Introduction to Analytical Models [1]

Eun Jung Kim   Ki Hwan Yum   Chita R. Das

[⋆]Department of Computer Science
Texas A&M University
College Station, TX 77843
ejkim@cs.tamu.edu

[†]Department of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249
yum@cs.utsa.edu

[‡]Department of Computer Science & Engineering
Pennsylvania State University
University Park, PA 16802
das@cse.psu.edu

[1]Submitted to Lizy John (ljohn@sunfire2.ece.utexas.edu) for review as a chapter in her book on performance evaluation, February 2005.

0.1 Introduction

Performance analysis of a computer system is required to provide intelligent answers to a variety of questions that arise during various stages of its life cycle. The three techniques used for performance analysis are measurement, simulation and analytical modeling. Measurement is the most credible approach to accurately evaluate the performance of a system, but it is very costly and many times not feasible simply because the target system may not be available or not yet built. Thus, simulation is an effective technique to predict the performance of any existing or new system. A simulation program, usually written in a high-level programming language, still requires tremendous amount of time and computing resources to model a relatively large or complex system precisely. On the other hand, analytical models attempt to capture the behavior of a computer system quite effectively and can provide quick answers to many questions. However, the model can become intractable as the system complexity increases.

For example, a very simple model of the average time required for a processor to access the main memory in the event of a cache miss is given by

$$T = T_c + (1 - h) \times T_m.$$

In this model, T is the average memory access time that we are trying to determine, h is the fraction of all memory references that hit in the cache (the hit ratio), T_c is the time required to read a block from the cache, and T_m is the time required to read a block from the main memory. While this model is an over-simplification of how the memory system of a computer behaves, it does provide some insights into the effect that certain parameter changes, such as the cache hit ratio or the memory access time, for instance, will have on the average memory access time.

While simple analytical models provide a bird's-eye view of a target system, sophisticated analytical models can be developed to obtain detailed information about how the system may behave under different conditions. For example, such models can be used to find an upper bound on the number of requests that a system is capable of handling or to quantify the average queueing time of jobs inside the system. These types of models are also useful for quickly analyzing the impact of system and workload parameters on system performance.

The motivation of this chapter is to give a quick review of the mathematical techniques that can be used for performance analysis of computer systems. It contains two different techniques that have been widely used; Queueing theory and Petri net model. Queueing theory is the study of "waiting in line" [1], and is used to predict, for example, how long a job stays in a queue, how many jobs are in the system, and what is the system throughput. Petri net [2], on the other hand, is a graphical and mathematical modeling tool, which is particularly useful for capturing concurrency and synchronization behaviors [3].

0.2 Queueing Theory

This section provides a concise review of the important queueing models used in analyzing computer systems. First, we summarize the relevant stochastic process concept that is required to understand queueing systems. Then, we describe a generic queueing system and introduce some fundamental laws of Queueing theory before summarizing the main results of single queue systems. Finally, the solution techniques for network of queues are included to aid in analyzing complete systems.

0.2.1 Stochastic Processes

In studying queueing systems, we need to deal with not only several random variables but also several different sequences that are functions of time. For example, the number of jobs in a queue at time t, $n(t)$, is a random variable. To specify its behavior, we need to know the probability distribution function for $n(t)$. Similarly, the waiting time in a queue, $w(t)$, is a random time dependent parameter. Such random functions of time, or sequences are called *stochastic processes*. Such processes are used to present the behavior of a queueing system in terms of its state space. The types of stochastic processes that are used in queueing system analysis are the following:

- Discrete-State and Continuous-State Processes: Depending on the values that the random variables can take, the process can be discrete or continuous. For example, $n(t)$ is a discrete-state process while $w(t)$ is a continuous-state process.
- Markov Processes: If the future states of a process are independent of its past and depends only on the present state, the process is called a Markov process. A discrete-state continuous-time Markov process is called a Markov chain. The most important property of a Markov process is its memoryless behavior, which is captured by the exponential distribution for continuous-time processes and geometric distribution for discrete-time processes.
- Birth-Death Processes: A Markov process in which the transitions are restricted to neighboring states only is called a birth-death process.
- Poisson Processes: If the interarrival times of a Markov process are independent and identically distributed (IID), the number of arrivals n over a given time interval $(t, t+x)$ has a Poisson distribution. The arrival process is called a Poisson process, or Poisson stream. A Poisson process has the following properties:

  (i) Merging of k Poisson processes with mean rates λ_i $(i = 1, 2, \cdots, k)$ results in a Poisson process with a mean rate given by

$$\lambda = \sum_{i=1}^{k} \lambda_i$$

(ii) If a Poisson process with a mean rate λ is split into k sub-processes such that the probability of a job going to the ith sub-process is p_i, each sub-process is also a Poisson process with a mean rate of $p_i\lambda$.

(iii) If the arrivals to a single server with exponential service time are a Poisson process with a mean rate λ, the departures are also a Poisson process with the same rate λ, provided that the arrival rate λ is less than the service rate μ.

(iv) If the arrivals to a service center with m servers are Poisson with a mean rate λ, the departures also constitute a Poisson stream with the same rate λ, provided that the arrival rate λ is less than the total service rate $\sum_i \mu_i$ of all m servers.

With these definitions, let us now look at a generic queueing system.

0.2.2 A Generic Queueing System

A generic queueing system is represented by a six-tuple notation, given by A/S/m/B/N/SD, where the first term stands for the arrival process, the second term represents the service time distribution, the third term denotes number of servers, the fourth term represents the buffer or queue size, the fifth term represents the population size, and the last term represents the service discipline [4]. A general queueing system depicting the six terms is shown in Figure 1.

