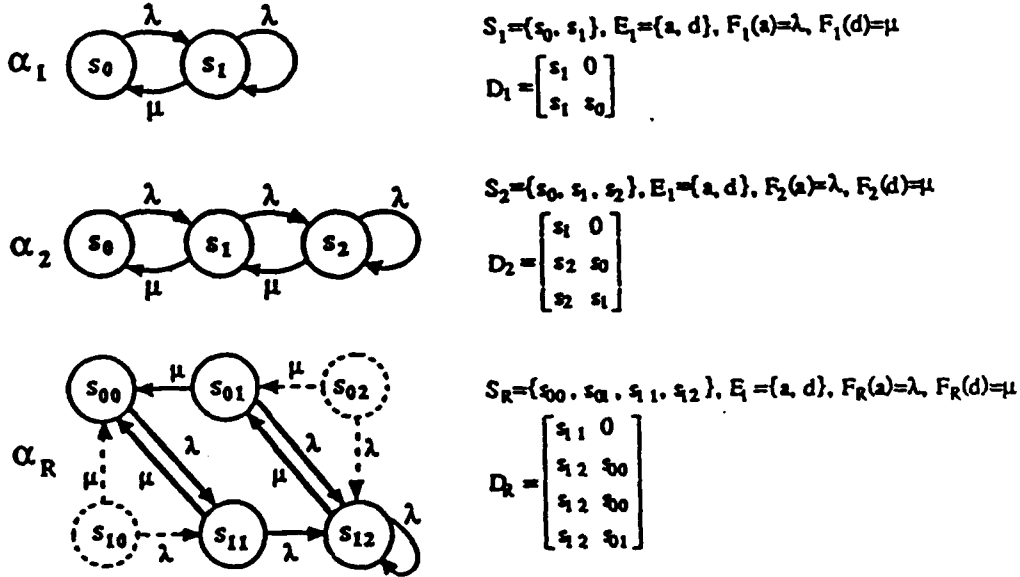


Figure 1: RAC For The M/M/1/1, M/M/1/2 System Pair

In order to provide a more compact representation for D_R , let us define

$$D_R^1(s_i, c) = \begin{cases} D_1(s_i, c) & \text{if } D_1(s_i, c) \neq 0 \\ s_i & \text{otherwise} \end{cases} \quad (4)$$

$$D_R^2(s_j, c) = \begin{cases} D_2(s_j, c) & \text{if } D_2(s_j, c) \neq 0 \\ s_j & \text{otherwise} \end{cases} \quad (5)$$

which allows us to rewrite (2) as

$$D_R[(s_i, s_j), c] = (D_R^1(s_i, c), D_R^2(s_j, c)) \quad (6)$$

2.2.1 The Stochastic Similarity Property

As previously mentioned, our motivation for constructing α_R is that it contains information about the joint behavior of α_1 and α_2 . In fact, if S_R is appropriately partitioned, α_R can be transformed into a Markov chain with the same stationary state probability vector as either α_1 or α_2 . We refer to this property as *stochastic similarity*.

Definition: Let $\alpha = \{S, E, D, F\}$ and $\alpha_0 = \{S_0, E_0, D_0, F_0\}$ be two Markov chains, with stationary state probability vectors π and π_0 respectively, and $\dim(S_0) = N$. The Markov chain α is said to be *stochastically similar* to α_0 with respect to P iff there exists a partition $P = \{p_i\}_{i=1, \dots, N}$ of S such that

$$\pi(p_i) = \pi_0(s_i) \text{ for all } s_i \in S_0$$

Clearly, the partition P results in a Markov chain α_P . Letting Q_P and Q_0 denote the infinitesimal generators of α_P and α_0 , an alternative definition is to require that $Q_P = Q_0$.

Remark: Stochastic similarity implies that the transitions between the partition sets $p_i \in P$ (as applied to S_0) constitute a realization of α_0 . Thus we can explicitly extract a realization of α_0 from the observed realization of α by simply ignoring all transitions internal to any p_i .

Given the definition of stochastic similarity above, we now wish to establish the fact that the RAC, α_R , defined above, is indeed stochastically similar to both α_1 and α_2 . Thus, we will show that elements of S_R can be aggregated in ways that allow us to express the stationary state probabilities of α_1 and α_2 in terms of such "aggregate" or "composite" states. The following lemma identifies the fundamental property of α_R which makes this possible.

Lemma 1: Let $\alpha = \{S, E, D, F\}$ and $\alpha_0 = \{S_0, E_0, D_0, F_0\}$ be two Markov chains, with infinitesimal generators Q and Q_0 respectively, and $\dim(S_0) = N$. If there exists a partition of S given by

$$P = \{p_i, i = 1, \dots, N\}$$

such that for all $a \in p_i$ and $j \neq i$,

$$\sum_{b \in p_j} Q(a, b) = Q_0(s_i, s_j) \quad (7)$$

where $s_i, s_j \in S_0$, then α is stochastically similar to α_0 with respect to P .

Proof: (see [2])

We shall now use this lemma to construct appropriate partitions of the RAC defined above, and establish stochastic similarity properties with α_1 and α_2 . First, as in our earlier work [1], we define the following partitions of α_R :

$$P_r = \{r_i : (s_k, s_j) \in r_i \text{ iff } s_k = s_i \in S_1, s_j \in S_2\} \quad (8)$$

$$P_c = \{c_i : (s_k, s_j) \in c_i \text{ iff } s_j = s_i \in S_2, s_k \in S_1\} \quad (9)$$

The definition of S_R as the cartesian product $S_1 \times S_2$ gives it a rectangular structure with the "rows" associated with elements of S_1 and the "columns" associated with the elements of S_2 . The partitions above simply formalize this fact. Thus, we will refer to $r_i \in P_r$ as the " i^{th} row" of α_R and $c_i \in P_c$ as the " i^{th} column" of α_R .

In the following result, we show that the partitions P_r and P_c allow us to establish that α_R is stochastically similar to α_1 and α_2 , respectively.

Theorem 1: Let S_R be partitioned through P_r and P_c . Then, α_R is stochastically similar to α_1 with respect to P_r , and to α_2 with respect to P_c , i.e.

$$\pi_R(r_i) = \pi_1(s_i) \text{ for all } s_i \in S_1 \quad (10)$$

$$\pi_R(c_j) = \pi_2(s_j) \text{ for all } s_j \in S_2 \quad (11)$$

Proof: (using Lemma 1; see [2]).

The key implication is that a single realization of α_R provides the same information as two distinct realizations, one of α_1 and the other of α_2 . Thus, we can use such a realization to obtain information regarding the behavior of both α_1 and α_2 . Suppose that α_1 represents a Markov chain model of a nominal system, and let α_2 be identical to α_1 except for some specified parameter perturbation. Thus, α_2 corresponds to a perturbed system. In this context, a realization of α_R provides sufficient information to estimate performance sensitivities of the nominal system with respect to the perturbed parameter. Neither α_1 nor α_2 need be observed.

2.2.2 The Observability Property

As already mentioned, the stochastic similarity property of the RAC, α_R , presented in Theorem 1, is of interest if sample paths of α_R can be conveniently constructed, given sample paths of α_1 (the directly observable Markov chain). This is indeed possible under an *observability* condition, which we shall formalise in this section.

Consider a Markov chain $\alpha_0 = \{S_0, E_0, D_0, F_0\}$ and let $s_0 \in S_0$ be a specified initial state. Then, for any event sequence $e = \{e_0, e_1, \dots\}$, with $e_i \in E_0$ for all i , there is a corresponding state sequence $s = \{s_0, s_1, \dots\}$, with $s_i \in S_0$ for all i . Thus, (s_0, e) describes a stochastic realisation of α_0 in terms of the sequence of states visited. Note that $s_i, i = 1, 2, \dots$ represents the state of the system just after event e_i occurs. In case $e_i \notin E^f(s)$, we have $s_i = s_{i-1}$ and no actual transition takes place.

Now suppose we consider a second Markov chain $\alpha = \{S, E, D, F\}$, and specify an initial state $t_0 \in S$. Given the same event sequence e , we may generate a state sequence $t = \{t_0, t_1, \dots\}$ with $t_i \in S$ for all i . We define the notion of observability in terms of the relationship between $E^f(s_i)$, the feasible set of events causing transitions when α_0 is in state s_i , and $E^f(t_i)$, the corresponding feasible set of events for $t_i \in S$.

Definition: Let $\alpha_0 = \{S_0, E_0, D_0, F_0\}$ and $\alpha = \{S, E, D, F\}$ be two Markov chains with specified initial states $s_0 \in S_0$ and $t_0 \in S$ respectively. Let $e = \{e_i\}$, $i = 0, 1, \dots$ be any event sequence with $e_i \in E_0$ for all i , and $s = \{s_i\}$, $t = \{t_i\}$ the corresponding state sequences for α_0, α . Then,

1. An event $e \in E$ (or the corresponding state transition from t_i to t_{i+1}) is said to be *observable* with respect to α_0 iff

$$e \in E^f(t_i) \Rightarrow e \in E^f(s_i) \text{ and } F(e) = F_0(e) \quad (12)$$

for any $i = 0, 1, \dots$

2. The chain α is said to be *observable* with respect to α_0 iff $e \in E^f(t_i)$ is observable for all $i = 0, 1, \dots$ and all event sequences e

Remark: If α is stochastically similar to α_0 with respect to some partition P , the definition is simplified by virtue of the fact that P constrains the sequence t in terms of s . Thus, if $\alpha = \alpha_R$, when α_0 is in state $s_i \in S_0$, α_R is in state $(s_i, s_j) \in S_R$ (for some s_j) by construction. This is true for all event sequences e used to generate s . Therefore, in view of (2) and (3), the observability condition for α_R with respect to the nominal chain α_1 becomes

$$E^f[(s_i, s_j)] \subseteq E^f(s_i) \text{ for all } s_i \in S_1, (s_i, s_j) \in S_R \quad (13)$$

or equivalently,

$$D_R[(s_i, s_j), e] \neq 0 \Rightarrow D_1(s_i, e) \neq 0 \text{ for all } e \in E_R \quad (14)$$

In the example of Figure 1, if the M/M/1/1 chain is observed, then RAC state s_{01} has an unobservable transition of rate μ to state s_{00} , since the nominal state corresponding to s_{00} ($s_0 \in S_0$) has no feasible μ transition (no departure can occur when the system is empty). Note that if the M/M/1/2 chain were observed, then all events/transitions would be observable.

The definition above specifies sufficient conditions for constructing a realisation of α from an observed realisation of α_0 . The feasible set of a state $s \in S$, along with $F(e)$ for all $e \in E^f(s)$, define the parameter of the holding time distribution (which is necessarily exponential) of the state, as well as the distribution of the type of the next event. Thus we can view observability as a condition which guarantees that the sequence of holding times and events which are observed in the realisation of α_1 have the correct distributions to be used in constructing a realization of α (see [2]).

3 FULLY OBSERVABLE REDUCED AUGMENTED CHAINS

As stated previously, our goal is the construction of a RAC, α_R , which is both observable with respect to α_1 and stochastically similar to α_2 . While α_R , as defined in Section 2.2, is always stochastically similar to α_2 , it will often not be observable with respect to α_1 due to the constraining nature of the observability conditions. In this section, we show that, subject to some general conditions, we can transform α_R to a new RAC, α'_R , which is fully observable with respect to α_1 , and retains a modified form of stochastic similarity, which we shall call ξ -similarity. In brief, we decompose the states of the RAC in a way that allows us to both eliminate the unobservable transitions and express the perturbed state probabilities in terms of aggregate states created by the decomposition.

3.1 ξ -Similarity

We begin by extending the definition of stochastic similarity given in section 2.2.1.

Definition : Let $\alpha = \{S, E, D, F\}$ and $\alpha_0 = \{S_0, E_0, D_0, F_0\}$ be two Markov chains with stationary state probability vectors π and π_0 respectively, and $\dim(S_0) = N$. The Markov chain α is said to be ξ -similar to α_0 with respect to $V = \{V_i\}_{1, \dots, N}$ iff there exists a set $V \subseteq S$ and a constant $\xi \in (0, 1]$ such that:

$$\pi(V_i) = \xi \pi_0(s_i) \quad \text{for all } s_i \in S_0$$

The next Lemma is a generalisation of Lemma 1, and establishes conditions under which two chains are ξ -similar. In what follows, given a state space S , $s \in S$, and $A \subset S$, we shall use $Q(s, A)$ to denote $\sum_{t \in A} Q(s, t)$. We shall also denote the complement of A with respect to S by $\bar{A}$.

Lemma 2: Let $\alpha = \{S, E, D, F\}$ and $\alpha_0 = \{S_0, E_0, D_0, F_0\}$ be two Markov chains with infinitesimal generators Q and Q_0 respectively, and $\dim(S_0) = N$. Let P be a partition of S such that

$$P = V \cup W = \{V_i\}_{1, \dots, N} \cup \{W_i\}_{1, \dots, N}$$

and the following conditions hold for all $i = 1, \dots, N$:

- (C1) $Q(s, \bar{W}_i) = Q(s, V_i)$, for all $s \in W_i$,
- (C2) $Q(s, V_i) + Q(s, W_i) = Q_0(s_k, s_i)$, for all $s \in V_k$, $k \neq i$
- (C3) W_i is not an absorbing aggregate state

Then, α is ξ -similar to α_0 with respect to V , i.e.

$$\pi(V_i) = \xi \pi_0(s_i)$$

and ξ is given by

$$\xi = \pi_R(V) = 1 - \pi_R(W)$$

Proof : (by flow balancing around each V_i and W_i ; see [2]).

Remark : Lemma 1 may be viewed as a special case of Lemma 2, where $V = S$ and $V_i = p_i$. In this case $\xi = 1$ and condition (C2) reduces to (7). Note that no assumptions are made regarding transitions originating in each W_i , except that any terminal states lying outside W_i must belong to the corresponding V_i .

In what follows, we will decompose S_R , the state space of α_R defined in section 2.2, so as to define a partition satisfying the conditions of Lemma 2, and also where all unobservable transitions originate within a W_i set. Using this partition, we define a transformation of α_R yielding a new RAC, α'_R , which is observable with respect to α_1 (nominal chain) and ξ -similar to α_2 (perturbed chain).

3.2 Decomposing the RAC: Active and Passive States

Without loss of generality, we assume that α_1 is the nominal chain (recall that we denote the nominal and perturbed chains, and the RAC by $\alpha_1, \alpha_2, \alpha_R$, respectively). Let U_i denote the set of RAC states in the composite state c_i (equivalently, the i^{th} column) defined in (9) which emit unobservable transitions, and let $R_i = c_i - V_i$.

Let us attach a binary indicator to the state of α_R which takes on two values: *active* and *passive* and is defined as follows. We assume the state is initially active at the start of the sample path. Entering any U_i while the state is active causes a switch to the passive state. The indicator remains passive until the system enters the R_i corresponding to the U_i which initiated the passive state. At this point it returns to the active state. Note that this indicator has no effect on the evolution of the original state vector. Further, entering any U_i while in the passive state has no effect on the binary indicator.

The RAC states $s \in S_R$ are thus decomposed into distinct component states $\{s_a, s_p^0, s_p^1, s_p^2, \dots\}$ defined by $s_a = s$, given the system is in the active state, and $s_p^i = s$, given the system is in the passive state as a result of entering U_i .

Based on this decomposition, let us define an aggregate state W_i as the set of all i^{th} passive components of states $s \in S_R$, i.e.

$$W_i = \bigcup_{s \in S_R} s_p^i \quad \text{for } i = 1, 2, \dots$$

$$\overbrace{\dots R_i R_j R_i R_k R_j \dots}^{\text{active}} \overbrace{U_i \dots}^{\text{passive}} \overbrace{R_i R_i R_k R_j \dots}^{\text{active}} \Rightarrow \overbrace{\dots V_i V_j V_i V_k V_j \dots}^{\text{active}} \overbrace{W_i \dots}^{\text{active}} \overbrace{V_i V_i V_k V_j \dots}^{\text{active}}$$

Figure 2: The RAC State Trajectory Decomposition

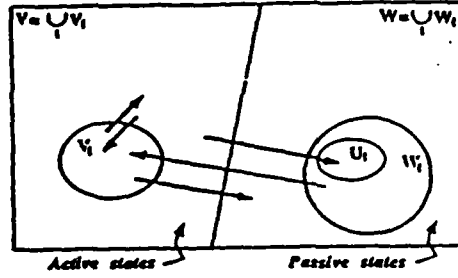


Figure 3: Active and Passive State Space Decomposition

In effect, the RAC enters W_i when it enters U_i while in the active state, and remains in W_i until the next visit to R_i . Thus, the RAC state trajectory can be viewed as a sequence of active segments connected by visits to a single W_i (Figure 2). Let us also use V_i to denote the set of all active components of states $s \in R_i$, and let $V = \cup_i V_i$. This defines the most important decomposition of S_R —into active and passive states (see Figure 3)—as

$$S_R = \bigcup_s s_a \cup \bigcup_{i,s} s_p^i = \bigcup_i V_i \cup \bigcup_i W_i = V \cup W \quad (15)$$

The next result establishes the fact that this decomposition of S_R satisfies the conditions of Lemma 2, and, therefore, allows α_R to be ξ -similar to α_2 with respect to the set of active states, V .

Lemma 3: Given the partition $V \cup W$ defined by (15), if $V_i \neq \emptyset$ for all i , then α_R is ξ -similar to α_2 with respect to V , i.e.

$$\pi_R(V_i) = \xi \pi_2(s_i), \text{ with } \xi = \pi_R(V) = 1 - \pi_R(W)$$

Proof: (using Lemma 2; see [2]).

Remark: Lemma 3 establishes ξ -similarity between α_R and α_2 with respect to the set of active states V . The portions of the nominal trajectory during which the system occupies active states (elements of V) effectively constitute a realization of the perturbed system (this is similar to the "cut-and-paste" idea of [7]). Thus we use observations made during these portions to estimate the perturbed state probabilities. The fraction of the total observation interval which these portions constitute is given by ξ , where $0 < \xi \leq 1$. For the fastest possible convergence of our estimates, we clearly want ξ as close to 1 as possible.

Note that, by construction, all unobservable transitions originate in W (if none exist, then $W = \emptyset$ and $\xi = 1$). Thus, our next objective is to transform α_R so as to eliminate all unobservable transitions without violating (C1)-(C3).

3.3 The Observability Transformation

In this section we present our main result which states that under certain general conditions, there exists a transformation of α_R yielding a new RAC, α'_R , which is both observable with respect to α_1 and ξ -similar to α_2 .

We begin by defining a state transition transformation for a Markov chain $\alpha = \{S, E, D, F\}$. Let Φ denote the set of all state transitions defined in α , i.e.

$$\Phi = \{(s, t, c) : s, t \in S, c \in E, D(s, c) = t, F(c) > 0\}$$

Then, a state transition transformation is defined to be a mapping:

$$T : \Phi \rightarrow (S \times S) \cup \emptyset$$

where the mapping to $\emptyset$ corresponds to removal of a transition $[(s, t), c]$. For simplicity, we shall limit ourselves to transforming transitions caused by a single event between each pair of states, and not affecting the transition rate of this event. Thus, we denote this transformation by

$$T(s, t) = (u, v) \text{ or } T(s, t) = \emptyset$$

where $(u, v) \in (S \times S)$, and the associated event is implied.

Now, we seek a transformation T^* which, when applied to all state transitions of α_R , generates a RAC observable with respect to α_1 and also ξ -similar to α_2 .

Theorem 2: Let Φ_R be the set of state transitions in α_R , and T^* a transformation applied to $(s, t) \in \Phi_R$ such that

$$T^*(s, t) = \begin{cases} \emptyset & \text{if } (s, t) \text{ is unobservable} \\ (s, v), v \in (R_j \cap V_i) & \text{if } s \in W_i, t \in r_j, i, j = 1, \dots, N \end{cases}$$

If $V_i \neq \emptyset$ for all i , the resulting RAC, α'_R , is observable with respect to α_1 and ξ -similar to α_2 with respect to V .

Proof: (see [2]).

In Figure 4 we show the application of this transformation to the example of Figure 1.

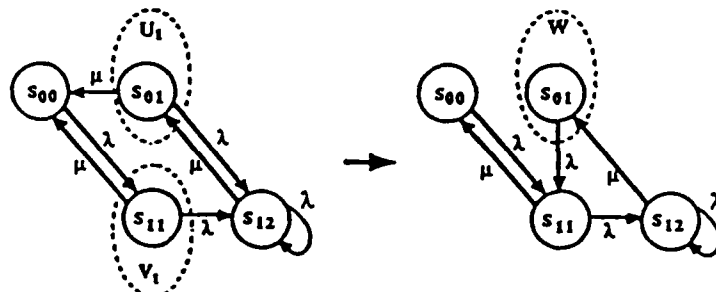


Figure 4: Observability Transformation of the $M/M/1/1, M/M/1/2$ RAC

Remark: Note that all but one of the conditions required are met either directly or indirectly by the construction of α_R . The only "real" condition is that $V_i \neq \emptyset$, for all i . This is equivalent to requiring that for each state $s \in S_2$, there exist at least one state $t \in S_1$ such that $E'(s) \subseteq E'(t)$. Also, note that simply removing the unobservable transitions is generally not sufficient since (as in our example—see Figure 4) it may make one or more W_i sets absorbing.

4 Extension To Semi-Markov Processes

We have investigated extensions of our method to semi-Markov processes via two directions. The first utilizes a discrete-time Markov chain imbedded in a continuous-time semi-Markov process. This only requires extending our existing methodology to discrete-time Markov chains. The second approach involves applying our continuous-time approach directly, simply relaxing the requirement that the event streams constitute Poisson processes. For certain restricted classes of semi-Markov processes, we obtain results equivalent to the pure Markovian case.

4.1 Imbedded RACs

The imbedded Markov chain approach is well known. We need only extend our augmented chain method to discrete-time Markov chains. While there are some complications due to the differing normalization conditions, this can be done. In Section 5 we apply the imbedded chain approach to the $M/GI/1/K$ queueing system. We note in passing that the utility of the imbedded chain approach rests on the ability to relate characteristics of the imbedded chain to those of the continuous-time chain in which it is imbedded. In the case of the $M/GI/1/K$ system, we are fortunate in that the stationary state probabilities of the (imbedded) discrete-time and continuous-time systems are identical. This is generally not the case.

4.2 Relaxation of the Markovian Assumption

In this approach, which we apply to the $GI/M/1/K$ system in Section 5, we simply relax the Markovian requirement for one or more of the event processes (allowing them to be arbitrarily distributed). In some cases (e.g. the $GI/M/1/K$ system) the efficacy of the method is unaffected by this relaxation. While this represents ongoing research, the basic idea can be outlined as follows.

