QUEUEING NETWORK MODELS FOR LOAD BALANCING IN DISTRIBUTED SYSTEMS

E.DE SOUZA E SILVA
M. GERLA *

NCE-06/88

Abril/88

Universidade Federal do Rio de Janeiro
Núcleo de Computação Eletrônica
Caixa Postal 2324
20001 - Rio de Janeiro - RJ
BRASIL

* UCLA Computer Science Department

Esta pesquisa contou com o apoio financeiro da NSF-USA INT-8514377 e CNPq.

Este trabalho foi publicado como relatório técnico do Departamento de Ciência da Computação da UCLA CSD-870069, Novembro 1987.

Resumo

Em sistemas distribuídos, o balanceamento de carga pode melhorar a eficiência de um sistema se *jobs* executando em computadores com elevada carga de trabalho forem transferidos para computadores com menor carga. Neste artigo apresentamos um método para o balanceamento ótimo de carga em um ambiente estático. Um modelo de redes de filas é usado para avaliar o tempo de resposta e técnicas de programação matemática são usadas para se achar a alocação de carga que minimiza o tempo médio de resposta. O método não é proposto como substituto para políticas heurísticas dinâmicas de balanceamento de carga; entretanto, o método é visto como uma ferramenta útil para alocação de recursos e planejamento de capacidade em sistemas distribuídos, e como um complemento promissor a políticas dinâmicas em estratégias híbridas de balanceamento de carga.

O método pode ser aplicado a diversas classes de problemas, incluindo: classes distintas de *jobs*; *jobs* com múltiplas tarefas, e; *jobs* que originam múltiplas tarefas. Vários exemplos ilustrando estas aplicações são apresentados.

Abstract

In distributed systems, load balancing can improve efficiency by migrating jobs from heavily loaded to lightly loaded sites. In this paper we present a method for optimal load allocation in a static environment. A queueing network model is used to evaluate response time; and mathematical programing techniques are used to find the load allocation that minimizes average response time. The method is not proposed as a substitute for dynamic, heuristic load balance policies; rather, it is perceived as a useful tool for resource allocation and capacity planning in distributed systems, and as a promising complement to dynamic policies in hybrid load balance strategies.

The method can handle very general classes of problems, including: distinct classes of jobs, multitasking within each job, and; jobs with spawned tasks. Several examples illustrating these applications are reported.

1 Introduction

The past few years have witnessed an increasing number of distributed computer system implementations based on local area networks. In these systems a number of resources (CPU's, file servers, disks, etc.) are shared among jobs originating at different computer sites. In a distributed system environment, it is desirable to equalize the usage of resources (balance the load) in order to reduce the response time of jobs and improve the utilization of the resources. This can be achieved by migrating jobs to the lightly loaded sites. An example of system permitting job migration is the LOCUS operating system [POPE81], where processes generated by users at one site in the network are allowed to run on other sites.

The load balancing problem in a distributed resource system is not new and can take different forms for different problems. For example, in a long haul packet computer network (where the resources are the channels), load balancing becomes the routing problem. The goal there is to find optimum paths on which to distribute the packets, so that some well defined performance criterion (overall delay, for instance) is optimized.

In a distributed computer system, the load balancing problem may be formulated as the problem of distributing the execution of processes throughout the network in such a way that the overall user response time is minimized. This load balancing problem is more complex than the routing problem for the following reasons:

(a) **Optimal selection** of the execution site and **optimal routing** from the originating site to the execution site must be simultaneously accomplished (although the routing problem may become trivial when a bus or ring network is used since in this case there is only one path between origin and execution site).

(b) A job in general consists of a **sequence of tasks**. The selection of the best site on which to run a task must be made on a task-by-task basis.

(c) **Multiple job classes** must be considered (e.g., interactive, batch, etc.) with drastically different resource usage characteristics. In particular, some classes behave as "closed" classes, in that their population remains constant during the life of the system (e.g., the number of interactive jobs in a multiprogrammed system). Other classes are best modeled as "open", in that the number of users in the class fluctuates statistically due to random inputs (e.g., database inquires originating from a very large terminal population).

(d) Some classes may be **restricted** to run on a subset of sites.

(e) A job may occasionally "spawn" other jobs. For example, an interactive user engaged in editing a file may periodically submit a text formatting (e.g., TROFF or

TEX) job to one of the servers. Since some of the users may be homed on server sites, proper balancing of spawned jobs is necessary to minimize user delay (and maximize user throughput).

Load balancing can be **static** or **dynamic**. Static load balancing implies finding the optimal job assignment at steady state. Dynamic load balancing requires that the job (or task) allocation to a site be made when the job (task) is generated, depending on the current traffic conditions. Needless to say, dynamic load balancing is much more difficult to analyze than static load balancing. In this paper, we limit ourselves to the static load balancing case, i.e., we assume that the system will reach steady state, and we seek the best distribution of jobs among sites at steady state, so that a proper performance measure, typically, the average delay, is optimized. The solution to the static load balancing problem is probabilistic in nature, in that it specifies the fraction of jobs in a given class which must be allocated to a given site. That is, with probability equal to that fraction, a job should be routed to that site. This is analogous to the optimal static routing solution in computer communication networks, where the traffic between origin and destination is probabilistically split among multiple paths ("bifurcated routing").

It may be argued that the usefulness of static load balancing solutions is very limited since operational systems implement dynamic procedures. The controversy is the same as that between static and dynamic routing policies in packet networks; and the answer is the same: they are both necessary, since they play different roles. First, static models are computationally effective for system sizing (e.g. allocation of resources, identification of bottlenecks, sensitivity studies, etc), while dynamic models are not suitable for this function (analytic models can handle only very small size systems; simulation models are too time consuming). Secondly, static solutions may actually be exploited for dynamic load balancing: a network management center can monitor loads and traffic patterns, periodically recompute the optimal load distribution, and dispatch load balancing instructions to the hosts. This periodic computation of optimal load balancing can be carried out also in a distributed fashion, without the network management center, as is done in optimal, distributed routing algorithms [GALL77]. The possibility also exists of combining the optimal static policy with the heuristic, dynamic policy: the static policy is used to optimize loads according to long term traffic trends, while the heuristic policy is used to react to sudden traffic changes and temporary perturbations.

The above arguments assume that static, optimal solutions are consistent with dynamic, heuristic solutions, so that a system can be sized using static models and then expected to perform well under heuristic rules. There is evidence that this is the case in distributed systems (as it happens also for optimal and heuristic routing in packet networks); however, additional, more systematic experimentation is required in this direction to corroborate the claim.

Most static load balancing problems can be formulated as non-linear, multicommodity flow problems. If the objective function is convex, a downhill search technique can be efficiently used for their solution. If the system includes only open classes and does not contain special features (such as site restriction, job multitasking, job spawning, etc), then any of the methods available for routing optimization in computer networks can be successfully used. In particular, the Flow Deviation method, a downhill search method specially designed for open queueing networks may be employed [FRAT73,KLEI76]. Furthermore, if the distributed system has a special structure (namely, there is only one class of jobs and the local network can be modeled by a single queue) the optimal equilibrium point may be obtained directly by solving (numerically) a set of non-linear equations, rather than using an iterative downhill descent method. An elegant "direct" solution method for this special structure was presented in [TANT85]. Unfortunately, the direct solution does not easily extend to multiple classes, site constraints, and general interconnecting networks.

If the distributed system under consideration includes only closed chains (i.e., closed classes), then an extension of the Flow Deviation method due to Kobayashi and Gerla [KOBA83] can be efficiently used. However, the delay function in this case is not convex, therefore, the routing problem with multiple closed chains typically leads to local minima. Thus, a further search for the minimum of the local minima is required.

In this paper we consider the more general situation of multiple mixed classes of jobs. A job in general consists of a sequence of tasks. As mentioned before, tasks of the same job can be run (in sequence) at different sites. A task, however, must be run to completion before the job is reallocated to another site.

Open classes correspond to jobs submitted from a large number of terminals or work stations, or spawn from other tasks (such as text formatting jobs spawn from interactive users, as mentioned before). These jobs may be permitted to execute remotely. At issue is the optimal selection of the remote site, and the optimal path to it. Restrictions may apply in the remote dispatcher. Closed classes correspond to jobs running locally on a computer site. The computer site is itself modeled as a network of queues (central server model).

The objective of our load balancing problem is to minimize a weighted sum of delays (over open and closed chains). Note that only the open chains need to be rerouted for load balancing. Routing is fixed within the closed chains. It can be shown that this guarantees the existence of only one local minimum, which is also the global minimum.

In summary, the main contribution of this paper is to provide an efficient algorithm for the solution of a very general class of static load balancing problems. Key capabilities of the algorithm is the handling of multitasking, sites constraints and job spawning. Examples will be presented to illustrate all the features.

In the following, in section 2 we describe in detail the model which will be used. In section 3 we present the solution approach. In sections 4 and 5 we demonstrate the application of our method to a few selected examples including site constraints, multitasking and spawned jobs. In section 6 we present our conclusions.

