EDITORS

Donald R. Deese

Robert J. Bishop

Jeffrey M. Mohr

H. Pat Artis

Proceedings of the

1981 Computer Measurement Group

International Conference

Proceedings of the

CMG XII INTERNATIONAL CONFERENCE

December 1-4, 1981

New Orleans, Louisiana

EDITORS

Donald R. Deese
FEDSIM/AY
Alexandria, Virginia

Robert J. Bishop
Ernst and Whinney
Atlanta, Georgia

Jeffrey M. Mohr
Arthur Young and Company
Washington, DC

H. Pat Artis
Morino Associates, Inc.
Vienna, Virginia

COMPUTING PERFORMANCE MEASURES OF COMPUTER SYSTEMS WITH VARIABLE DEGREE OF MULTIPROGRAMMING

Daniel A. Menascé*
Departamento de Informática
Pontifícia Universidade Católica
22453, Rio de Janeiro,RJ
Brazil

Virgilio A.F. Almeida
Centrais Elétricas de Minas Gerais, CEMIG
30000, Belo Horizonte, MG
Brazil

ABSTRACT

In actual computer systems, the number of jobs multiprogrammed in each job class is not fixed but varies dynamically with the nature of the class and with the amount of memory available for each workload. This paper presents an algorithmic way to compute performance measures such as throughput, mean response time and memory utilization, for system in which the number of jobs per workload varies within a given range. Our model alows the performance analyst to handle queueing for main memory and systems with several job classes of different types such as batch, timesharing and transaction processing. The techniques presented here have been succesfully used to model actual computer systems, and the results obtained with the model presented here are compared with those of a well known capacity planning software package.

1. INTRODUCTION

Queueing network models of computer systems have gained considerable attention from the performance analyst in the last few years. The first analytic results in queueing networks are due Jackson [1 and 2] and Gordon and Newell [3]. In 1971, Buzen [4] introduced a convenient model of multiprogrammed systems - the central server model - and presented [4 and 5] efficient computational algorithms for closed queueing networks. Baskett, Chandy, Muntz and Palacios [6] extended the theory to allow multiple classes, different queueing disciplines and non-exponential servers.

In 1976, Buzen [7 and 8] introduced operational analysis and explained why some of the classic results in queueing theory hold even when the assumptions upon which they were based are not verified. Later, Denning and Buzen [9 and 10] applied operational analysis to the study of queueing networks. Bouhana [11] gave an operational treatment of centralized queueing networks - a generalization of the central server model. Roode [12] extended these results to treat multiclass queueing networks.

The first closed queueing networks models to be studied suffered from a severe limitation since they required that the number of jobs in the system be fixed. It turns out that in actual systems, the number of jobs multiprogrammed in each job class is not fixed but varies dynamically depending on the nature of the class and on the amount of memory available for each class. In 1977, Lam [17] showed how to handle queueing network models with population size constraints and applied it to multiprogrammed computer systems with varying degree of multiprogramming. However, his approach, which is based on state-dependent lost and triggered arrivals, is not suitable for modeling queueing for main memory since jobs are lost instead of queued.

In this paper we present the solution of a multiclass queueing network model in which the number of jobs of a given class i ($1 \le i \le R$) is not fixed, but is allowed to vary within a given range. Our approach allows us to model queueing effects for main memory and to handle simultaneously many job classes of different types such as batch, timesharing and transaction processing.

Let a configuration be a tuple $(n_1, \ldots, n_R)$ such n_i is the current number of class i jobs being multiprogrammed. We present here an algorithm to obtain performance measures (e.g. throughput, response time) for each job class, averaged over all configurations observed during an observation period. This algorithm allows us to build convenient software packages for performance evaluation.

*This work was partially supported by Financiadora de Estudos e Projetos (FINEP), by CNPq, and by IBM Brasil.

Section 2 presents the computer system model considered here and characterizes the types of workloads analyzed. Section three presents the corresponding queueing network model. Section four presents an analysis of the different types of workload classes considered and presents the algorithm that should be used to treat models with variable degree of multiprogramming. Finally, section five presents some comparisons between numerical results obtained with our model and with those obtained with BEST/1. [13], a BGS Systems proprietary software package for capacity planning.

2. COMPUTER SYSTEM MODEL AND WORKLOAD CHARACTERIZATION

Our model of a computer system consists of a central subsystem which contains the CPU and I/O devices. A collection of terminals outside the central subsystem generate timesharing and transaction processing jobs which are executed by the central subsystem along with batch jobs. (see Figure 1).

Let us describe below the three types of workloads considered in our model, namely batch, timesharing and transaction processing.

The batch type of class is characterized by the average degree of multiprogramming of that class. We assume that a continuous backlog situation exists for batch jobs. For this type of class the main performance measures are the throughput and the turn-around time.

The timesharing class is described by the number of terminals, by the average think time and by the maximum level of multiprogramming for this class. Think time is defined as the interval between a response to a terminal and the submission of a new command (job) by the same terminal. The main performance measures in this case are response time and throughput.

Finally, the transaction processing class is characterized by the average transaction arrival rate to the central subsystem and by the maximum level of multiprogramming. The main performance measure in this case is response time.

The existence of these three types of workloads implies that a multiclass queueing network is needed. A job class characterizes a common behavior and resources usage statistic observed for a collection of jobs. We consider the existence of R classes numbered from 1 to R. Each class may be of one of the three types described above, i.e. batch, timesharing and transaction processing. However other types of classes may be imagined. The methods described in this paper work as well for any type of class, provided one can derive the fraction of time that the system is in a given state, in terms of the throughput for that particular class.

The central subsystem is assumed to be a centralized network, as introduced in [11]. Figure 2 illustrates such a type of network. As it can be seen, a centralized network is a generalization of the central server model [4] in which aside from having a central server (possessing a loop), the rest of the network can have an arbitrary topology.

The loop around the central server represents a conceptual exit of a job from the system, i.e. a fraction q_{00}^{r}, for $1 \le r \le R$, of all class r jobs that leave the central server are said to leave the system and are immediately replaced by a new job of the same class. The conceptual exit is an extremely interesting modeling device since it allows us to consider arrivals and departures of jobs in a closed network.