Consider the problem of constructing a perturbed realization in a simulation environment. Given an initial state, our task can be viewed as one of constructing a "stochastically correct" sequence of state

holding-times ($\tau_0, \tau_1, \dots$), with associated terminating events ($c_0, c_1, \dots$). Given a model of the system, this event sequence uniquely determines the state sequence. At each iteration i , we use c_i to determine the next state, and repeat the process. In a simulation environment, we can generate a (τ_i, c_i) pair which is "stochastically correct" by generating an exponentially distributed random number t_k for each feasible event c_k (parameterized by the event rate) and setting $\tau_{i+1} = \min_k \{t_k\}$ and $c_{i+1} =$ the associated event. The augmented chain essentially lets the nominal system perform this operation. If the current nominal state has the same feasible set as the current perturbed state (in the construction) then the observed (τ_i, c_i) pair has the appropriate stochastic characteristics required by the perturbed realization. This is what occurs when the RAC is in the active state. If the nominal feasible set does not match that of the perturbed state, we suspend the perturbed construction until the nominal system enters a state which *does* match that of the perturbed state, at which point we proceed as before. Suspension of the construction corresponds to the RAC entering the passive state via some U_i ; entering a nominal state where we can restart the construction corresponds to the RAC re-entering the active state via the corresponding V_i .

In a non-Markovian environment, things are much more difficult because the (τ_i, c_i) statistics are not functions of the state alone but also of the elapsed times since the previous occurrence of each non-Markovian event. In some cases, however, we can still extract nominal (τ_i, c_i) pairs with the correct statistics. While a complete discussion is beyond the scope of this paper, this appears to be possible only when we have no more than one non-Markovian event process active at any time, when each U_i with an unobservable non-Markovian event is reachable only by transitions corresponding to that event, and where each corresponding V_i is reachable by transitions corresponding to the same event. Such a case is the M/GI/1/K system.

5 Experimental Results

In this section we provide experimental results for three variations on the single server queueing system: the M/M/1/K, M/GI/1/K, and GI/M/1/K systems (more extensive results are contained in [2]). In each case considered below, $K=2$, the "GI" distribution is deterministic, and the utilization is 1 (i.e. $\lambda = \mu$). The perturbed systems represent a change in queue capacity of +1. The performance measures considered are: the utilization, U , the mean queue length, N_Q , and the mean delay (or system time), D . We applied the following variations of our augmented chain approach:

1. URAC/K where we use the original, unobservable RAC of Section 2 and handle unobservable transitions by generating artificial events, when required, using a random number generator parameterized by the event rate *which we assume is known*.
2. URAC/E which is identical to the URAC/K case except that we assume the event rate is unknown, thus we estimate it using observations made in the nominal path.
3. IRAC where we use an imbedded discrete-time RAC.
4. TRAC where we use the observability transformation of Section 4 to obtain a fully observable RAC.
5. SIM which is a straightforward simulation of the perturbed chain.

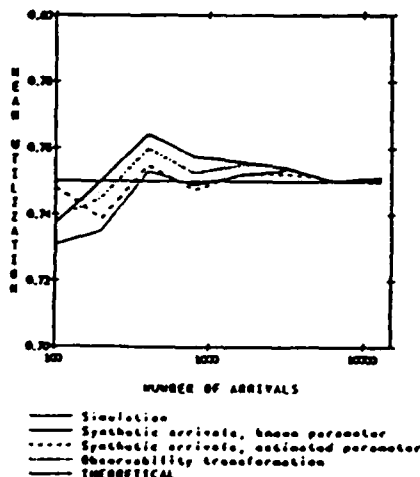


Figure 5: M/M/1/2 Utilization Estimates

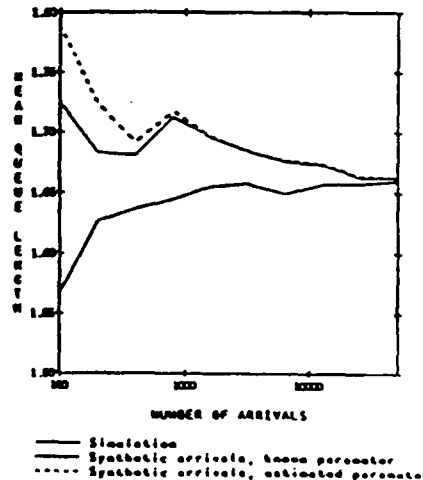


Figure 6: GI/M/1/2 Mean-Queue-Length Estimates

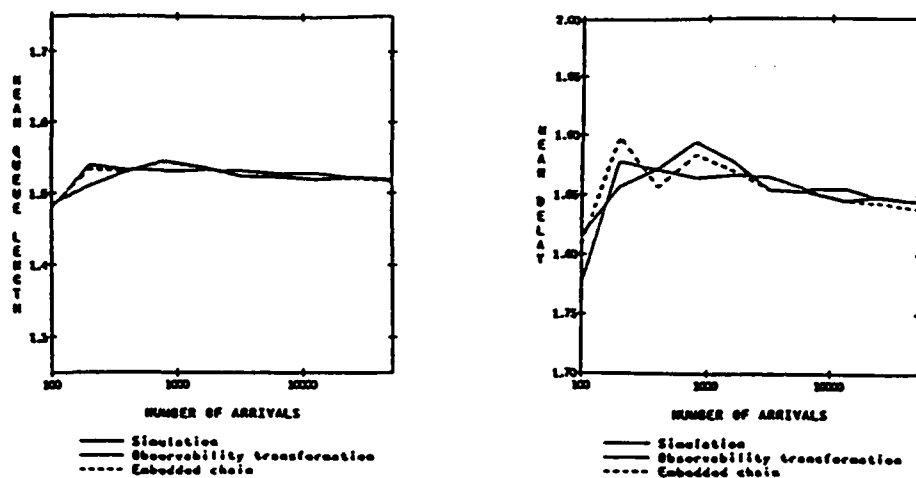


Figure 7: $M/GI/1/2$ Mean-Queue-Length and Delay Estimates

While all these techniques can be applied to Markov processes, not all can be applied to the same class of semi-Markov processes (see [2]).

For the $M/M/1/2$ system, we applied the URAC/K, URAC/E, TRAC, and SIM methods. In Figure 5 we show the resulting estimates of perturbed utilization as a function of the number of nominal arrivals. Also shown are the corresponding results using a straightforward simulation of the perturbed system. All curves are the average of 10 runs. Note that there is little degradation associated with estimating the artificial parameter from the nominal path as compared to a priori knowledge. Also note that the convergence of the observability-transformation method is slower than the other methods, indicating that $\xi < 1$.

We applied the and URAC/K, URAC/E, and SIM methods to the $GI/M/1/2$ system and the IRAC, TRAC, and SIM methods to the $M/GI/1/2$ system; the resulting estimates (averages of 10 runs) of perturbed mean-queue-length and delay (for the M/GI only) for these two systems are plotted versus the number of arrivals in Figures 5 and 7, respectively. The convergence of all methods is comparable.

Acknowledgement: The authors acknowledge many useful discussions with Don Towsley.

References

- [1] Cassandras, C.G. and Strickland, S.G., "An Augmented Chain Approach For On-Line Sensitivity Analysis of Markov Processes," to appear in *Proc. 26th IEEE Conf. on Decision and Control*, 1987.
- [2] Cassandras, C.G. and Strickland, S.G., "A General Approach For Sensitivity Analysis of Discrete Event Systems With Markovian Properties," unpublished manuscript, 1987.
- [3] Glynn, P.W. and Sanders, J.L., "Monte Carlo Optimization of Stochastic Systems: Two New Approaches," *Proc. 1986 ASME Computers in Engineering Conference*, 1986.
- [4] Gong, W. B. and Ho, Y.C., "Smoothed Perturbation Analysis of Discrete Event Dynamic Systems," to appear in *IEEE Trans. on Automatic Control*, 1987.
- [5] Ho, Y.C. and Cassandras C.G., "A New Approach to the Analysis of Discrete Event Dynamic Systems," *Automatica*, 19, 2, pp. 149-167, 1983.
- [6] Ho, Y.C. and Cao, X., "Perturbation Analysis and Optimization of Queueing Networks," *J. Optim. Theory and Applications*, 40, 4, pp. 559-582, 1983.
- [7] Ho, Y.C. and Li, S., "Extensions of Infinitesimal Perturbation Analysis," *subm. to IEEE Trans. on Automatic Control*, 1987.
- [8] Reiman, M.I. and Weiss, A., "Sensitivity Analysis For Simulations Via Likelihood Ratios," *Proc. of 1986 Winter Simulation Conference*, pp. 285-289.

APPENDIX C

A Comparison of the Processor Sharing and First Come
First Serve Policies for Scheduling Fork-Join Jobs in
Multiprocessors*

COINS Technical Report 87-83

August 26, 1987

D. Towsley
Dept. Computer and Information Sciences
Univ. of Massachusetts
Amherst, MA 01003

C. G. Rommel
Department of Electrical Engineering
Univ. of Massachusetts
Amherst, MA 01003

J. A. Stankovic
Dept. Computer and Information Sciences
Univ. of Massachusetts
Amherst, MA 01003

Abstract

In this paper a model of a shared memory multiprocessor that executes *fork-join* parallel programs as a bulk arrival $M^X/M/c$ queueing system is developed. Here a fork-join job is one that consists of a set of X tasks. All of the tasks arrive simultaneously to the system and the job is assumed to complete when the last task completes. We develop tight upper and lower bounds for the mean response time of such programs when the scheduling discipline is processor sharing under the assumptions of exponential task service times and a Poisson job arrival process. We study two processor sharing policies, one called *task scheduling* processor sharing and the other called *job scheduling* processor sharing. The first policy schedules tasks independently of each other and allows parallel execution, whereas the second policy schedules entire jobs as a unit and

*This work was supported in part by the National Science Foundation under grant MCS-8104203 and by RADC under contract RI-44896X and F302602-81-C-0169.

thereby does not allow parallel execution of an individual program. We find that the job scheduling policy exhibits better performance than task scheduling only on systems with a small number of processors, where the system is operating at high loads and is executing programs that can sustain a large degree of parallelism. Consequently, in general, task scheduling outperforms job scheduling. We also compare the performance of the processor sharing policy with first come first serve. We find that first come first serve exhibits better performance over a wide range of systems. The paper also studies the performance of processor sharing and first come first serve with two classes of jobs, and when a specific number of processors is statically assigned to each of these classes.

1 Introduction

With the advent of multiprocessors [Ost86] and programming languages that support parallel programming, (e.g., Concurrent Pascal [Han75], CSP [Hoa85], and Ada [Pyl81]) there is increasing interest in modeling the performance of parallel programs. In this paper, we evaluate the performance of a particular type of parallel program, a *fork-join* job, on a multiprocessor consisting of identical processors when the service discipline is processor sharing. In our model a fork-join job is composed of a set of tasks each of which can be scheduled independently of the others at any processor. All tasks in a given job arrive simultaneously to the system. The job completes when the last task completes.

The performance of parallel programs such as fork-join jobs is significantly affected by the choice of policy that is used to schedule tasks. We analyze the performance of a processor sharing (PS) policy that schedules *tasks* of a job independently of each other. We refer to this policy as *task scheduling PS*, TS-PS. We compare the performance of this TS-PS policy to that of a second PS policy that schedules entire *jobs* (as a single unit) independently of each other. We refer to this policy as *job scheduling PS*, JS-PS. The TS-PS policy is unaware that jobs exist whereas the JS-PS policy is unaware that tasks exist. We also compare the performance of TS-PS and JS-PS to the first come first serve (FCFS) policy. In these comparisons we consider different numbers of processors, sizes of fork-join jobs, multiple classes, and dedicated assignments of the processors of the multiprocessor to the different classes.

In the course of our study, we develop upper and lower bounds on the mean fork-join job response times under TS-PS. These bounds are generally very tight and we approximate the mean job response time by taking the average of the two bounds. Analyses of the other two policies, JS-PS and FCFS have already appeared in the literature ([RTS87, NTT87]).

We make the following observations from our study.

- FCFS provides better performance than TS-PS or JS-PS for a wide range of workloads and number of processors. It appears that the advantages that FCFS has over PS in

single processor systems carries over to multiprocessors executing parallel programs. This carries the implication that one should choose large quantum sizes for round robin policies operating on multiprocessors.

- TS-PS performs better than JS-PS most of the time. However, if the number of processors is small, the degree of parallelism high, and the processor utilization is high, JS-PS can perform better. This same phenomenon was observed on single processors in an earlier study, [RTS87].
- It may be useful to partition the processors in a multiprocessor into separate pools to handle different classes of jobs rather than having the jobs share the processors. We observe that jobs requiring the least amount of computation can benefit from such a partition.

In the remainder of this section we briefly review earlier work and outline the remainder of this paper. Processor-sharing has been addressed in the literature in several ways since its introduction [Kle64]. A survey of processor-sharing results may be found in [Kle76]. An exact analysis of the TS-PS policy operating on a *single* processor was performed by Rommel, et al. [RTS87]. Unfortunately, the approach used in that paper does not extend to multiple processors. This study first demonstrated that job scheduling can give better performance than task scheduling. In addition, there is a growing literature on fork-join queueing systems [BM85,BMT87,NT85]. Although these referenced papers analyze fork-join jobs, their analysis differs from that studied in this paper in that processors are allocated to specific tasks prior to execution. We are interested in systems where processors can be *dynamically* allocated to different tasks.

The format of this paper is as follows. We describe the queueing system under consideration in Section 2. Section 3 contains expressions for the upper and lower bounds on the mean response time for the TS-PS scheduling policy along with an approximate analysis of that policy. This is followed by our numerical results in Section 4. Finally, in Section 5 we summarize the results of the paper.

2 Model Description

We consider a system of c identical processors that serve a single queue. Fork-join jobs enter the system according to a Poisson process with parameter λ . A fork-join job consists of X tasks that can be processed independently of each other where X is a random variable (r.v.) with probability distribution $\alpha_i = P[X = i]$, $i = 1, 2, \dots$. The service time required by a task is assumed to be an exponential r.v. with parameter μ and is independent of the service requirements of all other tasks.

We are interested in the steady state behavior of this system when operating under the task scheduling processor sharing (TS-PS) and the job scheduling processor sharing (JS-PS) policies. As described in section 1, TS-PS is a policy that performs processor sharing at the task level and JS-PS is a policy that performs processor sharing at the job level. Thus, if the system contains two jobs, one with one task, the other with three tasks, then TS-PS provides an equal amount of service to each task and is capable of utilizing four processors. In this same example JS-PS sees two jobs, one whose service time is that of a single task, the other whose service time is the sum of the service times of the three tasks. JS-PS provides equal service to the two jobs and is only able to utilize two processors.

In both cases, we focus on the response time of a random job, i.e., the interval of time measured from the arrival of a job until the service completion of the *last task* associated with that job. The system can be visualized as a queue for tasks, c servers, and a waiting area for tasks that have completed service but are awaiting the completion of the last task associated with the job (Figure 1). This last queue is sometimes referred to as the *synchronization queue*. We denote this response time as T .

3 Analysis

In this section we concern ourselves with obtaining the mean response time $E[T]$ under both TS-PS and JS-PS. We consider JS-PS first as it is the simplest to analyze.

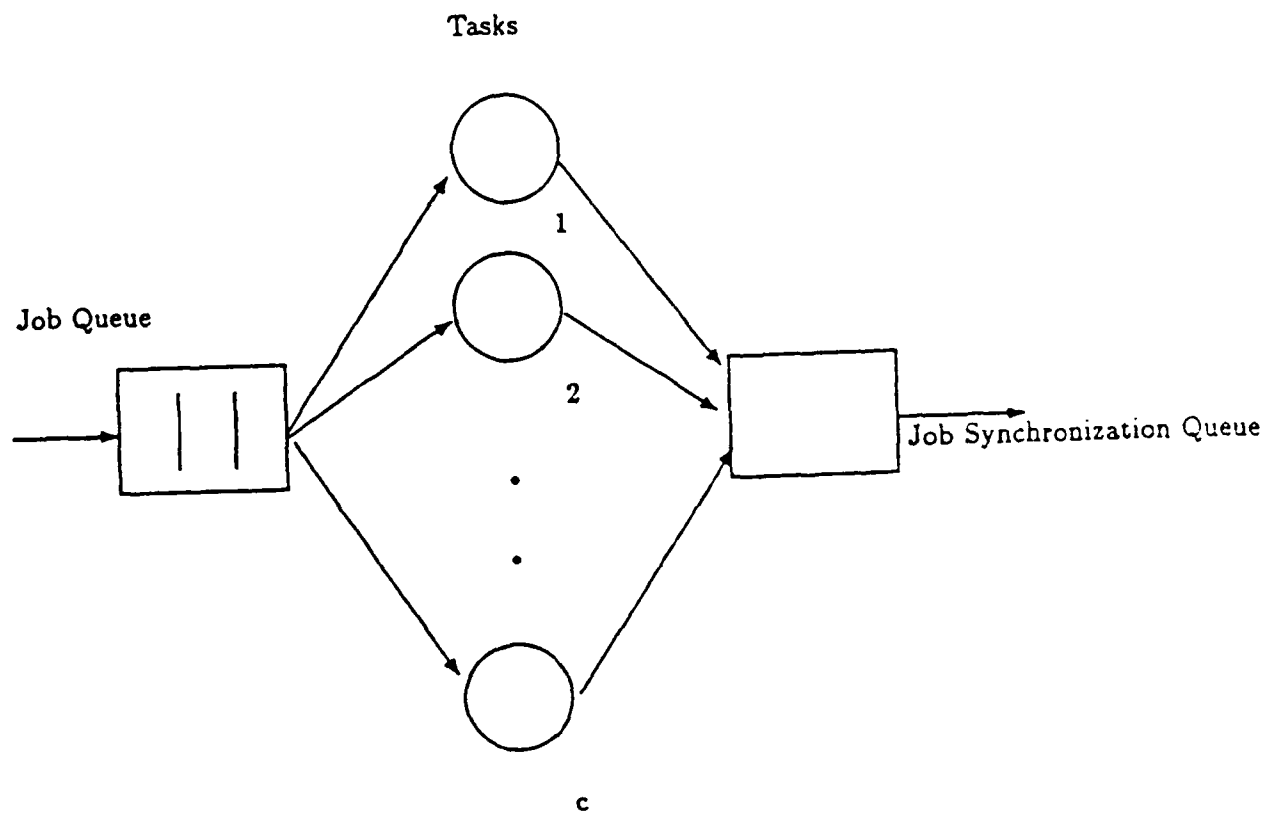


Figure 1: System Model

3.1 The JS-PS policy

Let L denote the number of jobs in the system under JS-PS. The distribution of L is identical to the queue length distribution of an $M/M/c$ system with arrival rate λ and average service time $E[X]/\mu$. Consequently, the average response time, $E[T]$, is ([All78])

$$E[T] = \frac{u^c(E[X]/\mu)}{cc![u^c/c! + (1 - u/c) \sum_{n=0}^{c-1} u^n/n!](1 - u/c)} + E[X]/\mu. \quad (1)$$

where $u = \lambda E[X]/\mu$. $E[T] = E[L]/\lambda$.

3.2 The TS-PS policy

To analyze the TS-PS policy, consider the delay that a randomly selected job incurs. Let J denote this job. Let N be a r.v. that denotes the number of tasks in the system at the time that J arrives. Let $\pi_n = P[N = n]$, $n = 0, 1, \dots$ denote the stationary distribution of N . Let $t_{i,n}$ denote the mean response time of J conditioned on the event that J consists of i tasks and that the system contains $N = n$ tasks at the time of its arrival, i.e. $t_{i,n} = E[T|X = i, N = n]$. We can write the following expression for the mean job response time,

$$E[T|X = i] = \sum_{n=0}^{\infty} \pi_n t_{i,n}, \quad i = 1, \dots \quad (2)$$

Removal of conditioning on the number of tasks in J yields

$$E[T] = \sum_{i=1}^{\infty} \alpha_i E[T|X = i]. \quad (3)$$

As described above, the number of tasks in the system is described by a Markov process. Fortunately, the behavior of this Markov process is independent of the policy used to schedule tasks so long as the policy does not schedule jobs based on service time information. Consequently, the distribution of N is identical to that for a bulk arrival $M^X/M/c$ system that schedules tasks in a FCFS manner. Expressions for the queue length distribution for this system can be found in earlier papers [CT83, Yao85, NTT87] and are omitted here.

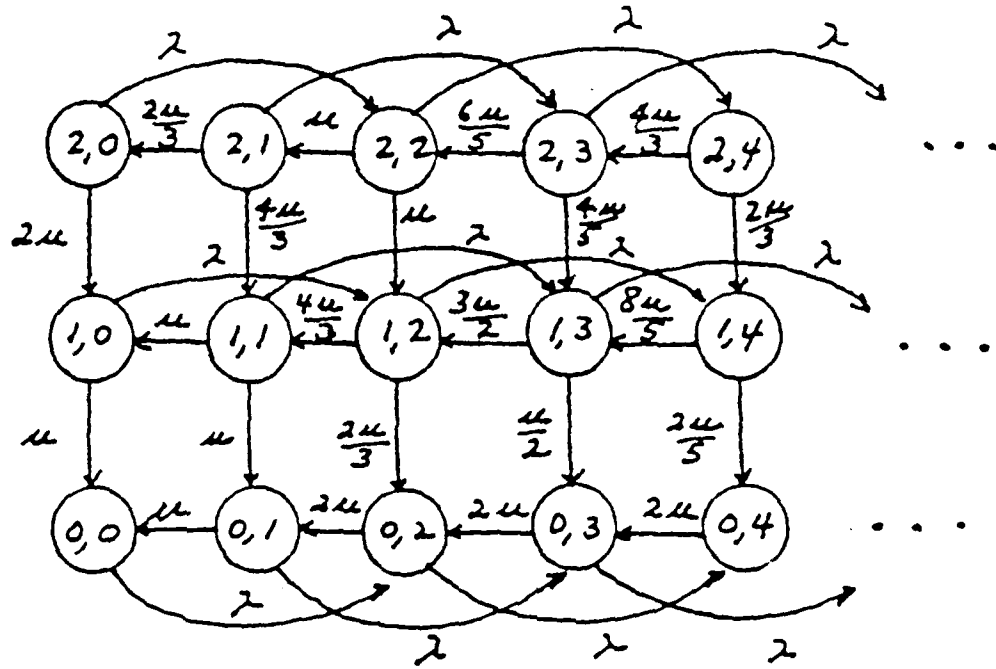


Figure 2: State diagram for the exact system when jobs consist of 2 tasks.