2 The Model.

We consider a distributed system of the type shown in Figure 1. In this system each site

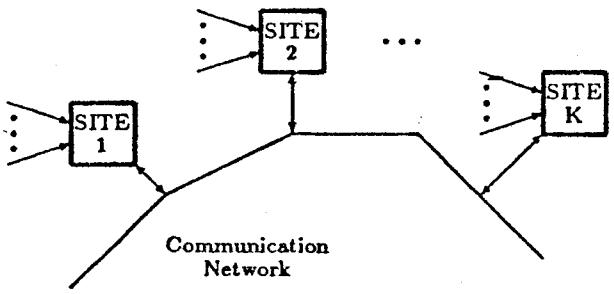

Figure 1: A Distributed System.

supports a number of resources (CPU's, disks, etc.), which are used by local processes. Sites may have different resource configurations, different capacities, etc. Each site is connected to a number of terminals which generate jobs with different processing requirements. A job consists of one or more tasks. Each task may require specialized resources, and therefore may execute only on a subset of the sites, since not all sites possess all needed resources. Tasks within a job are executed in sequence and, for each task, a "run site" is chosen. If the "run site" of a task is different from the site where the task was generated then all the information necessary for the task to run in this "foreign site" is transferred via the communication network. We assume that, once a "run site" is chosen for a task, it runs until completion on that site and no further transfers are allowed. A job completes execution after all individual tasks are executed.

This type of configuration may arise in a database application such as an airline reservation system where requests from local terminals may be forward to foreign sites. In addition, some of these sites are processing local application programs. We assume that application programs cannot execute in a foreign site. We refer to these applications as the "local load" of a site.

Our goal is to balance the load in the computer network, so that the overall delay is minimized. As a byproduct, the model will permit us to investigate several performance issues. For instance, we will be able to study the influence of local loads on the dispatching of database requests to sites; the effect of the communication network speed on load distribution, etc.

We model our distributed system as a collection of "central server models", one for each site, interconnected by the communication network queueing model. This kind of model was first proposed in [GOLD83] to model the LOCUS distributed system.

Local application programs (corresponding to the local load) are modeled by one or more closed chains for each site, as indicated in Figure 2. (In this figure we assume that these jobs are generated by a finite population of interactive users.) The remaining jobs

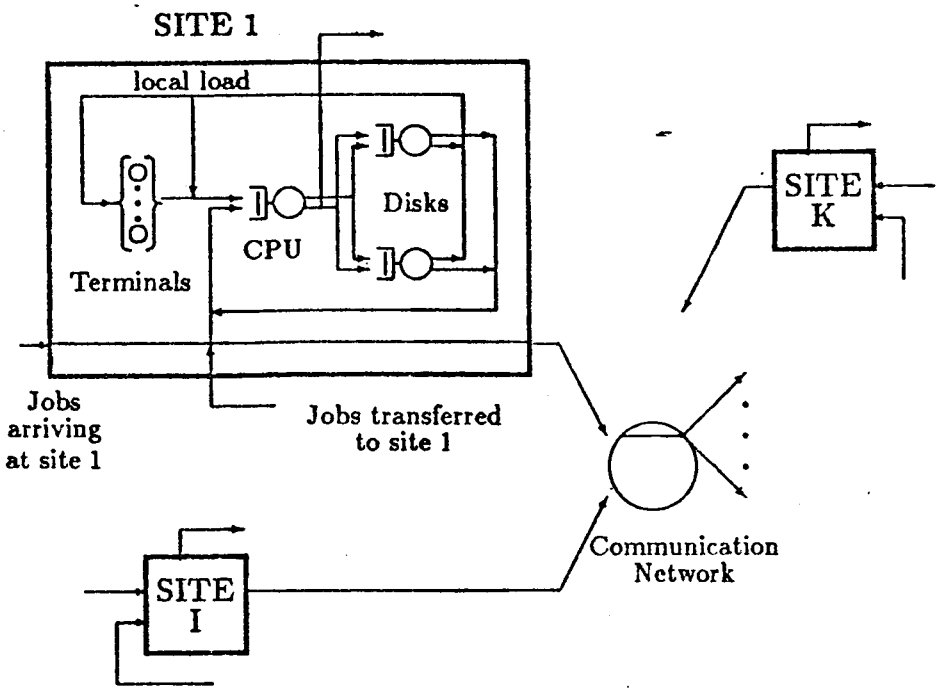

Figure 2: The model.

are generated by a large number of terminals, and so can be modeled as open chains with Poisson arrivals and total throughput Λ^o. (In section 5 we will investigate the case where jobs are generated from the local interactive population.) Jobs representing the local load are restricted to run on the site where they were generated. All other jobs in the model (represented by open chains) may execute at more than one site. These jobs require the execution of several tasks in sequence before completion. Suppose that a request r_i is composed by tasks $t_1, \ldots, t_M$. Upon arrival at site i, say the first task of r_i (t_1) may be executed locally or the request may be immediately forwarded to one of the other sites in the system which can execute task t_1. Task t_1 runs until completion in the chosen site. Then, a new site is chosen to execute task t_2, and the job completes when all M tasks are executed. We assume that the decision of running a task in a site is independent of the state of the network. Furthermore, the network is composed by single server fixed rate (SSFR) and infinite server (IS) service centers, and queue disciplines and service demands are such that they satisfy product form requirements.

The modeling of multiple tasks is handled as follows. Let us assume that each open chain job is composed by only one task which can be assigned to one of the sites that is able to execute this task. In this case, the "routing" of a job through the network is very

simple since, once assigned to a site, a job executes until it completes. Figure 3 shows the possible paths for a job of an open chain v in a distributed system with three sites (1, 2 and 3) and a communication network. We assume that site 1 is the local site of chain v

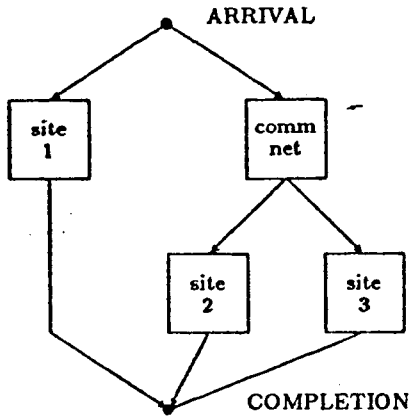

Figure 3: Possible paths of chain v jobs with only one task.

jobs and that they can be executed in any of the three sites. In this figure, we see that only three paths are possible: a job may execute in its local site (1) or it may be sent via the communication network to foreign sites 2 or 3. If a job has multiple tasks to be executed, the route of this job through the distributed system is more complex, since the job can visit several sites before completion. It is easy to see that the route of a chain v job through the system can be modeled by introducing classes for that chain, with one class mapped to each task of a job. Since a queueing network model with chains having multiple classes can be solved by an equivalent system with the same number of chains but with only one class [LAVE83], the introduction of classes does not increases the computational complexity of the model. However, the algorithm has to take into account a more complex routing structure. This is illustrated in Figure 4. In that figure, we assume that chain v jobs are composed of two tasks (t_1 and t_2). Task t_1 can only be executed in sites 1 or 2 and task t_2 can only be executed in sites 2 or 3. We see that there are four possible paths of execution for a job. In one of these paths, a job is sent through the communication network to have its first task executed at site 2. Then the job is sent again through the communication network to have its second task executed at site 3.

The following notation will be used throughout the paper:

J	=	total number of service centers in the network, including centers representing the communication network.
K	=	total number of chains in the network.
$j(k)$	=	a specified service center visited by chain k.
θ_{jk}	=	visit ratio of chain k jobs to center j, scaled so that $\theta_{j(k)k} = 1$.

T_{jk}	=	mean service time of chain k jobs at center j.
a_{jk}	=	$\theta_{jk}.T_{jk}$.
ρ_j	=	utilization of center j.
μ_{jk}	=	$1/T_{jk}$.
λ_{jk}	=	$\theta_{jk}.\lambda_k$ = throughput of chain k jobs at center j, where $\lambda_k = \lambda_{j(k)k}$.
L_{jk}	=	mean number of chain k jobs at center j.
W_{jk}	=	mean waiting time (queueing time + service time) of chain k jobs at center j.
$\vec{e}_k$	=	k-dimensional vector whose k-th element is one and whose other elements are zero.

In addition a superscript c denotes a quantity for a closed chain and a superscript o denotes a quantity for an open chain. For simplicity of notation we assume that there is at most one closed chain in each site representing the local load. Therefore, the index of the closed chain identifies the site. The results, however hold for multiple closed chains at a site. Let N_k be the number of jobs in the closed chain of site k, and $\vec{N} = (N_1, \ldots, N_{K^c})$.

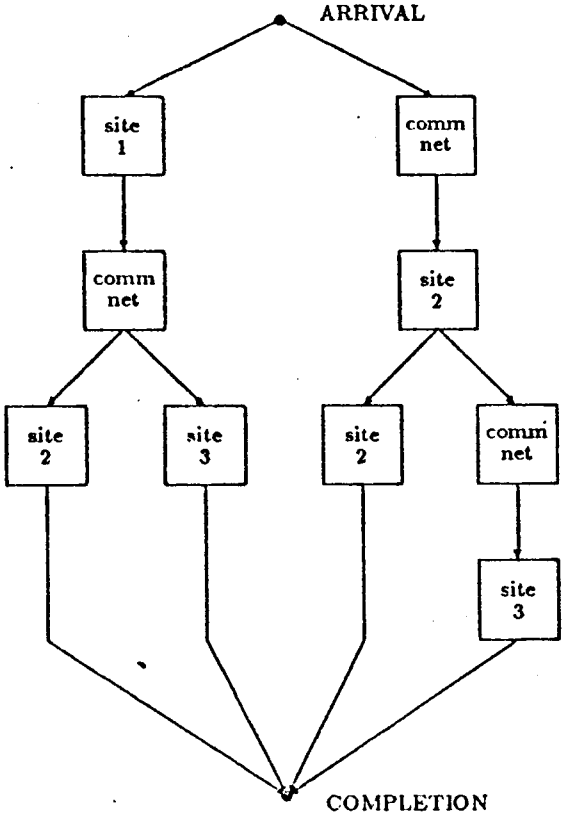

Figure 4: Possible paths of chain v jobs with two tasks.