The arrival process to a queueing system is characterized by the interarrival time distribution of the customers to the system. The interarrival times are usually assumed to be independent and identically distributed random variables. Similarly, the amount of time a customer spends at a server is given by the underlying service time distribution. While any known distribution can be used to represent the arrival and service time distributions, the commonly used distributions are Exponential or Memoryless (M), Deterministic (D) and General (G). Other distributions such as Erlang and hyperexponential have been used to

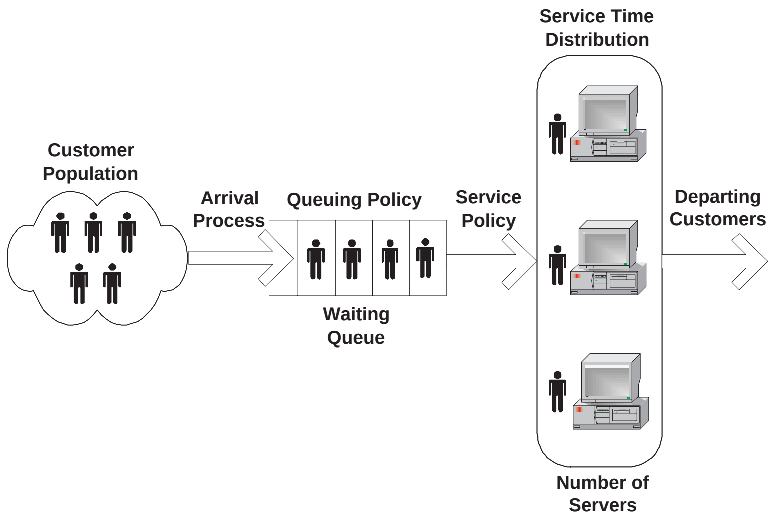


Figure 1: A Generic Queueing System

capture the service time variation of computer systems [5].

The third parameter, number of servers, specifies how many concurrent jobs can be served by the service center, while the fourth parameter captures the actual buffering capacity of the server. The fourth parameter is usually dropped from a queueing system notation if the buffer size is sufficiently large so that it can be considered an infinite capacity. Similarly, if the customer population is large, it can be assumed as infinite and the fifth parameter is dropped from the system representation. The last term, service discipline (SD), represents the order in which customers are serviced at the center. Starting from the most common First Come First Served (FCFS) scheduling to other policies such as round robin (RR), process sharing (PS) and other prioritized policies can be captured by this parameter.

Based on this notation, a few queueing systems are described next.

$M/M/1$: This is the simplest queueing system to analyze. The arrival and service times are exponentially distributed (Poisson processes), and the system consists of only one server. This queueing system can be applied to a wide variety of problems since any system with a very large number of independent customers can be approximated as a Poisson process. However, exponential service time distribution is not realistic for many applications, and thus, is only a crude approximation.

$M/D/n$: The arrival process is a Poisson process and the service time distribution is deterministic. The system has n servers (e.g. a ticket booking counter with n cashiers.), and the service time is the same for all customers.

$G/G/n$: This is the most general queueing system, where the arrival and service time distributions are both arbitrary. The system has n servers. This is the most complex system, for which no analytical solution is known.

For such queueing systems, we are interested in determining output parameters such as the average number of customers in the system (or in the queue), average response time and throughput of the system. Although the arrival and service time distributions are required to analyze a queueing system as a stochastic process for in-depth understanding of the system behavior, many times a set of simple relationships can be used for quick estimate of the system parameters. These relationships, called operational laws [6, 7] or fundamental laws are summarized in the following subsection.

0.2.3 Fundamental Laws

A large number of day-to-day problems in computer system performance analysis can be solved by using some simple relationships that do not require any assumptions about the distribution of service times and arrival times. Several such relationships, called operational laws, were identified originally by Buzen (1976) [6] and later extended by Denning and Buzen (1978) [7]. Operational means directly measurable quantities such as throughput and arrival rate.

Let us assume that A is the number of arrivals during time T to the queueing system, depicted in Figure 1, C is the number of completion during this observation period, and B is the system busy time. Using these measured quantities, we can define the following simple relations:

- Arrival rate
$$\lambda = \frac{A}{T}$$
- Throughput
$$X = \frac{C}{T}$$
- Utilization
$$U = \frac{B}{T}$$
- Mean service time
$$S = \frac{B}{C}$$

Utilization Law

$$U = \frac{B}{T} = \frac{C}{T} \times \frac{B}{C} = XS$$

Forced Flow Law: In a queueing system with several resources, the number of arrivals to a device i is equal to its number of departures,

$$A_i = C_i.$$

Simplification of this yields the throughput of device i as

$$X_i = \frac{C_i}{T} == XV_i,$$

where V_i denotes the number of visits (visit count) to device i. This is called the forced flow law since it enforces a balance system.

Little's Law: Little's theorem states that the average number of customers (N) can be determined as,

$$N = \lambda R, \tag{1}$$

where λ is the average customer arrival rate and R is the average service time of a customer. The proof of this theorem can be found in any standard textbook on queueing theory [1]. Here, we will focus on an intuitive understanding of the result. Consider an example of a restaurant where the customer arrival rate (λ) doubles, but the customers still spend the same amount of time in the restaurant (R). This will double the number of customers in the restaurant (N). By the same logic, if the customer arrival rate remains the same, but the customers service time doubles, this will also double the total number of customers in the restaurant. Little's Theorem is an important relation that relates the three important parameters of a queueing system.

General Response Time Law: Using Little's Law, we have

$$Q = XR$$

where Q is the total number of customers in the system and the job flow is balanced so that arrival is equal to departure. If Q_i denotes the queue length at device i, we have

$$Q = Q_1 + Q_2 + \cdots + Q_M \text{ for a system with } M \text{ queues.}$$

Since Little's Law can be applied to any device as long as the balance of the flow is maintained, we have

$$XR = X_1R_1 + X_2R_2 + \cdots + X_MR_M$$

Dividing both sides by X and using forced flow law, we get the average response time R as

$$R = \sum_{i=1}^{M} R_iV_i$$

This is called the general response time law, where the system response time is the summation of the product of the response time and visiting counts at each of the device. This is a very intuitive, yet extremely important result that relates the three system parameters.