We focus on the conditional expectations $t_{i,n}$. We define a Markov Chain with state (I_t, M_t) with infinitesimal generator Q where I_t is the number of tasks remaining in J at time t after J is introduced at time 0, M_t is the number of tasks in the system at time t that are not part of J , and $Q = [q_{(i,n),(l,m)}]$ where

$$q_{(i,n),(l,m)} = \begin{cases} \frac{i}{i+n} \mu_{i+n}, & l = i-1, n = m, \\ \frac{n}{i+n} \mu_{i+n}, & l = i, m = n-1, \\ \lambda \alpha_{m-n}, & i = l, m > n, \\ -(\lambda + \mu_{i+n}), & i = l, m = n, \\ 0 & , \text{otherwise} \end{cases} \quad (4)$$

where

$$\mu_k = \begin{cases} k\mu, & k = 1, \dots, c \\ c\mu, & k = c+1, \dots \end{cases}$$

The resulting chain is transient. Figure 2 illustrates the associated state diagram when all jobs consist of exactly 2 tasks.

It follows from the definition of Q that $t_{i,n}$ satisfies

$$\begin{aligned}
t_{1,0} &= \frac{1}{\lambda + \mu_1} + \frac{\lambda}{\lambda + \mu_1} \sum_{k=1}^{\infty} \alpha_k t_{1,k}, \\
t_{1,n} &= \frac{1}{\lambda + \mu_{n+1}} + \frac{\lambda}{\lambda + \mu_{n+1}} \sum_{k=1}^{\infty} \alpha_k t_{1,n+k} + \frac{n\mu_{n+1}/(n+1)}{\lambda + \mu_{n+1}} t_{1,n-1}, \quad n = 1, \dots, \\
t_{i,0} &= \frac{1}{\lambda + \mu_i} + \frac{\lambda}{\lambda + \mu_i} \sum_{k=1}^{\infty} \alpha_k t_{i,k} + \frac{\mu_i}{\lambda + \mu_i} t_{i-1,0}, \quad i = 2, \dots, \\
t_{i,n} &= \frac{1}{\lambda + \mu_{i+n}} + \frac{\lambda}{\lambda + \mu_{i+n}} \sum_{k=1}^{\infty} \alpha_k t_{i,n+k} \\
&\quad + \frac{n\mu_{i+n}/(i+n)}{\lambda + \mu_{i+n}} t_{i,n-1} + \frac{i\mu_{i+n}/(i+n)}{\lambda + \mu_{i+n}} t_{i-1,n}, \quad i = 2, \dots; n = 1, \dots.
\end{aligned} \tag{5}$$

Consider the last expression, $t_{i,n}$. The first term is the average time that the system spends in state (i, n) . The second term is the contribution to $t_{i,n}$ due to an arrival. The third and fourth terms are the contributions due to a departure of a task belonging to J and a task not belonging to J , respectively.

We are unable to obtain a closed form solution to equation (5). As there are a countably infinite number of unknown variables $t_{i,n}$, $i = 1, \dots; n = 0, \dots$, it is impossible to obtain exact numerical values for these quantities. Consequently, the remainder of this section is concerned with developing upper and lower bounds on the conditional expectations $t_{i,n}$. These can be used to obtain upper and lower bounds for $E[T|X=i]$, $i = 1, \dots$. We treat each in turn.

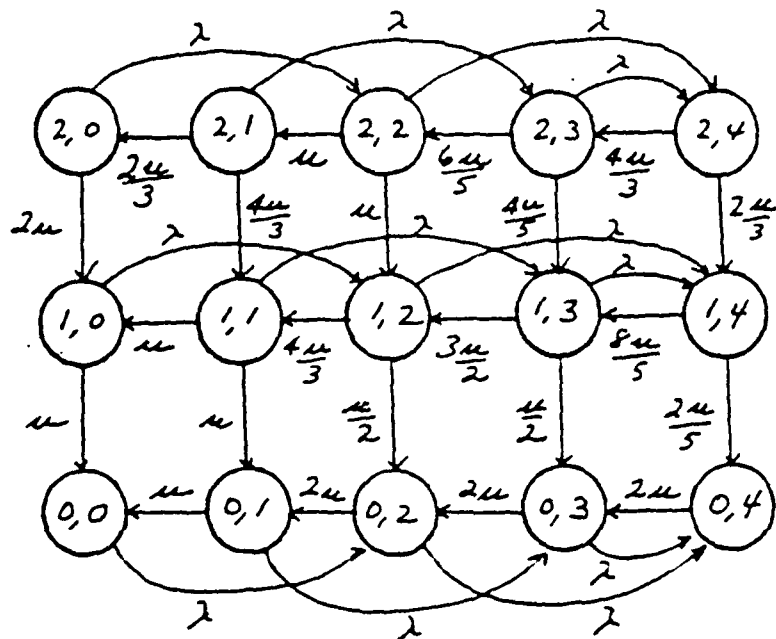


Figure 3: The state diagram associated with the lower bound, 2 tasks per job.

3.2.1 A Lower bound on $E[T|X=i]$

We study a Markov chain with state $(I_t^{(lb)}, M_t^{(lb)})$ that yields lower bounds $t_{i,n}^{(lb)}$ on $t_{i,n}$, $i = 1, \dots, n = 0, \dots$. This chain has infinitesimal generator $Q^{(lb)} = [q_{(i,n),(l,m)}^{(lb)}]$ where

$$q_{(i,n),(l,m)}^{(lb)} = \begin{cases} \frac{i}{i+n} \mu_{i+n}, & l = i-1, n = m; 0 \leq m \leq B, \\ \frac{n}{i+n} \mu_{i+n}, & l = i, m = n-1; 1 \leq m < B, \\ \lambda \alpha_{m-n}, & i = l, 0 \leq n < m < B, \\ \lambda \sum_{k=B-n}^{\infty} \alpha_k, & l = i, m = B, 0 \leq n < B, \\ -(\lambda + \mu_{i+n}), & i = l, m = n, 0 \leq m < B, \\ -\mu_{i+B}, & i = l, m = n = B, \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

This Markov chain corresponds to a system in which no more than B tasks not belonging to J are allowed in. Consequently, this modified system has fewer tasks that do not belong to J than the original system. The response time of J will be less in this system. Figure 3 illustrates the state diagram for this Markov chain when each job consists of exactly 2 tasks.

The conditional expectations, $t_{i,n}^{(lb)}$ satisfy

$$\begin{aligned}
t_{1,0}^{(lb)} &= \frac{1}{\lambda + \mu_1} + \frac{\lambda}{\lambda + \mu_1} \left(\sum_{k=1}^B \alpha_k t_{1,k}^{(lb)} + \sum_{k=B+1}^{\infty} \alpha_k t_{1,B}^{(lb)} \right), \\
t_{1,n}^{(lb)} &= \frac{1}{\lambda + \mu_{n+1}} + \frac{n\mu_{n+1}/(n+1)}{\lambda + \mu_{n+1}} t_{1,n-1}^{(lb)} + \\
&\quad \frac{\lambda}{\lambda + \mu_{n+1}} \left(\sum_{k=1}^{B-n} \alpha_k t_{1,n+k}^{(lb)} + \sum_{k=B-n+1}^{\infty} \alpha_k t_{1,B}^{(lb)} \right), \quad n = 1, \dots, B, \\
t_{i,0}^{(lb)} &= \frac{1}{\lambda + \mu_i} + \frac{\mu_i}{\lambda + \mu_i} t_{i-1,0}^{(lb)} + \\
&\quad \frac{\lambda}{\lambda + \mu_i} \left(\sum_{k=1}^B \alpha_k t_{i,k}^{(lb)} + \sum_{k=B+1}^{\infty} \alpha_k t_{i,B}^{(lb)} \right), \quad i = 2, \dots, \\
t_{i,n}^{(lb)} &= \frac{1}{\lambda + \mu_{i+n}} + \frac{\lambda}{\lambda + \mu_{i+n}} \left(\sum_{k=1}^{B-n} \alpha_k t_{i,n+k}^{(lb)} + \sum_{k=B-n+1}^{\infty} \alpha_k t_{i,B}^{(lb)} \right) + \\
&\quad \frac{n\mu_{i+n}/(i+n)}{\lambda + \mu_{i+n}} t_{i,n-1}^{(lb)} + \frac{i\mu_{i+n}/(i+n)}{\lambda + \mu_{i+n}} t_{i-1,n}^{(lb)}, \quad i = 2, \dots; n = 1, \dots, B. \quad (7)
\end{aligned}$$

Last, $t_{i,n}$, $n > B$ is bounded from below by $t_{i,B}^{(lb)}$, i.e.,

$$t_{i,n}^{(lb)} = t_{i,B}^{(lb)}, \quad i = 1, \dots; n = B+1, \dots. \quad (8)$$

Thus we have the following lower bound on $E[T|X=i]$,

$$E[T|X=i] \leq \sum_{n=0}^{B-1} \pi_n t_{i,n}^{(lb)} + P[N \geq B] t_{i,B}^{(lb)}, \quad i = 1, \dots. \quad (9)$$

additional $j < B$ tasks in the system. Now assume that k tasks arrive and that $n - k > B$. In this case, the time during which there are $B + 1$ or more additional tasks in the modified system is equal to the length of the busy period associated with a bulk arrival $M^X/M/1$ queue with rate μc that is initiated by the arrival of $n + k - B$ tasks. Consequently, we can write the following set of equations describing the expected response time of a job conditioned on the number of tasks at the time of arrival and the number of tasks in the arriving job, $t_{i,n}^{(ub)}$,

$$\begin{aligned}
t_{1,0}^{(ub)} &= \frac{1}{\lambda + \mu_1} + \frac{\lambda}{\lambda + \mu_1} \left(\sum_{k=1}^B \alpha_k t_{1,k}^{(ub)} + \sum_{k=B+1}^{\infty} \alpha_k (t_{1,B}^{(ub)} + b_{k-B}) \right), \\
t_{1,n}^{(ub)} &= \frac{1}{\lambda + \mu_{n+1}} + \frac{\lambda}{\lambda + \mu_{n+1}} \left(\sum_{k=1}^{B-n} \alpha_k t_{1,n+k}^{(ub)} + \sum_{k=B-n+1}^{\infty} \alpha_k (t_{1,B}^{(ub)} + b_{n+k-B}) \right) + \\
&\quad \frac{n\mu_{n+1}/(n+1)}{\lambda + \mu_{n+1}} t_{1,n-1}^{(ub)}, \quad n = 1, \dots, B, \\
t_{i,0}^{(ub)} &= \frac{1}{\lambda + \mu_i} + \frac{\lambda}{\lambda + \mu_i} \left(\sum_{k=1}^{\infty} \alpha_k t_{i,k}^{(ub)} + \sum_{k=B+1}^{\infty} \alpha_k (t_{i,B}^{(ub)} + b_{k-B}) \right) + \\
&\quad \frac{\mu_i}{\lambda + \mu_i} t_{i-1,0}^{(ub)}, \quad i = 2, \dots, \\
t_{i,n}^{(ub)} &= \frac{1}{\lambda + \mu_{i+n}} + \frac{\lambda}{\lambda + \mu_{i+n}} \left(\sum_{k=1}^{B-n} \alpha_k t_{i,n+k}^{(ub)} + \sum_{k=B-n+1}^{\infty} \alpha_k (t_{i,B}^{(ub)} + b_{n+k-B}) \right) + \\
&\quad \frac{n\mu_{i+n}/(i+n)}{\lambda + \mu_{i+n}} t_{i,n-1}^{(ub)} + \frac{i\mu_{i+n}/(i+n)}{\lambda + \mu_{i+n}} t_{i-1,n}^{(ub)}, \quad i = 2, \dots; n = 1, \dots, B. \quad (11)
\end{aligned}$$

where b_l is the average length of a busy period of an $M^X/M/1$ queue with arrival rate λ and service rate μc that is started by the arrival of i tasks. The value of b_l , $l = 1, \dots$ is ([GH76])

$$b_l = \frac{l}{\mu c} \left(1 + \frac{\lambda E[X]}{(\mu c - \lambda E[X])} \right).$$

Last, $t_{i,n}$, $n > B$ can be bounded by $t_{i,n}^{(ub)}$ given by

$$t_{i,n}^{(ub)} = t_{i,B}^{(ub)} + b_{n-B}, \quad n = B+1, \dots \quad (12)$$

These expressions can be substituted into the following relation to obtain an upper bound on $E[T|X=i]$,

$$\begin{aligned} E[T|X=i] &\leq \sum_{n=0}^{\infty} \pi_n t_{i,n}^{(ub)}, \quad i = 1, \dots, \\ &\leq \sum_{n=0}^B \pi_n t_{i,n}^{(ub)} + (1 - P[N \leq B]) t_{i,B}^{(ub)} \\ &\quad + b_1 \left(E[N] - B(1 - P[N \leq B]) - \sum_{n=1}^B n \pi_n \right), \quad i = 1, \dots. \end{aligned} \quad (13)$$

3.2.3 Approximate analysis of TS-PS

Let $T^{(lb)}$ and $T^{(ub)}$ denote the r.v.'s defined in the preceding sections that bound T from below and above. We use the following approximation for $E[T|X=i]$,

$$E[T^{(approx)}|X=i] = (E[T^{(lb)}|X=i] + E[T^{(ub)}|X=i])/2. \quad (14)$$

The accuracy of this approximation is high when the system load is low and/or when the parameter B takes a large value. We explore both of these effects in Table 1. Here we evaluate the upper and lower bounds on $E[T]$ for a system of 16 processors that process fork-join jobs containing exactly 16 tasks. The bounds are tabulated for different values of the processor utilization, $\rho = \lambda/\mu$ and for different values of B . We observe that sufficient accuracy is possible for processor utilizations up to .9 provided $B = 350$. In this case, the maximum error incurred by the approximation is 3.6% at $\rho = .9$ and less than .05% for $\rho \leq .8$. We shall use $B = 350$ throughout our studies.

ρ	B=50		B=100		B=200		B=350	
	lower	upper	lower	upper	lower	upper	lower	upper
.1	55.03	55.03	55.03	55.03	55.03	55.03	55.03	55.03
.2	56.68	56.41	56.68	56.68	56.68	56.68	56.68	56.68
.3	59.21	59.41	59.32	59.32	59.32	59.32	59.32	59.32
.4	62.95	63.91	63.47	63.47	63.47	63.47	63.47	63.47
.5	68.18	71.78	70.02	70.16	70.10	70.10	70.10	70.10
.6	75.17	87.01	80.60	81.70	81.17	81.17	81.17	81.17
.7	84.14	121.64	97.97	105.37	101.50	101.28	101.38	101.38
.8	95.20	225.95	126.68	175.40	142.94	148.56	144.88	145.04
.9	108.14	792.05	173.09	601.33	242.70	389.11	271.46	291.96

Table 1: Approximation Analysis

4 Comparison of Scheduling Policies

In this section we compare the performance of TS-PS, JS-PS, and FCFS. Specifically, we compare the mean job response time for different processor utilizations as we vary the number of processors and the job size. We also compare the performance of TS-PS and FCFS on a system that serves two classes of jobs: edit jobs and batch jobs. Edit jobs are assumed to consist of a single task whereas batch jobs consist of many tasks. Last, we consider the effects of partitioning the processors into two sets; one to serve edit jobs exclusively and the other to serve batch jobs exclusively. For this last study, we compare the performance of the partitioned system under TS-PS to one where the processors are available to all jobs under TS-PS.

4.1 Comparison of TS-PS, JS-PS, and FCFS

In this section we compare the TS-PS, JS-PS, and FCFS policies as a function of the processor utilization. In Figure 5 we plot the ratio of response times of TS-PS to JS-PS, and TS-PS to FCFS for two workloads as a function of the processor utilization, ρ . The workloads consist of jobs with a constant number of tasks that is equal to the number of processors, i.e., $X = 8, c = 8$ and $X = 16, c = 16$. The average task service time is taken to be $1/c$. From this figure we observe that FCFS provides uniformly better response over

the two PS policies for all processor utilizations. Furthermore, TS-PS gives lower response times than JS-PS for all processor utilizations less than 0.9. This is due to the fact that TS-PS takes advantage of the parallelism inherent in the fork-join job. We shall observe, however, TSPS is not always better than JSPS for very high utilizations in Section 4.2. The better performance exhibited by FCFS is due to the fact that TS-PS penalizes larger jobs, while no such penalty exists for FCFS (a more detailed discussion of this penalty phenomenon is given in the next section).

We also tested a workload consisting of two classes of jobs: edit jobs and batch jobs. Edit jobs consist of a single task and batch jobs consist of 16 tasks. Let f denote the fraction of jobs that are edit jobs. We considered three mixes, $f = .5, .95, .99$ operating on a system containing $c = 16$ processors. Figure 6 illustrates ratios of the mean job response time of TS-PS to FCFS as a function of the processor utilization ρ . We observe that the FCFS policy exhibits the best performance everywhere except when the utilization is high and the fraction of edit jobs is high ($f = .95, .99$). In this region TS-PS provides slightly lower response times.

This workload, ($f = .95, .99$), was chosen so as to increase the variability in the service job service times in an attempt to illustrate a setting in which TS-PS outperforms FCFS. It is surprising that the difference is so small. This is an indication that FCFS is a more robust policy on multiprocessors that execute parallel programs than it is in a system where jobs are executed serially.

From this figure we can also observe that TS-PS provides only slightly better service to edit jobs than FCFS, but significantly worse service to batch jobs.

4.2 Dependence on Number of Servers

In the last section we observed that TS-PS provides better performance than JS-PS for all of the examples. This appears to be at odds with observations that we noted in an earlier study [RTS87] on the performance of TS-PS and JS-PS in a *single processor* system. In a single processor system, JS-PS was shown to be uniformly better than TS-PS. This is due

Task Scheduling

VS

JSPS and FCFS

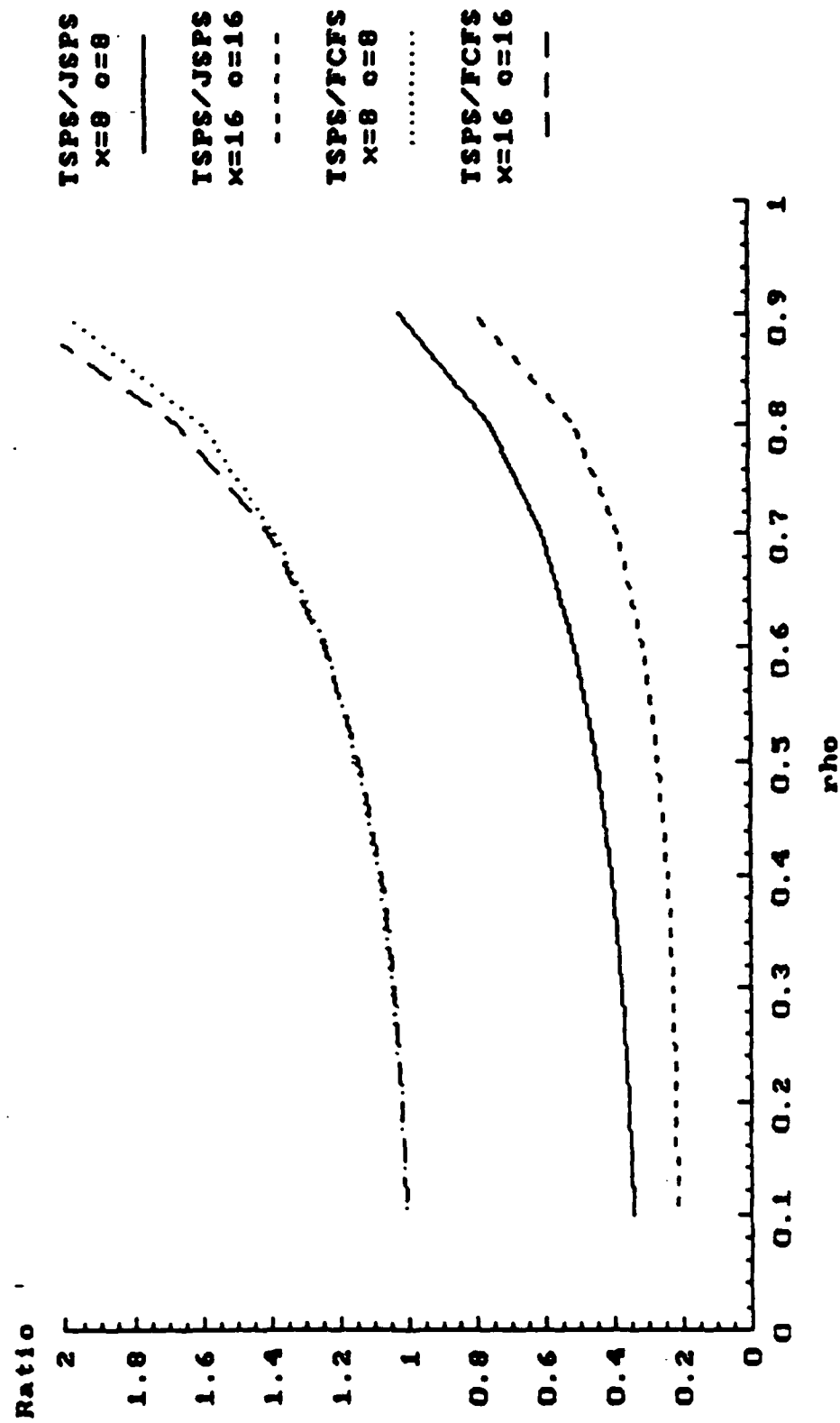
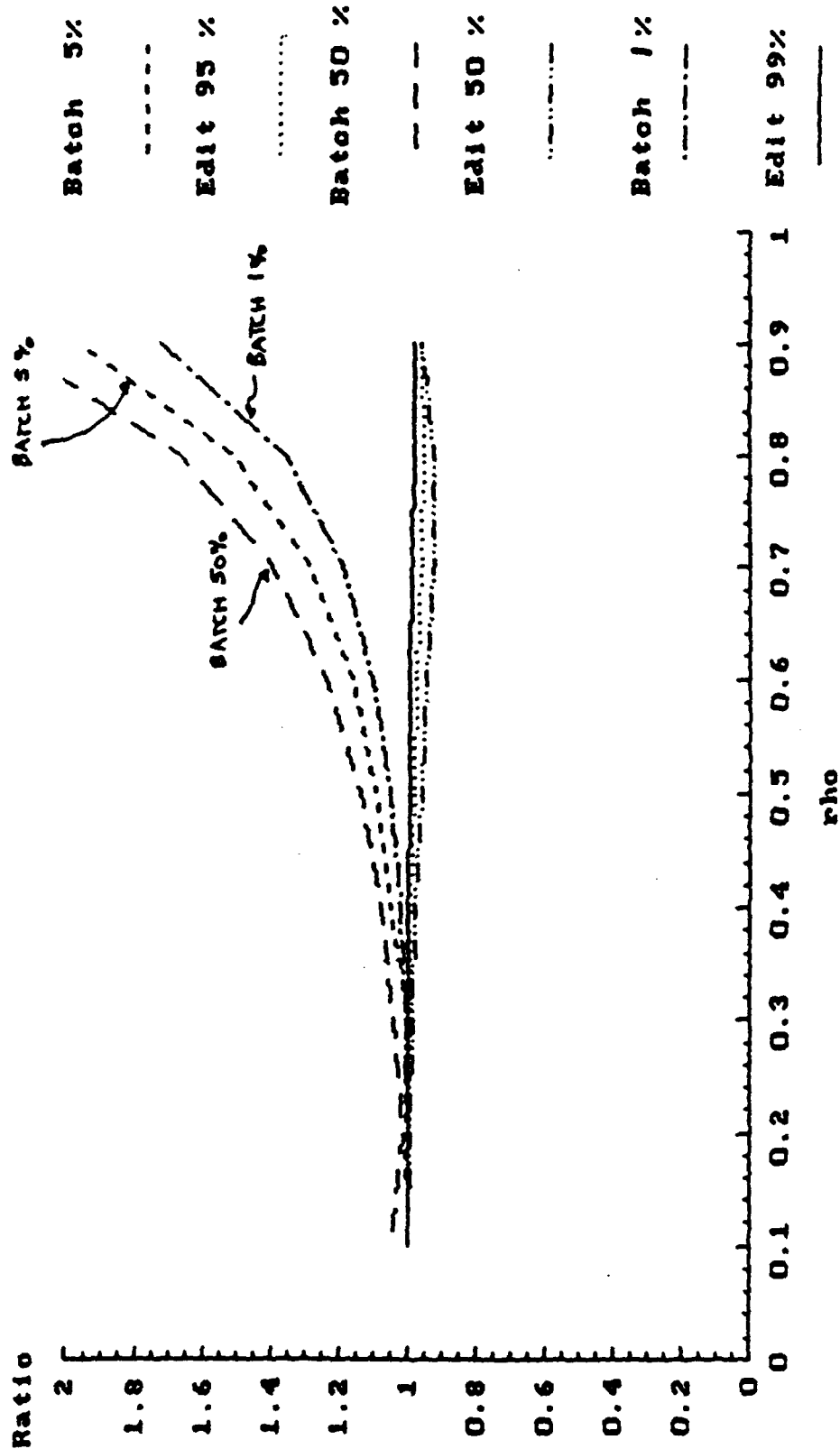


FIGURE 5

B=350

Batch and Edit Jobs with TSPS vs FCFS



$\sigma=16$
 $\kappa=16$
 $B=350$

FIGURE 6

to the fact that in such a system there is no possibility for parallelism and the following occurs. Assume that there are 2 jobs, one with 1 task and one with 9 tasks. Then TS-PS gives the job with 9 tasks, 9/10 of the processor, and the job with a single task only 1/10 of the processor. However, JS-PS would give each job 1/2 of the processor. So on a single processor, TS-PS penalizes jobs with a small number of tasks. In a multiprocessor, there exists sufficient possibilities for parallelism so that this anomaly found in a single processor for TS-PS does not exist.

