12. Networks of Queues

Once we have become familiar with the behavior of a single-processor queuing system, we can start reasoning about how a more complex system behaves and how it can be analyzed. The simplest multi-processor variations are those in which you simply use $N > 1$ more identical processors to serve requests. In this case, there are two main possibilities, which we cover in Section 12.1 and Section 12.2, respectively.

12.1 Multiple Single-server Queues

In the first scenario, illustrated in Figure 12.1, upon arriving to the system, customers choose one of N queues. Each of these queues is served by an individual server. This scenario is analogous to queues at the check-out counters of grocery stores. We call this system a multiple single-server queue and we denote it with the notation N*M/M/1.

The case of N queues in Figure 12.1 is quite easy to handle. Indeed, the different M/M/1 sub-systems that compose the N*M/M/1 system could be analyzed independently. Hence, we can apply the M/M/1 results we obtained for the single server queue with an arrival rate of λ/N. We can then obtain the various other metrics — such as W, q, T_w, and T_q. Care must be take, however, in understanding what the derived metrics apply to. For instance, once you solve a single M/M/1 sub-system and derive the average number of requests in the sub-system (q), the total average number of requests in the entire N*/M/M/1 system will be $q_{tot} = Nq$.

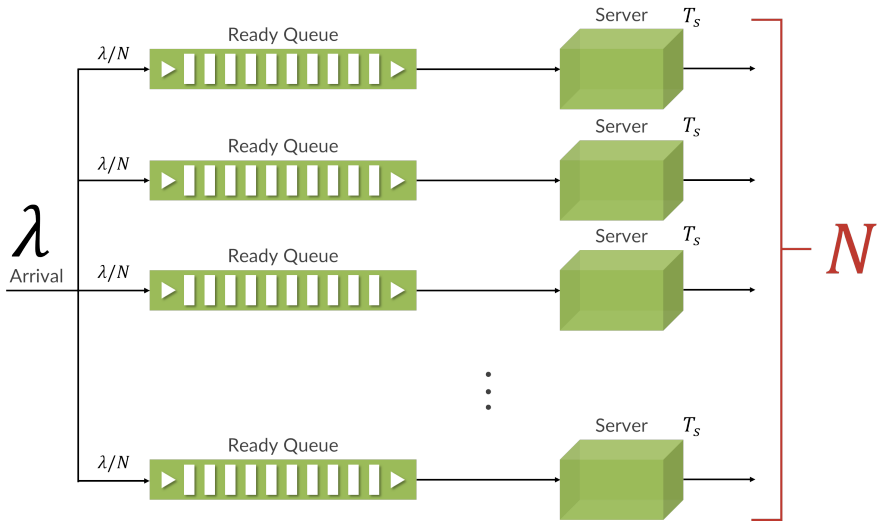

Figure 12.1: A system with N servers, each with its own independent queue, hence called an N*M/M/1 system.

12.2 Multi-server Queues

In the second scenario, illustrated in Figure 12.2, upon arriving to the system, customers join a single queue and the customer at the front of the queue is served by the "next available server" out of the N servers in the system. This scenario is analogous to queues at airport airline check-in. We call this system a multi-server queue and we denote it with M/M/N.

In this case, our analysis for the M/M/1 queue needs to be generalized. While we are not going to delve into the derivation of the stationary probabilities for S_j as we did for the M/M/1 queue, we will try to understand the reason why the performance of the multi-server system outperforms that of a multiple single-queue system.

The "difference" between the multi-server queue illustrated in Figure 12.2 and the multiple single- server queues illustrated in Figure 12.1 lies in the "sharing" of the queue.

In the multiple single-server queues system, it is quite possible (and for that matter likely) that a server would be idle when another server's queue is not empty — i.e. there would be customers waiting in a queue when a server is in effect available. This situation does not exist for the multi-server queue because the only way a server would be idle is for the main queue to be empty, implying no requests are waiting anywhere in the system.