3. QUEUEING NETWORK MODEL

This section presents the quantities which describe the queueing network model used to model computer systems. Consider a queueing network consisting of M devices (e.g. processor, i/o devices). Jobs in the system may belong to any one of R classes. Jobs are not allowed to change classes.

Let,

n_{ir} = number of class r jobs at device i.

$n_r = \sum_{i=1}^{M} n_{ir}$ = total number of class r jobs

$N = \sum_{r=1}^{R} n_r$ = total number of jobs in the system.

We assume that all devices in the network have a load independent behavior, i.e. the time a job takes to be serviced at the device does not depend on the size of the queue at the device

3.1 NETWORK SOLUTION

A network state $\underline{n}$ is defined as $\underline{n} = (\underline{n}_1, \underline{n}_2, \ldots, \underline{n}_M)$ where $\underline{n}_i$ is the state of device i given by

$$\underline{n}_i = (n_{i1}, n_{i2}, \ldots, n_{iR})$$

In order for $\underline{n}$ to be a feasible network state it is necessary that

$$\sum_{i=1}^{M} \sum_{r=1}^{R} n_{ir} = N \qquad (1)$$

Let $T(\underline{n})$ be the total time that the network was in state $\underline{n}$ during the observation period T. The proportion of time that the system is in state $\underline{n}$ is given by

$$p(\underline{n}) = \frac{T(\underline{n})}{T} \qquad (2)$$

and $\sum_{\underline{n}} p(\underline{n}) = 1$ where the summation is over all possible states $\underline{n}$.

The network solution can be shown to be given by the expression below:

$$p(\underline{n}) = \frac{1}{G} \prod_{i=1}^{M} Y_{i1}^{n_{i1}} Y_{i2}^{n_{i2}} \cdots Y_{iR}^{n_{iR}} \qquad (3)$$

where Y_{ir} is the total amount of service time accumulated by a job in all visits to device i averaged over all class r jobs. G is a normalization constant defined as follows

$$G = \sum_{\underline{n} \in S(N,M,R)} \prod_{i=1}^{M} Y_{i1}^{n_{i1}} Y_{i2}^{n_{i2}} \cdots Y_{iR}^{n_{iR}} \qquad (4)$$

The summation in (4) is over the set S(N,M,R) defined as follows

$$S(N,M,R) = \{\underline{n} = (n_{11}, n_{12}, \ldots, n_{1R}, n_{21}, \ldots, n_{2R}, \ldots, n_{MR}) \mid \sum_{i=1}^{M} n_{ir} = n_r \;\&\; \sum_{r=1}^{R} n_r = N \;\&\; n_{ir} \ge 0 \;\forall\; i, r\} \qquad (5)$$

It should be emphasized that no assumption regarding service time distribution must be made in order to obtain expression (3). However the following operational assumptions, which are defined in [10], must hold for expression (3) to be true:

Flow Balance: arrivals equal departures at each system device or network state.

One-Step Behavior: the only observable state changes result from single jobs either entering or leaving the system, or moving from one device into another.

Device Homogeneity: the output rate of a device is independent of the system state and may depend only on the queue size at the device.

Routing Homogeneity: the routing frequencies between device is independent on the system state.

3.2 THE NORMALIZATION CONSTANT

Computationally efficient algorithms for calculating the normalization constant in closed queueing networks were first presented by Buzen in [4 and 5]. Brüel and Balbo in their recent book [14] present a nice and comprehensive study of computational algorithms for closed networks.

We briefly review here the results we need. The normalization constant G, given in equation (4), can be efficiently obtained as $G_M(n_1, \ldots, n_R)$ using the recursive equation below and the initial conditions given by equations (7) and (8).

$$G_i(K_1, \ldots, K_R) = G_{i-1}(K_1, \ldots K_R) + \sum_{r=1}^{R} Y_{ir}\, G_i(K_1, \ldots, K_r - 1, \ldots, K_R)$$

$$\text{for } i = 1, \ldots, M \qquad (6)$$

$$G_i(K_1, \ldots, K_R) = 1 \quad \text{if } K_1 = \ldots = K_R = 0 \qquad (7)$$

and

$$G_0(K_1, \ldots, K_R) = \begin{cases} 1 & \text{if } K_1 = \ldots = K_R = 0 \\ 0 & \text{otherwise} \end{cases} \qquad (8)$$

One of the nice properties of the normalization constant is that several useful performance measures, such as the ones listed below, can be derived from it.

$U_i(n_1, \ldots, n_R)$: utilization of device i, when the system contains $n_1, \ldots, n_R$ jobs of classes 1 through R, respectively.

$U_{ir}(n_1, \ldots, n_R)$: utilization of device i by class r jobs, when there are $n_1, \ldots, n_R$ jobs of classes 1 through R, respectively, in the system.

$X_0(n_1,\ldots,n_R)$: system throughput when the system contains $n_1,\ldots,n_R$ jobs of classes 1 through R respectively.

$X_{or}(n_1,\ldots,n_R)$: throughput of class r jobs when the system contains $n_1,\ldots,n_R$ jobs of classes 1 through R respectively.

$Q_{ir}(n_1,\ldots,n_R)$: average number of class r jobs present at device i when there are $n_1,\ldots,n_R$ jobs of classes 1 through R, respectively, in the system.

4. VARYING THE DEGREE OF MULTIPROGRAMMING PER JOB CLASS

In actual computer systems the number of jobs multiprogrammed in each job class is not fixed but varies dynamically depending on the class nature and mainly on the amount of memory available for each class during the period of observation.

For instance in a given installation one may have the following upper limits on the level of multiprogramming per workload type.