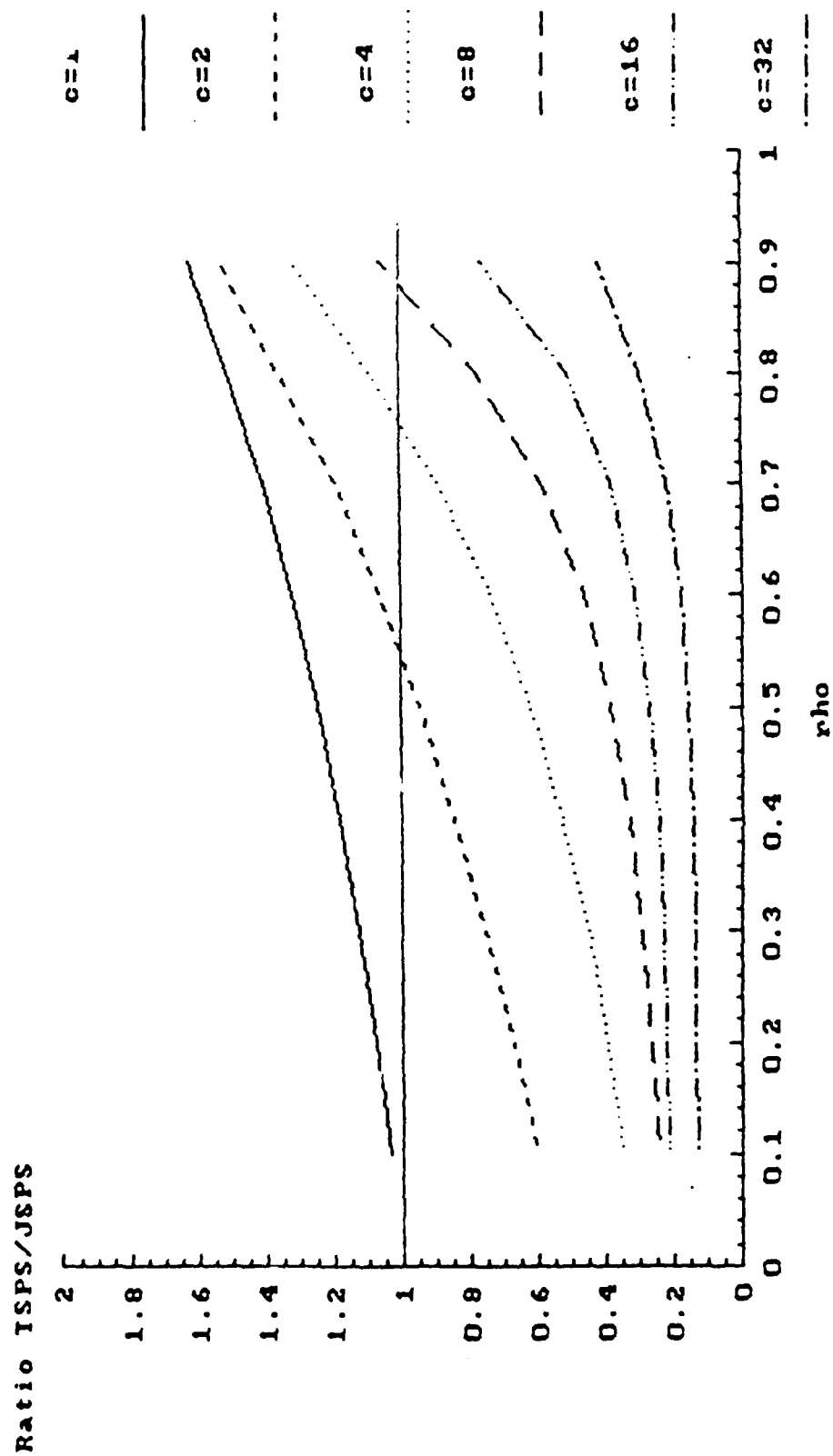
To study the effect of parallelism, we consider a workload of jobs consisting of 16 tasks and study the performance of TS-PS and JS-PS on systems with $c = 1, 2, 4, 8, 16, 32$ processors as a function of processor utilization. Figure 7 illustrates the results of this study plotting the response time ratios of TS-PS to JS-PS. We observe that TS-PS is always better than JS-PS in multiple processor systems when processor utilization is low. However, when the number of processors is small (< 8), there exists a utilization value, say ρ_0 such that system performance is better under JS-PS when $\rho > \rho_0$. This threshold is an increasing function of c the number of processors. This results because as the number of processors increases, the capability of sustaining parallel program execution under TS-PS increases.

4.3 Processor Partitioning

We now study the effect of dedicating a portion of the multiprocessor to each of the batch and edit classes. For edit jobs we assume that the computation time is small and equivalent to one task unit. Batch jobs are assumed to be large, consisting of fork-join tasks. The individual tasks from either class are assumed to have the same service requirements.

In order to examine the effect of statically dedicating a portion of the multiprocessor to each class, we compare the performance of a system composed of 16 servers where each server can run either class of job, to a partitioned system where some fraction of the processors are dedicated to each class. The combined system is composed of $c = 16$ servers. The partitioned system is composed of $c = 16$ servers such that K servers are dedicated to edit jobs and $c - K$ servers are dedicated to batch jobs. Our performance metric is the ratio of the response time of the partitioned system to that of the combined system.

TSPS/JSPS vs Number of Processors



$x=16$
 $B=350$

FIGURE 7

In this experiment the independent parameter is the combined system utilization. Our first experiment consists of an arrival of 50 percent edit jobs and 50 percent batch jobs. Note that this arrival pattern results in the total computation time of edit jobs to be 1/16 of batch jobs. The partitioned system is defined by K and the equivalent flow of jobs. We plot our results in (Figure 8).

We can observe several interesting phenomena from Figure 8. First, by dedicating only one server to the edit jobs, $K = 1$, both edit and batch jobs degrade. Thus, a poor partitioning choice negatively effects both classes of jobs. Second, improvements can be made in the edit jobs by allocating enough additional servers, $K = 2, 3$, to handle the computational load of edit jobs, but this is done at the expense of the batch jobs. This phenomena is especially striking at high utilizations.

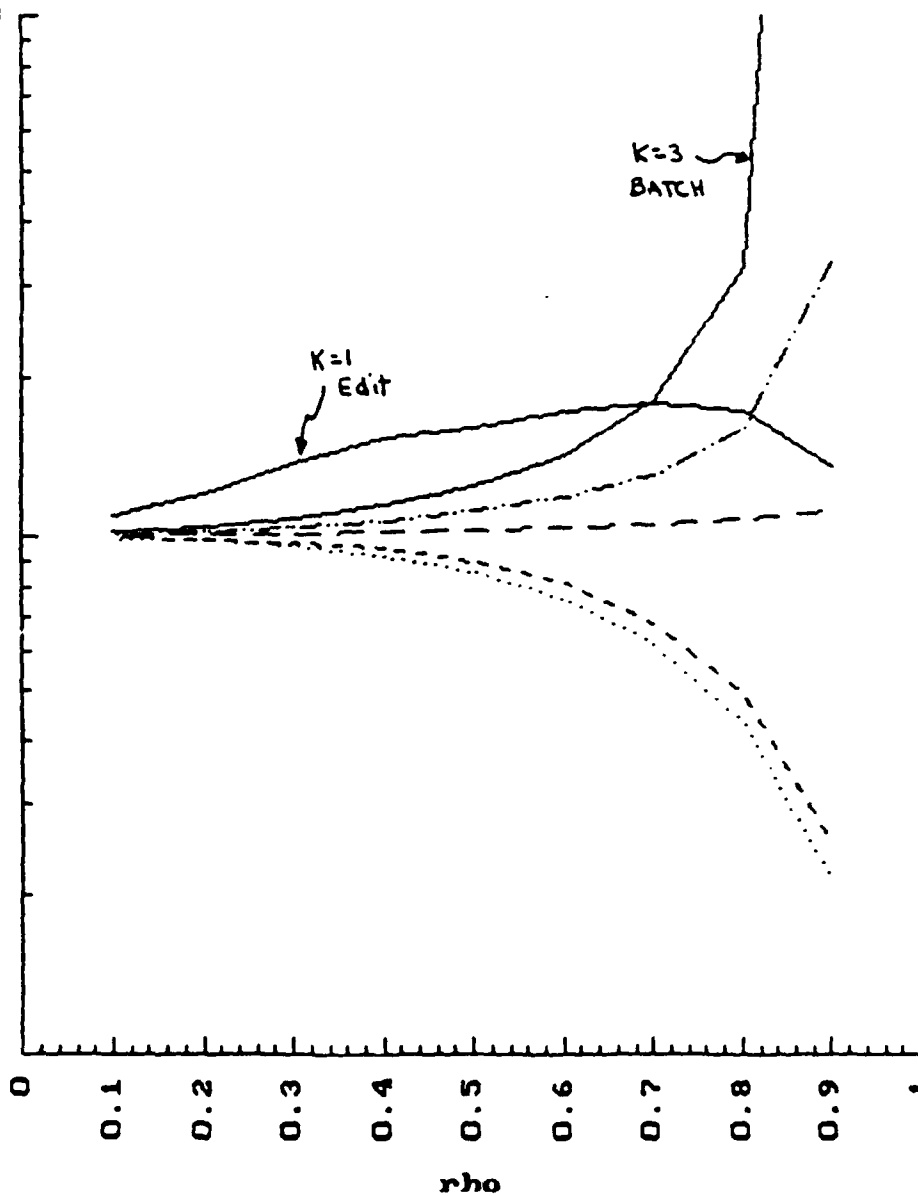
As the relative arrival rate between edit and batch jobs increases, as show in (Figure 9) where the proportion of edit jobs is 95 percent, we see that more servers must be dedicated to edit jobs before the mean response time is decreased. Note that in this case the total computation time required by edit jobs is greater than needed by batch jobs. The result is that 9 of the 16 processors are required to reduce the edit job response times (see (Figure 9)). There are regions in the figure in which the performance of both jobs classes decrease, however, we observe no region in which both classes improve performance. This phenomena has also been reported in [NTT87] for FCFS scheduling.

Figure 10 reports the results when batch jobs are composed of 4 tasks and the workload contains 50% batch and 50% edit jobs. The results in this figure are similar to the 95% edit jobs and 5% batch job tests shown in the previous figure. The reason for this is that when batch jobs are fairly small, $x = 4$, and there are 50% edit jobs and 50% batch jobs in the workload, then the total computational requirements of edit jobs is high (as in the 95% test) for a given utilization. Therefore, edit jobs will saturate a small number of processors. Notice that only when the number of processors dedicated to editing reaches 4, does editing perform well.

Partitioning at 50 %

Ratio Partition/Not

10



K=1 Edit

K=2 Edit

K=3 Edit

K=1 Batch

K=2 Batch

K=3 Batch

FIGURE 8

x=16
B=350

Partitioning at 95 %

Ratio Partition/Not

10

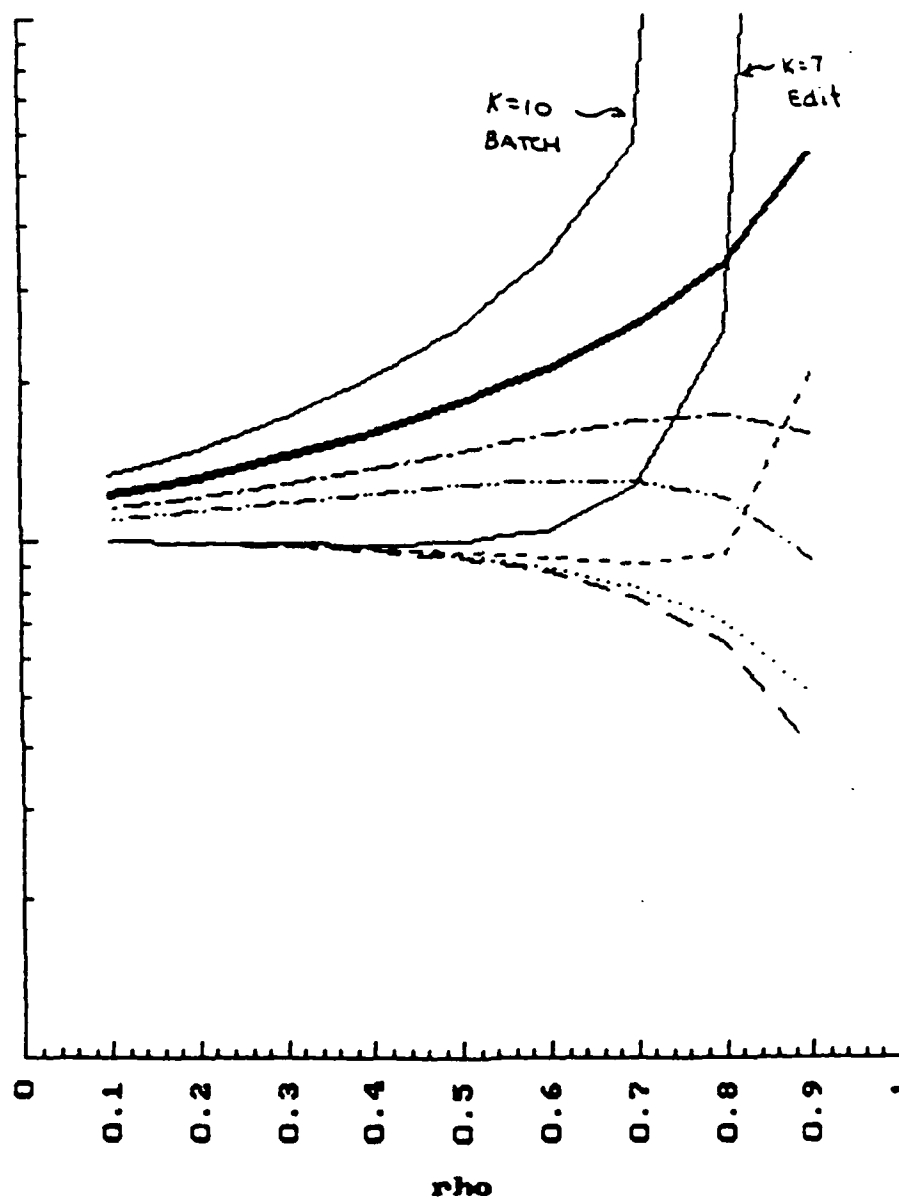
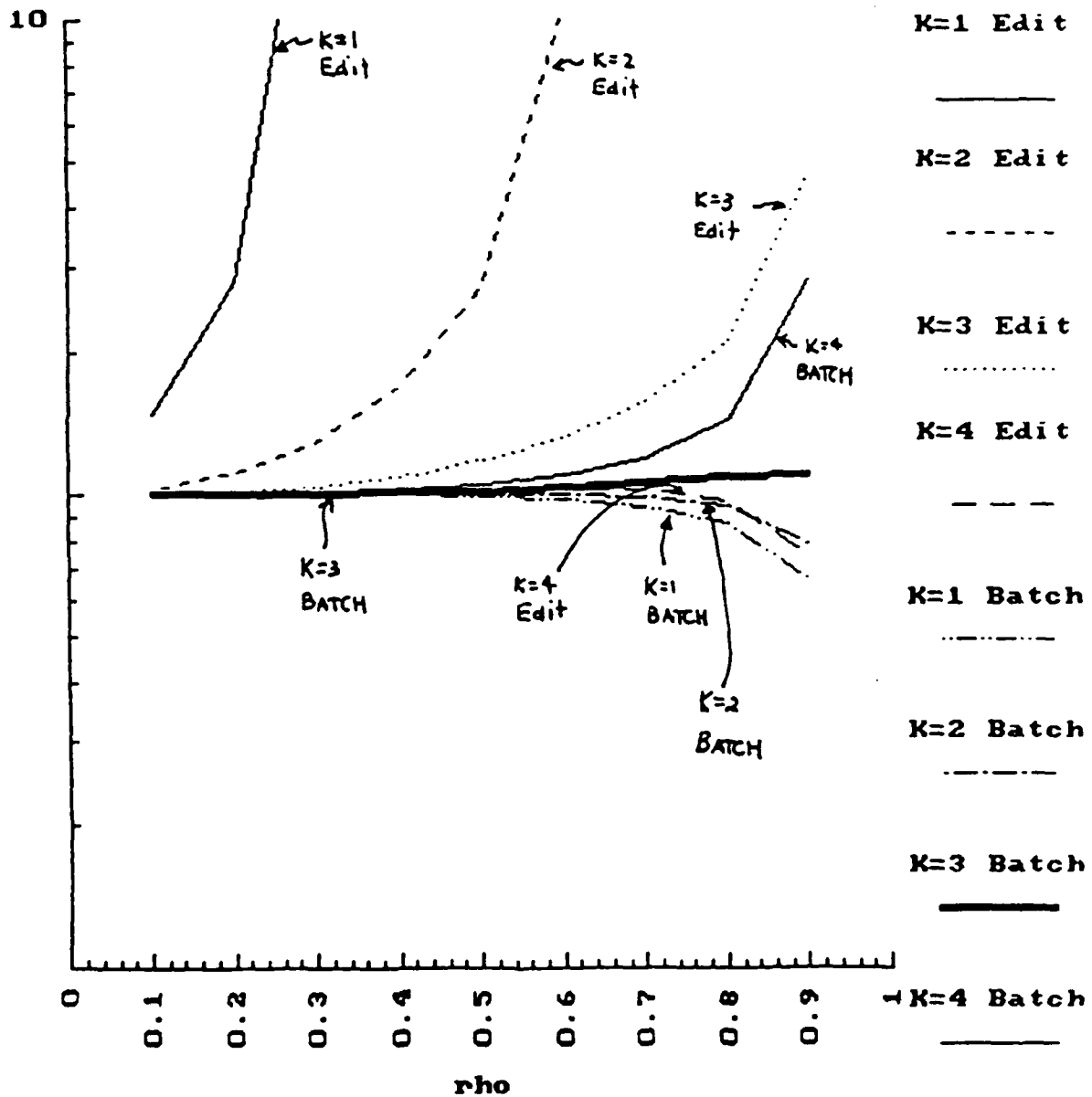


FIGURE 9

$x=16$
 $B=350$

Partitioning at 50 %

Ratio Partition/Not



x=4
B=350

FIGURE 10

5 Summary

We have analyzed fork-join programs as a $M^X/M/c$ queueing system. We have obtained an expression for the mean response time of a fork-join task under processor-sharing. Since our expression is not in closed form, but given as a set of recurrent equations, we have obtained expressions for both lower and upper bounds. Our bounds become tight as the number of states increase.

We have compared three scheduling approaches: TS-PS, JS-PS and FCFS. We have observed that in general FCFS out performs both TS-PS and JS-PS. Likewise, we have observed that TS-PS performs better than JS-PS unless that number of servers is small compared to the number of tasks.

We have considered the interesting problem of partitioning the system into two subsystems. Each subsystem is dedicated to one of two job classes: edit jobs and batch jobs. We determined several interesting results. When half the jobs are edit jobs and one server is dedicated for edit jobs exclusively, both classes experience an increase in response time. Improvements in edit jobs always cause a reduction in the performance of batch jobs in the partitioned system. This suggests that a parallel system should have a controllable boundary for processor partitioning.

References

- [All78] Arnold Allen. *Probability, Statistics and Queueing Theory*. Academic Press, New York, New York, 1978.
- [BM85] F. Baccelli and A. Makowski. Simple computable bounds for the fork-join queue. *Proc. Conf. Inform. Sci. Systems*, 1985.
- [BMT87] F. Baccelli, W. Massey, and D. Towsley. Acyclic fork-join queueing networks. *submitted to JACM*, 1987.
- [CT83] Chaudhry and Templeton. *First Course in Bulk Queues*. J. Wiley, New York,

New York, 1983.