We define the overall delay $D(\vec{N})$ as the following weighted sum of all chain delays:

$$D(\vec{N}) = \frac{w^o \sum_{v=1}^{K^o} \lambda_v^o D_v^o(\vec{N}) + w^c \sum_{k=1}^{K^c} \lambda_k^{Nc}(N_k) D_k^c(N_k)}{w^o \sum_{v=1}^{K^o} \lambda_v^o + w^c \sum_{k=1}^{K^c} \lambda_k^{Nc}(N_k)} \quad (1)$$

where: (1) $D_v^o(\vec{N})$ and $D_k^c(N_k)$ represent the response time of open chain v jobs and closed chain k jobs, respectively; (2) $\lambda_k^{Nc}(N_k)$ represents the "nominal" throughput of a closed chain k obtained when all open chain throughputs are set to zero; (3) w^o and w^c are arbitrary constant weighting factors for open and closed chains, respectively. The use of the nominal (instead of actual) throughput as a weighting factor is justified as follows: if the nominal throughput for a closed chain is replaced by the actual throughput, and the weights are set to 1, the expression above gives the total average delay. However, using total average delay as the objective function leads to unfairness to closed chains, since an increase in open chain utilization at a site causes an increase in local chain delay, and also a decrease in local chain throughput (the product remains constant by Little's result). This implies that for very heavy loads from external sources the impact of closed chain delays on the objective function becomes negligible with respect to the open chain delays. For this reason, we chose to assign a fixed coefficient, the nominal throughput, to each closed chain. In addition, we introduced the weighting factors $\{w\}$ to reflect the relative importance of open and closed chains in the model. Without loss of generality we assume $w^o = w^c = 1$ throughout the rest of the paper.

The goal of load balancing is to minimize the non-linear function $D(\vec{N})$ by optimally distributing the open chain traffic among the various sites. This is equivalent to optimizing $D(\vec{N})$ with respect to the flows $\{\lambda_{jv_t}^o\}$, $\forall\, j, v_t$, where the subscript t identifies a task of an open chain v job.

3 The Solution Approach.

We define our load balancing problem as follows:

- **Given:**
  - The number of tasks in each job.
  - The service demands of jobs (and their tasks) from all chains at all service centers in the network.
  - The visit ratios of the closed chain jobs (local load jobs) of a site at each service center at that site, as well as the visit ratios of open chain jobs at centers of a site.
  - The number of interactive jobs representing the local load at each site.
  - The throughput rate of each open chain v.
  - The set of sites where task t of open chain v jobs can execute.
  - The "local" site of each open chain job.
- **Minimize:** The overall delay $D(\vec{N})$.
- **With respect to:** The flows at each service center $\lambda^o_{jv_t}$. (Where the subscript t identifies a task of an open chain v job.)

The solution method we use is a downhill technique based on the "flow deviation" method [FRAT73,KLEI76]. We restrict the development to a single closed chain per site as previously indicated, for notational convenience. However, the approach is applicable to multiple closed chains at each site. To compute the steepest descent direction for the downhill technique we need to compute the (partial) derivatives of the overall delay function with respect to each open chain flow at each service center. For that we manipulate the expression for the overall delay in equation (1) as follows.

From Little's result:

$$D^o_v(\vec{N}) = \frac{\sum_{j=1}^{J} L^o_{jv}(\vec{N})}{\lambda^o_v} \tag{2}$$

and

$$D^c_k(N_k) = \frac{N_k - \lambda^c_k(N_k) T^c_{term,k}}{\lambda^c_k(N_k)} \tag{3}$$

where $T^c_{term,k}$ is the average think time of an interactive closed chain k job at the terminal, and $\lambda^c_k(N_k)T^c_{term,k}$ gives the average number of closed chain k jobs at the terminals.

Substituting (2) and (3) into (1),

$$D(\vec{N}) = \frac{\sum_{j=1}^{J} L_j^o(\vec{N}) + \sum_{k=1}^{K^c} \lambda_k^{Nc}(N_k)(N_k/\lambda_k^c(N_k) - T_{term,k}^c)}{\Lambda^o + \sum_{k=1}^{K^c} \lambda_k^{Nc}(N_k)} \tag{4}$$

Then, we observe that a mixed product form queueing network with SSFR and IS service centers can be reduced to an equivalent closed queueing network where the service request $T_{jk}^{'c}$ of closed chain k jobs at center j is given by the following expression [LAVE83]:

$$T_{jk}^{'c} = \frac{T_{jk}^c}{1 - \rho_j^o} \tag{5}$$

We note that in our model, closed chain jobs (local load) at a site do not interfere with closed chain jobs from another site. Therefore, we can decompose our model into K^c independent mixed network models each with one closed chain, plus the queueing model of the communication network.

Finally, we recall that in a mixed network the open chain population at center i, L_i^o, can be expressed in terms of the closed chain by [LAVE83]

$$L_i^o(N_k) = \begin{cases} \frac{\rho_i^o}{1-\rho_i^o}[1 + L_i^c(N_k)] & i \in \text{site } k \text{ or } i \in \text{commun. network}, i \neq \text{IS center} \\ \rho_i^o & i \in \text{site } k \text{ or } i \in \text{commun. network}, i = \text{IS center} \end{cases} \tag{6}$$

where the vector notation was dropped due to the reduction of the problem to $K^c + 1$ independent single chain mixed networks.

Therefore, equation (4) can be rewritten as:

$$D(\vec{N}) = \frac{\sum_{\substack{i=1 \\ i \neq IS}}^{J} \frac{\rho_i^o}{1-\rho_i^o}[1 + L_i^c(N_k)] + \sum_{\substack{i=1 \\ i=IS}}^{J} \rho_i^o + \sum_{k=1}^{K^c} \lambda_k^{Nc}(N_k)[N_k/\lambda_k^c(N_k) - T_{term,k}^c]}{\Lambda^o + \sum_{k=1}^{K^c} \lambda_k^{Nc}(N_k)} \tag{7}$$

The independent variables in our optimization problem are the $\{\lambda_{jv_t}^o\}$. Recalling that $\rho_j^o = \sum_{v_t} \lambda_{jv_t}^o T_{jv_t}^o$, we note that we can also use the ρ_j^o as independent variables. In the following, for convenience, we take the derivative of $D(\vec{N})$ with respect to the total utilization of open chain jobs at center j (ρ_j^o). In this computation we exploit the fact that: (1)

the performance measures of site (or communication network) u are not altered when we vary the open chain throughput at a center j in site (or communication network) k, $k \neq u$; and, (2) the total open throughput remains constant as do the nominal throughputs of the closed chains. For convenience of notation, we label the centers belonging to the same site (or communication network) as center j as $l, \ldots, L$.

$$\frac{\partial D(\vec{N})}{\partial \rho_j^o} = \begin{cases} \dfrac{\dfrac{1+L_j^c(N_k)}{(1-\rho_j^o)^2} + \sum\limits_{\substack{i=l \\ i \neq \text{IS}}}^{L} \dfrac{\rho_i^o}{1-\rho_i^o} \dfrac{\partial L_i^c(N_k)}{\partial \rho_j^o} - \delta(j) \dfrac{\lambda_k^{Nc}(N_k) N_k}{(\lambda_k^c(N_k))^2} \dfrac{\partial \lambda_k^c(N_k)}{\partial \rho_j^o}}{\Lambda^o + \Lambda^{Nc}(\vec{N})} & j \neq \text{IS} \\ \dfrac{1}{\Lambda^o + \Lambda^{Nc}(\vec{N})} & j = \text{IS} \end{cases} \quad (8)$$

where j is a center which belongs to site k or the communication network, $\Lambda^{Nc}(\vec{N}) = \sum_{k=1}^{K^c} \lambda_k^{Nc}(N_k)$, and

$$\delta(j) = \begin{cases} 1 & j \notin \text{communication network} \\ 0 & \text{otherwise} \end{cases}$$

It remains to find the derivatives of the closed chain throughputs and queue lengths with respect to the utilization of open chain jobs at center j. To this end, we use the following theorems and corollary:

Theorem 1 *In a product form, mixed queueing network model*

(a)

$$\frac{\partial L_j^c(N_k)}{\partial \rho_j^o} = \frac{V_j^c(N_k)}{(1-\rho_j^o)} \quad (9)$$

(b)

$$\frac{\partial L_l^c(N_k)}{\partial \rho_j^o} = \frac{V_{jl}^c(N_k)}{(1-\rho_l^o)} \quad (10)$$

where $V_j^c(N_k)$ is the variance of the queue length of closed queue jobs at center j of site k and $V_{jl}^c(N_k)$ is defined as the covariance of queue lengths of closed chain jobs at centers j and l of site k.

Corollary 1 *In a product form mixed queueing network model, the variances and covariances in (9) and (10) above can be easily obtained using mean value analysis (MVA) and the following recursive equations (the subscript k in N_k was dropped to simplify the notation):*

$$V_j(N) = \lambda(N)\frac{a_j}{1-\rho_j}([1+L_j(N-1)-L_j(N)][1+L_j(N-1)]+V_j(N-1)) \quad (11)$$

$$V_{jl}(N) = \lambda(N)\frac{a_l}{1-\rho_l}([1+L_j(N-1)-L_j(N)][1+L_l(N-1)]+V_{jl}(N-1)) \quad (12)$$

where $V_j(0) = V_{jl} \stackrel{\text{def}}{=} 0$.