Interactive Response Time Law: In an interactive system, where users have an average think time Z before generating requests that are serviced by the system with an average response time R, the total time spent in the system is $R + Z$. Each user generates about $T/(R + Z)$ requests in the time period T. $(R + Z)$ is the average response time per request, and thus, $1/(R+Z)$ is the number of requests per unit time. System throughput is expressed as

$$X = \frac{N[T/(R + Z)]}{T} = \frac{N}{R + Z}$$

or

$$R = (N/X) - Z.$$

0.2.4 The $M/M/1$ Queue

In this subsection we summarize the $M/M/1$ queueing system. As we have seen earlier, $M/M/1$ refers to negative exponential arrival and service times with a single server. This is the most widely used queueing system and is a good approximation for a large number of systems. It assume a Poisson arrival process, which is a very good approximation for the arrival processes in real systems that meet the following conditions:

- The number of customers in the system is very large;
- Impact of a single customer on the performance of the system is very small, i.e. a single customer consumes a very small percentage of the system resources;
- All customers are independent, i.e. their decisions to use the system are independent of each other.

Now that we have established scenarios where we can assume an arrival process to be Poisson, let us look at the probability density distribution for a Poisson process. The probability of seeing n arrivals in a period 0 to t is given by

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t},$$

where t is the time interval 0 to t and n is the total number of arrivals in the interval 0 to t, and λ is the average arrival rate in arrivals/sec.

Suitability of an $M/M/1$ queue is easy to identify from the server standpoint. For example, a single transmitting queue feeding an outgoing link is a single server and can be modeled as an $M/M/1$ queueing system. If a single transmit queue feeds two load-sharing links, it should be modeled as an $M/M/2$ queue.

Queueing models are usually solved using Markov Chain (MC) models. For example, an $M/M/1$ queue is represented by the following state transition diagram in Figure 2, where each state of the queue gives the number of jobs in the system. Arrival of a job, with an average rate λ, increases the number of jobs in the system, while departure of a job after completing service at the server reduces the number of jobs by one. The solution of this MC gives the probabilities of all states in the system [5, 4]. The state probabilities are used to estimate the performance parameters.

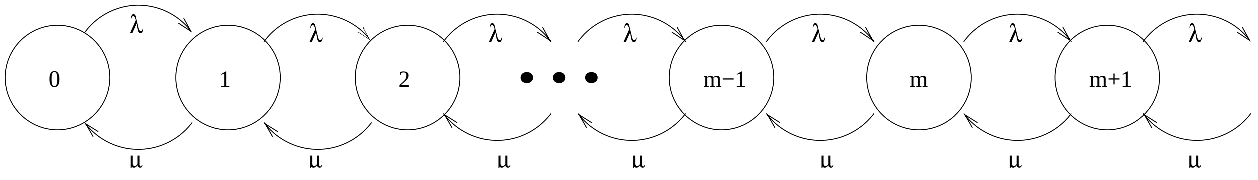


Figure 2: State Transition Diagram of the $M/M/1$ system

First we define ρ, the traffic intensity (sometimes called occupancy) as $\rho = (\lambda \ / \ \mu)$. For a stable system, the average service rate should always be higher than the average arrival rate. (Otherwise the queues would grow indefinitely). Thus, ρ should always be less than one. Also, note that since we are using average rates here, the instantaneous arrival rate may exceed the service rate. Over a longer time period, the service rate should always exceed the arrival rate.

The solution of the above MC gives the state probabilities, which can be used to find the performance parameters.

Now, using the state probabilities, the mean number of customers in the system (N) becomes

$$E[N] = \sum_{i=1}^{\infty} np_n = \frac{\rho}{1-\rho}(1)$$

Note that as ρ approaches 1, the number of customers would become very large. This can be easily justified intuitively. ρ will approach 1 when the average arrival rate starts approaching the average service rate. In this situation, the server would always be busy, and a lead to a queue build up.

The average response time including service time, is computed using Little's Law as $N = \lambda R$ or

$$E[R] = \frac{1}{\mu - \lambda}$$

Again we see that as mean arrival rate (λ) approaches mean service rate (μ), the waiting time becomes very large. Figure 3 depicts the response time behavior of an $M/M/1$ queue. For other parameters, the readers should refer to [5, 4].

0.2.5 The $M/M/m$ Queue

The $M/M/m$ queueing system is identical to the $M/M/1$ system except that there are m servers. A customer at the head of the queue is routed to any server that is available. The Markov model for the $M/M/m$ system is given below.