We now proceed to the analysis for the M/M/N system, which we provide without derivation. Note however that one could follow the same approach we used for the M/M/1 system, with the caveat that the transition diagram would be different because we would have special cases anytime the number of customers in the system is less than N.

The probability that all servers are busy in a multi-server queuing system — i.e. the probability that there are N or more customers in the system — is given by the Erlang-C Function defined below:

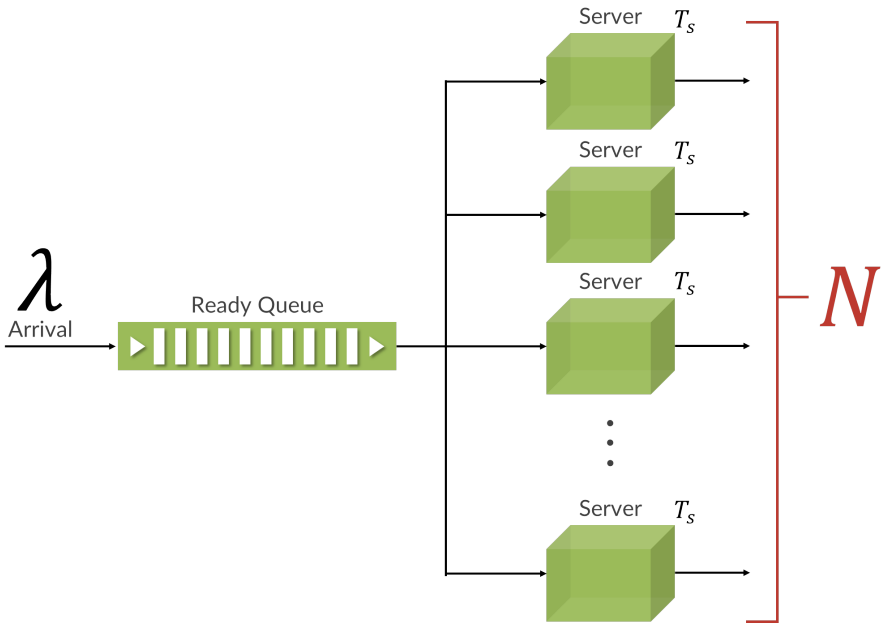

Figure 12.2: A system with N servers, sharing the same ready queue, hence called an M/M/N system.

$$C = \frac{1-K}{1-\rho K} \tag{12.1}$$

Where the term K is known as the *Poisson ratio* function and is defined as follows:

$$K = \frac{\sum_{i=0}^{N-1} \frac{(N\rho)^i}{i!}}{\sum_{i=0}^{N} \frac{(N\rho)^i}{i!}} = 1 - \frac{\frac{(N\rho)^N}{N!}}{\sum_{i=0}^{N} \frac{(N\rho)^i}{i!}} \tag{12.2}$$

Note that in the above equations, ρ represents the utilization of **each** of the servers in the system. Given the total traffic λ arriving at the system, the value of ρ is calculated as:

$$\rho = \frac{\lambda T_s}{N} = \frac{\lambda}{N\mu} \tag{12.3}$$

In the context of an M/M/N system, there is also another way to look at ρ. That is, to consider it as the overall utilization of the N servers. Since each server has a maximum service rate of μ, then it follows that the N servers could be thought of as being able to provide service at a maximum rate of $N\mu$.

Clearly, for any value of ρ, the value of K approaches 1 for large values of N, and the value of C (the probability that all servers are busy) approaches 0. Conversely, for any value of N, the value of K approaches 0 for values ρ that approach 1. Consequently, the value of C (the probability that all servers are busy) approaches 1.