Theorem 2 *In a product form, mixed queueing network model*

$$\frac{\partial\lambda_k^c(N_k)}{\partial\rho_j^o} = \frac{\lambda_k^c(N_k)}{1-\rho_j^o}\left[L_j^c(N_k-1)-L_j^c(N_k)\right] \quad (13)$$

where service center j is assumed to be in site k.

A proof of the above theorems and corollary can be found in [DeSO84a] and is included in the appendix for completeness. For a general discussion of partial derivatives of queueing measures with respect to parameters in closed queueing network models we refer to [DeSO84b].

Finally, substituting (9), (10) and (13) into (8) and noting that $\partial D(\vec{N})/\partial\lambda_{jv_t}^o = (1/\mu_{jv_t}^o)\partial D(\vec{N})/\partial\rho_j^o$, where the subscript t identifies a class of open chain v, we obtain:

$$\frac{\partial D(\vec{N})}{\partial\lambda_{jv_t}^o} = \quad (14)$$

$$\begin{cases} \dfrac{1+L_j^c(N_k)}{1-\rho_j^o}+\dfrac{\rho_j^o V_j(N_k)}{1-\rho_j^o}+\displaystyle\sum_{\substack{i=1\\ i\neq j\\ i\neq IS}}^{L}\frac{\rho_i^o V_{ij}(N_k)}{1-\rho_i^o}+\delta(j)\frac{\lambda_k^{Nc}(N_k)}{\lambda_k^c(N_k)}N_k[L_j^c(N_k)-L_j^c(N_k-1)] \\ \hline \mu_{jv_t}^o(1-\rho_j^o)(\Lambda^o+\Lambda^{Nc}(\vec{N})) \\ j\neq \text{IS} \\ \\ \dfrac{1}{\mu_{jv_t}(\Lambda^o+\Lambda^{Nc}(\vec{N}))} \qquad\qquad j=\text{IS} \end{cases}$$

Equation (14) is given in terms of mean values of throughputs and queue lengths, variance and covariance of queue lengths which can be easily obtained from MVA recursion and equations (11) and (12).

The key to the flow deviation method is to associate a length or weight for an open chain job to each queue, given by:

$$l_{jv_t} \stackrel{\text{def}}{=} \frac{\partial D(\vec{N})}{\partial \lambda^o_{jv_t}} \qquad j = 1, \ldots, J \tag{15}$$

However, due to the particular characteristics of our problem, we can further simplify the solution by noting that, once a task is assigned to a site, its behavior is preestablished by the routing probabilities given as input parameters for that site. In other words, the "route" of tasks within a site is fixed. This observation allows us to define a "site" weight, as:

$$lsit_{kv_t} \stackrel{\text{def}}{=} \sum_{j \in k} \theta^o_{jv_t} l_{jv_t} \tag{16}$$

where the index k represents the site and index v_t, class t of open chain v. In the same way, the weight of the communication network ($lnet_{v_t}$) can be defined. The weight given by (16) represents the linear rate at which $D(\vec{N})$ increases with an infinitesimal increase of the flow of open chain v, class t at site k. We also define $lsit_{kv_t} \stackrel{\text{def}}{=} \infty$ if task t of chain v jobs is not allowed to execute at site k.

We outline a flow deviation algorithm for load balancing (FDLB) in a distributed computer network of the type described in section 2. The algorithm is similar to the one described in [KLEI76].

We define the assignment tri-dimensional matrix $\vec{A}$ where element A_{kv_t} gives the percentage of jobs of type v, task t, assigned to site k. Similarly we define the network matrix $\vec{F}$ where element F_{v_t} gives the percentage of jobs of type v, task t, assigned to a foreign site, i.e., $F_{v_t} = \sum_{k \neq \text{home site of } v} A_{kv_t}$ We assume that we have an initial feasible assignment, i.e., initial values for $\vec{A}$ so that $\rho^o_j < 1$ for all service centers in the network which are not IS centers.

FDLB algorithm [1]:

Step 1: let $n = 0$ and let $\vec{A}^{(0)}$ be an initial feasible assignment. Identify all possible execution paths for jobs of a particular chain (see Figure 4 for an example).

Step 2: Compute weights l_{jv_t} using MVA and equations (11) and (12) for computing variances and covariances of queue lengths. Compute the weights of each site ($lsit_{kv_t}$) and the weight of the network ($lnet_{v_t}$) using equation (16).

[1] The description of the algorithm follows similar descriptions given in [KLEI76]. However, in the implementation, we deviate the flow for one chain at a time, since it was observed that this equivalent approach tends to reduce the overall number of iterations.

Step 3: For each class of open chain jobs, find the cheapest way to execute the job, i.e., find the matrix $\vec{A}^*$. If each job in the system has only one task, then this computation is trivial. If jobs have multiple tasks, any known optimum path algorithm can be used (e.g., see [KLEI76]). Find $\vec{F}$ as defined above.

Step 4: Compute the incremental delay $b^{(n)}$ and b^* for the assignment $\vec{A}^{(n)}$ and cheapest assignment $\vec{A}^*$, respectively:

$$b^{(n)} = \sum_{v_t} \left[\sum_k lsit_{kv_t} \times A^{(n)}_{kv_t} + lnet_{v_t} \times F^{(n)}_{v_t} \right]$$

$$b^{(n)} = \sum_{v_t} \left[\sum_k lsit_{kv_t} \times A^{*}_{kv_t} + lnet_{v_t} \times F^{*}_{v_t} \right]$$

Step 5: Stopping rule:
if $|b^{(n)} - b^*| < \epsilon$, where $\epsilon > 0$ is a properly chosen tolerance, stop. Otherwise go to Step 6.

Step 6: Find the value α in the range $0 \leq \alpha \leq 1$ such that the assignment $\vec{A}' = (1 - \alpha)\vec{A}^{(n)} + \alpha\vec{A}^*$ minimizes $D(\vec{N})$.
Let $\vec{A}^{(n+1)} = \vec{A}'$.

Step 7: Let $n = n + 1$ and go to Step 2.

In order to find a feasible initial assignment or, more generally, to determine whether a feasible solution exists we can apply one of the several methods already developed for routing in open networks [KLEI76].

The FDLB algorithm finds the global minimum for $D(\vec{N})$ due to the convexity of $D(\vec{N})$ with respect to the open chain flows and the convexity of the space of the feasible flows. The convexity of $D(\vec{N})$ can be deduced from the fact that the function $D(\vec{N})$ in (4) is a sum of convex functions over the open chain flows. An intuitive (but not rigorous) explanation for that can be given as follows: we observe that equation (4) can be decomposed into a weighted sum of the number of open chain jobs in a site plus the response time of the closed chain jobs in that site. We can verify that both the number of open chain jobs in a site and the response time of closed chain jobs in that site are increasing without bound (and with a non-decreasing rate) as a function of the open chain flow at that site. Thus, the delay of each site is a convex increasing function over the open chain flows, and the weighted sum of these delays is also convex. Convergence is guaranteed by the fact that each iteration provides an improvement in the objective function.

The amount of computation required by each iteration of the FDLB algorithm is only on the order of the computation of the "lowest cost path" in Step 3, and on the order of the cost

of the MVA algorithm for site k $\forall k$, since the closed chains at each site are independent, plus the cost of solving the queueing model of the communication network. Note that in Step 6 the MVA algorithm is repeated several times, and the MVA computation becomes the dominant term.

4 Applications.

In this section we present two applications which follow directly from the theory developed in previous sections, i.e., jobs which can execute at more than one site are independent of the local load and are composed by one or more tasks. In the next section we extend the theory to account for jobs which are spawned from jobs representing the local load.

In the first application we apply the FDLB algorithm to a distributed computer system consisting of 3 sites linked by a slotted ring, as illustrated in Figure 5. For the slotted ring, we used the closed chain model proposed by Bux [BUX81].