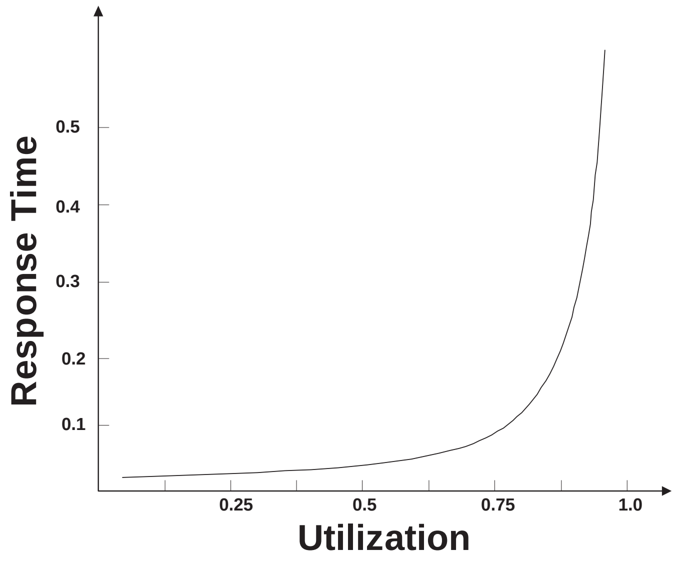


Figure 3: Response Time for the $M/M/1$ system

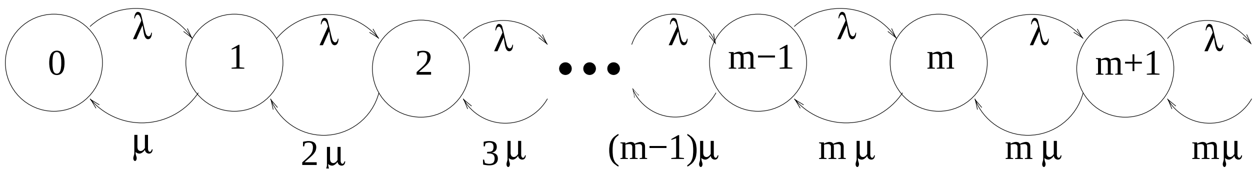


Figure 4: State Transition Diagram of the $M/M/m$ system

As shown in Figure 4, in the first m stages, the service rate increases linearly because more servers are available. After the m customers are busy, the service rate remains constant at $m\mu$. Using the state diagram we obtain the state probabilities as

$$p_n = \begin{cases} p_0 \frac{(m\rho)^n}{n!}, & n \leq m \\ p_0 \frac{m^m \rho^n}{m!}, & n > m, \end{cases}$$

where ρ is given by

$$\rho = \frac{\lambda}{m\mu} < 1.$$

We can calculate p_0 using the relation of $\sum_{n=0}^{\infty} p_n = 1$, which gives

$$p_0 = 1/(1 + \frac{(m\rho)^m}{m!(1-\rho)} + \sum_{n=0}^{m-1} \frac{(m\rho)^n}{n!}).$$

The probability that a customer has to wait is denoted by

$$\begin{aligned} P_Q &= \sum_{n=m}^{\infty} p_n = \sum_{n=m}^{\infty} \frac{p_0 m^m \rho^n}{m!} \\ &= \frac{p_0 (m\rho)^m}{m!} \sum_{n=m}^{\infty} \rho^{n-m} \end{aligned}$$

and reduces to

$$P_Q \approx \frac{p_0 (m\rho)^m}{m!(1-\rho)}.$$

This is known as the well known *Erlang C formula.* This equation is widely used in telephony to estimate the probability of a call request finding all of the m lines busy.

We can now calculate the mean number of customers (N) in the system. It is easier to calculate the number of customers in the queue and number in service separately. The number of customers in the queue is

$$E[N] = \sum_{n=m+1}^{\infty} (n-m) p_n = P_Q \frac{\rho}{1-\rho}, \text{ and}$$

the number of customers in the server is

$$E[N] = \sum_{n=0}^{m-1} n p_n + \sum_{n=m}^{\infty} m p_n = m\rho.$$

The total number of customers in the system is

$$E[N] = E[n_q] + E[n_s] = m\rho + P_Q \frac{\rho}{1-\rho}.$$

The service time is computed as

$$E[R] = \frac{E[N]}{\lambda} = \frac{1}{\mu}(1 + \frac{P_Q}{m(1-\rho)}).$$

0.2.6 $M/M/m/B$ Queue with finite buffer

The $M/M/m/B$ system is similar to the $M/M/m$ queue except that the number of buffers B is finite. After the B buffers are full, all arrivals are lost. We assume that B is greater than or equal to m.

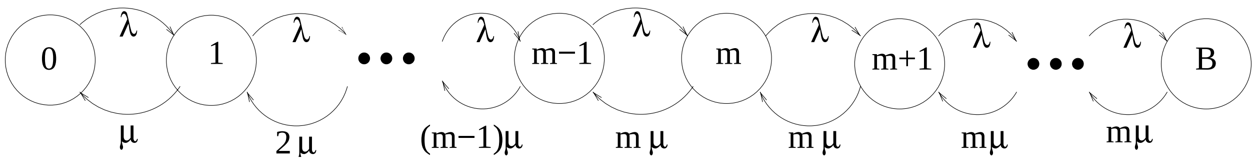


Figure 5: Discrete-time Markov chain for the $M/M/m/B$ system

The state transition diagram for an $M/M/m/B$ queue is shown in Figure 5. The arrival and service rates in the system with n jobs are

$$\lambda_n = \lambda, \;\; n = 0, 1, 2, \ldots, B-1$$

$$\mu_n = \begin{cases} n\mu, & n = 0, 1, 2, \ldots, m-1, \text{and} \\ m\mu, & n = m, m+1, \ldots, B. \end{cases}$$

The probability of n jobs in the system in terms of the traffic intensity $\rho = \lambda/m\mu$ is

$$p_n = \begin{cases} \frac{(m\rho)^n}{n!} p_0, & n = 1, 2, \ldots, m-1, \text{ and} \\ \frac{\rho^n m^m}{m!} p_0, & n = m, m+1, \ldots, B. \end{cases}$$

The probability of zero jobs in the system is calculated by the following equation.

$$\sum_{n=0}^{B} p_n = 1.$$

This gives

$$p_0 = [1 + \frac{1 - \rho^{B-m+1}(m\rho)^m}{m!(1-\rho)} + \sum_{n=1}^{m-1} \frac{(m\rho)^n}{n!}]^{-1}.$$

The mean number of customers (N) in the system is

$$E[N] = \sum_{n=1}^{B} np_n$$

For $m = 1$,

$$E[N] = \frac{\rho}{1-\rho} - \frac{(B+1)\rho^{B+1}}{1-\rho^{B+1}}.$$

The mean number of jobs in the queue is

$$N_Q = \sum_{n=m+1}^{B} (n-m)p_n.$$

For $m = 1$,

$$N_Q = \frac{\rho}{1-\rho} - \rho\frac{1+B\rho^B}{1-\rho^{B+1}}.$$

In the state $n = B$, the effective arrival rate is

$$\lambda' = \sum_{n=0}^{B-1} \lambda p_n = \lambda(1-p_B),$$

The average service time becomes

$$E[R] = \frac{E[N]}{\lambda'} = \frac{E[N]}{\lambda(1-p_B)}.$$

0.2.7 The $M/G/1$ Queue

Many queueing models used in performance analysis assume exponential interarrival times and exponential service times, which are covered by $M/M/m$ systems in the previous sections. In a $M/G/1$ queue, general service times are used.