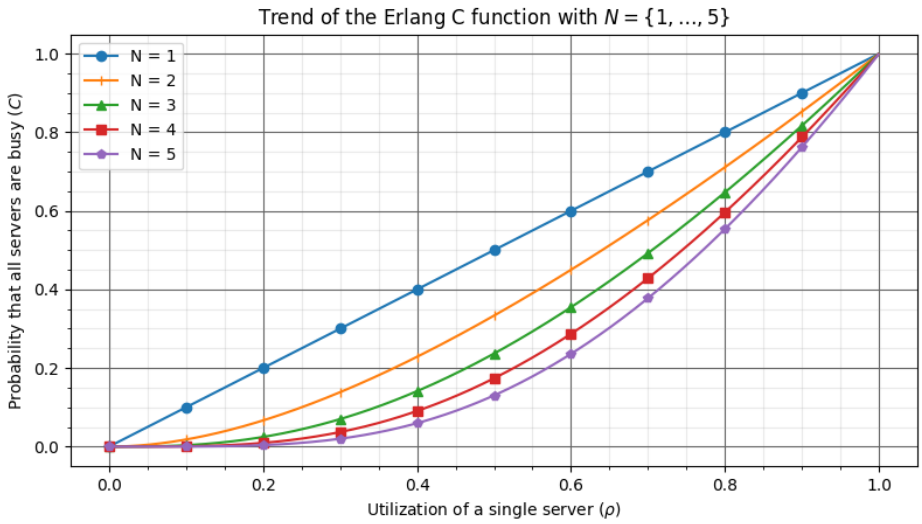

Figure 12.3: Trend of the Erlang-C function which captures the probability of N servers being busy with N between 1 and 5.

Notice that when $N = 1$, the value of C (which is the probability that service cannot start immediately; i.e., the request will experience a queuing delay) will be ρ, which is indeed what we would expect for an M/M/1 system. For larger values of N, we will find out that the Erlang function grows slowly when ρ is small and grows faster when ρ is large. For example, one can show that for $N = 2$ we have:

$$C(N = 2) = \frac{2\rho^2}{1+\rho} \tag{12.4}$$

Figure 12.3 illustrates this point by showing how the value of C changes as we increase the per-server utilization ρ. The figure shows the trend of C for values of N between 1 and 5. Note that for $\rho < 0.5$, we can see that C grows slowly, but that for $\rho > 0.5$, C grows faster.

12.2.1 Average Number of Requests in an M/M/N System

The average number of customers q in a M/M/N system is given by the following equation:

$$q = N\rho + \frac{C\rho}{1-\rho} \tag{12.5}$$

The above formula for q has two "components." The first (namely $N\rho$) is the expected number of customers being served, which follows directly from the fact that we have N servers and that each of them is "busy" with a probability ρ.

The second component accounts for the number of customers waiting in the (shared) queue. Notice that there are two pieces to that component. The first is $\rho/(1-\rho)$ which is identical to the

expression we derived for q in the M/M/1 queue. The second is the probability of "waiting," which is given by the Erlang-C function. Thus, another way to interpret that second component is that the queue will build up just like an M/M/1 system with similar (single) server utilization, but that the buildup will be scaled by an additional term which is the value of C.

12.2.2 Average Number of Requests Waiting for Service in an M/M/N System

Since the effective utilization of every server is ρ, it must hold that $q = w + N\rho$. Indeed, as we mentioned above, the number of customers in the system is the number of customers waiting in the single queue, plus the number of customers being served. Since there are N servers each with utilization ρ , the expected number of customers "receiving" service is $N\rho$. We can then write:

$$w = q - N\rho = \frac{C\rho}{1-\rho} \tag{12.6}$$

12.2.3 Average Time in an M/M/N System

To derive the average time that a request spends in an M/M/N system, i.e. T_q, we can simply use Little's Law. Recall that we computed $\rho = \lambda T_s/N$. We have:

$$T_q = \frac{q}{\lambda} = \frac{N\rho}{\lambda} + \frac{C\rho}{\lambda(1-\rho)} = T_s + \frac{CT_s}{N(1-\rho)} \tag{12.7}$$

One can also compute the deviation of the response time T_q in an M/M/N system. This is given by the following:

$$\sigma_{T_q} = \frac{T_s}{N(1-\rho)}\sqrt{C(2-C) + N^2(1-\rho)^2} \tag{12.8}$$

12.2.4 Average Time Waiting in a M/M/N System

Once again, to derive the average time that a request spends in an M/M/N system waiting for service, i.e. T_w, we can use Little's Law. We have:

$$T_w = \frac{w}{\lambda} = \frac{C\rho}{\lambda(1-\rho)} = \frac{CT_s}{N(1-\rho)} \tag{12.9}$$

12.3 Comparison Between an M/M/N system and an N*M/M/1 System

We start with a simple example to show how considerable the difference in performance could be for this seemingly simple difference in organization (servers sharing a queuing buffer versus servers with independent queues or buffers). We do this with two servers (i.e. $N = 2$).