TABLE 1 - Example of Varying Degree of Multiprogramming

WORKLOAD #	WORKLOAD NAME	MAXIMUM LEVEL OF MULTIPROGRAMMING
1	FAST BATCH	3
2	NORMAL BATCH	1
3	TIMESHARING	3
4	DB QUERY	2
5	DB UPDATE	1

Therefore, at a given instant, the number of FAST BATCH jobs being multiprogrammed may be 0, 1, 2 or 3. Let a configuration be the tuple$(n_1,\ldots,n_R)$ where n_i, for $1\leq i\leq R$, is the current number of class i jobs being multiprogrammed. In section 3 this number was assumed to be fixed for each job class (workload). In this section we let this number be any integer from zero up to a given limit N_i (the maximum level of multiprogramming of class i). Some examples of possible configurations for the table above are (2, 0, 2, 1, 1), (3, 1, 1, 2, 0) and (1, 1, 3, 2, 1).

This section presents an algorithm which allows us to obtain performance measures for each class when several different configurations exist during the observation period. In other words, we do not require that the number, n_i, of class i jobs be fixed, but we allow it to vary from zero up to the maximum level of multiprogramming N_i.

The next three subsections show how to model workloads of the three different types (batch processing, time - sharing and transaction processing). Then we present the algorithms used to obtain performance measures in the varying degree of multiprogramming case.

4.1 BATCH PROCESSING CLASS

The batch processing (BP) class is modeled assuming a continuous backlog situation. Therefore, the number of batch jobs in the central subsystem (see Figure 3) oscillates between two consecutive integers N_L and N_H. The average number of jobs in the system is N.

Let,

N: average number of batch jobs in the central subsystem.
$X_0(N)$: throughput of the batch class when there are N jobs of this class in the system.
R: average response time of batch jobs.
N_L: greatest integer smaller than N. ($\lfloor N \rfloor$)
N_H: smallest integer greater than N. ($\lceil N \rceil$)
$p(\ell)$: fraction of time that there are ℓ jobs in the system.
$I(\ell)$: amount of time during which there are ℓ jobs in the system.

If N is an integer, the average response time of batch jobs can be obtained from Little's Law [15][1]. Thus

$$R = \frac{N}{X_0(N)} \tag{9}$$

Let us now examine the case where N is not an integer. From the definitions of N, N_L and N_H and from our continuous backlog assumption, it follows that,

$$p(N_L) + p(N_H) = 1 \tag{10}$$

$$\frac{N_L\, I(N_L) + N_H\, I(N_H)}{T} = N \tag{11}$$

If we observe that $p(\ell) = I(\ell)/T$, expression (11) can be rewritten as

$$N_L\, p(N_L) + N_H\, p(N_H) = N \tag{12}$$

From (10) and (12) it follows that

$$p(N_L) = N_H - N \tag{13}$$

$$p(N_H) = N - N_L \tag{14}$$

The average throughput is given by

$$X_0(N) = p(N_L)\, X_0(N_L) + p(N_H) X_0(N_H) \tag{15}$$

and the average response time of batch jobs can be obtained by applying Little's Result.

Hence,

$$R = \frac{N}{X_0(N)} \tag{16}$$

Since N_L and N_H are integers, the throughputs $X(N_L)$ and $X(N_H)$ can be obtained through the normalization constant.

Notice therefore that formula (15) allows us to obtain the throughput when the average number of batch jobs in the system is not an integer.

4.2 TIMESHARING

Figure 4 shows a computer system devoted exclusively to the processing of timesharing jobs.

Let,

Z : average think time (i.e., average time between the system response to a user request and the submission of a new request)

N_T : number of terminals

J : maximum level of multiprogramming (i.e. maximum number of active timesharing jobs in the central subsystem)

N : current number of active jobs in the system.

$X_0(N)$: throughput of the timesharing class when there are N jobs in the central subsystem.

(1) The operational coounterpart of Little's Law was proved by Buzen in [7].

service completions equal to $S_0(N) = 1/X(N)$. The principle of decomposition is used again in this analysis There may be up to J jobs in memory. The remaining jobs must wait for memory. Therefore, the central subsystem is again viewed as a server and a queue. The throughput of this server is $X_0(N)$ if $N \leq J$ and $X_0(J)$ if $N > J$.

Assuming that the system is in operational equilibrium and assuming single arrivals and departures one can derive balance equations for p(n) in a way similar to Buzen in [8]. Let N^* be the maximum number of jobs in the central system during the observation period. Then

$$p(N) = \frac{L}{X_0(n)}\, p(n-1) \qquad n=1,\ldots,J \tag{28}$$

$$p(n) = \frac{L}{X_0(J)}\, p(n-1) \qquad n=J,\ldots,N^* \tag{29}$$

The normalization equation is

$$\sum_{n=0}^{J-1} p(n) + \sum_{n=J}^{N^*} p(n) = 1 \tag{30}$$

From (29) we can obtain the value of p(n) as a function of p(J) for $n \geq J$.

$$p(n) = \left(\frac{L}{X_0(J)}\right)^{n-J} p(J) \qquad n=J,\ldots,N^* \tag{31}$$

If N^* is large and $L < X_0(J)$ we can write [1]

$$\sum_{n=J}^{N^*} p(n) = \sum_{n=J}^{N^*} \frac{L}{X_0(J)}^{n-J} p(J) = \frac{p(J)}{1-\frac{L}{X_0(J)}} \tag{32}$$

We are now ready to find the true values of p(n) for any value of n as follows.

1. Set p(0) = 1
2. Calculate p(n), for n=1,....J, iteratively using (28)
3. Use the values obtained in step 2 to calculate
$$C = \sum_{n=0}^{J-1} p(n) + \frac{p(J)}{1-\frac{L}{X_0(J)}}$$
4. Divide the values of p(n) obtained in step 2 by C. These are the true values of p(n) for n=1,...,J. The true value of p(0) is 1/C.
5. The true value of p(n) for n>J is obtained from expression (31) where p(J) is the one obtained in step 4.