The parameters are:

- λ: arrival rate in jobs per unit time
- $E[s]$: mean service time per job
- C_s: coefficient of variation of the service time.

The traffic intensity, ρ, is defined as $\lambda E[s]$ and the system is stable if the traffic intensity ρ is less than 1. The probability of zero jobs in the system is $p_0 = 1 - \rho$ and the mean number of jobs in the system is

$$E[N] = \rho + \rho^2(1 + C_s^2)/[2(1-\rho)].$$

This equation is known as the Pollaczek-Khinchin (P-K) mean-value formula. It should be noted that the mean number in the queue grows linearly with the variance of the service time distribution. The variance of number of jobs in the system is

$$\mathrm{Var}[n] = E[n] + \lambda^2 \mathrm{Var}[s] + \frac{\lambda^3 E[s^3]}{3(1-\rho)} + \frac{\lambda^4 (E[s^2])^2}{4(1-\rho)^2}.$$

Then the mean number of jobs in the queue is

$$E[n_q] = \rho^2(1 + C_s^2)/[2(1-\rho)]$$

and the variance of the number of jobs in the queue is

$$\mathrm{Var}[n_q] = \mathrm{Var}[n] - \rho + \rho^2.$$

The mean response time is

$$E[R] = E[N]/\lambda = E[s] + \rho E[s](1 + C_s^2)/[2(1-\rho)]$$

and the variance of the response time is

$$\mathrm{Var}[r] = \mathrm{Var}[s] + \lambda E[s^3]/[3(1-\rho)] + \lambda^2 (E[s^2])^2/[4(1-\rho)^2].$$

The mean waiting time is

$$E[w] = \rho E[s](1 + C_s^2)/[2(1-\rho)].$$

For example, if we consider the $M/D/1$ system then $C_s = 0$ and

$$E[w] = \rho E[s]/[2(1-\rho)].$$

0.2.8 Networks of Queues

While all the systems we have seen so far have only one queue, there exist many systems that contain multiple queues. Unlike single queueing systems, there is no easy notation for specifying the type of a queueing network. The simplest way to classify a queueing network

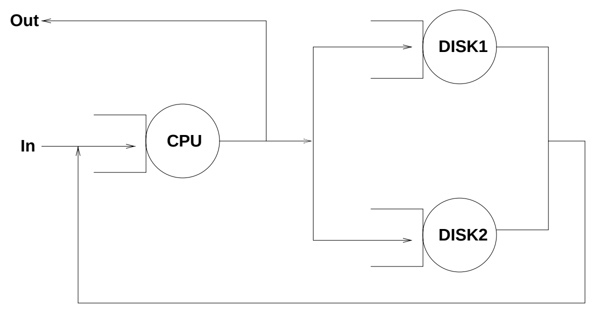


Figure 6: An Open Queueing Network

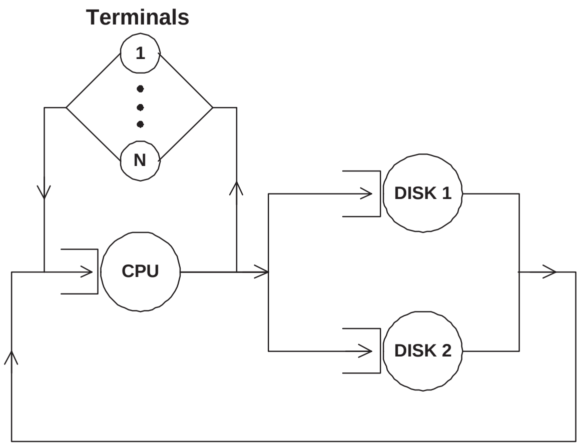


Figure 7: A Closed Queueing Network

is either it is open or closed. An open queueing network has external arrivals and departures as shown in Figure 6. The jobs enter the system at “In” and exit at “Out.” The number of jobs in the system varies with time. In analyzing an open system, we assume that the throughput is known and is equal to the total arrival rate, and the goal is to characterize the distribution of number of jobs in the system, and average response time.

Solution of Open Queueing Networks: Typically assuming it, each queue can be analyzed independently. Open queueing networks are used to represent transaction processing systems such as airline reservation systems or banking systems. The key feature is that the arrival rate does not depend on the load of the system. For all fixed-capacity service centers in an open queueing network, the response time is

$$R_i = S_i(1 + Qi).$$

Intuitively, an arriving job sees Q_i jobs ahead of it, each of which will spend S_i time in service, thus the total response time is all of its predecessors plus itself. This is not an operational law since it assumes the memory-less property of the arrivals (i.e. Poisson arrival) which cannot be tested.