Consider two multiprocessor systems. In the first, jobs are submitted to the ready queue of one of the processors at random, whereas in the second, jobs are submitted to a shared ready queue for all

processors. Assume that jobs arrive at the rate of $\lambda = 20$ jobs/sec and that each arriving job needs an average of 80 milliseconds of processor time. Assume that arrivals are Poisson and service times are exponential. Let us compare the average response time of the two systems. We proceed as follows.

1. Using the N*M/M/1 analysis, we can easily calculate T_q to be 400 milliseconds. Since the service time is 80 milliseconds, the waiting time T_w would be 320 milliseconds, on average.
2. Now using M/M/N analysis, we can easily calculate $C = 0.71$ and substituting in the equation for T_q , we get $T_q = 220$ milliseconds, which is about half of the response time for M/M/1!

We can compare the other metrics too. For the 2*M/M/1 system let us also compute q, and σ_{T_s}. These are as follow: $q^{2*M/M/1} = 8$, $\sigma_{T_s}^{2*M/M/1} = 0.4$. Similarly, for the M/M/2 system we have $q^{M/M/2} = 4.44$, $\sigma_{T_s}^{M/M/2} = 0.2$. Here, we note that not only the *average* number of requests in the M/M/2 system is about half compared to the 2*M/M/1 system, but also that the uncertainty on the response time that will be observed in the M/M/2 system is exactly half what we have in the 2*M/M/1 case.

12.3.1 When does an M/M/N System Differs more from an N*M/M/1 System?

As we hinted earlier, the difference in performance between these two systems is because it is possible for the latter (N*M/M/1 systems) to have an idle server while the queue of some other server in the system is not empty — an impossibility for an M/M/N system. This translates to worse performance (e.g., turnaround time). Intuitively, the likelihood of such a condition is higher when the utilization of the servers is low and/or when the number of servers is large. One can show this analytically by comparing the total number of requests in the two systems.

For N*M/M/1, the total number of requests in the N queues is the following:

$$q^{N*M/M/1} = \frac{N\rho}{1-\rho}, \tag{12.10}$$

whereas for M/M/N the total number of requests was given in Equation 12.5 and is provided below once again:

$$q^{M/M/N} = N\rho + \frac{C\rho}{1-\rho}. \tag{12.11}$$

Therefore, on average, the N*M/M/1 would have $q^{N*M/M/1} - q^{M/M/N}$ more customers, which is a quantity that can be computed as:

$$q^{N*M/M/1} - q^{M/M/N} = (N-C)\frac{\rho}{1-\rho} - N\rho = \frac{\rho}{1-\rho}(N\rho - C). \tag{12.12}$$

It can be observed that the quantity computed in Equation 12.12 is non-negative as long as $C \leq N\rho$, which is always true. It follows that given N processors, there is always improvement in going from an N*M/M/1 organization to an M/M/N arrangement.