The average number of jobs in memory, $\bar{N}_M$, is given by

$$\bar{N}_M = \sum_{i=1}^{J-1} ip(i) + J.\sum_{i \geq J} p(i) \tag{33}$$

Using (32) in (33) we obtain

$$\bar{N}_M = \sum_{i=1}^{J-1} ip(i) + \frac{Jp(J)}{1-\frac{L}{X_0(J)}} \tag{34}$$

The average number of jobs waiting for memory, $\bar{N}_Q$, is obtained by

$$\bar{N}_Q = \sum_{i \geq 1} ip_Q(i) \tag{35}$$

where $P_Q(i)$ is the fraction of time that there are i jobs waiting for memory. Since $P_Q(i) = p(i+J)$ for $i \geq 1$, it follows that

$$\bar{N}_Q = \sum_{i \geq 1} ip(i+J) = \sum_{i \geq 1} i \left[\frac{L}{X_0(J)}\right]^i p(J)$$

$$= p(J)\frac{L}{X_0(J)}\frac{\partial}{\partial\left(\frac{L}{X_0(J)}\right)}\sum_{i \geq 1}\left[\frac{L}{X_0(J)}\right]^i = p(J)\frac{\frac{L}{X_0(J)}}{\left(1-\frac{L}{X_0(J)}\right)^2} \tag{36}$$

Notice that the distribution of jobs waiting for memory is easily obtained from p(n) as follows.

$$P_Q(0) = \sum_{i=0}^{J} p(i)$$

$$P_Q(i) = p(i+J) \text{ for } i \geq 1$$

The average number of jobs in the central subsystem, $\bar{N}_S$, is given by

$$\bar{N}_S = \bar{N}_M + \bar{N}_Q$$

From Little's law we obtain the average response time of transaction type jobs, as

$$R = \frac{\bar{N}_S}{L} \tag{37}$$

4.4 COMPOSITION OF JOB CLASSES

As mentioned already, queueing network theory literature contains algorithms to analyze queueing networks with multiple classes. However, these results require that the number of jobs in the system be fixed. This section presents an extension to the operational analysis of queueing networks that allows the performance analyst to deal with models which contains multiple classes of jobs and where the number of jobs in each class may vary in a given range, determined by the maximum degree of multiprom ming of each class. Therefore, the number of jobs in the system is not fixed anymore. This is achieved by the class composition algorithm given below.

First we introduce the notation used in this section. Then, we present the algorithm followed by a set of explanatory comments about the crucial steps. The reader is advised to follow the algorithm and the comments in parallel.

Let,

$p_r(n_r|n_{r-1},\ldots,n_1)$: fraction of time that there are n_r class r jobs in the central subsystem given that there are $n_{r-1},\ldots,n_1$ jobs in classes r-1,...,1 respectively.

$p_r(n_r)$: fraction of time that there are n_r class r jobs in the central subsystem, independently of the number of jobs in other classes.

$X^+_{0r}(n_1,\ldots,n_r)$: throughput of class r jobs when there are n_1 class 1 jobs,..,n_r class r jobs, independently of how many jobs there are of classes r+1 through R. For r=R, $X^+_{0r}(n_1,\ldots,n_R) = X_{0R}(n_1,..,n_R)$.

$X_{0r}(n_r)$: throughput of class r jobs when there n_r jobs of this class in the central subsystem, independently of the number of jobs in the other classes.

We assume that any class may be of one of the three types defined in section two, namely batch, timesharing and

[1] More precisely we require that $\left[\frac{L}{X_0(J)}\right]^{N^* - J+1} \ll 1$.

L(N) : average arrival rate of timesharing jobs when there are N jobs in the central subsystem.

$\bar{N}_M$: average number of jobs in main memory

$\bar{N}_Q$: average number of jobs waiting for memory

$\bar{N}_S$: average number of jobs in the central subsystem. $\bar{N}_S = \bar{N}_Q + \bar{N}_M$.

p(N) : fraction of time that there are N jobs in the central subsystem.

Timesharing systems were first analyzed using decomposition methods by Brandwajn [20]. His model is a single job class one. The analysis of the timesharing case done here considers the central subsystem as a load dependent server with mean time between service completions equal to $S_0(N) = 1/X(N)$. Notice that X(N) is the throughput of the central subsystem obtained, under a constant load of N jobs, i.e. studying the offline behavior of the central subsystem. This kind of approach is the decomposition principle introduced by Courtois in [16]. This principle allows the analyst to replace a subsystem by a single state dependent server. The service rate of this server is determined by studying the subsystem in isolation. Little error is obtained with this approximation if the rate at which transitions occur within the subsystem is much greater than the rate at which the subsystem interacts with the rest of the system.

Let us consider the central subsystem in more detail as in Figure 5. If there are N jobs in the central subsystem then min(N,J) jobs are in memory. These jobs may be in one of three different states: ready (R), executing (E) and suspended (S). If there are more than J jobs in the central subsystem, then there will be (N-J) jobs waiting for memory. The central subsystem may now be viewed as being formed by a server and a queue. The throughput of this server is equal to $X_0(N)$ if $N \leq J$ and $X_0(J)$ if $N < J$.

Let us write balance equations for the fraction of time, p(N), that there are N jobs in the central subsystem. If one assumes that the system is in operational equilibrium and if one assumes single arrivals and departures [8] one can obtain the balance equations given below, following an argument similar to the one presented by Buzen [8] to derive the General Birth-Death Formula.

$$p(N) = \frac{N_T - N + 1}{Z} \cdot \frac{1}{X_0(N)} \cdot p(N-1) \quad \text{for } N = 1, \ldots, J \tag{17}$$

$$p(N) = \frac{N_T - N + 1}{Z} \cdot \frac{1}{X_0(J)} \cdot p(N-1) \quad \text{for } N = J+1, \ldots, N_T \tag{18}$$

We also have the normalization equation

$$\sum_{i=0}^{N_T} p(i) = 1 \tag{19}$$

Setting p(0)=1 we obtain a first set of values of p(N) for $1 \leq N \leq N_T$ recursively from equations(17) and (18). In order to get the true values of p(N) one must divide each of the values obtained by the normalization constant $C = \sum_{i=0}^{N_T} p(i)$, where the p(i)'s of the previous summation are the ones obtained in the first place.