Assuming job flow balance, the throughput is equal to the arrival rate $X = \lambda$. The throughput of the ith device using the forced law is $X_i = XV_i$. The utilization of the ith device using the utilization law is

$$U_i = X_iS_i = XV_iS_i = \lambda D_i.$$

The queue length of the ith device using Little's Law is

$$Q_i = X_iR_i = X_iS_i(1 + Q_i) = U_i(1 + Q_i)$$

or

$$Q_i = \frac{U_i}{1 - U_i}$$

and the response time for device i is

$$R_i = \frac{S_i}{1 - U_i}.$$

Closed Queueing Networks: On the other hand, a closed queueing network has no external arrivals or departures as shown in Figure 7. The jobs in the system keep circulating from one queue to the next. The total number of jobs in the system is constant. In analyzing a closed system, we assume that the number of jobs (N) is given, and we attempt to determine the throughput or the average service time. Two types of technique can be used to analyze closed network of queues: MVA and Convolution [1].

[1] For convolution algorithm, please refer to [5, 4].

Mean value analysis allows solving closed queueing networks in a manner similar to that used for open queueing networks. Key differences are no outside arrivals, no departures, arrival rate at a device may depend on the load of the device. We will consider only fixed-capacity device here. Other variations are more complicated.

Given a closed queueing network with N jobs and the service is memoryless (exponential), Reiser and Lavenberg (1980) showed that the response time of the ith device is given by

$$R_i(N) = S_i[1 + Qi(N-1)].$$

This form is similar to the open queueing network. The response time with N jobs depends on the one with $N-1$ jobs. We can calculate the one with zero jobs in it and then extend to $1, 2, \cdots N-1, N$.

Given the response times at individual devices, the system response time using the general response time law is

$$R(N) = \sum_{i=1}^{M} V_i R_i(N).$$

The system throughput using the interactive response time law is

$$X(N) = \frac{N}{R(N) + Z}.$$

The device throughput measured in terms of jobs per second is $X_i(N) = X(N)V_i$. The device queue length with N jobs in the network using Little's Law is

$$Q_i(N) = X_i(N)R_i(N) = X(N)V_iR_i(N).$$

Consider a model of a computer CPU connected to an I/O device. We have a closed network with each job reentering the CPU directly with probability p_1 or after using the I/O device with probability $p_2 = 1 - p_1$. There are N jobs in the system. We can assume that $X_1(N) = \mu_1, X_2(N) = p_2\mu_1$. With these device throughput(CPU throughput X_1, I/O throughput X_2, we can obtain each device queue length and the system throughput using the above equations.

0.3 Petri Nets

Petri nets are a graphical and mathematical modeling tool applicable to many systems [3]. A Petri net is a particular kind of directed graph, together with an initial state called the initial marking, M_0. The underlying graph Z of a Petri net is a directed, weighted, bipartite graph consisting of two kinds of nodes, called places and transitions, where arcs are either from a place to a transition or from a transition to a place. In graphical representation, places are drawn as circles, and transitions as bars or boxes. Arcs are labeled with their weights (positive integers), where a k-weighted arc can be interpreted as the set of k parallel arcs. Labels for unity weight are usually omitted. A marking (state) assigns to each place a nonnegative integer. If a marking assigns to place p a nonnegative integer k, we say that p is marked with k tokens. Pictorially, we place k black dots (tokens) in place p. A marking is denoted by M, an m-vector, where m is the total number of places. The pth components of M, denoted by $M(p)$, is the number of tokens in place p.

In modeling, places represent conditions and transitions represent events. A transition (an event) has a certain number of input and output places representing the pre-conditions and post-conditions of the event, respectively. The presence of a token in a place is interpreted as holding the truth of the condition associated with the place. In another interpretation, k tokens are put in a place to indicate that k data items or resources are available.

A formal definition of a Petri net is as follows.

Definition 1 *A Petri net is a 5-tuple,* $PN = (P, T, F, W, M_0)$ *where*

$P = \{p_1, p_2, \ldots, p_m\}$ *is a finite set of places,*

$T = \{t_1, t_2, \ldots, t_n\}$ *is a finite set of transitions,*

$F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs (flow relation),*

$W : F \rightarrow \{1, 2, \ldots\}$ *is a weight function,*

$M_0 : P \rightarrow \{0, 1, 2, \ldots\}$ *is the initial marking,*

$P \cap T = \emptyset$ *and* $P \cup T \neq \emptyset$.

A Petri net structure $Z = (P, T, F, W)$ *without any specific initial marking is denoted by* Z.

A Petri net with the given initial marking is denoted by (Z, M_0).

In order to simulate the dynamic behavior of a system, a state or marking in a Petri nets is changed according to the following transition (firing) rule:

1. A transition t is said to be enabled if each input place p of t is marked with at least $w(p, t)$ tokens, where $w(p, t)$ is the weight of the arc from p to t.

2. An enabled transition may or may not fire (depending on whether or not the event actually takes place).

3. A firing of an enabled transition t removes $w(p, t)$ tokens from each input place p of t, and adds $w(t, p)$ tokens to each output place p of t, where $w(t, p)$ is the weight of the arc from t to p.

A transition without any input place is called a source transition, and one without any output place is called a sink transition. It should be noted that a source transition is unconditionally enabled, and that the firing of a sink transition consumes tokens, but does not produce any. A pair of a place p and a transition t is called a self-loop if p is both an input and output place of t. A Petri net is said to be pure if it has no self-loops. A Petri net is said to be ordinary if all of its arc weights are 1's.

The firing of a transition may transform a PN from one marking into another. With respect to a given initial marking M_0, the *reachability set* is defined as the set of all markings reachable through any possible firing sequences of transitions, starting from the initial marking [5]. The evolution of a PN can be completely described by its *reachability graph*, in which each marking in the reachability set is a node in the graph, while the arcs describe the possible marking-to-marking transitions. Arcs are labeled with the name of the transition whose firing caused the associated changes in the marking.