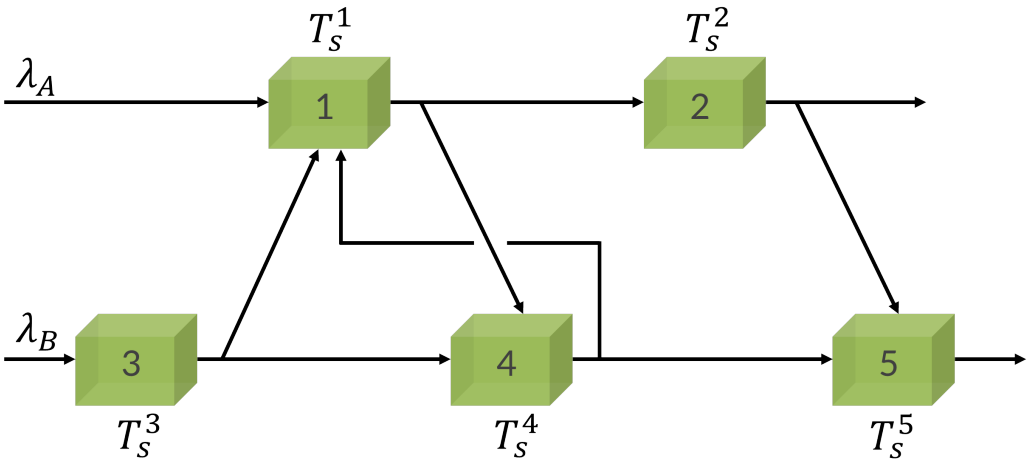

Figure 12.4: An illustration of a generic network of queues.

12.4 General Network of Queues

In a real system, customers seldom need service from a single server. Rather, they move from one server to the next. For example, a job may request the CPU and then the Disk and then may await user input, and so on[1]. Thus, it would be useful to analyze the behavior of a network of queues, where multiple servers exist, each with its own queue. Customers join the system by adding themselves to one queue and moving from one server to next, until at some point they leave the system. Figure 12.4 provides a generic example of a network of queues with two sources of traffic (λ_A and λ_B) and 5 resources and corresponding characteristic service times $T_1, \ldots, T_5$.

Before diving into how to decompose and analyze an arbitrarily complex network of queues, let us introduce important properties for the traffic that flows through this type of systems.

12.4.1 How does a Network of Queues form?

The network of servers (or queues) is formed through 3 basic processes. (1) The pipelining of servers where the output of a server becomes the input to the next server; (2) The splitting of a stream into a number of streams; and (3) the joining of a number of streams into a single stream.

1. **Pipelining**: The stream of customers out of one queue could join a following queue. This is also called "queues in tandem" or "daisy-chain" of servers. At steady state, the rate of arrivals to a queue must equal the rate of "departure" from that queue — which obviously is the same as the rate of arrival to the next queue, and so on. Figure 12.5 illustrates precisely the concept of steady-state pipelining.
2. **Splitting**: The stream of customers being released from a server may be partitioned into several streams with some probability of going to either stream. In particular, if the stream is divided into two, as shown in Figure 12.6, then the probability of a customer (or request)

[1] Recall our discussion of how to represent the state of a system using a set of interconnected queues for processes that are "ready", "blocked", or "running".

Figure 12.5: An illustration of a pipeline of queues.

joining the first stream (A) is some p and the probability of a customer joining the second stream (B) is $(1-p)$. Therefore, if the rate of requests before the split was λ, then the effective traffic through path A will be $p\lambda$, with $(1-p)\lambda$ being directed through path B instead.

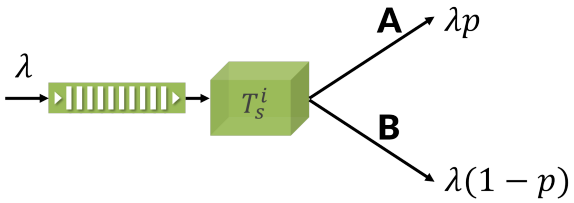

Figure 12.6: An illustration of traffic splitting between two paths.

3. **Joining:** Two (or more) streams being released from different servers may be joined into the queue of a single server. In particular, if the rate at which the two streams are released is λ_A and λ_B, then the resulting stream will have a rate of $\lambda_A + \lambda_B$. Furthermore, if both streams are Poisson, then the combined stream also follows a Poisson distribution. This is illustrated in Figure 12.7.