- [GH76] Gross and Harris. *Introduction to Queueing Theory*. J. Wiley, New York, New York, 1976.
- [Han75] P. Brinch Hansen. The programming language concurrent pascal. *IEEE Transaction on Software Engineering.*, 1, 1975.
- [Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, London, 1985.
- [Kle64] Leonard Kleinrock. Analysis of a time-shared processor. *Naval Research Logistics Quarterly*, 11, 1964.
- [Kle76] Leonard Kleinrock. *Queueing Systems Volume II: Computer Applications*. J. Wiley, New York, New York, 1976.
- [NT85] R. Nelson and A.N. Tantawi. Approximate analysis of fork/join synchronization in parallel queues. *IBM Report RC11481*, 1985. to appear in *IEEE Transactions on Computers*.
- [NTT87] R. Nelson, D. Towsley, and A. Tantawi. Performance analysis of parallel processing systems. to appear in *IEEE Trans. on Software Engineering*.
- [Ost86] Anita Osterhaug. *Guide to Parallel Programming*. Sequent Computer Systems, Inc, Beaverton, Oregon, 1986.
- [Pyl81] I.C. Pyle. *The Ada Programming Language*. Prentice-Hall International, London, 1981.
- [RTS87] C. Gary Rommel, D. Towsley, and J. Stankovic. An analysis of fork-join jobs using processor-sharing. *Submitted to Operations Research*, 1987.
- [Yao85] D. D. Yao. Some results for queues $M^X/M/c$ and $GI^X/G/c$. *Operations Research Letters*, 4, 1985.

APPENDIX D

ACYCLIC FORK-JOIN QUEUEING NETWORKS

François BACCELLI ¹

William A. MASSEY ²

Don TOWSLEY ³

COINS Technical Report 87-40

ABSTRACT

In this paper we study the class of acyclic fork-join queueing networks that arise in various applications, including parallel processing and flexible manufacturing. In such queueing networks, a fork describes the simultaneous creation of several new customers which are sent to different queues. The corresponding join occurs when the services of all these new customers are completed. We derive the evolution equations that govern the behavior of such networks. From this, we obtain the stability conditions and develop upper and lower bounds on the network response times. These bounds are based on stochastic ordering principles and apply under general workload assumptions.

¹ INRIA B.P. 105 - 78153 Le Chesnay (France)

² ATT BELL LABORATORIES, Murray Hill - NJ - 07940 (U.S.A.)

³ University of MASSACHUSETTS Amherst - MA - 01003 (U.S.A.) Professor Towsley's work was supported by RADC under contract F302602-81-C-0169.

In this paper we study the class of *acyclic fork-join queueing networks*, in short "AFJQN's", that arise in the performance analysis of parallel processing applications and flexible manufacturing systems. We obtain the stability conditions and develop upper and lower bounds on the performance of this class of networks under very general workload assumptions.

AFJQN's arise very naturally in parallel processing applications. Many parallel programs are decomposed into tasks, each of which can execute on a separate processor. The division of the parallel program into tasks can be described by a directed graph where the nodes correspond to tasks and the directed edges represent the precedence relations between the tasks. In many cases, the underlying graph is acyclic and the program is implemented with the use of *fork* and *join* constructs. Briefly, a fork exists at each point in a parallel program that one or more tasks can be initiated simultaneously. A join occurs whenever a task is allowed to begin execution following the completion of one or more other tasks. Forks and joins reflect themselves in the underlying computation graph in the following manner. A task that has one or more outgoing edges corresponds to a fork. A task with one or more incoming edges corresponds to a join. These are exemplified by the *parbegin* and *parend* constructs that are available in parallel programming languages such as Concurrent Pascal [Br 75], Concurrent Sequential Processes (CSP) [Ho 78], and Ada [Py 81].

Consider a multiple processor system where each task in a specific program is mapped onto a separate processor. The execution of a single program request can be described as follows: (i) Upon completion of a marked task, tokens associated with the program are routed to each processor handling the tasks that follow the marked task in the underlying computation graph; (ii) Once a processor has received tokens from all tasks that precede a marked task in the computation graph, this processor is allowed to execute it. Let this system be required to service a stream of requests corresponding to different instances of that program and assume each processor executes its tasks in the order defined by the program arrival dates. We have described, in brief, an AFJQN. Figure 1a illustrates a hypothetical parallel program using forks and joins and Figure 1b illustrates the associated fork-join queueing network.

AFJQN's also arise naturally in the context of flexible manufacturing systems. In production lines, objects are built by assembling multiple parts together. The successive assembly steps are described by an acyclic graph where the nodes correspond to assembly operations and the edges to precedence constraints between these operations. Here, a join occurs whenever all the parts to be produced by the operations that precede a marked operation have to be available in order to begin assembling. A fork occurs at points where several assembly operations are initiated simultaneously (for instance at points where the production of some part is followed in the underlying graph by several assembly operations to be done on this same part). Assume each assembly operation is allocated to a specific machine. We have another instance of AFJQN when identifying assembly machines with the servers of the queueing network and the parts with its customers.

Apart from the subclass of Jackson series networks, the type of queueing networks we consider here remain basically unsolved. It can be shown that the "synchronisations" induced by the forks and the joins destroy all nice properties like insensitivity or product form, so that every problem becomes computationally hard. Initially, most attention focussed on fork-join networks consisting of B queues in parallel. In this case, exact solutions have been provided for $B = 2$ in [FH 84] and [Ba 85]. Approximate solutions and bounds have been provided for arbitrary values of B in [BM 85], [NT 85], [TY 86] and [BMS 87]. Conditions for stability have been presented for arbitrary values of B in [BM 85] and [Si 87]. Last, models have been developed for programs exhibiting parallel fork-join structures that are executed on multiple processors serving a single queue in [KW 85] and [NTT 87]. Series-parallel Fork Join queueing networks have been introduced in [BM 85], where stability condition and bounds were derived.

Several classes of stochastic ordering principles have been considered in the queueing literature

(see [St 84] for a comprehensive treatment of the issue). It was shown for instance, that an increased input (resp. decreased output) intensity leads to higher (resp. reduced) moments of the waiting or response times for wide classes of queueing systems (see [Wh 81]). Another type of ordering comes from the idea that an increased variability of either the input or the service statistics should also lead to higher waiting or response times. This has been discussed by several authors in the context of isolated queues (see [St 84], [Ha 84], [Wh 84], [BM 85b]). The latter ordering principle was used in [BM 85] (resp. [BM 85b]) to compare the moments of the delays experienced by customers traversing parallel (resp. series-parallel) fork-join queueing networks to the related moments of product form networks. Both upper and lower bounds were derived using this principle.

A third type of ordering arises when a set of random variables (RV's) are associated. In this case the statistics of the maximum over these RV's are bounded by the maximum of the marginals of these RV's. This approach was used in [NT 85] and [BMS 87] to develop upper bounds on the moments of the delays experienced by customers traversing a parallel fork-join network.

The aim of this paper is to extend the scope of these ordering and bounding techniques to the class of arbitrary AFJQN's which are rigorously defined in Section 2. The equations governing the behavior of these networks are provided in Section 3. This section also contains necessary and sufficient conditions for the stability of these networks under fairly general statistical assumptions. This stability result is based on an extension of Loynes' method [Lo 62] to this class of queueing networks. Bounds based on convex ordering are described in Section 4. Although these arguments yield upper and lower bounds on the moments of customer delays, tighter upper bounds are obtained in Section 5 using stochastic ordering properties of associated RV's. Sections 6 and 7 are devoted to the derivation of bounds of practical interest based on convex ordering and associated RV's respectively. All these bounds exhibit the same stability condition as the initial queueing system.

2 Notation and definitions

We are concerned with the delays that customers experience when they traverse an Acyclic Fork-Join Queueing Network β . Here β is represented by an acyclic graph $G = (V, E)$ where V is a set of B FIFO queues labeled $i = 1, \dots, B$ and E is a set of links such that $(i, j) \in E$ implies $j > i$ (such an ordering is always possible in an acyclic graph).

Define the set of immediate predecessors of queue i , $p(i)$, to be the set of queues that have a direct link to queue i

$$p(i) = \{ j \in (1, B) \mid (j, i) \in E \} \quad (2.1)$$

and the set of immediate successors of queue i , $s(i)$, to be the set of queues to which i has a direct link

$$s(i) = \{ j \in (1, B) \mid (i, j) \in E \}. \quad (2.2)$$

Define the set of predecessors of queue i , $\pi(i)$, to be the set of queues that have a (possibly) indirect link to queue i :

$$\pi(i) = \{i\} \cup p(i) \cup p^2(i) \dots \cup p^{B-1}(i), \quad (2.3)$$

where $p(X)$ denotes the set of immediate predecessors of the queue of X , a subset of $(1, \dots, B)$ and $p^n(X)$ denotes $p(p(\dots p(X) \dots))$.

We also denote as $s(0)$ the set of queues with no incoming links and as $p(B+1)$ the set of queues with no outgoing link. It will be assumed that the numbering of queues is such that

$$s(0) = (1, \dots, B_0), \quad B_0 \leq B \quad (2.4)$$

and

$$p(B+1) = (B_1, \dots, B), \quad B_1 \leq B. \quad (2.5)$$

Observe that $p(i) = \emptyset$ if $i \in s(0)$ and $s(i) = \emptyset$ if $i \in p(B+1)$.

We associate with queue j , $1 \leq j \leq B$, a sequence $\{\sigma_n^j\}_0^\infty$, where $\sigma_n^j \in R^+$ represents the service requirement of the n -th customer to enter this queue. Queue j behaves as a single server FIFO queue so that an arrival pattern $\{a_n^j\}_0^\infty$ to this queue together with the sequence $\{\sigma_n^j\}_0^\infty$ fully determine the sequence of service completion dates (using the Lindley-Loynes equations).

Definition 0

An acyclic queueing network, as defined above, is an it Acyclic Fork-Join Queueing Network if it obeys the following rules:

- (i) There is a single exogeneous arrival stream with pattern $a_0 = 0 < a_1 < \dots < a_n < \dots \in R^+$. The n -th customer arrival to queue i , $1 \leq i \leq B_0$, coincides with the n -th date of this exogeneous stream. As stated above, this fully determines the sequence of service completion in the queues $1 \leq j \leq B_0$.
- (ii) A service completion in queue i does not systematically trigger an arrival to a queue of $s(i)$. The arrivals to queue j , $j > B_0$, are precisely generated as follows: assume the sequence of service completions is known for all queues $1 \leq i \leq j$, where $B_0 < j \leq B$. The n -th customer arrival to queue j , a_n^j , coincides with the latest of the n -th service completions in the queues of $p(j)$. Due to the acyclic structure of V , this successively defines the arrival patterns in queue $B_0+1, B_0+2, \dots, B$.
- (iii) There is a single output stream out of this network. Its n -th event coincides with the latest of the n -th service completions in the queues $B_1, B_1+1, \dots, B$.

As it will be seen in the next section, these three rules fully determine the evolution of the queueing network.

Some of the bounds discussed in the application sections 6 and 7 will only apply to certain subclasses of AFJQN's, namely parallel and series networks. An AFJQN β is said to be a parallel one with $K \geq 2$ subnetworks with respective underlying graphs $G_k = (V_k, E_k)$, $1 \leq k \leq K$, if its graph G is decomposable into the K disconnected subgraphs $G_1, \dots, G_K$. An AFJQN β is said to be a series one with $K \geq 2$ subnetworks with respective underlying graphs $G_k = (V_k, E_k)$, $1 \leq k \leq K$, if its graph G is connected and exhibits the following property: There are $K-1$ vertices $1 < i_1 < i_2 < \dots < i_{K-1} < B$ such that there are no direct links between the vertices of $(1, \dots, i_k - 1)$ and those of $(i_k + 1, \dots, B)$ for all $1 \leq k \leq K-1$. The graph G_k is defined as the restriction of G to the vertices $(i_{k-1} + 1, \dots, i_k)$, where $i_0 = 0$ and $i_K = B$. Figure 2 illustrates a parallel AFJQN and a series AFJQN.

3 Evolution equations and steady state

For $n \geq 0$ and $1 \leq i \leq B$, let $\sigma_n^i \in R^+$ be the service requirement of the n -th customer to be served in queue i (there is hence a zero-th customer !) and r_n be the n -th interarrival of the exogeneous stream : $r_n = a_{n+1} - a_n$, $n \geq 0$. Similarly, let $d_n^i \in R^+$ be the delay between the

n -th exogeneous arrival date and the beginning of the n -th service in queue i and R_n be the n -th network response time defined as the delay between the n -th exogeneous arrival and the n -th date of the global departure process.

Lemma 1

Assume the network is empty at time 0. Then, for $n \geq 0$,

$$d_{n+1}^j = \max(\max_{i \in p(j)} (d_{n+1}^i + \sigma_{n+1}^i), d_n^j + \sigma_n^j - r_n), \quad (3.1)$$

where the maximum over an empty set is zero by convention and

$$d_0^j = \max_{i \in p(j)} (d_0^i + \sigma_0^i). \quad (3.2)$$

The n -th network response time, R_n , is given by

$$R_n = \max_{i \in p(B+1)} (d_n^i + \sigma_n^i). \quad (3.3)$$

Proof

The boundary condition (3.2) follows from the assumption on the initial condition and from rules (i) and (ii) that define AFJQN's. For j such that $1 \leq j \leq B_0$, the inputs in queue j coincide with the exogeneous arrivals and d_n^j is thus the n -th waiting time in a FIFO queues with interarrival sequence $\{r_n\}_0^\infty$ and service requirements $\{\sigma_n^j\}_0^\infty$. We have hence the classical Lindley-Loynes equations

$$d_{n+1}^j = \max(0, d_n^j + \sigma_n^j - r_n), \quad n \geq 0, \quad 1 \leq j \leq B_0, \quad (3.4)$$

which is exactly equation (3.1) since $p(j) = \emptyset$.

Let j be such that $p(j) \neq \emptyset$, and assume that $\{d_n^i\}_0^\infty$ is known for all $i \in p(j)$ so that the n -th service completion in queue $i \in p(j)$ takes place at $d_n^i + \sigma_n^i$. According to rule (ii), the $n+1$ -st arrival to queue j takes place at

$$a_{n+1} + \max_{i \in p(j)} (d_{n+1}^i + \sigma_{n+1}^i). \quad (3.5)$$

Since the server of queue j becomes available for serving the $n+1$ -st customer at time

$$a_n + d_n^j + \sigma_n^j, \quad (3.6)$$

it follows that d_{n+1}^j is equal to the expression in the r.h.s of equation (3.1). Equations (3.1) and (3.2) are the basic evolution equations of the network, from which the transient bounds of section 4 and 5 will be derived.

The remainder of this section is devoted to the construction of the stationary regime of such networks. This construction will be essential in the continuation of the transient bounds to steady state bounds. Consider the following set of assumptions.

H_0 The sequence $\{r_n, \sigma_n^j, 1 \leq j \leq B_0\}_0^\infty$ on $(R^+)^{B+1}$ forms a stationary and ergodic sequence of integrable RV's on the probability space (Ω, F, P) .

Theorem 2

Let j be fixed $1 \leq j \leq B$. Assume H_0 holds and that for all $i \in p(j)$, d_n^j converges weakly to a finite and integrable RV d_∞^j when n goes to ∞ . Assume in addition that

$$E[\sigma_n^i] < E[r_n] \quad \forall i \in \pi(j). \quad (3.7)$$

Then the distribution functions of the RV's d_n^j converge weakly to a finite RV d_∞^j when n goes to ∞ . More precisely, under these conditions, there exists a sequence of RV's δ_n^j , $n \geq 0$ on (Ω, F, P) such that δ_n^j and d_n^j are equivalent in law for all $n \geq 0$ ($d_n^j =_{st} \delta_n^j$) and δ_n^j increases pathwise to a finite limit δ_∞^j when n goes to ∞ .

The proof is presented in Appendix 1.

4 Bounds based on convex ordering

We are now in position to prove the stochastic ordering result. Consider a network β in C and assume that all the RV's $\{a_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty$, $1 \leq j \leq B$ are defined on the probability space (Ω, F, P) and are all integrable.

Let now $\{\bar{a}_n\}_0^\infty$ and $\{\bar{\sigma}_n^j\}_0^\infty$, $1 \leq j \leq B$, be a set of "smoother" arrival and service processes on (Ω, F, P) in the sense that there exists a sub σ -algebra say G of F such that for all $n \geq 0$,

$$\bar{r}_n = \bar{a}_{n+1} - \bar{a}_n = E[r_n | G] \quad \text{a.s.} \quad (4.1)$$

and for all j in B ,

$$\bar{\sigma}_n^j = E[\sigma_n^j | G] \quad \text{a.s.} \quad (4.2)$$

These new variables are smoother than the original ones in the following sense : let $\bar{b}$ and b be two non-negative and integrable RV's on (Ω, F, P) such that

$$\bar{b} = E[b | G] \quad \text{a.s.} \quad (4.3)$$

Owing to Jensen's theorem for conditional expectations, (4.3) entails

$$f(\bar{b}) = f(E[b | G]) \leq E[f(b | G)], \quad \text{a.s.} \quad (4.4)$$

for all convex nondecreasing function $f: R^+ \rightarrow R^+$ such that the expectations exist. This in turn entails that for all such f

$$E[f(\bar{b})] \leq E[f(b)] \quad (4.5)$$

which can be rephrased in terms of the convex increasing stochastic ordering of Stoyan [St 84] as follows :

$$\bar{b} \leq_{ci} b. \quad (4.6)$$

Observe that b and $\bar{b}$ have hence the same first moment and higher moments are always larger for b than for $\bar{b}$.

Let $\tilde{d}_n^j$ be the delay variable obtained with the new arrival and service pattern $\{\tilde{r}_n\}_0^\infty \{\tilde{\sigma}_n^j\}_0^\infty$ $j = 1, B$. The main result of this section is the following theorem :

Theorem 3

For all $n \geq 0$ and $1 \leq j \leq B$,

$$d_n^j \text{ is integrable and } \tilde{d}_n^j \leq E[d_n^j|G] \text{ a.s.} \quad (4.7)$$

Proof

Basis step

Consider the case $n = 0$ we shall show that (4.7) holds for all $j = 1, B$ by induction on j

Basis step

Consider all j such that $p(j) = \emptyset$, equation (3.2) shows that

$$d_0^j = \tilde{d}_0^j = 0, \quad (4.8)$$

so that (4.7) holds.

Inductive step

Assume that the hypothesis is true for all $i, 1 \leq i < j$ where $B_0 \leq j \leq B$. It is plain from (3.2) that d_0^j is then integrable. Applying Jensen's inequality for conditional expectations to (3.2) yields

$$E[d_0^j|G] \geq \max_{i \in p(j)} (E[d_0^i|G] + \tilde{\sigma}_0^i), \quad (4.9)$$

so that if the predecessors of j satisfy property (4.7), so does queue j since (4.9) implies then :

$$E[d_0^j|G] \geq \max_{i \in p(j)} (\tilde{d}_0^i + \tilde{\sigma}_0^i). \quad (4.10)$$

This completes the proof of the basis step.

Inductive step Assume now that the property (4.7) was established for all queues up to rank n . We now show that the property holds also for $n + 1$. This is done by induction on $1 \leq j \leq B$.

Basis step

Consider all j such that $p(j) = \emptyset$. (3.1)

$$d_{n+1}^j = \max(d_n^j + \sigma_n^j - r_n, 0), \quad (4.11)$$

so that d_{n+1}^j is also integrable. Jensen's inequality together with (4.1) and (4.2) imply that

$$E[d_{n+1}^j|G] \geq \max(E[d_n^j|G] + \tilde{\sigma}_n^j - \tilde{r}_n, 0) \quad (4.12)$$

Hence, since (4.7) is satisfied for rank n , we get from (4.12) that

$$E[d_{n+1}^j|G] \geq \max(\tilde{d}_n^j + \tilde{\sigma}_n^j - \tilde{r}_n, 0) = \tilde{d}_{n+1}^j \text{ a.s., } 1 \leq j \leq B_0. \quad (4.13)$$

so that the property is also true for rank $n + 1$.

Inductive step

Assume (4.7) holds for all $i, 1 \leq i < j$, where $B_0 < j \leq B$ we now show that the property holds for j . It follows from (3.1) that d_{n+1}^j is also integrable. Applying Jensen's inequality to (3.1) and using (4.1) and (4.2), we get

$$E[d_{n+1}^j|G] \geq \max(\max_{i \in p(j)} (E[d_{n+1}^i|G] + \bar{\sigma}_{n+1}^i), E[d_n^j|G] + \bar{\sigma}_n^j - \bar{r}_n). \quad (4.14)$$

Using now the ordering property for rank 'n, we get

$$E[d_{n+1}^j|G] \geq \max(\max_{i \in p(j)} (E[d_{n+1}^i|G] + \bar{\sigma}_{n+1}^i), \bar{d}_n^j + \bar{\sigma}_n^j - \bar{r}_n) \text{ a.s.} \quad (4.15)$$

Since the property is satisfied for the predecessors of j , we get that it is then satisfied by queue j too since (4.15) entails that

$$E[d_{n+1}^j|G] \geq \max(\max_{i \in p(j)} (\bar{d}_{n+1}^i + \bar{\sigma}_{n+1}^i), \bar{d}_n^j + \bar{\sigma}_n^j - \bar{r}_n) = \bar{d}_{n+1}^j \text{ a.s.} \quad (4.16)$$

This complete the induction step on j .

This completes the induction step on n and proves the lemma.

Remark

Observe that theorem 3 also holds under the weaker assumptions.

$$\bar{r}_n \geq E[r_n|G], \quad n \geq 0 \quad (4.17)$$

and

$$\bar{\sigma}_n^j \leq E[\sigma_n^j|G], \quad n \geq 0, \quad j = 1, B. \quad (4.18)$$

Corollary 4

For all $n \geq 0$ and $i = 1, B$.

$$d_n^j \geq \bar{d}_n^j. \quad (4.19)$$

Proof

Due to Jensen's inequality

$$E[f(d_n^j)|G] \geq f(E[d_n^j|G]), \quad (4.20)$$

so that using equation (4.7) and the increasingness of f ,

$$E[f(d_n^j)|G] \geq f(\bar{d}_n^j). \quad (4.21)$$

Equation (4.19) follows now directly from (4.21).

The next corollary shows that if the network achieves steady state in the sense of Theorem 2, the transient bounds of corollary 4 extend to steady state.