For the above rule of transition enabling, it is assumed that each place can accommodate an unlimited number of tokens. Such a Petri net is referred to as an infinite capacity net. For modeling many physical systems, it is natural to consider an upper limit to the number of tokens that each place can hold. Such a Petri net is referred to as a finite capacity net. For a finite capacity net (Z, M_0), each place p has an associated capacity $K(p)$, the maximum number of tokens that p can hold at any time. For finite capacity nets, for a transition t to be enabled, there is an additional condition that the number of tokens in each output place p of t cannot exceed its capacity $K(p)$ after firing t.

This rule with the capacity constraint is called the strict transition rule, whereas the rule without the capacity constraint is called the (weak) transition rule. Given a finite capacity net (Z, M_0), it is possible to apply either the strict transition rule to the given net (Z, M_0) or, equivalently, the weak transition rule to a transformed net (Z', M_0'), the net obtained from (Z, M_0) by the following complementary-place transformation, where it is assumed that N is pure.

Step 1: Add a complementary place p' for each place p, where the initial marking of p' is given by $M_0'(p') = K(p) - M_0(p)$.

Step 2: Between each transition t and some complementary places p', draw new arcs (t, p')

or (p', t) where $w(t, p') = w(p, t)$ and $w(p', t) = w(t, p)$, so that the sum of tokens in place p and its complementary place p' equals its capacity $K(p)$ for each place p, before and after firing the transition t.

0.3.1 Modeling with Petri Nets

Petri nets are used in the modeling of a specific class of problems, the class of discrete-event systems with concurrent or parallel events. Petri nets model systems, and particularly two aspects of systems, events and conditions, and the relationships among them.

For example, consider the following description of a computer system [8]:

- Jobs appear and are put on an input list. When the processor is free, and there is a job on the input list, the processor starts to process the job.
- When the job is complete, it is placed on an output list, and if there are more jobs on the input list, the processor continues with another job; otherwise it waits for another job.

We can identify several conditions of interest: The processor is idle; A job is on the input list; A job is being processed; A job is on the output list. Also we can identify several events: A new job enters the system; Job processing is started; Job processing is completed; A job leaves the system.

Figure 8 illustrates the modeling of this system. The "job enters" transition in this example is a source and the "job leaves" transition is a sink. This example shows several characteristics of Petri nets and the systems they can model. One is inherent *concurrency* or *parallelism*. There are two main kinds of independent entities in the system: the job and the processor. In the Petri net model, the events which relate solely to one or the

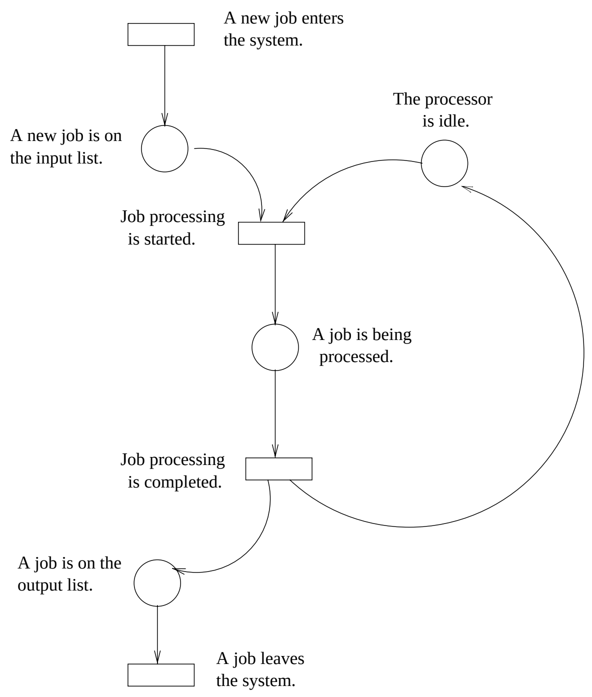

A new job enters the system.
The processor is idle.
A new job is on the input list.
Job processing is started.
A job is being processed.
Job processing is completed.
A job is on the output list.
A job leaves the system.

Figure 8: Modeling of a Simple Computer System

other can occur independently; there is no need to synchronize the actions of the jobs and the processor. However, when synchronization is necessary (for example, when both a job and an idle processor must be available for processing to start), the situation can be easily modeled.

Another major feature of Petri nets is their asynchronous nature. There is no inherent measure of time or the flow of time in a Petri net. This reflects a philosophy of time which states that the only important property of time, from a logical point of view, is in defining a partial ordering of the occurrence of events. Events take variable amounts of time in real life; the Petri net model reflects this variability by not depending on a notion of time to control the sequence of events. Therefore, the Petri net structure itself must contain all necessary information to define the possible sequences of events of a modeled system. Thus in Figure 8, the event 'Job processing is completed' must follow the corresponding event 'Job processing is started' because of the structure of the Petri net although no information is given or considered concerning the amount of time required to process a job.

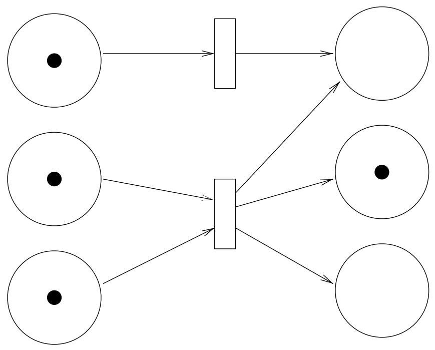
A Petri net is viewed as a sequence of discrete events whose order of occurrence is one of possibly many allowed by the basic structure. If at any time more than one transition is enabled, then any of the several enabled transitions may fire. The choice as to which transition fires is made in a nondeterministic manner, i.e., randomly or by forces that are not modeled. While this nondeterminism is advantageous from a modeling point of view, it introduces considerable complexity into the analysis of Petri nets.