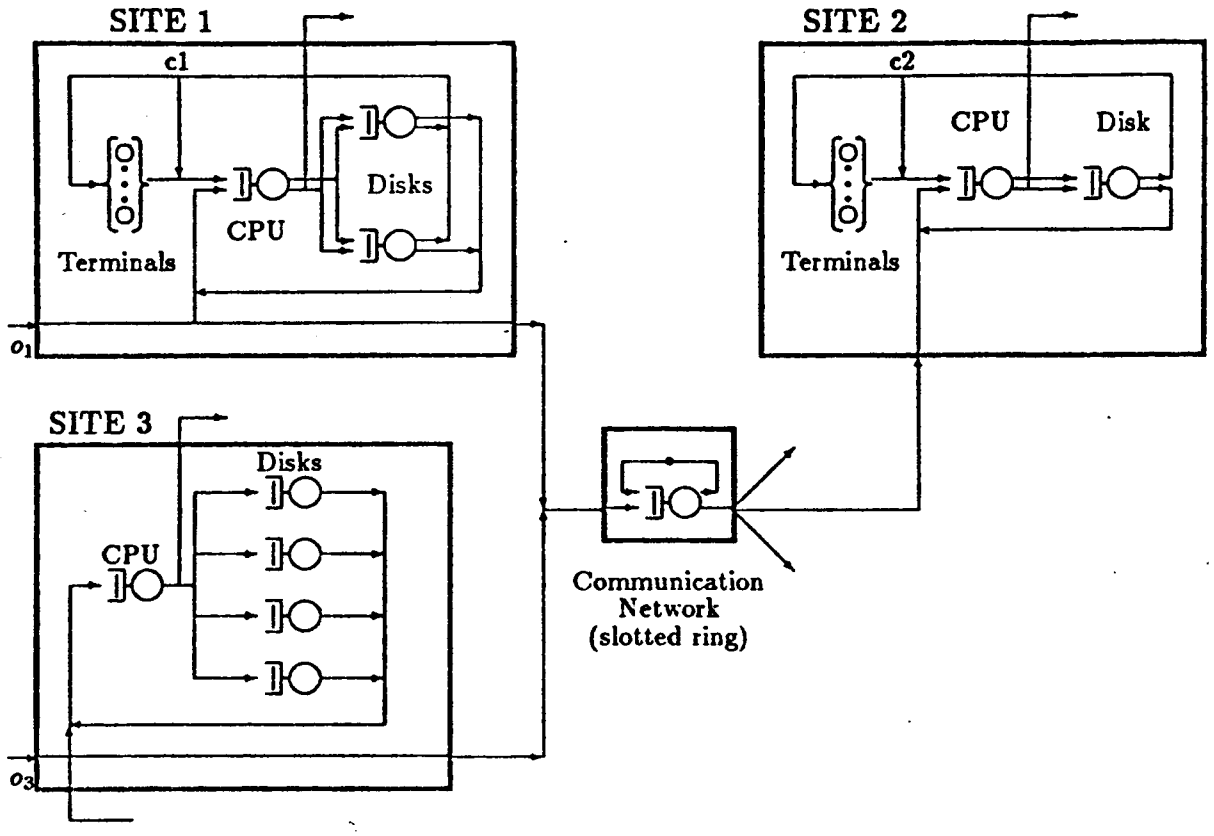

Figure 5: Application: a distributed computer system with 3 sites.

There are 2 classes of open chain jobs $o1$ and $o3$ arriving at sites 1 and 3, respectively. Class $o1$ can request service from any of the 3 sites, but class $o3$ can only request service from sites 2 and/or 3. Sites 1 and 2 are running local application programs, modeled as closed chains $c1$ and $c2$ respectively. Table 1 shows the visit ratios and service requirements for each class of job at each service center in the network, as well as other input parameters.

JOB CLASS	SITE	CENTER	TYPE	VISIT RATIOS	SERVICE TIMES (msec)
$o1$	1	CPU	PS	5	30
6 jobs/sec		disk(1 or 2)	FCFS	2	50
	2	CPU	PS	5	21
		disk	FCFS	4	15
	3	CPU	PS	5	30
		disk(1 ... 4)	FCFS	1	140
	com. net.		PS		10
$o3$	1	CPU	PS	5	50
3.5 jobs/sec	(*)	disk(1 or 2)	FCFS	2	50
	2	CPU	PS	5	35
		disk	FCFS	4	15
	3	CPU	PS	5	50
		disk(1 ... 4)	FCFS	1	140
	com. net.		PS		10
$c1$	1	terminals	IS	1	4000
5 jobs		CPU	PS	10	90
		disk(1 or 2)	FCFS	5	50
$c2$	2	terminals	IS	1	3000
X jobs		CPU	PS	10	40
(variable)		disk(1 or 2)	FCFS	10	15
(*) To illustrate the case where class $o3$ is not restricted to run at site 1.					

Table 1: Parameters for the first application.

As an illustration, let us consider the behavior of jobs $o1$ and $c1$ at site number 1. A job of class $o1$ spends an average of 30 msec at the CPU before issuing an I/O request. The service time of each I/O device is 50 msec. On the average, a job of class $o1$ visits the CPU 5 times and the I/O devices 4 times before completion. On the other hand, a job of class $c1$ spends an average of 90 msec at the CPU and 50 msec at each I/O device, and it visits the CPU and I/O devices 10 times before a visit to the terminals.

Figure 6 shows the percentage of foreign jobs processed at site 2 (after balancing the load) when the number of jobs at site 2 representing the local load (N_2) varies from 0 to 20. When $N_2 = 0$, site 2 processes 80% of the jobs of class $o1$ and 36% of the jobs of class $o3$. When $N_2 = 20$, site 2 processes only 37% and 4% of the jobs of class $o1$ and $o3$, respectively. It is interesting to mention that when jobs of class $o3$ are allowed to be processed at site 1, and load balancing is applied, 22% of these jobs are processed at site 1 and 15% at site 2 (when $N_2 = 0$). Furthermore, all jobs of class $o1$ are sent to site 2.

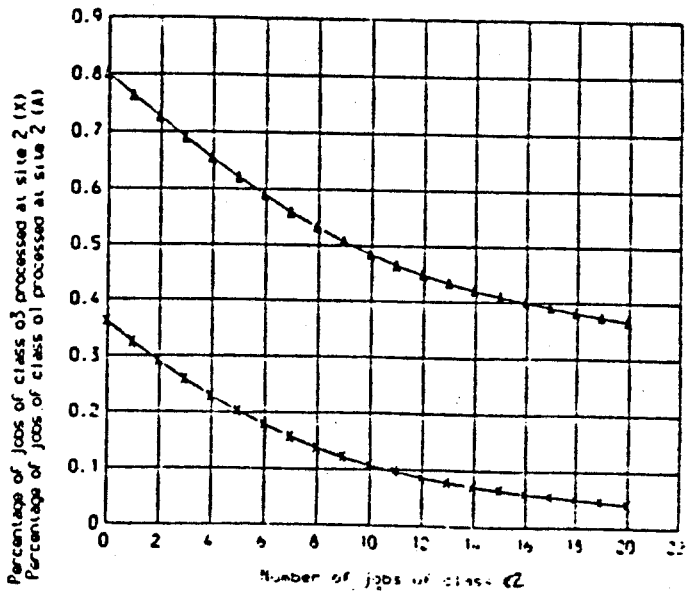

Figure 6: Percentage of foreign jobs processed at site 2.

Figure 7 shows the effect of load balancing on overall delay as a function of N_2. The effect is illustrated by plotting the relative difference between the delays when load balancing is not used and when it is used [2]. For instance, when $N_2 = 0$ and load balancing is used the average overall delay is 0.95 sec in contrast with 9.2 sec when load balancing is not used (the relative difference is 870% in this case).

The second application presents results for a distributed system model in which one class of jobs has multiple tasks which can execute in different sites. The model is similar to the one considered for the previous application. We assume that there is only one class of open chain jobs. Jobs in this class have the same parameters as jobs from class o_1 in the previous application. However, each job from this class is formed by two tasks which have to be executed in sequence before completion of this job. We assume that the two tasks have identical computational requirements, but the first task (t_1) can only be executed in sites 1 or 2 and the second task (t_2) can only be executed in sites 2 or 3. The possible paths of execution for chain o_1 jobs is illustrated in Figure 4.

Table 2 shows the optimum load-balancing strategy for the network described above for three cases: when the number of jobs representing the local load of site 2 (N_2) is equal to 1 and the time to transfer a task over the communication network is assumed to be 10 msec; when N_2 is equal to 20; and when the transfer time over the communication network

[2]We define the relative difference between delays as: (Delay without load balancing - Delay with load balancing) / Delay with load balancing).

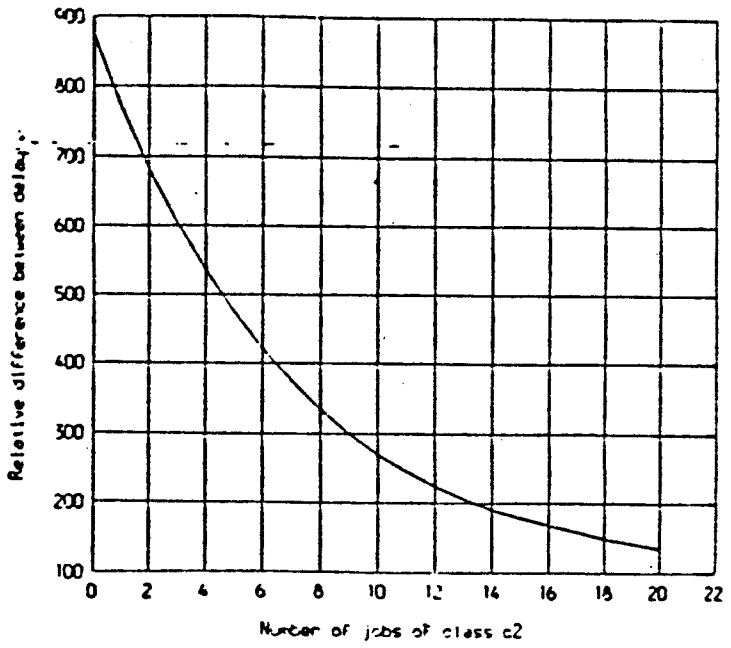

Figure 7: Relative difference (%) of the overall delays.

is increased to 160 msec, for N_2 equal to 1. As we can observe from the results, in the first case site 1 processes 33.9% of tasks t_1, since its local load is considerably large. Site 2 processes the majority of tasks t_1 and almost half of tasks t_2. As the local load of site 2 increases to 20 jobs, we see a decrease of the percentage of tasks processed at this site. Now the majority of tasks t_1 is processed at site 1 and the majority of tasks t_2 is processed at site 3. In the third case, when the communication delay increases significantly (N_2 is equal to 1 in this case), we observe that tasks are migrated so that the communication network stays with the minimum utilization possible as we can observe by the value of the throughput in the communication network, which is equal to the value of the throughput of chain o_1 jobs. Finally, it is interesting to observe again the importance of load balancing. For the first case above, (N_2 equal to 1 and the communication delay equal to 10 msec), if all tasks t_1 are executed in the first site and tasks t_2 are split equally between sites 2 and 3, the overall delay is 12.6 sec, a 556% increase in relation to the same network when load balancing is used.

	tasks	Percentage of tasks of jobs from class $o1$ processed at a site			commun. net. throughput	overall delay
		site 1	site 2	site 3	(jobs/sec)	(sec)
$c2$: 1 job	t_1	33.9	66.1		7.08	1.92
time to transfer	t_2		48.2	51.8		
a task: 10 msec						
$c2$: 20 jobs	t_1	67.3	32.7		6.96	5.95
time to transfer	t_2		17.2	82.8		
a task: 10 msec						
$c2$: 1 job	t_1	42.4	57.6		6.00	8.75
time to transfer	t_2		57.6	42.4		
a task: 160 msec						

Table 2: Optimal load balancing for the second application.