Corollary 5

Assume that both $\{\tau_n, \sigma_n^j, j = 1, B\}_0^\infty$ and $\{\bar{\tau}_n, \bar{\sigma}_n^j, j = 1, B\}_0^\infty$ satisfy the condition H_0 and that d_n^j and $\bar{d}_n^j$ converge weakly to finite RV's d_∞^j and $\bar{d}_\infty^j$ respectively. Then

$$\bar{d}_\infty^j \leq_{st} d_\infty^j. \quad (4.22)$$

Proof

Assume $f(\delta_\infty^j)$ is integrable. Since $\delta_n^j \leq \delta_\infty^j$, and $d_n^j =_{st} \delta_n^j$, $\bar{d}_n^j =_{st} \bar{\sigma}_n^j$, it is then easy to prove that $f(d_n^j)$ and $f(\bar{d}_n^j)$ are both integrable for all $n \geq 0$ so that corollary 4 entails

$$E[f(\delta_n^j)] = E[f(d_n^j)] \leq E[f(\bar{d}_n^j)] = E[f(\bar{\sigma}_n^j)]. \quad (4.23)$$

Letting n go to infinity in the inequality

$$E[f(\delta_n^j)] \leq E[f(\bar{\sigma}_n^j)] \quad (4.24)$$

yields the desired result using the bounded convergence theorem.

Remark 1

Consider a two queue series network and denote as W_n^j , $n \geq 0$, $j = 1, 2$ the waiting time of the n -th customer to enter queue j . We have the following inductions for the RV's W_n^j initialized by the condition $W_0^j = 0$:

$$W_{n+1}^1 = \max(W_n^1 + \sigma_n^1 + a_n - a_{n+1}, 0), \quad n \geq 0 \quad (4.25)$$

and

$$W_{n+1}^2 = \max(W_n^2 + \sigma_n^2 + d_n - d_{n+1}, 0), \quad n \geq 0, \quad (4.26)$$

where the RV's $\{d_n\}_0^\infty$ are the departure epochs from queue 1:

$$d_{n+1} - d_n = \sigma_{n+1}^1 + \max(a_{n+1} - a_n - \sigma_n^1 - W_n^1, 0). \quad (4.27)$$

Observe that due to the decreasingness of the r.h.s of (4.27), considered as a function of W_n^1 , we cannot derive from this any simple comparison result between $(d_{n+1} - d_n)$ and $(\bar{d}_{n+1} - \bar{d}_n)$ when using Jensen's inequality as before.

We prove in Appendix 2 that there is actually no such general ordering result by considering two simple stationary queueing systems where an increased variability of the sequence (r_n, σ_n^1) has the following respective effects:

- It increases the variability of the interdeparture distribution for the first one,
- It decreases it for the second one.

This strongly suggests that the stochastic ordering result of this section, which apply to the total delays d_n^j , does not extend to the individual waiting times W_n^j .

5 Bounds based on association

5.1 Association of the delays

Before entering the core of this section, we introduce some terminology that will be useful in the forthcoming analysis and review the properties of stochastic ordering and associated RV's that will be useful to us.

Definition 6 ([BP 75])

Real valued RV's $a_1, \dots, a_n$ are said to be associated if

$$\text{cov}[h(a_1, \dots, a_n), g(a_1, \dots, a_n)] \geq 0 \quad (5.1.1)$$

for all pairs of increasing functions $h, g : R^n \rightarrow R$. Association of RV's entails the following properties :

1. Any subset of associated RV's are associated,
2. Increasing functions of associated RV's are associated,
3. Independent RV's are associated,
4. If two sets of associated RV's are independent of one another, then their union forms a set of associated RV's,
5. If $a_1, \dots, a_n$ are associated RV's, then

$$P\left[\max_{1 \leq i \leq n} a_i \leq t\right] \geq \prod_{i=1}^n P[a_i \leq t]. \quad (5.1.2)$$

We are now in position to derive the main results. Network β , $\{\tau_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty$, $j = 1, B$ are defined as in section 2. The following assumptions will be made throughout the section

- H_1 $\{\tau_n\}_0^\infty$ is independent of $\{\{\sigma_n^j\}_0^\infty\}$, $1 \leq j \leq B$,
 $\{\tau_n\}_0^\infty$ is a set of independent RV's and
 $\{\{\sigma_n^j, 1 \leq j \leq B\}_0^\infty\}$, is a set of independent RV's.

Lemma 7

Assume H_1 holds. For all $m \geq 0$, $\{d_n^j, 1 \leq j \leq B, 0 \leq n \leq m\}$ is a set of associated RV's.

Proof

We shall actually prove the more general result that $\{d_n^j, 1 \leq j \leq B, 0 \leq n \leq m-1\} \cup \{-r_n, n \leq 0\} \cup \{\sigma_n^j, n \geq 0, 1 \leq j \leq B\}$ is a set of associated RV's for $1 \leq k \leq B$, $m > 0$. This is done by induction on m .

Basis step

Consider the case $m = 1$. We shall show that $\{d_0^j, 1 \leq j \leq k\} \cup \{-r_n, n \geq 0\} \cup \{\sigma_n^j, n \geq 0, 1 \leq j \leq B\}$ is a set of associated RV's for all $1 \leq k \leq B$ by induction on k .

Basis step

Consider all j such that $p(j) = \emptyset$. d_0^j can be expressed as

$$d_0^j = 0. \quad (5.1.3)$$

Consequently, $\{d_0^j\}$, $1 \leq j \leq B_0$ is a set of independent RV's which along with $\{-r_n, n \geq 0\} \cup \{\sigma_n^j, n \geq 0, 1 \leq j \leq B\}$ form a set of associated RV's according to property 4.

Inductive step

Assume that the hypothesis is true for all i , $1 \leq i < k$ where $B_0 \leq k \leq B$. We now show that it is also true for k . Note that $p(k) \neq \emptyset$. By definition,

$$d_0^k = \max_{i \in p(k)} (d_0^i + \sigma_0^i) \quad (5.1.4)$$

which is an increasing function of associated RV's (note that $i < k$ if $i \in p(k)$). Therefore it follows that $\{d_0^j, 1 \leq j \leq k\} \cup \{-r_n, n \geq 0\} \cup \{\sigma_n^j, n \geq 0, 1 \leq j \leq B\}$ is a set of associated RV's.

This completes the proof of the basis step.

Inductive step

Assume that the hypothesis is true up to m . We now show that the hypothesis holds also for $m+1$. This is done by showing that the RV's $\{d_n^j, 1 \leq j \leq k, 0 \leq n \leq m\} \cup \{-r_n, n \geq 0\} \cup \{\sigma_n^j, n \geq 0, 1 \leq j \leq B\}$ are associated for all $1 \leq k \leq B$ by induction on k .

Basis step

We first show that $\{d_n^j, 0 \leq n \leq m, 1 \leq j \leq B\} \cup \{-r_n, n \geq 0\} \cup \{\sigma_n^j, n \geq 0, 1 \leq j \leq B\} \cup \{d_{m+1}^j, 1 \leq j \leq B_0\}$ is a set of associated RV's. By hypothesis we already know that $\{d_n^j, 1 \leq n \leq m, 1 \leq j \leq B\} \cup \{-r_n, n \geq 0\} \cup \{\sigma_n^j, n \geq 0, 1 \leq j \leq B\}$ is a set of associated RV's. Now, for $1 \leq j \leq B_0$,

$$d_{m+1}^j = \max(d_m^j + \sigma_m^j - r_m, 0) \quad (5.1.5)$$

is an increasing function of associated RV's which proves the result.

Inductive step

Assume $\{d_n^j, 1 \leq n \leq m, 1 \leq j \leq B\} \cup \{-r_n, n \geq 0\} \cup \{\sigma_n^j, n \geq 0, 1 \leq j \leq B\} \cup \{d_{m+1}^j, 1 \leq j \leq k\}$ is a set of associated RV's for $B_0 \leq i < k$ where $B_0 \leq k \leq B$. We now show that the hypothesis holds for k . The expression for d_{m+1}^k is

$$d_{m+1}^k = \max(\max_{i \in p(k)} (d_{m+1}^i + \sigma_{m+1}^i), d_m^k + \sigma_m^k - r_m) \quad (5.1.6)$$

which is an increasing function of associated RV's, hence the result.

This completes the induction step on k and the hypothesis is true for $k = B$.

This completes the induction step on m and proves the lemma.

Remark

Lemma 7 holds under the weaker assumptions

$$\begin{aligned} H'_1 \quad & \{r_n\}_0^\infty \text{ is independent of } \{\{\sigma_n^j\}_0^\infty\}, 1 \leq j \leq B, \\ & \{r_n\}_0^\infty \text{ is a set of associated RV's and} \\ & \{\{\sigma_n^j, 1 \leq j \leq B\}_0^\infty\}, \text{ is a set of associated RV's.} \end{aligned}$$

5.2 Bounds based on stochastic ordering

This section will mainly deal with distribution functions rather than with RV's.

Definition 8

Let F and G be two distributions functions on R . F is said to stochastically dominate G , $F \geq_{st} G$, iff

$$G(x) \leq G(x), \forall x \in R. \quad (5.2.1)$$

If a and b are two real valued RV's, we shall say that $a \geq_{st} b$ whenever

$$P[a \leq x] \leq P[b \leq x], \forall x \in R. \quad (5.2.2)$$

A consequence of the above definition and property 5 of associated RV's is

Lemma 9

Let $(a_1, \dots, a_n)$ be a set of associated real valued RV's with respective distribution function $F_1, \dots, F_n$. Let F be the distribution function of $\max(a_1, \dots, a_n)$. Then

$$F \leq_{st} \prod_{i=1}^n F_i. \quad (5.2.3)$$

Last, we state the following obvious lemma.

Lemma 10

Let $(F_1, \dots, F_n)$ and $(G_1, \dots, G_n)$ be two families of distribution functions on R . If $F_i \geq_{st} G_i = 1, n$, then

$$F_1 \cdot F_2 \dots F_n = \prod_{i=1}^n F_i \geq_{st} \prod_{i=1}^n G_i = G_1 \cdot G_2 \dots G_n \quad (5.2.4)$$

and

$$F_1 * F_2 * \dots * F_n \geq_{st} G_1 * G_2 * \dots * G_n, \quad (5.2.5)$$

where $\cdot$ and $*$ respectively denote the product and the convolution of distribution functions.

In the sequel, network β is given as in the preceding sections. We denote as Σ_n^j (resp. T_n^-) the distribution functions on R of the RV σ_n^j (resp. $-r_n$). Notice that Σ_n^j has its support on R^+ and T_n^- on R^- .

We define a sequence D_n^j , $n \geq 0$, $1 \leq j \leq B$ of distribution function on R by the following recursion

$$\hat{D}_0^j = \prod_{i \in p(j)} (\hat{D}_0^i * \Sigma_0^i), \quad j = 1, B \quad (5.2.6)$$

and

$$\hat{D}_{n+1}^j = \left(\prod_{i \in p(j)} \hat{D}_{n+1}^i * \Sigma_{n+1}^i \right) \cdot (\hat{D}_n^j * \Sigma_n^j * T_n^-). \quad (5.2.7)$$

In these definitions, the product over an empty set is always understood as the step distribution function U defined by

$$U(t) = 0, \quad t < 0, \quad U(t) = 1, \quad t \geq 0 \quad (5.2.8)$$

It can be checked by induction that the RV's $\hat{D}_n^j$ have their support on R^+

Theorem 11

Assume H_1 is satisfied. Let D_n^j be the distribution function of the RV's d_n^j , $n \geq 0$, $1 \leq j \leq B$. We have then

$$D_n^j \leq_{st} \hat{D}_n^j, \quad n \geq 0, \quad j = 1, B. \quad (5.2.9)$$

Proof

The proof is by induction on n . Here $df(a)$ denotes the distribution function of the RV a .

Basis step $n = 0$. This step is shown by induction on j .

Basis step

Consider queue j where $p(j) = \emptyset$. $\hat{D}_0^j = D_0^j = U$, so that the result holds true.

Inductive step

Assume the theorem is true for $B_0 \leq j \leq B$. We now show that it is true for $j + 1$.

Note that $p(j) \neq \emptyset$. We have

$$\hat{D}_0^{j+1} = \prod_{i \in p(j+1)} (\hat{D}_0^i * \Sigma_0^i) \geq_{st} \prod_{i \in p(j+1)} (D_0^i * \Sigma_0^i) \quad (5.2.10)$$

(by induction hypothesis and lemma 10)

$$\geq_{st} df\left(\max_{i \in p(j+1)} d_0^i + \sigma_0^i\right)$$

(by lemma 7 and lemma 9 plus assumption H_1 which entails that d_0^i and σ_0^i are independent RV's).

$$= D_0^{j+1}$$

(by definition).

This proves the basis step for n .

Inductive step

Assume that the theorem is true for n . We now show that it is true for $n + 1$ by induction on j .

Basis step

We first do it for $j \in V$ such that $p(j) = \emptyset$:

$$\hat{D}_{n+1}^j = U.(\hat{D}_n^j * \Sigma_n^j * T_n^-) \geq_{st} U.(D_n^j * \Sigma_n^j * T_n^-) \quad (5.2.11)$$

(induction assumption)

$$= df(\max(d_n^j + \sigma_n^j - r_n, 0))$$

(by assumption H_1 which entails that d_n^j is independent of $\sigma_n^j - r_n$)

$$= D_{n+1}^j.$$

This completes the basis step.

Induction step

We now assume the theorem is true for $B_0 \leq j \leq B$ and prove it for $j + 1$. We have

$$\begin{aligned} \hat{D}_{n+1}^{j+1} &= \left(\prod_{i \in p(j+1)} (\hat{D}_{n+1}^i * \Sigma_{n+1}^i) \right) \cdot (\hat{D}_n^{j+1} * \Sigma_n^{j+1} * T_n^-) \\ &\geq_{st} \left(\prod_{i \in p(j+1)} (D_{n+1}^i * \Sigma_{n+1}^i) \right) \cdot (D_n^{j+1} * \Sigma_n^{j+1} * T_n^-). \end{aligned} \quad (5.2.12)$$

(inductive hypothesis and lemma 10).

$$\geq_{st} df(\max(\max_{i \in p(j+1)} (d_{n+1}^i + \sigma_{n+1}^i), d_n^{j+1} + \sigma_n^{j+1} - r_n))$$

(where we used that d_{n+1}^i is independent of σ_{n+1}^i and d_n^{j+1} of $\sigma_n^{j+1} - r_n$ due to H_1 , then that $(d_{n+1}^i + \sigma_{n+1}^i)$ and $(d_n^{j+1} + \sigma_n^{j+1} - r_n)$ form a set of associated RV's due to lemma 7 and finally lemma 9)

$$= D_{n+1}^{j+1}$$

(by definition).

This concludes the proof of the inductive step, and the proof of the theorem.

The next result concerns the extension of the transient bounds of theorem 12 to steady state. H_2 will denote the following set of assumptions :

H_2 Assumption H_1 ,

The sequence $\{r_n\}_0^\infty$ is i.i.d. with r_n integrable,

The sequence $\{\sigma_n^j\}_0^\infty$ is i.i.d. with σ_n^j integrable for all $j = 1, B$.

Theorem 12

Let j be fixed $1 \leq j \leq B$. Assume H_2 holds and that for all $i \in p(j)$, $\hat{D}_n^i$ converges weakly to a finite and integrable distribution function $\hat{D}_\infty^i$ when n goes to ∞ . Assume in addition that

$$E[\sigma_n^j] < E[r]. \quad (5.2.13)$$

Then the distribution functions $\hat{D}_n^j$ converge weakly to a finite distribution function $\hat{D}_\infty^j$ when n goes to ∞ . Denote as D_n^j the distribution function of d_n^j . Under the foregoing assumptions, the distribution functions D_n^j converge weakly to a finite distribution function D_∞^j when n goes to ∞ and $\hat{D}_\infty^j$ stochastically dominates D_∞^j , namely

$$D_\infty^j \leq_{st} \hat{D}_\infty^j. \quad (5.2.13)$$

The proof is found in Appendix 3.

6 Applications of bounds based on convex ordering

The following set of assumptions will be assumed to hold throughout the section:

H_3 . The $j+1$ sequences $\{r_n\}_0^\infty$, $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$ are mutually independent

6.1 Determinism minimizes response times

The property that under certain independence assumptions, deterministic interarrival times (resp. service times) minimize response times in G/G/1 queues, as shown in [St 84] and [Wh 84], can be extended to AFJQN's using Theorem 3.

Let $\{\bar{d}_n^j\}_0^\infty$ (resp. $\{\bar{\sigma}_n^j\}_0^\infty$), $j = 1, \dots, B$ be the response times obtained for the constituting sequences $\{\bar{r}_n\}_0^\infty$ and $\{\bar{\sigma}_n^j\}_0^\infty$, $j = 1, \dots, B$ (resp. $\{r_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$) respectively defined by the equations:

$$\bar{r}_n = E[r_n], \quad n \geq 0 \quad (6.1.1)$$

$$\bar{\sigma}_n^j = \sigma_n^j, \quad j = 1, \dots, B, \quad n \geq 0 \quad (6.1.2)$$

and

$$\bar{r}_n = r_n, \quad n \geq 0 \quad (6.1.3)$$

$$\bar{\sigma}_n^j = \sigma_n^j, \quad j = 1, \dots, B, \quad j \neq j_0, \quad n \geq 0 \quad (6.1.4)$$

$$\bar{\sigma}_n^{j_0} = E[\sigma_n^{j_0}], \quad n \geq 0 \quad (6.1.5)$$

where j_0 is any fixed integer $1 \leq j_0 \leq B$.

Corollary 13

For all $n \geq 0$ and $j = 1, \dots, B$, the following inequalities hold

$$\bar{d}_n^j \leq_{ci} d_n^j, \quad n \geq 0 \quad (6.1.6)$$

and

$$\bar{d}_n^j \leq_{ci} d_n^j, \quad n \geq 0 \quad (6.1.7)$$

Proof

Let $\tilde{G}$ (resp. $\tilde{G}$) be the sub σ -fields of F generated by the RV's $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$ (resp. $\{r_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$, $j \neq j_0$). We first get from the independence assumption that

$$\bar{r}_n = E[r_n | \tilde{G}], \quad n \geq 0 \quad (6.1.8)$$

$$\bar{r}_n = E[r_n | \tilde{G}], \quad n \geq 0 \quad (6.1.9)$$

and

$$\bar{\sigma}_n^j = E[\sigma_n^j | \tilde{G}], \quad n \geq 0 \quad j = 1, \dots, B, \quad (6.1.10)$$

$$\bar{\sigma}_n^j = E[\sigma_n^j | \tilde{G}], \quad n \geq 0 \quad j = 1, \dots, B, \quad (6.1.11)$$

so that Theorem 3 entails

$$\bar{d}_n^j \leq E[d_n^j | \tilde{G}], \quad n \geq 0 \quad j = 1, \dots, B \quad (6.1.12)$$

and

$$\bar{d}_n^j \leq E[d_n^j | \bar{G}], \quad n \geq 0 \quad j = 1, \dots, B. \quad (6.1.13)$$

Equations (6.1.6) and (6.1.7) are mere rephrasing of (6.1.12) and (6.1.13) respectively.

The lower bounds (6.1.12) and (6.1.13) on d_n^j extend to steady state when the constituting sequences $\{r_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$ satisfy the assumptions of Theorem 2. Indeed, these conditions entail that both the constituting sequences $\{\bar{r}_n\}_0^\infty$, $\{\bar{\sigma}_n^j\}_0^\infty$, $j = 1, \dots, B$ and $\{\bar{r}_n\}_0^\infty$ and $\{\bar{\sigma}_n^j\}_0^\infty$, $j = 1, \dots, B$ satisfy the assumptions of Theorem 2. Hence Corollary 5 applies to show that the bounds of Corollary 13 extend to steady state, namely

$$\bar{d}_\infty^j \leq_{cs} d_\infty^j, \quad j = 1, \dots, B, \quad (6.1.14)$$

$$\bar{d}_\infty^j \leq_{cs} d_\infty^j, \quad j = 1, \dots, B. \quad (6.1.15)$$

6.2 Networks in random environment

The problem of determining the statistics of isolated queues with time varying interarrival times was considered in the markovian case in [Ma 85]. For the general G/G/1 FIFO queue, bounds are also available when the variations depend upon an independent stationary and ergodic "environment" process. It was shown in [BM 86] that the waiting time statistics in such a queueing system are bounded from below by those of the same queue with the environment process kept to its mean value (see also [Ro 83]). Theorem 3 allows to extend this result to any AFJQN β . As in [BM 86], the environment process is assumed to be a non-negative real-valued stochastic process $V(t)$, $t \in R$ on (Ω, F, P) being ergodic and stationary. Two stationary and ergodic sequences of nonnegative RV's are assumed to be given: $\{r_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$. All these RV's are assumed to be integrable with $E[V(t)] = 1$ holding in particular. The modulation of the arrival process is obtained by accelerating time proportionally to V , so that the effective interarrival times in the random environment network are given by the sequence $\{\bar{r}_n\}_0^\infty$ defined by

$$\bar{r}_n = \int_{a_n}^{a_{n+1}} V(s) ds, \quad n \geq 0. \quad (6.2.1)$$

Let $\{\bar{d}_n^j\}_0^\infty$ (resp. $\{d_n^j\}_0^\infty$) be the response times obtained for the constituting sequences $\{\bar{r}_n\}_0^\infty$ (resp. $\{r_n\}_0^\infty$) and $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$.

Corollary 14

If the stochastic process $V(t)$, $t \in R$ is independent of $\{r_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$, then the following inequality holds for all $n \geq 0$ and $j = 1, \dots, B$

$$\bar{d}_n^j \geq_{cs} d_n^j. \quad (6.2.2)$$

Proof

Let $\dot{G}$ be the sub σ -fields of F generated by the RV's $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$ and $\{r_n\}_0^\infty$. It was shown in [BM 86] that under the enforced assumptions, for all $n \geq 0$

$$E[\bar{r}_n | \dot{G}] = r_n. \quad (6.2.3)$$

Equation (6.2.2) is now obtained as a direct consequence of Theorem 3.