The average number of jobs in memory is then calculated as

$$\bar{N}_M = \sum_{i=1}^{J-1} i p(i) + J \sum_{i=J}^{N_T} p(i) \tag{20}$$

The average number of jobs waiting for memory may be calculated as

$$\bar{N}_Q = \sum_{i=1}^{N_T - J} i \, p_Q(i) \tag{21}$$

where $p_Q(i)$ is the fraction of time that there are i jobs waiting for memory. The values of $p_Q(i)$ are obtained directly in terms of the values of p(N) as follows.

$$p_Q(i) = p(i+J) \text{ for } i = 1, \ldots, N_T - J \tag{22}$$

$$p_Q(0) = \sum_{i=0}^{J} p(i) \tag{23}$$

The average number of jobs in the system is now given by

$$\bar{N}_S = \bar{N}_M + \bar{N}_Q \tag{24}$$

Notice, that while $\bar{N}_S$ could be obtained directly as $\sum_{i=0}^{N_T} i \, p(i)$, the analysis above yields several interesting memory utilization reports which include the average degree of multiprogramming ($\bar{N}_M$) and the distribution of jobs in execution and waiting for memory.

Let X_{NT} be the central subsystem throughput when there are N_T terminals.

From the values of p(N) one can calculate X_{NT} as

$$X_{NT} = \sum_{N=0}^{N_T} X_0(N) \, p(N) \tag{25}$$

Job flow balance implies that

$$X_0(N) = \frac{N_T - N}{Z} \tag{26}$$

Replacing (26) in (25) it follows that

$$X_{NT} = \sum_{N=0}^{N_T} \frac{N_T - N}{Z} p(N) = \frac{N_T}{Z} - \frac{1}{Z} \sum_{N=0}^{N_T} N p(N) = \frac{N_T - \bar{N}_S}{Z}$$

Finally, applying Little's law we get the response time, R, of timesharing jobs.

$$R = \frac{\bar{N}_S}{X_{NT}} \tag{27}$$

4.3 TRANSACTION PROCESSING CLASS

Figure 6 presents a computer system subject to a load of transaction type jobs which arrive from terminals at an average arrival rate of L transactions per second.

Let,

L: average arrival rate of transaction type jobs.

J: maximum level of multiprogramming for the transaction class

$X_0(N)$: throughput of the transaction class when there are N transaction type jobs in the central subsystem.

N: current number of active jobs in the system

$\bar{N}_M$, $\bar{N}_Q$ and $\bar{N}_S$: as defined in section 4.2.

p(N) : fraction of time that there are N jobs in the central subsystem.

Similarly to the analysis of the timesharing class done in section 4.2 we are going to consider the central subsystem as a load dependent server with mean time between

transaction. It should be noted however that there may be any number of classes of a given type.

Consider the graphical representation of classes given in Figure 7. This example will be used to elucidate the class composition algorithm. There are three classes of jobs with multiprogramming levels varying between 0 and 1, 0 and 2, and 0 and 3 for classes 1 through 3 respectively. Notice that a path from the root to a leaf of the tree shown in Figure 7 represents one of the possible configurations, and the tree represents all possible configurations.

Class Composition Algorithm

Step 1 - Set r=R

Step 2 - [calculation of class r throughput]. Determine the class r throughput for every possible configuration $(n_1,.....,n_R)$, according to the formula given below

$$X_{0r}(n_1,....,n_r,...,n_R) = \frac{G_M(n_1,...,n_r-1,..,n_R)}{G_M(n_1,.....,n_R)}$$

for $n_1=0,...,N_1,\ n_2=0,..,N_2;...;n_R=0,....,N_R$

Step 3 - [step 4 must be skipped for class R jobs] If r=R then go to step 5.

Step 4 - [calculation of class r throughput independent of higher classes] Determine the class r throughput for every possible configuration $(n_1,.....,n_r)$ independently of the number of jobs in classes r+1 through R, i.e., calculate

$$X^+_{0r}(n_1,....,n_r) = \sum_{n_R=0}^{N_R}\cdots\sum_{n_{r+1}=0}^{N_{r+1}} X_{0r}(n_1,..n_R)\times p_R(n_R|n_{R-1},...,n_r,...,n_1)\times ..... \times p_{r+1}(n_{r+1}|n_r,....,n_1)$$

for $n_1=0,.....,N_1;.....;n_r=0,.....,N_r$

Step 5 - [calculation of conditional fraction of time] Using the values of $X^+_{0r}(n_1,....,n_r)$ obtained in the previous step calculate the probability $p_r(n_r|n_{r-1},..,n_1)$ for $n_1=0,..,N_1;..;n_r=0,..,N_r$. These probabilities are calculated accordingly to the methods described in section 4.1 through 4.3 for each type of class.

Step 6 - Set r=r-1. if r≥1 then go to step 2.

Step 7 - [Obtaining the unconditional fractions of time and throughputs] Calculate the unconditional fraction of time $p_r(n_r)$ for r=2,....,R and for $n_r=0,...,N_r$, as follows.

$$p_r(n_r) = \sum_{n_1=0}^{N_1}\cdots\sum_{n_{r-1}=0}^{N_{r-1}} p_r(n_{r-1},...,n_1)\times p_{r-1}(n_{r-1}|n_{r-2},...,n_1)\times..\times p_2(n_2|n_1)\times p_1(n_1)$$

Calculate the unconditional throughput $X_{0r}(n_r)$ for r=2,...,R and for $n_r=0,...,N_r$ as follows.

$$X_{0r}(n_r) = \sum_{n_1=0}^{N_1}\cdots\sum_{n_{r-1}=0}^{N_{r-1}} X^+_{0r}(n_1,...,n_r) \times p_{r-1}(n_{r-1}|n_{r-2},.....,n_1)\times...\times p_2(n_2|n_1)\times p_1(n_1)$$

Step 8 - [Obtaining the average throughput for each class, over all observed configurations] Calculate

$$X_{0r} = \sum_{n_r=0}^{N_r} X_{0r}(n_r)p_r(n_r) \quad \text{for } r=1,.....,R.$$

Comments on the Class Composition Algorithm

Comment on Step 2: In the example of figure 7, one would calculate $X_{0r}(0,0,0)$, $X_{0r}(1,0,0),.....,X_{0r}(3,2,1)$.

Comment on Step 4: Let us first derive the expression given in Step 4 for $X^+_{0r}(n_1,....,n_r)$. Some definitions are in order. Let

$C_{0r}(n_1,.....,n_R)$: number of class r system completions when there are n_1 class 1 jobs,...,n_R class R jobs.