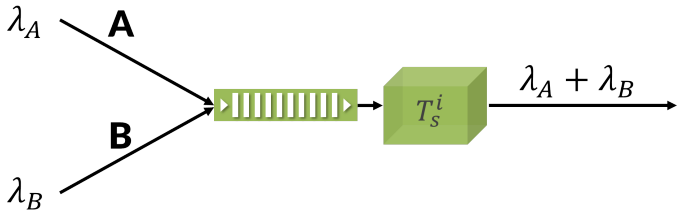

Figure 12.7: An illustration of traffic joining into as single queue from two separate paths.

The analysis of networks of queues is complex — and beyond the scope of the basic treatment of queuing theory for this class. However, under some simplifying assumptions, we can analyze networks of queues by analyzing the individual servers in the network (whether M/M/1, N*M/M/1 or M/M/N) independently and then combining the results. The conditions under which this independent analysis is possible are:

1. All arrivals (from outside the network of queues) are Poisson;
2. All service times are exponential;
3. All customers are either being served (by a server) or are waiting in some queue.

Under the above conditions, the extremely powerful Jackson's Theorem applies. We discuss the Jackson's Theorem in the next section.

12.4.2 Jackson's Theorem

In a nutshell, the Jackson's Theorem states that under the conditions mentioned above, each server/queue can be analyzed independently and the results can be combined to compute overall "delays" through the network of servers.

The reason the Jackson's Theorem works is simple. At steady state, if the input stream to a M/M/1 (or M/M/N) system is Poisson with parameter λ, then the output stream of that system is also Poisson with parameter λ. Also, if a Poisson stream with parameter λ is being split into two streams according to probabilities p and $(1-p)$, then the resulting streams are also Poisson with parameters $p\lambda$ and $(1-p)\lambda$, respectively.

Let us consider a simple example! Here is the breakdown of a system. In this case, a restaurant:

- Customers arrive to a restaurant according to a Poisson arrival process with mean rate of 10 customers per hour.
- First, customers wait to be seated by a host. Next, they wait for the waiter to get food. Next, they eat. Next, they wait for the cashier to process their payment. Finally, they leave.
- Assume that the time it takes the host to seat a customer is 3 mins, the time for the waiter to respond to an order is 4 mins, the time for the customers to eat their food is 5 mins, and the time for the cashier to process payment is 2 mins.
- Calculate the mean response time for the system, the utilization of the various entities involved, the total number of customers waiting for various stages of service, and the like.

Pipelining Example

Clearly, in this example, each customer goes through a sequence of steps, so we are dealing with a pipeline of queues. Hence, we can compute the utilization of the various "resources" as follows:

- Host: $3 \cdot 10/60 = 0.5$
- Waiter: $4 \cdot 10/60 = 0.67$
- Customer: $5 \cdot 10/60 = 0.83$
- Cashier: $2 \cdot 10/60 = 0.16$

Similarly, we can figure out all the other metrics. For instance:

- Total average number of customers at the restaurant $q_{tot} = 0.5/(1-0.5) + 0.67/(1-0.67) + 0.83/(1-0.83) + 0.16/(1-0.16) = 8.1$ customers;
- Average time spent by each customer in the restaurant (response time) $T_{q_{tot}} = 8.1/(10/60) = 48.6$ minutes;
- Average total time spent by each customer waiting for service $T_{w_{tot}} = 34.6$ minutes.

Multi-Resource Computing System Example #1

Consider the example depicted in Figure 12.8 in which requests arrive to a system at the rate of 10 per second, use one resource (say the CPU) and then with a probability p exit the system, otherwise (with probability $1-p$) request another resource (say the disk) and then they go back to the first resource. Given the service times of 0.03 and 0.055 seconds for the CPU and disk, respectively,