Consider a fixed queue j . Observe that under the foregoing assumptions, if $\{\tilde{r}_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty, j = 1, \dots, B$ satisfy the conditions of Theorem 2 for j , then, $\{r_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty, j = 1, \dots, B$ also satisfy the conditions of Theorem 2 for j , so that the bounds of Corollary 14 then extend to steady state, namely

$$d_\infty^j \geq_{cs} d_\infty^j. \quad (6.2.4)$$

6.3 Bounds on parallel networks

Theorem 3 also provides lower and upper bounds for the following problem, a particular case of which was considered in [BM 85]. Let β be any AFJQN made of K AFJQ subnetworks $\alpha_1, \dots, \alpha_K$ in parallel with respective underlying graphs $G_l = (V_l, E_l), 1 \leq l \leq K$. Denote as R_n the n -th network response time:

$$R_n = \max_{i \in p(B+1)} (d_n^i + \sigma_n^i) \quad (6.3.1)$$

for the constituting sequences $\{r_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty, j = 1, \dots, B$. Let R_n^l denote the n -th response time in the subnetwork $\alpha_l, 1 \leq l \leq K$ for the constituting sequences $\{r_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty, j \in V_l$.

$$R_n^l = \max_{i \in p_l(B+1)} (d_n^i + \sigma_n^i), \quad (6.3.2)$$

where $p_l(B+1)$ denotes the queues of $p(B+1)$ which belong to V_l . Owing to the parallel structure of β , we have

$$R_n = \max_{1 \leq l \leq K} R_n^l. \quad (6.3.3)$$

Let finally $\tilde{R}_n^l$ denote the n -th response time in α_l for the constituting sequence $\{\tilde{r}_n\}_0^\infty$ and $\{\tilde{\sigma}_n^j\}_0^\infty, j = 1, \dots, B$, defined by equations (6.1.1) and (6.1.2).

Corollary 15

For all $n \geq 0$

$$R_n \geq_{cs} \max_{1 \leq l \leq K} \tilde{R}_n^l. \quad (6.3.4)$$

Proof

It was established in the proof of Corollary 13 that

$$\tilde{d}_n^j \leq E[d_n^j | \tilde{G}], \quad n \geq 0 \quad j = 1, \dots, B \quad (6.3.5)$$

This and Jensen's Theorem can be used in (6.3.3) to yield

$$\tilde{R}_n^l \leq E[R_n^l | \tilde{G}], \quad n \geq 0 \quad l = 1, \dots, K. \quad (6.3.6)$$

Using now this last inequality and Jensen's Theorem in (6.3.4), we get

$$E[R_n | \tilde{G}] \geq \max_{1 \leq l \leq K} E[R_n^l | \tilde{G}], \quad n \geq 0. \quad (6.3.7)$$

Combining equations (6.3.6) and (6.3.7), we finally obtain

$$E[R_n|\tilde{G}] \geq \max_{1 \leq l \leq K} \bar{R}_n^l, \quad n \geq 0, \quad (6.3.8)$$

which implies (6.3.4).

Remark

Notice that due to our mutual independence assumption on the sequences $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$, the sequences $\{\bar{R}_n^l\}_0^\infty$, $l = 1, \dots, K$ are mutually independent as well. In other words, Corollary 14 allows us to derive lower bounds for the network response times that reduce to computing the maximum of K independent RV's being the response times of subnetworks of smaller size than the initial one.

Upper bounds can also be obtained using convex ordering in the following particular case: assume the arrival process is divisible in the sense that there exist K mutually independent sequences of RV's $\{t_n^l\}_0^\infty$ which satisfy the mean condition:

$$r_n = \frac{\sum_{l=1}^K t_n^l}{K}, \quad n \geq 0. \quad (6.3.9)$$

Let d_n^j (resp. $\bar{R}_n^l$) denote the delay between the n -th arrival and the beginning of the n -th service in queue j (resp. the n -th response time) in V_l for the constituting sequence $\{t_n^l\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$, $j \in \alpha_l$.

Corollary 16

For all $n \geq 0$

$$R_n \leq \max_{1 \leq l \leq K} \bar{R}_n^l. \quad (6.3.10)$$

Proof

Let $\tilde{G}$ be the sub σ -algebra of F generated by the RV's $\{r_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$. For all $n \geq 0$, We get from the exchangeability of the RV's $\{t_n^l\}_0^\infty$ and the independence assumptions that for all $n \geq 0$,

$$E[t_n^l] = r_n. \quad (6.3.11)$$

Using Jensen's inequality in

$$\bar{R}_n = \max_{1 \leq l \leq K} \bar{R}_n^l, \quad (6.3.12)$$

we get

$$E[\max_{1 \leq l \leq K} \bar{R}_n^l | \tilde{G}] \geq \max_{1 \leq l \leq K} \max_{j \in \alpha_l} (E[d_n^j | \tilde{G}] + \sigma_n^j). \quad (6.3.13)$$

This together with Theorem 3 entail

$$E[\max_{1 \leq l \leq K} \bar{R}_n^l | \tilde{G}] \geq \max_{1 \leq l \leq K} \max_{j \in \alpha_l} (d_n^j + \sigma_n^j) = T_n, \quad (6.3.14)$$

which completes the proof of (6.3.10).

Notice that for this upper bound too, the RV's $\bar{R}_n^l$ are mutually independent and can be obtained by considering subnetworks of smaller dimensions than the initial one. Observe that if $\{r_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$ (resp. $\{t_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$) satisfy the conditions of

Theorem 2 for all $j = 1, \dots, B$, the bounds of Corollary 15 (resp. 16) then extend to steady state namely,

$$R_{\infty} \geq c, \max_{1 \leq l \leq K} \tilde{R}_{\infty}^l. \quad (6.3.15)$$

and

$$R_{\infty} \leq c, \max_{1 \leq l \leq K} \tilde{R}_{\infty}^l. \quad (6.3.16)$$

6.4 Bounds on series networks

Let β be any AFJQN made of K AFJQ subnetworks $\alpha_1, \dots, \alpha_K$ in series with respective underlying graphs $G_l = (V_l, E_l)$, $1 \leq l \leq K$. Owing to the series structure of the network, the subnetworks β_l^* , $1 \leq l \leq K$ of β obtained by considering only the queues of $V_1 \cup \dots \cup V_l$ are also in the AFJQN class. Let R_n^l denote the n -th response time in β_l^* for the constituting sequences $\{\tau_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$, $j \in V_1 \cup \dots \cup V_l$. Let also t_n^l denote the n -th interdeparture time of the output stream of β_l^* :

$$t_n^l = R_{n+1}^l - R_n^l, \quad n \geq 0. \quad (6.4.1)$$

Owing to the series structure of β , R_n can be decomposed into the sum:

$$R_n = \sum_{l=1}^K \rho_n^l, \quad n \geq 0 \quad (6.4.2)$$

where ρ_n^l denotes the n -th response time in the AFJQN α_l , for the interarrival times sequence $\{t_n^{l-1}\}_0^\infty$ and the service times sequence $\{\sigma_n^j\}_0^\infty$, j in V_l and where t_n^0 stands for τ_n , $n \geq 0$.

Similarly, let $\tilde{\rho}_n^l$ denote the n -th response time in the AFJQN α_l , for the constituting sequences $\{\tilde{t}_n^{l-1}\}_0^\infty$ and $\{\tilde{\sigma}_n^j\}_0^\infty$, j in V_l , where

$$\tilde{t}_n^l = E[t_n^l], \quad n \geq 0 \quad (6.4.3)$$

$$\tilde{\sigma}_n^j = \sigma_n^j, \quad j \in V_l, \quad n \geq 0. \quad (6.4.4)$$

Corollary 17

For all $n \geq 0$, the following inequality holds

$$E[R_n] \geq \sum_{l=1}^K E[\tilde{\rho}_n^l]. \quad (6.4.5)$$

Proof

Let $\tilde{G}_l$ be the sub σ -algebra of F generated by the RV's $\{\tau_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty$, $j = 1, \dots, B$, j in $V_1 \cup \dots \cup V_l$. Owing to the independence assumptions, we have, for all $n \geq 0$, $1 \leq l \leq K$

$$E[t_n^l | \tilde{G}_l] = E[t_n^l] \quad (6.4.6)$$

and, for all $j \in V_l$

$$E[\sigma_n^j | \tilde{G}_l] = \sigma_n^j, \quad j \in V_l. \quad (6.4.7)$$

Hence, Theorem 3 applied to the network α_l , entails that, for all $n \geq 0$, $1 \leq l \leq K$

$$E[\rho_n^l | \tilde{G}_l] \geq \tilde{\rho}_n^l. \quad (6.4.8)$$

This together with equation (6.4.2) readily entail (6.4.3).

Observe that (6.4.5) obviously holds at steady state provided the first moments involved in this equation converge.

7 Applications of bounds based on association

The condition H_1 will be assumed to hold throughout the section so that the assumptions of Lemma 7 and Theorem 12 are satisfied.

7.1 Bounds on parallel networks

The notations are those of section 6.3: R_n (resp. R_n^l) denotes the n -th response time in β (resp. α_l , $1 \leq l \leq K$) for the constituting sequences $\{r_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty$ $j = 1, \dots, B$. Under the foregoing assumptions, we have the following strengthening of corollary 16

Corollary 18

For all $n \geq 0$

$$df(R_n) \leq_{st} \prod_{1 \leq l \leq K} df(R_n^l). \quad (7.1.1)$$

Proof

It was established in Lemma 7 that the RV's $\{d_n^j\}_0^\infty$, $j = 1, \dots, B$, are associated. Hence, the RV's R_n^l , $n \geq 0$, $1 \leq l \leq K$, which are given by (6.3.2) in terms of increasing functions of associated RV's, are also associated, owing to property 2 of associated RV's. Equation (7.1.1) is hence a direct consequence of property 5 (equation (5.1.2)) of association.

Assume the stability condition of Theorem 2 is satisfied. (Observe that condition H_1 is stronger than condition H_0 .) Then, the random vectors $\{d_n^j\}$, $j = 1, \dots, B$ converge weakly to a finite random vector $\{d_\infty^j\}$, $j = 1, \dots, B$ when n goes to ∞ . This in turn implies that the random vectors $\{R_n^l\}$, $l = 1, \dots, K$ (resp. the RV's R_n) converge weakly to a finite random vector (resp. RV) $\{R_\infty^l\}$, $l = 1, \dots, K$ (resp. R_∞) when n goes to ∞ .

Applying now proposition (1.2.3) of [St 84] to the weakly converging sequences $df(R_n)$ and $\prod_{1 \leq l \leq K} df(R_n^l)$ it is plain that equation (7.1.1) extends to steady state, namely

$$df(R_\infty) \leq_{st} \prod_{1 \leq l \leq K} df(R_\infty^l). \quad (7.1.2)$$

The upper bounds of equation (7.1.2) and the lower bounds of equation (6.3.4) are exemplified in Figure 3.

7.2 More general bounds. Relation to resequencing

We consider now the case of more general AFJQN's. For these networks, we show that Theorems 11 and 12 can be used to provide computable upper bounds which relate to resequencing models analyzed earlier in [BGP 84]. The discussion of these bounds will be limited to steady

state. It is assumed that each queue satisfies the assumptions of Theorem 12, so that the distribution functions D_n^j (resp. $\hat{D}_n^j$), $j = 1, \dots, B$ converge weakly to a proper distribution function D_∞^j (resp. $\hat{D}_\infty^j$) when n goes to ∞ and $D_\infty^j \leq \hat{D}_\infty^j$ for all $1 \leq j \leq B$. Denoting as Σ^j (resp. T^-) the common distribution function of the RV's $\{\sigma_n^j\}$, $j = 1, \dots, B$. (resp. $-\{\tau_n\}$), it follows from equation (5.2.7) that the distribution functions $\hat{D}_\infty^j$, $j = 1, \dots, B$ satisfy the set of equations (7.2.1)-(7.2.3) below :

$$\hat{D}_\infty^j = U.(\hat{D}_\infty^j * \Sigma^j * T^-), \quad (7.2.1)$$

for j such that $p(j) = 0$ and

$$\hat{D}_\infty^j = A^j.(\hat{D}_\infty^j * \Sigma^j * T^-), \quad (7.2.2)$$

for j such that $p(j) \neq 0$, where

$$A^j = \prod_{i \in p(j)} \hat{D}_\infty^i * \Sigma^i. \quad (7.2.3)$$

This set of functional equations can be solved recursively as follows:

First compute the solution $\hat{D}_\infty^j$ of equation (7.2.1) for all $1 \leq j \leq B_0$. This equation is the functional equation satisfied by the distribution function of the stationary waiting times in a GI/GI/1 queue with service times distributed according to Σ^j and (negative) interarrival times according to T^- .

Next, compute by induction $\hat{D}_\infty^{B_0+1}, \dots, \hat{D}_\infty^B$ as follows. Assume that the distribution functions $\hat{D}_\infty^1, \dots, \hat{D}_\infty^{j-1}$ are known for some $j > B_0$. Notice first that this and equation (7.2.3) fully determine the distribution function A^j on R^+ . Hence, the only unknown in equation (7.2.2) is $\hat{D}_\infty^j$. This equation is the functional equation satisfied by the distribution function of the stationary end-to-end delays in a GI/GI/GI/1 resequencing queue as considered in [BGP 84] with desordering times distributed according to A^j , service times distributed according to Σ^j and (negative) interarrival times according to T^- .

The end of this section is devoted to computational problems related to the solution of these functional equations. General techniques for solving (7.2.1) are well known (see for instance [Co 85] for a detailed discussion)

We consider now equation (7.2.2), the general form of which is

$$D = A.(D * \Sigma * T^-), \quad (7.2.4)$$

where A , Σ and T^- are known distribution functions on R with their support on R^+ , R^+ and R^- respectively, $C = \Sigma * T^-$ has a negative mean and D is the unknown distribution function on R_+ . Closed form solutions have been derived for the solution of (7.2.4) in [BGP 84] for certain classes of distribution functions A and T^- namely A hyperexponential and T^- exponential. For more general classes of distribution functions, it is established in Appendix 3 that the following numerical schema converges towards the solution of (7.2.1):

$$F_{n+1}^j(t) = A^j(t). \int_{-\infty}^t F_n^j(t-u) dC^j(u), \quad n \geq 0, \quad t \in R, \quad (7.2.5)$$

where $C^j = \Sigma^j * T_-$ and

$$F_0^j = A^j. \quad (7.26)$$

Here, the functions $F_n(t)$, $t \in R$ are distribution functions on R with support on R^+ and the convergence of F_n towards the solution of (7.2.1) has to be understood in the sense of the weak convergence.

In conclusion, Theorems 11 and 12 provide a general method to compute upper bounds on the stationary delays through AFJQN's with i.i.d. constituting sequences. The computation of these bounds reduces to determining B_0 stationary waiting time distribution functions of GI/GI/1 queues and $B - B_0$ stationary state end-to-end delays in GI/GI/GI/1 resequencing queues.

Appendix 1

The basic idea for proving theorem 2 consists in generalizing the schema of Loynes for the response time of a $G/G/1$ queue ([Lo 62]), to the response times d_n^j of our network. Let us first consider the sequence $\{\tau_n\}_0^\infty$ and $\{\sigma_n^j\}_0^\infty$ for all $j \in 1, B$ as the right half of certain bi-infinite sequences $\{\tau_n\}_{-\infty}^{+\infty}$ and $\{\sigma_n^j\}_{-\infty}^{+\infty}$ on (Ω, F, P) . We shall assume that (Ω, F, P) is the canonical space. Hence P will be assumed to be θ -invariant (stationary) and θ -ergodic. Let us denote by r the difference $a_1 - a_0$, and by σ^j the variable σ_0^j . Consider now the schema $\{\delta_n^j\}_0^\infty$ defined by $\delta_0^j = d_0^j$, and for $n \geq 0$:

$$\delta_{n+1}^j \circ \theta = \max(\max_{i \in p(j)} ((\delta_{n+1}^i + \sigma^i) \circ \theta), \delta_n^j + \sigma^j - r). \quad (A.1.1)$$

Lemma 1

For any $j \in B$, the sequence $\{\delta_n^j\}_{n \geq 0}$ is increasing.

Proof

Let us first prove this for $1 \leq j \leq B_0$. It is clear that $\delta_1^j \geq 0 = \delta_0^j$. Assume now that $\delta_n^j \geq \delta_{n-1}^j$ for some $n \geq 1$. From (A.1.1), we get:

$$\delta_{n+1}^j \circ \theta = \max(0, \delta_n^j + \sigma^j - r) \geq \max(0, \delta_{n-1}^j + \sigma^j - r) = \delta_n^j \circ \theta, \quad 1 \leq j \leq B_0. \quad (A.1.2)$$

By induction, the δ_n^j 's are thus increasing.

Now consider j such that $p(j) \neq \emptyset$. By the induction hypothesis, we can assume that the RV's δ_n^i are increasing in n for $i \in p(j)$. We prove first that $\delta_1^j \geq \delta_0^j$. We have

$$\delta_1^j \circ \theta = \max(\max_{i \in p(j)} (\delta_1^i + \sigma^i) \circ \theta, \delta_0^j + \sigma^j - r) \geq \max_{i \in p(j)} ((\delta_1^i + \sigma^i) \circ \theta) \geq \max_{i \in p(j)} ((\delta_0^i + \sigma^i) \circ \theta), \quad (A.1.3)$$

where we have used our assumption $\delta_1^i \geq \delta_0^i$. Notice that the last expression is $\delta_0^j \circ \theta$ so that the property is proved. Assuming now that $\delta_n^j \geq \delta_{n-1}^j$, by (A.1.1) we get

$$\delta_{n+1}^j \circ \theta \geq \max(\max_{i \in p(j)} ((\delta_{n+1}^i + \sigma^i) \circ \theta), \delta_{n-1}^j + \sigma^j - r). \quad (A.1.4)$$

Since the δ_n^i are increasing for $i \in p(j)$ we get from the last expression that

$$\delta_{n+1}^j \circ \theta \geq \max(\max_{i \in p(j)} ((\delta_{n+1}^i + \sigma^i) \circ \theta), \delta_{n+1}^j + \sigma^j - r) = \delta_n^j \circ \theta \quad (A.1.5)$$

and so δ_n^j increases in n .

Lemma 2

Let δ_∞^j be the limiting value of the increasing sequence δ_n^j when n goes to infinity. Under the assumptions of theorem 2, $\delta_\infty^j < \infty$. If there exists an $i \in \pi(j)$ such that $E[\sigma_n^i] > E[r_n]$ then $\delta_\infty^j = \infty$ a.s.

Proof The limiting variables δ_∞^j satisfy the pathwise equation :

$$\delta_\infty^j \circ \theta = \max(\max_{i \in p(j)} ((\delta_\infty^i + \sigma^i) \circ \theta), \delta_\infty^j + \sigma^j - r) \quad (A.1.6)$$

For $1 \leq j \leq B_0$, (A.1.6) reduces to

$$\delta_\infty^j \circ \theta = \max(0, \delta_\infty^j + \sigma^j - r). \quad (A.1.7)$$

Equation (A.1.7) shows that the event $\{\delta_\infty^j = \infty\}$ is θ -invariant. Therefore, this event is either of probability 0 or 1. Assume that it is of probability 1. By the increasingness property we have

$$E[\max(0, \delta_n^j + \sigma^j - r) - \delta_n^j] = E[\delta_{n+1}^j \circ \theta - \delta_n^j] = E[\delta_{n+1}^j - \delta_n^j] \geq 0. \quad (A.1.8)$$

From this we get

$$\lim_{n \rightarrow \infty} E[\max(0, \delta_n^j + \sigma^j - r) - \delta_n^j] \geq 0. \quad (A.1.9)$$

Using now Lebesgue's theorem, this inequality is preserved with limit taken inside the expectation. If we assume that $\delta_\infty^j \uparrow \infty$, then we get

$$E[\sigma^j] \geq E[r]. \quad (A.1.10)$$

Now taking the contrapositive of this argument, we see that

$$E[\sigma^j] < E[r] \quad (A.1.11)$$

is sufficient to have δ_∞^j finite a.e. This completes the proof of the first part of the lemma for $1 \leq j \leq B_0$.

Let j be such that $B_0 < j \leq B_0$. Assume now that for all $i \in \pi(j)$, δ_∞^i is a.e. finite and integrable. The proof that condition (A.1.11) entails δ_∞^j finite a.e. proceeds as follows. The event $\{\delta_\infty^j = \infty\}$ is shown to be θ -invariant from (A.1.6). The inequality

$$\limsup_{n \rightarrow \infty} E[(\max(\max_{i \in p(j)} ((\delta_{n+1}^i + \sigma^i) \circ \theta), \delta_n^j + \sigma^j - r) - \delta_n^j)] \geq 0 \quad (A.1.12)$$

is then established using the increasingness of δ_n^j and its integrability as in (A.1.8). One also gets from elementary manipulations that

$$X_n = (\max(\max_{i \in p(j)} ((\delta_{n+i}^1 + \sigma^1) \circ \theta), \delta_n^2 + \sigma^2 - r) - \delta_n^2) \leq (\max_{i \in p(j)} ((\delta_{n+1}^1 + \sigma^1) \circ \theta) + \sigma^2 - r. \quad (A.1.13)$$

From the increasingness of δ_n^1 , $i \in p(j)$, we get hence

$$X_n \leq (\max_{i \in p(j)} ((\delta_\infty^1 + \sigma^1) \circ \theta) + \sigma^2 - r. \quad (A.1.14)$$

Owing to the integrability assumptions, it follows from (A.1.14) that the RV's X_n are uniformly bounded from above by an integrable RV. The Fatou-Lebesgue lemma and (A.1.12) entail then

$$E[\limsup_n X_n] \geq \limsup_n E[X_n] \geq 0. \quad (A.1.15)$$

Under the assumption $\delta_\infty^1 < \infty$ a.e. for all $i \in \pi(j)$, the hypothesis $\delta_n^2 \uparrow \infty$ implies that

$$\limsup_n X_n = \sigma^2 - r, \quad (A.1.16)$$

so that queue j satisfies condition (A.1.10). The rest of the proof follows exactly as before.

Proof of Theorem 2

We get by induction that $d_n^2 = \delta_n^2 \circ \theta^n$ (use the fact $r_n = r \circ \theta^n$, $\sigma_n^2 = \sigma \circ \theta^n$, $n \geq 0$). Hence d_n^2 and δ_n^2 have the same distribution due to the θ -invariance of P . The weak convergence of the law of d_n^2 to a proper distribution is now a direct consequence of the increasing a.e. of δ_n^2 to the finite random variable δ_∞^2 .

Appendix 2

1 - A stationary queuing system where an increased variability of interarrivals decreases the variability of interdeparture times.

Consider a $GI/M/1$ queue. The steady state distribution for the number of customers just after a departure is geometrically distributed with parameter σ which is the smallest positive real root of the equation

$$\sigma = A^*(\mu(1 - \sigma)), \quad (A.2.1)$$

where A^* denotes the Laplace transform of the interarrival times and μ^{-1} the mean service time. The interdeparture distribution function has hence the following Laplace transform

$$D^*(s) = (1 - \sigma) \sum_{k \geq 1} \sigma^k \frac{\mu}{\mu + s} + (1 - \sigma) A^*(s) \frac{\mu}{\mu + s} \quad (A.2.2)$$

$$= (1 - \sigma) A^*(s) \frac{\mu}{\mu + s} + \sigma \frac{\mu}{\mu + s}. \quad (A.2.3)$$

The mean interdeparture time is hence

$$d = \frac{1}{\mu} + (1 - \sigma) \frac{1}{\lambda}, \quad (\text{A.2.4})$$

where denotes the mean interarrival time. Consider the two cases where A^* is exponential and deterministic with the same mean λ^{-1}

$$A_1^*(s) = \frac{\lambda}{\lambda + s}, \quad (\text{A.2.5})$$

$$A_2^*(s) = \exp(-\frac{s}{\lambda}). \quad (\text{A.2.6})$$

The distribution function corresponding to A_1^* is larger for convex ordering than the one corresponding to A_2^* . However, $\sigma_1 > \sigma_2$ so that $d_1 < d_2$.