$C^+_{0r}(n_1,.....,n_r)$: number of class r system completions when there n_1 class 1 jobs,.....,n_r class r jobs.

$I(n_1,.....,n_R)$: amount of time during which there are n_1 class 1 jobs,....,n_R class R jobs.

$I_r(n_1,.....,n_r)$: amount of time during which there are n_1 class 1 jobs,.....,n_r class r jobs.

From the above definitions we may write the obvious relationship below.

$$X^+_{0r}(n_1....,n_r) = \frac{C^+_{0r}(n_1,.....,n_r)}{I_r(n_1,.....,n_r)} \tag{38}$$

$$X_{0r}(n_1,.....,n_R) = \frac{C_{0r}(n_1,....,n_R)}{I(n_1,.....,n_R)} \tag{39}$$

$$C^+_{0r}(n_1,...,n_r) = \sum_{n_R=0}^{N_R}\cdots\sum_{n_{r+1}=0}^{N_{r+1}} C_{0r}(n_1,...,n_R) \tag{40}$$

Using (39) in (40) it follows that

$$C^+_{0r}(n_1,...,n_r) = \sum_{n_R=0}^{N_R}\cdots\sum_{n_{r+1}=0}^{N_{r+1}} X_{0r}(n_1,..,n_R)I(n_1,..,n_R) \tag{41}$$

Dividing both sides of (41) by $I_r(n_1,.....,n_r)$ we get

$$X^+_{0r}(n_1,....,n_r) = \sum_{n_R=0}^{N_R}\cdots\sum_{n_{r+1}=0}^{N_{r+1}} X_{0r}(n_1,....,n_R) \times \frac{I(n_1,.....,n_R)}{I_r(n_1,....,n_r)} \tag{42}$$

Rewriting the ratio $I(n_1,....,n_R)/I_r(n_1,.....,n_r)$ in the form below, we get

$$\frac{I(n_1,.....,n_R)}{I_r(n_1,....,n_R)} = \frac{I(n_1,..,n_R)}{I_{R-1}(n_1,..,n_{R-1})}\times\frac{I_{R-1}(n_1,..,n_{R-1})}{I_{R-2}(n_1,..,n_{R-2})}\times..\times \frac{I_{r+1}(n_1,...,n_{r+1})}{I_r(n_1,.....,n_r)} \tag{43}$$

If we use the fact that

$$p_r(n_r|n_{r-1},....,n_1) = \frac{I_r(n_1,....,n_r)}{I_{r-1}(n_1,...,n_{r-1})}$$

in (42) and (43) we get the expression for $X^+_{0r}(n_1,...,n_r)$ given in Step 4.

In the example of figure 7, for r=2 one would calculate $X^+_{02}(0,0)$, $X^+_{02}(1,0),.....,X^+_{02}(2,1)$ using the values of

$X_{02}(0,0,0)$, $X_{02}(1,0,0),\ldots,X_{02}(3,2,1)$ obtained in step 2 and the values of $p_3(0|0,0),\ldots,p_3(3|2,1)$ obtained in step 5 for r=3.

Comment on Step 5: Assume that class 2 is of type time-sharing. Then the balance equations for $p_2(n_2|n_1)$ would be given by

$$p_2(n_2|n_1) = \frac{N_T-n_2+1}{Z} \times \frac{1}{X^+_{02}(n_1,n_2)} \times p_2(n_2-1|n_1)$$

for $n_2 = 1,\ldots,N_2$

and

$$p_2(n_2|n_1) = \frac{N_T-n_2+1}{Z} \times \frac{1}{X^+_{02}(n_1,N_2)} \times p_2(n_2-1|n_1)$$

for $n_2=N_2+1,\ldots,N_T$

accordingly to equations (17) and (18). The normalization equation would be

$$\sum_{n_2=0}^{N_T} p(n_2|n_1) = 1$$

Comment on Step 7: The expression for $p_r(n_r)$ follows directly from the operational counterpart of the theorem of total probability, which we call theorem of total fraction of time. We state this theorem below. A proof of this theorem can be found in [18].

Theorem of Total Fraction of Time: Let S be a system configuration and let S_i=1,.....n, be system configurations such that

i) if the system is in configuration S, it must be in exactly one of configurations S_i

ii) if the system is in configuration S_i it must not be at configurations S_j for $j\neq i$.

iii) the system must be in configuration S_i for an amount of time greater than zero, for every i=1,....,n

Then

$$p(S) = \sum_{i=1}^{n} p(S|S_i)p(S_i)$$

where $p(S|S_i)$ is the fraction of time that the system is observed in configuration S given that it is in configuration S_i.

From the definition of $p(S|S_i)$ it follows that

$$p(S|S_i) = \frac{I(SS_i)/T}{I(S_i)/T} = \frac{p(SS_i)}{p(S_i)} \quad (44)$$

Now, from this theorem we have that

$$p_r(n_r) = \sum_{n_1=0}^{N_1} \cdots \sum_{n_{r-1}=0}^{N_{r-1}} p_r(n_r|n_{r-1},..,n_1)p(n_{r-1},..n_1) \quad (45)$$

From (44) it follows that

$$p(n_{r-1},\ldots,n_1) = p_{r-1}(n_{r-1}|n_{r-2},..,n_1)\times p(n_{r-2},\ldots,n_1) =$$

$$p_{r-1}(n_{r-1}|n_{r-2},..,n_1)\times p_{r-2}(n_{r-2}|n_{r-3},..,n_1)\times p(n_{r-3},..,n_1) =$$

$$\ldots = p_{r-1}(n_{r-1}|n_{r-2},\ldots,n_1)\times\ldots\times p_2(n_2|n_1)\times p_1(n_1) \quad (46)$$

Finally, using (46) in (45) we get the expression for $p_r(n_r)$ used in step 7.

In order to derive the expression for $X_{0r}(n_r)$ let us first state and prove theorem below.

Theorem of Total Throughput: Let S and S_i for i=1,...,n be system configurations as defined for the theorem of total fraction of time.