5 Jobs with Spawned Tasks.

In the previous section we considered jobs with single and/or multiple tasks which execute in sequence. These jobs were assumed to be generated by a large number of terminals and were independent of the closed chain jobs representing the "local load". In this section we consider jobs which can spawn other jobs consisting of one or more tasks. These tasks then execute in sequence and do not require synchronization at any point during execution. In a distributed environment similar to the one portrayed in Figure 1, these spawned tasks may represent background load generated by interactive users. As an example, in our current LOCUS configuration some interactive user generated batch jobs (e.g., compilation, troff and others) are routed to one of the "server nodes" dedicated to this type of background processing. Spawned jobs in queueing networks, generated by interactive users, have been approximately modeled by Heildelberger and Trivedi [HEID82] as open Poisson sources with arrival rate dependent on the (closed chain) rate of the "parent" job. Comparison with simulation shows that their approach is sufficiently accurate for load balance optimization (errors in server utilization are within a few percent for typical cases).

In this section we combine the basic approach of [HEID82] with the method developed in previous sections to calculate the optimum (i.e., minimum delay) routing strategy for spawned jobs. The assumptions are the same as in previous sections. We define the overall delay $D(\vec{N})$ as follows:

$$D(\vec{N}) = \frac{w^s \sum_{k=1}^{K^c} \lambda_k^{Nc}(N_k) P_k D^s_{v(k)}(\vec{N}) + w^c \sum_{k=1}^{K^c} \lambda_k^{Nc}(N_k) D_k^c(N_k)}{\sum_{k=1}^{K^c} \lambda_k^{Nc}(N_k)[w^c + w^s P_k]} \tag{17}$$

where superscript s indicates a spawned job. P_k is defined as the probability that a job from closed chain k spawns a job, after visiting a designated service center (say, for instance, the node representing the terminals for chain k in our distributed system model). $v(k)$ is the subscript representing a job spawned by a closed chain k job. $D_k^c(N_k)$ represents the response time of closed chain k jobs and $D^s_{v(k)}(\vec{N})$ the response time for jobs spawned by closed chain k jobs. $\lambda_k^{Nc}(N_k)$ is the nominal throughput of closed chain k jobs when $P_k = 0 \; \forall \, k$, i.e., no jobs are spawned. w^s and w^c are arbitrary constant weighting factors for the spawned jobs and interactive jobs, respectively. As explained in section 2, we use the nominal throughputs as waiting factors to avoid unfairness to the closed chain jobs.

We assume that the interactive jobs (modeled as closed chains) can run only in their local sites. This assumption could actually be easily relaxed without change in the solution approach. The spawned tasks, on the other hand, can be reassigned to a subset of the sites in the network (i.e., the "server sites"). Therefore, similarly to the previous section,

the model is reduced to K^c independent mixed network models, each which one or more closed chains. For notational convenience we assume that the behavior of interactive jobs in a site can be represented by a single closed chain, and that spawned jobs are composed of a single task. Furthermore, we assume $w^s = w^c = 1$. Applying Little's result and using equation (6), we obtain:

$$\begin{aligned} D^s_{v(k)}(\vec{N}) &= \sum_{\substack{j=1 \\ j\neq IS}}^{J} \frac{\rho^s_{jv(k)}}{1-\rho^s_j} \frac{[1+L^c_j(N_k)]}{\lambda^c_k(N_k)P_k} + \sum_{\substack{j=1 \\ j=IS}}^{J} \frac{\rho^s_{jv(k)}}{\lambda^c_k(N_k)P_k} \\ &= \sum_{\substack{j=1 \\ j\neq IS}}^{J} \frac{a^s_{jv(k)}}{1-\rho^s_j}[1+L^c_j(N_k)] + \sum_{\substack{j=1 \\ j=IS}}^{J} a^s_{jv(k)} \end{aligned} \tag{18}$$

since $\rho^s_{jv(k)} = \lambda^c_k(N_k)P_k a^s_{jv(k)}$. (Also recall that k is the index of the single closed chain that visits center j.) Substituting (18) in (17) and exchanging summations:

$$D(\vec{N}) = \frac{\sum_{\substack{j=1 \\ j\neq IS}}^{J} \frac{1+L^c_j(N_k)}{1-\rho^s_j}\Upsilon_j + \sum_{\substack{j=1 \\ j=IS}}^{J} \Upsilon_j + \sum_{k=1}^{Nc} \lambda^{Nc}_k(N_k)\left[\frac{N_k}{\lambda^c_k(N_k)} - T^c_{term,k}\right]}{\sum_{k=1}^{K^c} \lambda^{Nc}_k[1+P_k]} \tag{19}$$

where $\Upsilon_j = \sum_{k=1}^{K^c} \Upsilon_{jv(k)}$ and $\Upsilon_{jv(k)} = \lambda^{Nc}_k(N_k)P_k a^s_{jv(k)}$

The independent variables in our optimization problem are the $\{\lambda^s_{jv(k)}\}$. Therefore, since

$$\frac{\partial \Upsilon_j}{\partial \lambda^s_{jv(k)}} = \frac{1}{\lambda^s_{v(k)}} \frac{\partial \Upsilon_j}{\partial \theta^s_{jv(k)}} = \frac{\Upsilon_{jv(k)}}{\lambda^s_{jv(k)}}$$

$$\frac{\partial \rho^s_j}{\partial \lambda^s_{jv(k)}} = T^s_{jv(k)}$$

and from Theorem 1,

$$\frac{\partial L^c_j(N_k)}{\partial \lambda^s_{jv(k)}} = \frac{T^s_{jv(k)} V_j(N_k)}{(1-\rho^s_j)}$$

We finally have:

$$\frac{\partial D(\vec{N})}{\partial \lambda^s_{jv(k)}} = \begin{cases} \dfrac{\Delta_1 + \Delta_2}{\mu^s_{jv(k)}(1+\rho^s_j)\sum_{k=1}^{N^c} \lambda^{N^c}_k[1+P_k]} & j \neq \text{IS} \\ \dfrac{\Upsilon_{jv(k)}}{\rho^s_{jv(k)}\mu^s_{jv(k)}\sum_{k=1}^{N^c} \lambda^{N^c}_k[1+P_k]} & j = \text{IS} \end{cases} \tag{20}$$

where

$$\Delta_1 = \left[\frac{\Upsilon_{jv(k)}}{\rho^s_{jv(k)}} + \frac{\Upsilon_j}{1-\rho^s_j}\right][1 + L^c_j(N_k)] + \frac{\Upsilon_j}{1-\rho^s_j}V_j(N_k) + \sum_{\substack{i\neq j \\ i\neq \mathrm{IS}}} \frac{\Upsilon_i}{1-\rho^s_i}V_{ij}(N_k)$$

and

$$\Delta_2 = \frac{\delta(j)\lambda^{Nc}_k(N_k)}{\lambda^c_k(N_k)}N_k[L^c_j(N_k) - L_j(N_k - 1)]$$

Equation (20) is the key to the flow deviation method. We associate a length or weight for a spawned task $v(l)$ to each site, given by:

$$lsite_{k_v(l)} \stackrel{\text{def}}{=} \sum_{j\in k} \theta^s_{jv(l)} \frac{\partial D(\vec{N})}{\partial \lambda^s_{jv(l)}}$$

and, similarly, $lnet_v(l)$ is defined for the communication network.

The flow deviation algorithm for load balancing spawned tasks remains basically the same as outlined in previous section. In the initialization step (Step 1) the open chain rates are initialized to the nominal throughput of correspondent closed chain jobs. Furthermore, after deviating the flow for an open chain $v(k)$, the throughput of the open chain $v(k)$ is updated to $\lambda^s_{v(k)} = \lambda^c_k P_k$. At the end of the algorithm, $\lambda^s_{v(k)}$ converges to λ^c_k $\forall$ k in the network.

Application:

As an application of the approach, consider the distributed system model illustrated in Figure 5. However, in this example, the open chains represent tasks which are spawned by closed chain jobs. Site 3 is reserved as a "server" site (similarly to the scheme used at UCLA-LOCUS). Therefore, no interactive users can login at that site and only batch tasks originating at other sites are allowed to run at site 3. Site 3 is assumed to have the same CPU characteristics as site 1. We assume that interactive jobs which run at site 1 and 2 spawns batch tasks with probability 0.25 and 0.1, respectively. Batch tasks generated from site 1 and 2 have the requirements shown in Table 3.