2 - A stationary queueing system where an increased variability of interarrivals increases the variability of interdeparture times.

Consider a stable D/D/1 queue. Let λ denote the intensity of the arrival process. The stationary interdeparture times have deterministic distribution with mean λ^{-1} . Here, an increased variability of interarrivals increases the variability of interdeparture times.

Appendix 3

In this section, weak convergence of distribution functions on R will be denoted as $\Rightarrow$. We establish first that under the assumptions of Theorem 12

$$D_n^j \Rightarrow D_\infty^j \quad (\text{A.3.1})$$

and

$$\hat{D}_n^j \Rightarrow \hat{D}_\infty^j \quad (\text{A.3.2})$$

when n goes to ∞ , where D_∞^j and $\hat{D}_\infty^j$ are proper distribution functions on R^+ . We establish the convergence (A.3.2) first. The property is first proved for j such that $p(j) = 0$. For such a j , $\hat{D}_n^j$ represents the distribution function of the n -th waiting time in a GI/GI/1 FIFO queue and classical results in queueing theory [Co 85] can be used to establish (A.3.1) provided $E[\sigma_n^j] < E[\tau_n]$.

The convergence (A.3.2) is now established by induction for all $B_0 \leq j \leq B$. Assume queues $1, \dots, j-1$ to be in steady state for some j such $B_0 < j < B$. Then equations (5.2.6) and (5.2.7) read respectively

$$\hat{D}_0^j = A^j \quad (\text{A.3.3})$$

and

$$\hat{D}_{n+1}^j = A^j \cdot (\hat{D}_n^j \cdot \Sigma^j \cdot T^-), \quad n \geq 0, \quad (\text{A.3.4})$$

where

$$A^j = \prod_{i \in p(j)} \hat{D}_\infty^i \cdot \Sigma^i. \quad (\text{A.3.5})$$

Let $\{\alpha_n^j\}_{n=0}^\infty$, $\{\sigma_n^j\}_{n=0}^\infty$ and $\{r_n\}_{n=0}^\infty$ be independent sequences of i.i.d. RV's with respective distribution functions A^j , Σ^j and T^- . Consider the R^+ -valued Markov chain $\{y_n^j\}_0^\infty$ defined by the recursion

$$y_{n+1}^j = \max(\alpha_{n+1}^j, y_n^j + \sigma_n^j - r_n), \quad n \geq 0, \quad (\text{A.3.6})$$

where

$$y_0^j = \alpha_0^j. \quad (\text{A.3.7})$$

Using the independence assumptions, it is plain from (A.3.3)-(A.3.5) that $df(y_n^j) = \hat{D}_n^j$ for all $n \geq 0$.

Denote α_0^j , σ_0^j and r_0 as α^j , σ^j and r respectively. Using the same formalism as in Appendix 1, define the Loynes' schema $\{z_n^j\}_0^\infty$ by the recursion

$$z_{n+1}^j \circ \theta = \max(\alpha^j \circ \theta, z_n^j + \sigma^j - r), \quad n \geq 0, \quad (\text{A.3.8})$$

where

$$z_0^j = \alpha^j. \quad (\text{A.3.9})$$

One proves as in Appendix 1 that z_n^j increases pathwise with n , $z_n^j =_{st} y_n^j$ for all $n \geq 0$ and

$$z_{n+1}^j \circ \theta - z_n^j \leq \alpha^j \circ \theta + \sigma^j - r. \quad (\text{A.3.10})$$

The integrability assumptions are then used in (A.1.10) to prove that the RV's $\{z_n^j\}_0^\infty$ are bounded from above by an integrable RV. The remainder of the proof is as in Appendix 1.

The numerical schema (7.2.5)-(7.2.6) is a mere rephrasing of equations (A.3.3)-(A.3.5), so that its convergence towards the solution of (7.2.1) is a direct consequence of (A.3.2).

We prove now the convergence (A.3.1). It was established in Theorem 11 that under the assumption H_2

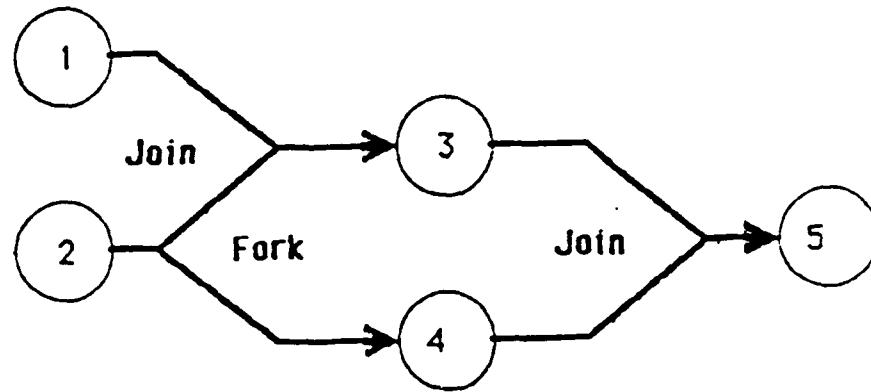
$$D_n^j \leq_{st} \hat{D}_n^j, \quad n \geq 0, \quad j = 1, B. \quad (\text{A.3.11})$$

It follows from the discussion of Appendix 1 that

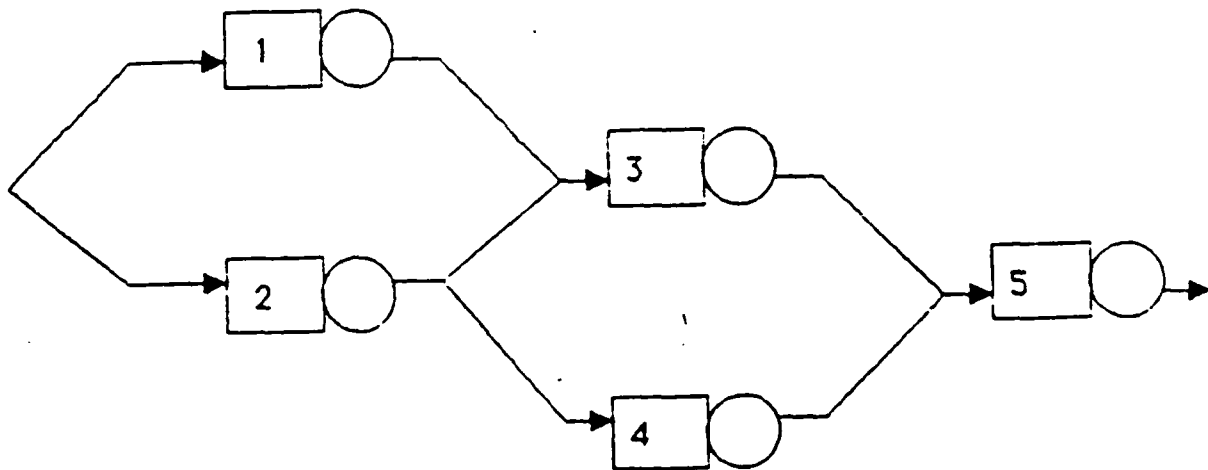
$$D_n^j = df(\delta_n^j), \quad n \geq 0, \quad j = 1, B. \quad (\text{A.3.12})$$

Hence, the convergence (A.3.2) of $\hat{D}_n^j$ towards a finite distribution function used in (A.3.11) entails that the increasing sequence δ_n^j cannot converge to ∞ almost surely, which establishes (A.3.1).

FIGURE 1



(a)

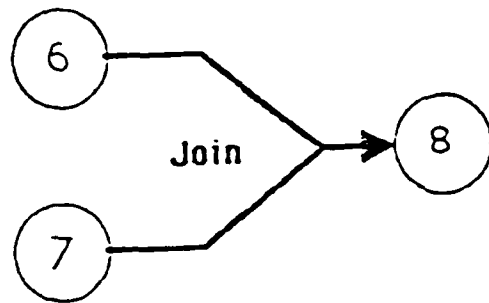
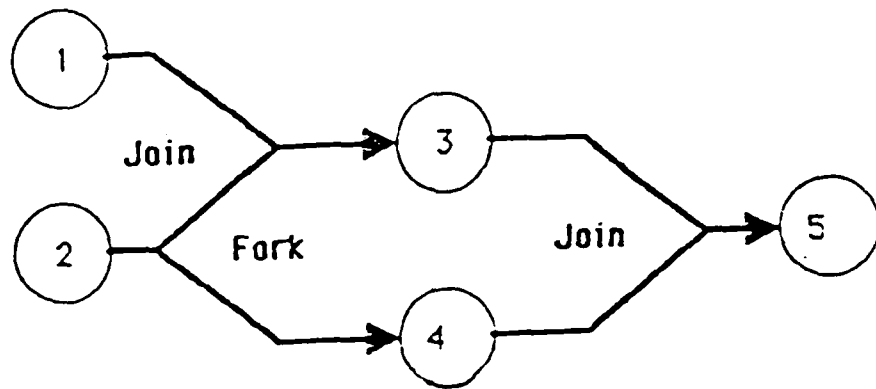


(b)

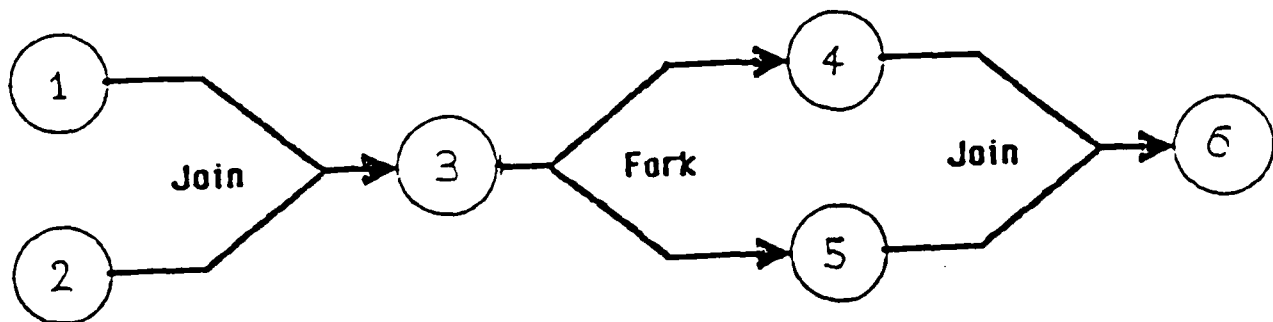
(a) A program

(b) The associated
Fork join
Queueing network

FIGURE 2



(a)

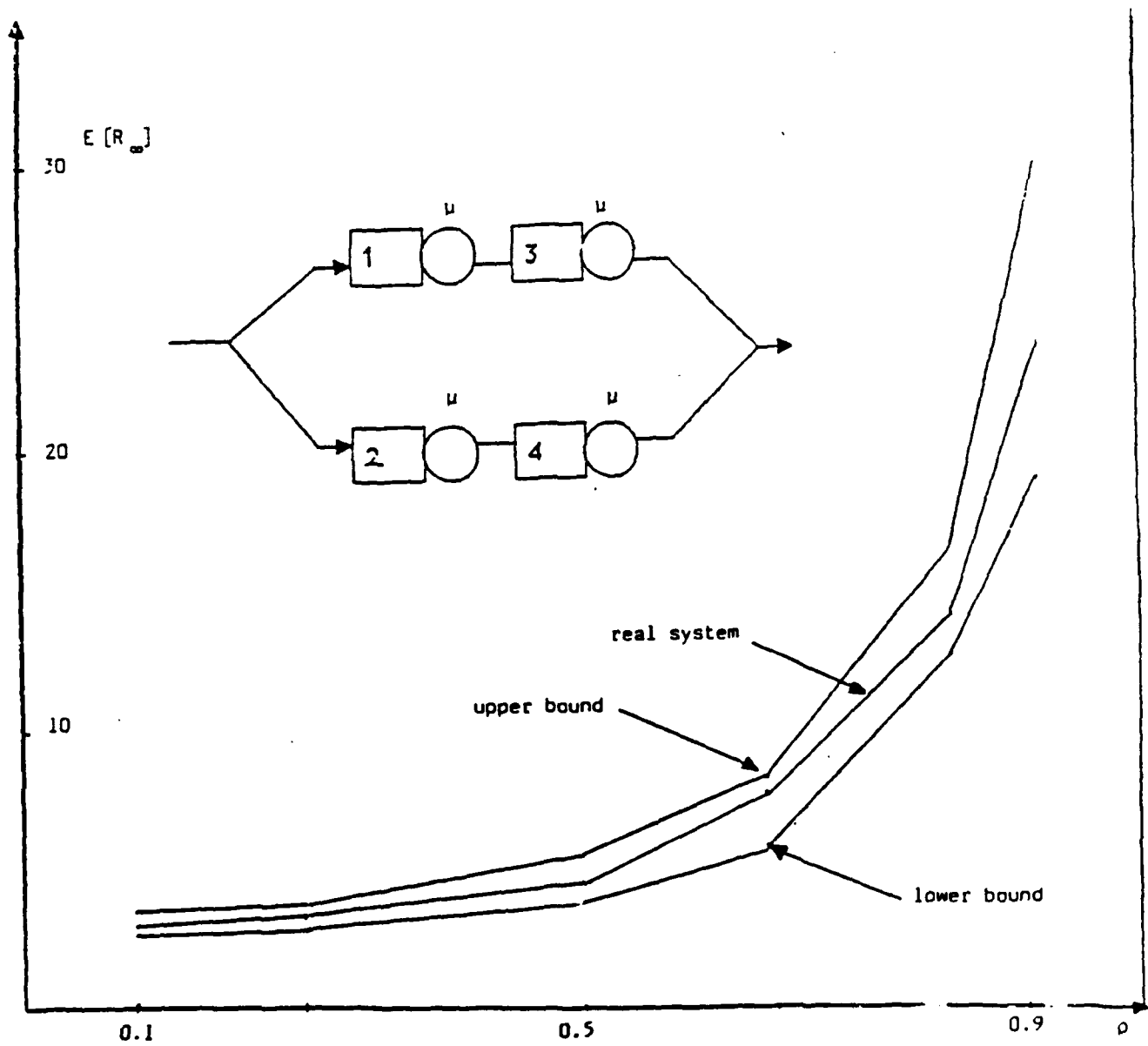


(b)

(a) A parallel network

(b) A series network

FIGURE 3



Arrival process : Poisson of parameter λ

Service times : exponentially distributed of parameter μ

BIBLIOGRAPHY

- [Ba 85] F. Baccelli. "Two Parallel Queues Created by Arrivals with Two Demands: The M/G/2 Symmetrical case," Report INRIA No. 426, July 1985.
- [BGP 84] F. Baccelli, E. Gelenbe and B. Plateau. "An End to End Approach to the Resequencing Problem," *JACM*, VI. 31 No. 3, july 1984, pp.474-485.
- [BM 85] F. Baccelli, A. Makowski, "Simple Computable Bounds for the Fork-Join Queue," *Proc. Conf. Inform. Sci. Systems*, John Hopkins Univ. pp 436-441. March 1985.
- [BM 85b] F. Baccelli, W.A. Massey, "Series-Parallel, Fork-Join Queueing Networks and Their Stochastic Ordering," AT&T Bell Laboratories Memorandum, May 1985.
- [BM 86] F. Baccelli, A. Makowski. "Stability and Bounds for Single Server Queues in Random Environment," *Stochastic Models*, Vol.2, n. 2, Marcel Dekker. 1986.
- [BMS 87] F. Baccelli, A. Makowski and A. Shwartz, " Fork-Join Queue and related systems with synchronization constraints: Stochastic ordering, approximations and computable bounds ," Electrical Engineering Technical Report, University of Maryland, College Park, jan. 87.
- [BMT 87] F. Baccelli, W. A. Massey and D. Towsley, "Acyclic Fork-Join Queueing Networks", Internal Report, Computer Sc. Dept., University of Massachusetts, April 1987.
- [BP 75] R. Barlow, F. Proschan, "Statistical Theory of reliability and life testing," Holt, Rinehart and Winston, 1975.
- [Br 75] P. Brinch Hansen, " The Programming Language Concurrent Pascal," *IEEE Trans. Soft. Engng.*, SE-1, pp. 199-207, June 1975.
- [FH 84] L. Flatto, S. Hahn, "Two Parallel Queues Created by Arrivals with Two Demands,I," *SIAM J. Appl. Math.* Vol. 44, pp. 1041-1053, 1984.
- [Co 82] J.W. Cohen . " The Single Server Queue," North Holland, 1982.
- [Ha 84] B. Hajek. " The Proof of a Folk Theorem on Queueing Delay with Applications to Routing in Networks," *JACM* Vpl. 30, pp. 834-851, 1983.
- [HHK 79] U. Herzog, W. Hoffmann, W. Kleinöder, "Performance Modeling and Evaluation for Hierarchically Organized Multiprocessor Computer Systems," proceedings of the international conference on parallel processing, 1979, pp. 103-114.
- [Ho 78] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall International, London, 1985.
- [Ho 79] W. Hoffmann, "Queuing Models for Parallel Processing and their Application to a Hierarchically Organized Multiprocessing System," *Proc. 1st European Conf. Parallel Distr. Proc.*, Toulouse France, pp. 221-227, Feb. 1979.
- [KW 85] C.P. Kruskal, A. Weiss. "Allocating Independent Subtasks on Parallel Processors," *Trans. Soft. Engng.* SE-11, pp. 1001-1016, Oct. 1985.
- [Ma 85] W.A. Massey, "Asymptotic Analysis of the Time Dependent M/M/1 Queue," *Mathematics of Operations Research*, Vol. 10, No. 2, May 1985, pp 305-327.
- [Nels85] R. Nelson, A.N. Tantawi, "Approximate Analysis of Fork/Join Synchronization in Parallel Queues," IBM Report RC11481, Oct. 1985.

- [NTT 87] R. Nelson, D. Towsley, A.N. Tantawi, "Performance Analysis of Parallel Processing Systems," to be presented at SIGMETRICS'87, 1987.
- [Py 81] I.C. Pyle, *The Ada Programming Language*, Prentice-Hall International, London, 1981.
- [Ro 83] T Rolski, "Comparison Theorems for Queues with dependent inter-arrival times ", *Modelling and Performance Evaluation Methodology, Lecture Notes in Control and Information Sciences, 60*, Springer Verlag, 1984.
- [Si 87] K. Sigman, "Regeneration in Queues with Regenerative Input," submitted to *Queueing Systems*.
- [St 84] D. Stoyan. *Comparison Methods for Queues and Other Stochastic Models*, English translation (D.J. Daley editor), J. Wiley and Sons, New York, 1984.
- [TY 87] D. Towsley and S.P. Yu. "Bounds for Two Server Fork-Join Queueing Systems," submitted to *Operations Research*.
- [Wh 81] W. Whitt. "Comparing and Counting Processes and Queues," *Adv. Appl. Prob.* Vol. 13, pp. 207-220, 1981.
- [Wh 84] W. Whitt. "Minimizing Delays in the GI/G/1 Queue," *Opns. Res.* Vol. 32, pp. 41-51, 1984.

DISTRIBUTION LIST

addresses	number of copies
Williams, Alan N RADC/COTC Griffiss AFB NY 13441-5700	2
RADC/COVL GRIFFISS AFB NY 13441-5700	1
RADC/CAP GRIFFISS AFB NY 13441-5700	2
ADMINISTRATOR DEF TECH INF CTR ATTN: DTIC-DDA CAMERON STA EG 5 ALEXANDRIA VA 22304-6145	5
University of Massachusetts Department of Electrical Engr & Computer Engr Attn: Christos Cassandras Amherst, Massachusetts 01003	4
SDIO/S-BM (Lt Col Sowa) The Pentagon Washington DC 20301-7100	1
SDIO/S-BM (Lt Col Rirad) The Pentagon Washington DC 20301-7100	1

IDA (SDIO Library) 2
(Albert Ferrella)
1801 N Beauregard Street
Alexandria VA 22311

SAF/AGSD (Lt Col Ben Greenway) 1
The Pentagon
Washington DC 20330

AFSC/CV-D (Lt Col Flynn) 1
Andrews AFB MD 20334-5000

HQ SD/XR (Col Heimach) 1
PO Box 92960
Worldway Postal Center
Los Angeles CA 90009-2960

HQ SD/CNI (Col Hchman) 1
PO Box 92960
Worldway Postal Center
LA CA 90009-2960

HQ SD/CNk 1
PO Box 92960
Worldway Postal Center
Los Angeles CA 90009-2960

ESD/AT (Col Paul) 1
Hanscom AFB MA 01731-5000

AFSTC/XX (Lt Col Detucci) 1
Kirtland AFB, NM 87117

USA SEC/CASD-H-SE (Larry Tubbs) 1
PO Box 1500
Huntsville, AL 35807

ANSER Corp 1
Suite 300
Crystal Gateway 3
1215 Jefferson Davis Highway
Arlington VA 22202

AFOTEC/XPP (Capt Wrobel) 1
Kirtland AFB NM 87117

AF Space Command/XPXIS 1
Peterson AFB CO 80914-5001

Director NSA (V43 George Hoover) 1
9800 Savage Road
Ft George G Meade MD 20755-6000

SDIO/S-BM (Capt Hart) 1
The Pentagon
Washington DC 20301-7100

SDIO/S-BM (CCR Newton) 1
The Pentagon
Washington DC 20301-7100

HQ SD/CN (COL WILKENSCH) 1
PO BOX 97960
WORLDWAY POSTAL CENTER
LA CA 90009-2960

HQ SD/CNIS
(LT COL FENNELL)
PO BOX 92960
WORLDWAY POSTAL CENTER
LA CA 90009-2960

1

HQ SD/CWX
PO BOX 92960
WORLDWAY POSTAL CENTER
LA CA 90009-2960

1

HQ SD/CNE
PO BOX 92960
WORLDWAY POSTAL CENTER
LA CA 90009-2960

1

ESD/ATS (LT COL CLDENBERG)
HANSCOM AFB MA 01731-5000

1

ESD/ATN (LT COL LEIB)
HANSCOM AFB MA 01731-5000

1



MISSION of Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C³I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C³I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.