Then,

$$X(S) = \sum_{i=1}^{n} X(S|S_i)p(S_i)$$

where $X(S|S_i)$ is the throughput of the system when it is in configuration S given that it is in configuration S_i, and X(S) is the throughput when the system is in configuration S.

Proof: By definition X(S) = C(S)/I(S) where C(S) is the number of completion when the system is in configuration S. Also, by definition

$$X(S|S_i) = \frac{C(SS_i)}{I(S_i)} \quad (47)$$

If we observe that

$$C(S) = \sum_{i=1}^{n} C(SS_i) \quad (48)$$

and if we use (47) in (48) it follows that

$$C(S) = \sum_{i=1}^{n} X(S|S_i)I(S_i) \quad (49)$$

Dividing both sides of (49) by I(S) the theorem is proved.

The expression for $X_{0r}(n_r)$ used in step 7 is obtained as a direct consequence of the theorem of total throughput and of expression (46).

5. CONCLUSIONS

The algorithm presented in section 4 was implemented in PL/1 and several experiments were conducted on an IBM/370 Model 165. Several BEST/1 case studies published in the literature were used to validate our model. Table 3 below presents the comparison between our model and Best/1. We also provide CPU time and memory requirements to run our model.

Cases 1 and 2 have 3 different workload of the types batch processing (BP), transaction processing (TP) and timesharing (TS). Cases 3 through 6 have two workloads of the types TP and BP. Case 1 was run with data from [13] while the remaining ones were run with data from [19].

As shown in section 4, other results such as memory reports (e.g. average number of jobs in memory per class and probability distribution of jobs in memory and waiting for memory) can also be obtained. The class composition algorithm presented here can be used to obtain many other performance measures. For instance, given the expression for the average number of jobs per device for any fixed configuration one can obtain the average number of jobs per device for the varying degree of multiprogramming case using the class composition algorithm presented in section 4.

6. ACKNOWLEDGEMENTS

The authors would like to thank Jeffrey P. Buzen and Alex Thomasian who read this paper and made several useful comments.

REFERENCES

[1] Jackson, J.R. Networks of Waiting Lines, Operations Research, 5, 518-521,(1957).

[2] Jackson, J.R., Jobshop-Like Queueing Systems, Management Science, 10, No.1, 131-142, (1963).

TABLE 2 - Comparison between Our Model and Best/1 Results.

	RESPONSE TIME (seg)		THROUGHPUT		CPU time	MEMORY
	Our Model	Best/1	Our Model	Best/1	(seg)	Kbytes
CASE 1	133.97 3.17 3.23	133.98 3.17 3.31	102 4000 3825	102 4000 3815	2.41	196
CASE 2	4.76 106.15 4.42	4.77 106.18 4.47	4000 203 2614	4000 203 2611	1.79	190
CASE 3	4.44 85.97	4.45 85.98	3000 251	3000 251	0.63	168
CASE 4	17.94 92.38	17.94 92.38	4000 233	4000 234	0.56	168
CASE 5	3.83 92.75	3.83 92.75	4000 232	4000 233	0.69	170
CASE 6	11.90 88.29	11.87 88.29	4000 245	4000 245	0.60	168

[3] Gordon, W.J. and G.F. Newell, Closed Queueing Sys - tems with Exponential Servers, Operations Research, 15, 254-265, (1967)

[4] Buzen, J.P., Queueing Networks Models of Multiprogram ming, Ph.D. Thesis, Div. of Engineering and Applied Science, Harvard Univ. Cambridge, Mass., May 1971 (NTIS AD 731 575, Aug. 1971).

[5] Buzen, J.P., Computational Algorithms for Closed Queueing Networks with Exponential Servers, CACM, 16,527-531, (1973).

[6] Baskett, F., K.M. Chandy, R.R. Muntz, and F. Pala - cios - Gomez, Open, Closed, and Mixed Networks of Queues with Different Classes of Customers, JACM,22, 248-260, (1975).

[7] Buzen, J.P., Fundamental Operational Laws of Compu- ter System Performance, Acta Informatica, 7, 2, 167-182, (1976).

[8] Buzen, J.P., Operational Analysis: the key to the New Generation of Performance Prediction Tools, Proceedings of the IEEE COMPCON, IEEE, New York, (1976).

[9] Denning, P.J. Buzen, Operational Analysis of Queue- ing Networks, Measuring, Modelling and Evaluating Computer Systems, eds. H. Beilher and E. Gelenbe, North-Holland Publishing Company, 151-172,(1977).

[10] Denning, P.J. and J.P. Buzen, The Operational Analy- sis of Queueing Network Models, ACM Computing Survey, 10,3 225-261, (1978).

[11] Bouhana, J., "Operational Aspects of Centralized Queueing Networks", Ph.D. Thesis, Computer Science Dept., Univ. of Wisconsin, Madison, January 1978.

[12] Roode, J.D., Multiclass Operational Analysis of Queue ing Networks Models, Performance of Computer Sys - tems, North Holland Publishing Company, 339-352, (1979).

[13] Buzen, J.P. et al, BEST/1 - design of a tool for computer system capacity planning, Proceedings of 1978 AFIPS NCC, Vol. 47, pp 447-455, (1978).

[14] Bruel,S.C. and G. Balbo, Computational Algorithms for Closed Queueing Networks, North Holland, New York, Oxford, (1980).

[15] Little, J.D.C., A Proof of the Queueing Formula L=λW, Operations Research, 9, 383-387, (1961).

[16] Courtois, P.J., Decomposability, Instabilities and Saturation in Multiprogrammed Systems, CACM, 18, 7 371-377, (1975).

[17] Lam, S.S., Queueing Networks with Population Size Constraints, IBM Journal of Research and Develop - ment, Vol. 21, Nº 4, 370-378, (1977).

[18] Menasce, D.A. and Virgilio A.F. Almeida, A Multi - class Queueing Network Model of Computer Systems with Variable Degree of Multiprogramming, Technical Report PE 108001, Departamento de Informatica, PUC/RJ, October 1980.