Initially we assume that batch tasks can run in any of the sites. Figure 8 shows the percentage of batch tasks spawned from interactive jobs of site 1, which run on site 2 (squares) and 3 (dots), after balancing the load, as a function of the number of interactive jobs of site 2 (N_2). From the figure we can observe that if the number of interactive users running at site 2 is less or equal than 5 then site 2 is the best site to run batch tasks spawned from site 1. When the number of interactive users at site 2 increases beyond 5, then we see that a percentage of batch tasks from site 1 should also run on sites 1 or 3,

TASK	SITE	CENTER	TYPE	VISIT RATIOS	SERVICE TIMES (msec)
$v1$	1	CPU	PS	5	300
		disk(1 or 2)	FCFS	2	50
	2	CPU	PS	5	150
		disk	FCFS	4	15
	3	CPU	PS	5	300
		disk(1 ... 4)	FCFS	1	140
	com. net.		PS		10
$v3$	1	CPU	PS	5	500
		disk(1 or 2)	FCFS	2	50
	2	CPU	PS	5	250
		disk	FCFS	4	15
	3	CPU	PS	5	500
		disk(1 ... 4)	FCFS	1	140
	com. net.		PS		10

Table 3: Computational requirements of spawned tasks.

for achieving minimal delay. When the number of interactive users at site 2 reaches 14, no tasks spawned from site 1 should run on site 2. Similarly, Figure 9 shows the percentage of batch tasks spawned from interactive jobs of site 2, which run on site 2 (squares) and 3 (dots), after balancing the load, as a function of the number of interactive jobs of site 2 (N_2). From the figure we can observe that if the number of interactive users running at site 2 is equal to 1 then site 2 is the best site to run batch tasks spawned from this site 2 interactive user. As the number of interactive users at site 2 increases, we see that a percentage of batch tasks from site 2 should also run on site 3, for achieving minimal delay. When the number of interactive users at site 2 reaches 5, all tasks spawned from site 2 should run on site 3.

Figure 10 compares the overall delay when batch tasks are allowed to run in any site and when the tasks are restricted to run only on the "server" site. The comparison is illustrated by plotting the relative difference between the delays when load balancing is not used and when it is used. It can be easily observed that restricting batch tasks to server sites may greatly increase the overall delay. As we see from Figure 10, as the number of interactive users increases balancing the load become increasingly important for the overall delay (relative difference increases). The slight decrease in relative difference when N_2 is increased beyond 15 is explained by the fact that after this point the network is near saturation even when batch load is balanced.

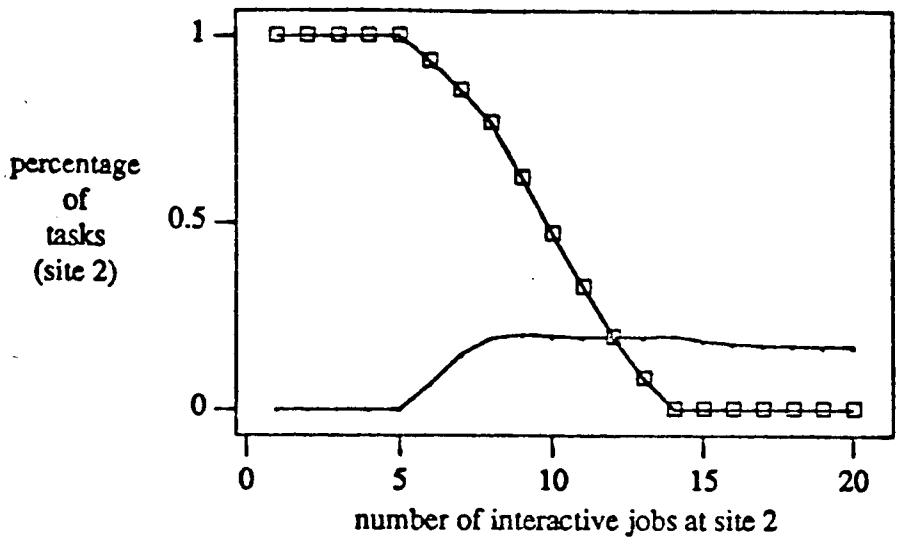

Figure 8: Percentage of spawned tasks from site 1.

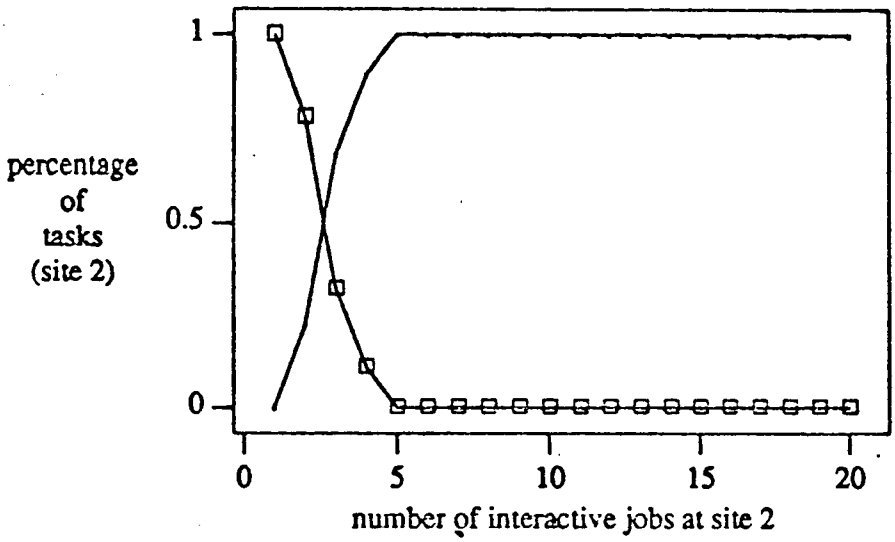

Figure 9: Percentage of spawned tasks from site 2.

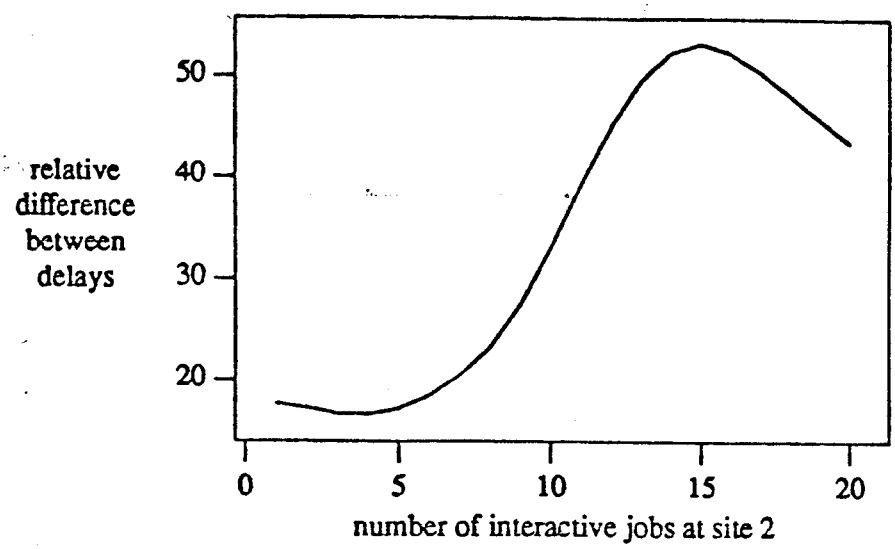

Figure 10: Relative difference (%) between delays.

6 Conclusions.

We have presented an efficient method for static load balancing in distributed systems. The main advantage of this method over previous schemes is its applicability to very general models. Namely, models with multiple classes, site constraints, job spawning and multitasking can be easily handled. The success of the method rests on the following key properties/results:

- the queueing network model is of product form type, or can be reasonably approximated as such.
- a computationally efficient method for finding "improving paths" was developed and implemented in the steepest descent procedure.
- the local minimum is a global minimum (i.e., the objective function is convex).

The numerical results have demonstrated the efficacy of the method and have indicated some of its potential uses. Typical applications include comparisons of the different systems and different resource allocation policies (e.g., how many servers should be installed in the system, or; which site constraints should be imposed on different classes, etc), sensitivity studies (e.g., impact of background load on interactive job delays) and, more generally, capacity planning.

A possible limitation of our model in predicting the performance of a system yet to be built is in fact that the model is a **static** load balancing model, while actual systems apply **dynamic** load balancing, where the job/task relocation decision is based on a dynamically changing system status rather than on predefined fraction allocations. However, dynamic models are very difficult to analyze (in the limit one must use simulation) and very cumbersome to use in an interactive design process. Thus, static models, albeit inaccurate, are the only practical approach to capacity planning and design. This is not unlike the situation found in wide area packet switched networks, where optimal, steady state routing schemes are used for network topology design (while actual networks are driven by dynamic routing schemes). For packet networks, good agreement was found between static (multipath) and dynamic routing performance [KLEI76]. An important issue for future investigation is the consistency between static and dynamic load balance policies in distributed systems. Another promising direction for further research is the combination of optimal (static) and heuristic (dynamic) rules in a hybrid load management scheme.

The method presented here can be easily extended to account for several modifications in the basic model. In particular, the performance measure can be modified to minimize delays of a subset of user classes. Alternatively, the delay measure may be replaced by a

proper fairness measure [GERL85], where the intent is to "equalize" closed chain throughputs by distributing open chain classes among the various sites. Furthermore, the use of multiple classes, as done for modeling multitasking, can also be used to model tasks which may temporarily suspend execution at one site, start a remote process at another site (e.g., remote procedure call, request of a page/file transfer. etc) and then resume execution once the remote process has terminated. Finally we note that complex interconnection networks can also be handled, assuming that they satisfy product form requirements.