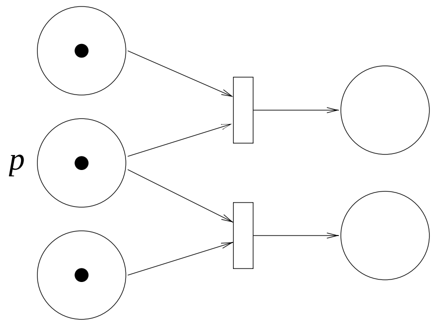
To reduce the complexity, one generally accepts that the firing of a transition (occurrence of an event) is considered instantaneous, i.e., to take zero time. Since time is a continuous variable, the probability of any two or more events happening simultaneously is zero, and two transitions cannot fire simultaneously. The events modeled are considered primitive events. For example, in Figure 8 the event 'Process a job' was modeled. But since this event is not a primitive one, it is decomposed into a *beginning* and an *ending*, which are instantaneous events, and the noninstantaneous occurrence.

The nondeterministic and nonsimultaneous firing of transitions in the modeling of concurrent systems takes two forms:

- Simultaneous events that may occur in any order (Figure 9 (a));
- Conflicting transitions where the firing of one will disable the other (Figure 9 (b)).

In Figure 9 (a), the two enabled events do not affect each other in any way and the possible sequences of events include some in which one event occurs first and some in which the other occurs first. In the other type of situation, shown in Figure 9 (b), the two enabled transitions are in conflict. Only one transition can fire, since that removes the token from p and disables the other transition.


(a) Simultaneous events that may occur in either order



(b) Conflicting transitions

Figure 9: Simultaneous and Conflicting Transitions

An important aspect of Petri nets is that they are uninterpreted models. The net of Figure 8 has been labeled with statements that indicate to human observer the intent of the model, but these labels do not, in any way, affect the execution of the net. We only deal with the abstract properties inherent in the structure of the net.

Another valuable feature of Petri nets is their ability to model a system hierarchically. An entire net may be replaced by a single place or transition for modeling at a more abstract level (abstraction) or places and transitions may be replaced by subnets to provide more detailed modeling (refinement).

0.3.2 Stochastic Petri Nets

Stochastic Petri nets (SPNs) are obtained by associating stochastic and timing information to Petri nets [5]. We do this by attaching *firing time* to each transition, representing the time that must elapse from the instant that the transition is enabled until the instant it actually fires in isolation, that is, assuming that it is not affected by the firing of other transitions. If two or more transitions are enabled at the same time, the firing of transitions is determined by the *race policy*; that is, the transition whose firing time elapses first is chosen to fire next. If the firing times can have general distributions, SPNs can be used to represent a wide range of well-known stochastic processes. However, choices about execution policy and memory policy, besides the firing time distributions, must be specified. The firing times are often restricted to have an exponential distribution to avoid policy choices. A more important fact in this case is that an SPN can be automatically transformed into a Continuous-Time Markov Chain (CTMC). In a graphical representation, transitions with exponentially distributed firing times are drawn as rectangular boxes.

When an SPN is applied to performance analysis of computer networks, places can be used to denote the number of packets or cells in the buffer or the number of active users, or flows in the system, while the arrival and departure of packets, cells, users or flows can be represented by firing or transitions.

In the following we show an SPN model for a simple $M/M/1$ queueing system to illustrate the transformation from an SPN into a CTMC.

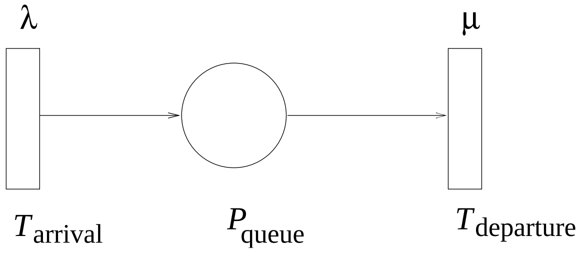


Figure 10: SPN Model of $M/M/1$ Queue

In Figure 10, the number tokens in P_{queue} represents the number of customers in the system, including the one receiving service, if any. Whenever there is a customer (one or more tokens) in the system (in place P_{queue}), a customer may complete service when the transition T_{service} fires and the firing time is exponentially distributed with rate μ. The reachability graph of this SPN is, in fact, a simple birth-death process with an infinite state space, as shown in Figure 2. Measures such as system size, response time, and throughput can be computed by solving this CTMC.

0.4 Conclusions

In this chapter we have introduced Queuing theory and Petri nets, which are important modeling tools for performance analysis of computer systems. Unlike other performance analysis techniques such as measurement and simulation, analytical modeling can capture the behavior of a computer system effectively and can provide quick answers to many question. We have explained Queuing theory and related fundamental laws to help the readers comprehend the basic principles. Then, according to Queuing system classification, we discussed the parameters and related equations. The chapter also gives a quick review of Petri nets.

The motivation of this chapter is to introduce different analytic techniques that can be used in performance analysis of computer systems. It is impossible to provide an in-depth treatment of the vast area in a chapter. Interested readers should refer to [4, 5] for detailed analyses.

References

[1] L. Kleinrock. *Queueing Systems Volume I: Theory.* Wiley-Interscience, 1975.

[2] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 3, 1962. Also, English translation, "Communication with Automata," New York: Griffiss Air Force Base, Tech. Rep. RADC-TR-65-377, Vol. 1, Suppl. 1, 1966.

[3] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[4] R. Jain. *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, Inc., 1992.

[5] K. S. Trivedi. *Probability and Statitics with Reliability, Queuing and Computer Science Applications.* Wiley-Interscience, 2002.

[6] J. P. Buzen. A Queueing Network Model of MVS. *ACM Computing Surveys*, 10(3):319–331, September 1978.

[7] P. J. Denning and J. P. Buzen. The Operational Analysis of Queueing Network Models. *ACM Computing Surveys*, 10(3):225–261, September 1978.

[8] J. L. Peterson. Petri Nets. *ACM Computing Survey*, 9(3):223–252, September 1977.