[19] Saxton, W.A and M. Edwards, Modeling Tool Exposes Teleprocessing Threats, Infosystems, October 1979.

[20] Brandwajn, A., A Model of a Time Sharing Virtual Memory Systems Solved Using Equivalence and Decom- position Methods, Acta Informatica, 4, 11-47 (1974).

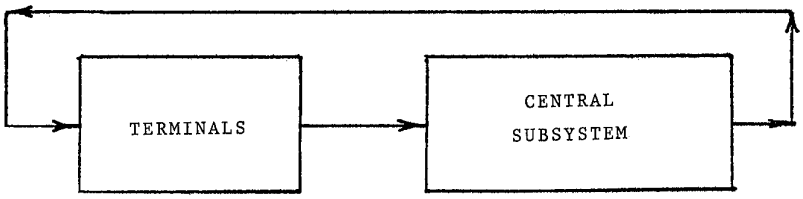

Figure 1 - Computer System Model

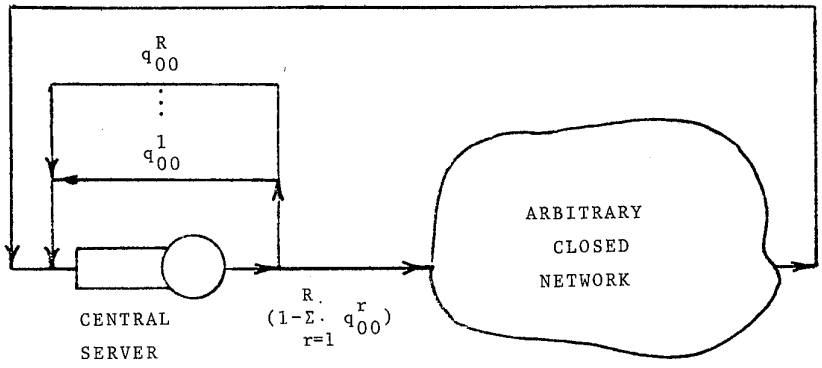

Figure 2 - Centralized Network

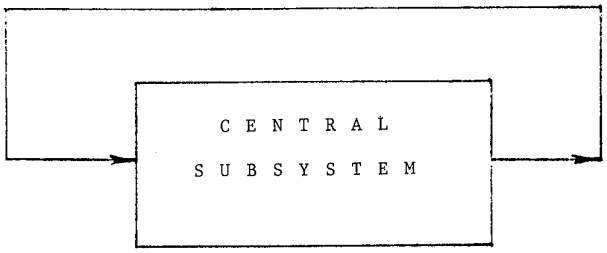

Figure 3 - Batch Processing

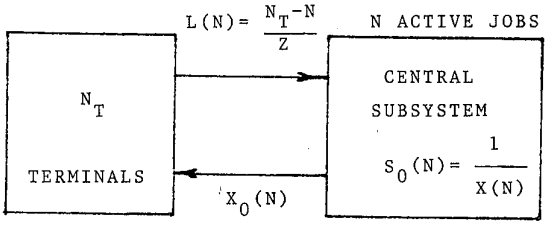

Figure 4 - Timesharing Class

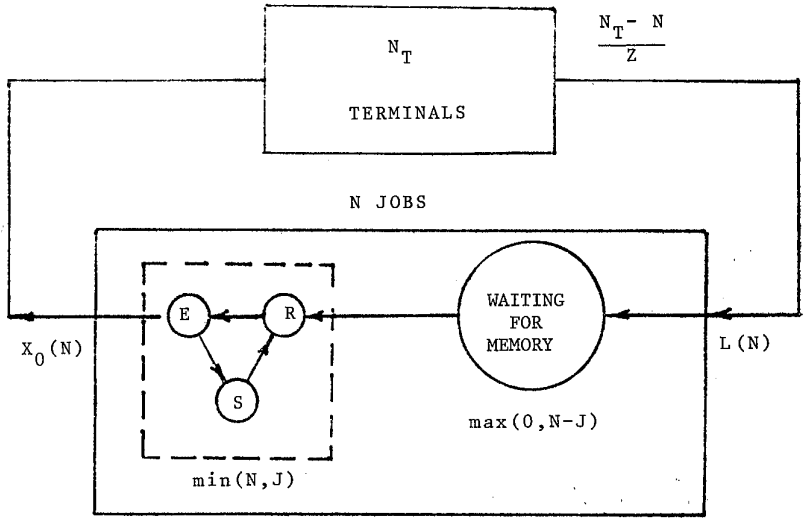

Figure 5. Detail of Central Subsystem

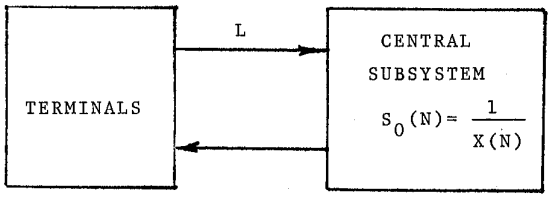

Figure 6 - Transaction Processing Class

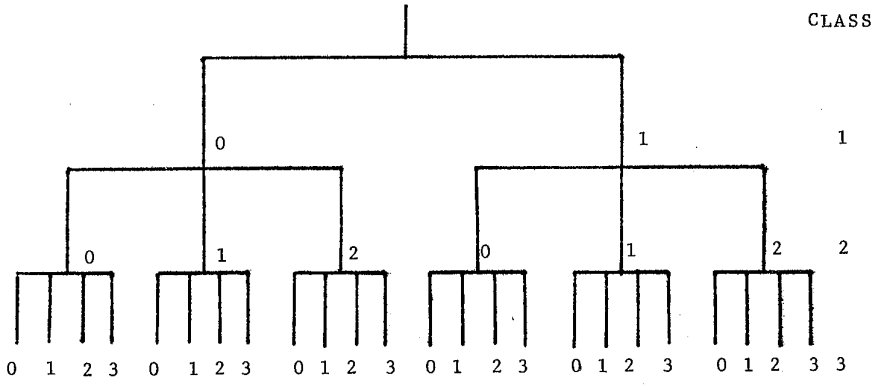

Figure 7 - Graphical Representation of three classes.