Appendix.

Proof of Theorem 1:

In a product form mixed queueing network model with SSFR and IS service centers, the equilibrium state probability for a site k is given by [LAVE83] (we are assuming single closed chain network for simplicity):

$$P_k^c(\vec{n}_k) = \frac{1}{G_k^c(N_k)} \prod_{\substack{i=l \\ i \neq IS}}^{L} \left[\frac{a_{ik}^c}{1-\rho_i^o}\right]^{n_i} \prod_{\substack{i=l \\ i=IS}}^{L} \frac{(a_{ik}^c)^{n_i}}{n_i!} \tag{21}$$

where centers which belong to site k are labeled $l, \ldots, L$ and $G_k^c(N_k)$ is the normalization constant of the queueing network model representing site k, i.e., $G_k^c(N_k) = \sum_{|\vec{n}_k|=N_k} P_k^c(\vec{n}_k)$, $\vec{n}_k = (n_l, \ldots, n_L)$, $|\vec{n}_k| = n_l + \ldots + n_L$.

Taking the (partial) derivative of $G_k^c(N_k)$ with respect to ρ_j^o,

$$\frac{\partial G_k^c(N_k)}{\partial \rho_j^o} = \frac{1}{1-\rho_j^o} \sum_{|\vec{n}_k|=N_k} n_j \prod_{\substack{i=l \\ i \neq IS}}^{L} \left(\frac{a_{ki}^c}{1-\rho_j^o}\right)^{n_i} \times \prod_{\substack{i=l \\ i=IS}}^{L} \frac{(a_{ik}^c)^{n_i}}{n_i!} \quad j \neq \text{IS} \tag{22}$$

$$= \frac{G_k^c(N_k)}{1-\rho_j^o} \sum_{|\vec{n}_k|=N_k} n_j P_k^c(\vec{n}_k) \tag{23}$$

The summation on the right hand side of (23) is by definition the average queue length of closed chain jobs at service center j, therefore:

$$\frac{\partial G_k^c(N_k)}{\partial \rho_j^o} = \frac{G_k^c(N_k)}{1-\rho_j^o} L_j^c(N_k) \tag{24}$$

To prove equation (9) we Take the derivative of (22) again with respect to ρ_j^o,

$$\frac{\partial^2 G_k^c(N_k)}{\partial (\rho_j^o)^2} = \frac{G_k^c(N_k)}{(1-\rho_j^o)^2} \sum_{|\vec{n}_k|=N_k} n_j P_k^c(\vec{n}_k) + \frac{G_k^c(N_k)}{(1-\rho_j^o)^2} \sum_{|\vec{n}_k|=N_k} n_j^2 P_k^c(\vec{n}_k)$$

$$= \frac{G_k^c(N_k)}{(1-\rho_j^o)^2} \left(L_j^c(N_k) + S_j^c(N_k)\right) \tag{25}$$

where $S_j^c(N_k)$ is the second moment of queue lengths of the closed chain jobs at center j of site k.

The second derivative of $G_k^c(N_k)$ with respect to ρ_j^o can also be obtained by taking the derivative of equation (24) with respect to ρ_j^o. We obtain:

$$\frac{\partial^2 G_k^c(N_k)}{\partial (\rho_j^o)^2} =$$

$$= \frac{1}{(1-\rho_j^o)^2} G_k^c(N_k) L_j^c(N_k) + \frac{1}{1-\rho_j^o} \frac{\partial G_k^c(N_k)}{\partial \rho_j^o} L_j^c(N_k) + \frac{1}{1-\rho_j^o} G_k^c(N_k) \frac{\partial}{\partial \rho_j^o} L_j^c(N_k)$$

$$= \frac{G_k^c(N_k)}{(1-\rho_j^o)^2} \left(L_j^c(N_k) + \left(L_j^c(N_k)\right)^2 + (1-\rho_j^o) \frac{\partial}{\partial \rho_j^o} L_j^c(N_k) \right) \quad (26)$$

Now, equating (25) and (26) and using the definition of the variance of queue lengths of closed chain jobs at center j, (9) is obtained. Equation (10) can be proved in a similar way and we omit the details for conciseness. □ •

Proof of Corollary 1:

Equations (11) and (12) are obtained from MVA equations and Theorem 1. We prove the result for single closed chain networks, but the same approach can be used for multiple chain networks.

The well known MVA equation which relates the mean queue lengths of closed chain jobs at a service center as a function of queue lengths of closed chain jobs with one less closed chain job in the network becomes, for mixed networks:

$$L_j(N) = \lambda(N) \frac{a_j}{1-\rho_j^o} [1 + L_j(N-1)] \quad (27)$$

where the superscript c was dropped to simplify the notation and j is a SSFR service center.

In order to prove equation (11) we take the derivative on both sides of equation (27) with respect to ρ_j^o:

$$\frac{\partial L_j(N)}{\partial \rho_j^o} = \frac{\partial \lambda(N)}{\partial \rho_j^o} \frac{a_j}{1-\rho_j^o} [1 + L_j(N-1)]$$

$$+\lambda(N) \frac{a_j}{(1-\rho_j^o)^2} [1 + L_j(N-1)] + \lambda(N) \frac{a_j}{1-\rho_j^o} \frac{\partial L_j(N-1)}{\partial \rho_j^o} \quad (28)$$

Now, substituting (9) and (13) in (27), equation (11) is obtained. Using similar steps, the recursive expression for the covariance $V_{jl}(N)$ (equation (12)) can be obtained, and we omit the details for conciseness. □

Proof of Theorem 2

In order to find the derivatives of the closed chain throughput of site k ($\lambda_k^c(N_k)$) with respect to the open chain utilization at center j in k (ρ_j^o), we express $\lambda_k^c(N_k)$ in terms of

the normalization constant, [LAVE83]:

$$\lambda_k^c(N_k) = \frac{G_k^c(N_k - 1)}{G_k^c(N_k)}$$

Now, taking the derivative of (29) with respect to ρ_j^o and using (24),

$$\begin{aligned}
\frac{\partial \lambda_k^c(N_k)}{\partial \rho_j^o} &= \frac{\partial G_k^c(N_k - 1)/\partial \rho_j^o}{G_k^c(N_k)} - \frac{G_k^c(N_k - 1)}{(G_k^c(N_k))^2} \frac{\partial G_k^c(N_k)}{\partial \rho_j^o} \\
&= \frac{G_k^c(N_k - 1)}{G_k^c(N_k)} \frac{L_j^c(N_k - 1)}{1 - \rho_j^o} - \frac{G_k^c(N_k - 1)}{G_k^c(N_k)} \frac{L_j^c(N_k)}{1 - \rho_j^o} \\
&= \frac{\lambda_k^c(N_k)}{1 - \rho_j^o} \left(L_j^c(N_k - 1) - L_j^c(N_k) \right)
\end{aligned}$$

where service center j is assumed to be in the same site of closed chain k. □

References

[BUX81] W. Bux, "Local-Area Subnetworks: A Performance Comparison", *IEEE Transactions on Communications* COM-29, no. 10, 1465-1473, 1981.

[DeSO84a] E. de Souza e Silva and M. Gerla, "Load Balancing in Distributed Systems with Multiple Classes and Site Constraints", *Performance 84 Proceedings*, E. Gelenbe (Editor), 17-33, 1984.

[DeSO84b] E. de Souza e Silva and R.R. Muntz "Simple Relationships Among Moments in Product Form Queueing Network Models", UCLA Computer Science Department, 1984, to appear in IEEE Transactions on Computers.

[FRAT73] L. Fratta, M. Gerla and L. Kleinrock, "The Flow Deviation Method - An Approach to Store-and-Forward Communication Network Design", *Networks* 3, 1973.

[GALL77] R. G. Gallager, "A Minimum Delay Routing Algorithm Using Distributed Computation", *IEEE Transactions on Communications*, COM-23, 73-85, 1977.

[GERL85] M. Gerla and H.W. Chan, Window Selection in Flow Controlled Networks", *Proceedings of Data Communication Symposium*, September 1985.

[GOLD83] A. Goldberg, G. Popek and S.S. Lavenberg, "A Validated Distributed System Performance Model", *Performance 83 Proceedings*, A.K. Agrawala and S.K. Tripathi (Editors), 251-268, 1983.

[HEID82] P. Heidelberger and K. S. Trivedi, "Queueing Network Models for Parallel Processing with Asynchronous Tasks", *IEEE Transactions on Computers*, vol. C-31, pp.1099-1108, 1982.

[KLEI76] L. Kleinrock, "Queueing Systems, Volume II: Computer Applications", *Wiley-Interscience*, New York, 1976.

[KOBA83] H. Kobayashi and M. Gerla, "Optimal Routing in Closed Queueing Networks", *ACM Transactions on Computer Systems*, vol. 1, no. 4, 294-310, November 1983.

[LAVE83] S.S. Lavenberg (Editor), "Computer Performance Modeling Handbook", *Academic Press, New York*, 1983.

[POPE81] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin and G. Thiel, "LOCUS: A Network Transparent, High Reliability Distributed System", *Proceedings of the Eighth Symposium on Operating Systems Principles*, California, December 1981.

[REIS80] M. Reiser & S.S. Lavenberg, "Mean Value Analysis of Closed Multichain Queueing Networks" *Journal of ACM*, vol. 27, pp. 313-322, 1980.

[TANT85] A.N. Tantawi and D. Towsley, "Optimal Static Load Balancing in Distributed Computer Systems", *Journal of ACM*, vol. 32, pp. 445-465, 1985.