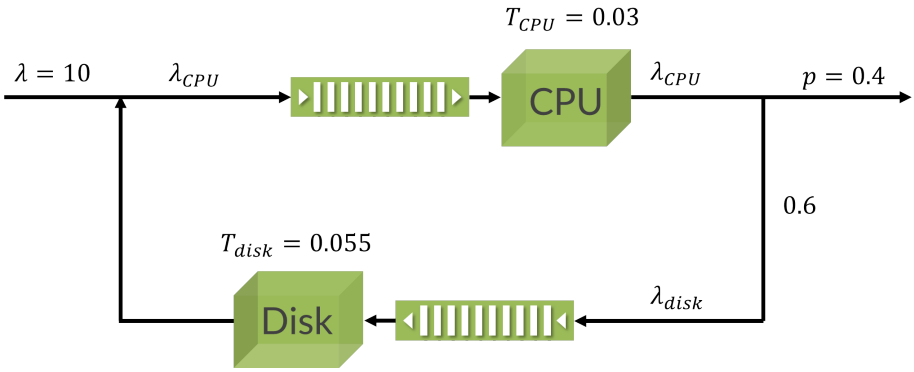

Figure 12.8: Example of a queuing system to model processes using two resources (CPU and Disk) alternately.

we could compute the overall turnaround time of the above system (from entry to exit).

In order to be able to apply the Jackson's Theorem, we need to figure out the steady-state rate of arrival to each queue in the above network. To do so, we need to solve for an unknown λ_{CPU} (e.g., the rate of arrival/departure through the CPU queue). We can write the following equation — by noting that the rate into the CPU queue must be equal to the rate out of the CPU queue.

$$\lambda_{CPU} = 10 + 0.6\lambda_{CPU} \tag{12.13}$$

Solving the above equation, we get $\lambda_{CPU} = 25$. Thus the rate of arrivals for the CPU queue is 25 and the rate of arrival for the disk queue is $\lambda_{disk} = 0.6 \cdot 25 = 15$. Now we can compute the utilization of the CPU $\rho_{CPU} = 25 \cdot 0.03 = 0.75$ and the utilization of the disk $\rho_{disk} = 15 \cdot 0.055 = 0.825$. Using M/M/1 analysis we can compute the value of $q_{CPU} = 3$ for the CPU and the value of $q_{disk} = 4.7$ for the disk. This means that on average, there would be $q_{tot} = 7.7$ requests in the system. Using Little's law, we can compute the average turnaround time to be $T_{q_{tot}} = 7.7/10 = 0.77$ seconds.

Multi-Resource Computing System Example #2

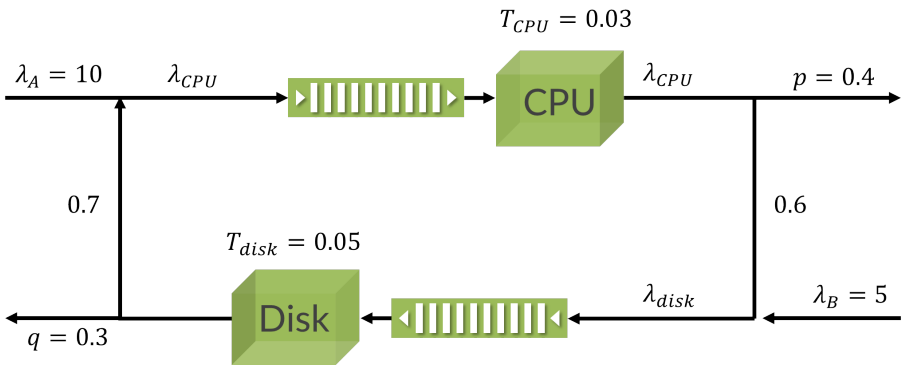

Figure 12.9: Example of a queuing system to model processes using two resources (CPU and Disk) and with two traffic entrance points.

Again, to solve the above system hinges on our ability to figure out the rate of arrivals/departures from each queue. First, note that this system has points where the traffic enters the system. These are (1) at the CPU queue with rate $\lambda_A = 10$ requests per second; and (2) at the Disk queue with rate $\lambda_B = 5$ requests per second.

We have to be able to write one (or more) equations to calculate the rate of traffic through each of the queues. Let us start with the rate of arrivals to the CPU queue λ_{CPU}, which can be computed as:

$$\lambda_{CPU} = 10 + 0.7(0.6\lambda_{CPU} + 5) \tag{12.14}$$

The above equation can be easily solved for λ_{CPU} and the solution of the above system will follow as before.