the Availability Digest

Configuring to Meet a Performance SLA – Part 2: Multiserver Response Times

January 2009

Many applications carry with them a performance Service Level Agreement (SLA) that specifies the response times that they must achieve. After all, if an application's response time is so slow that the application is not useful, the application is, in effect, down.

The performance requirement is often expressed as a probability that the system's transaction response time will be less than a given interval. For instance, "98% of all transactions must complete within 500 milliseconds."

In Part 1 of this series, we derived the basic response-time expression for a single-server system. Here in Part 2, we extend that result to a multiserver system in which multiple servers work off a common work queue. In Parts 3 and 4, we will further extend our results to answer the SLA question posed above for servers with exponential and arbitrary service time distributions. We conclude with an example of a complex system in Part 5.

Reviewing the Basic Single-Server Queuing Relationship

In Part 1, we showed that the average response time for a single-server system was given by the Khintchine-Pollaczek equation:

$$T_r = \frac{T_s}{1-L}[1-(1-k)L] \qquad (1)$$

where

- T_r is the average transaction response time.
- T_s is the average service time of the server.
- L is the load on the server.
- k is the distribution coefficient of the server's service time.

The distribution coefficient k depends upon the probability distribution of the server's service time. For instance, k = 1 applies to server distribution times that are random (that is, exponentially distributed), which is the common assumption.[1] In this case, Equation (1) reduces to

$$T_r = \frac{T_s}{(1-L)}, \qquad (2)$$

which for many of you is the well-recognized transaction response time expression relating transaction response time to server load.

[1] More specifically, k is 1/2 the ratio of the service time's second moment to the square of its mean.

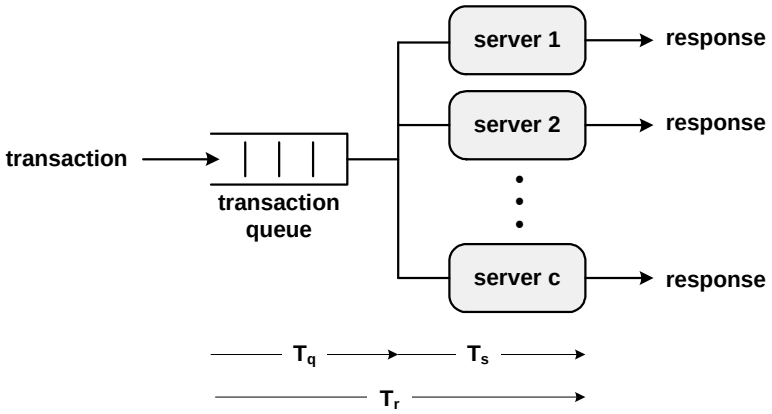
Note that the transaction response time, T_r, given by Equation (1) is the time that the transaction must wait in the server plus its service time, T_s. The time that the transaction must wait in the server is called its *queuing time, T_q*, and from Equation (1) is

$$T_q = T_r - T_s = \frac{T_s}{1-L}[1-(1-k)L] - T_s = \frac{kL}{1-L}T_s \qquad (3)$$

Response Time for a Multiserver System

The Model

We now consider the case in which multiple servers process transactions from a common transaction queue, as shown in Figure 1. A transaction enters the common transaction queue and waits until it arrives at the head of the queue. It is then serviced by the next server that becomes available.



Multiserver System
Figure 1

You have probably experienced the advantage of this arrangement yourself. If you are waiting in line for an airline agent, would you rather have individual lines in front of each agent and try to determine which will be faster; or would you rather wait in a common line and be serviced by the next available agent when you arrive at the head of the line? I vote for the latter.

Multiserver systems abound in today's technology. For instance, web farms are often configured this way. Multiple web servers provide pages upon request to remote browsers. Each browser request is satisfied by the next available web server. Since these requests are context-free, each may be serviced by any server in the server pool.

Likewise, transaction monitors such as Tuxedo or HP NonStop's Pathway route transaction requests to the next available process (the server in this case) in the process pool.

We define the following parameters:

T_r	is the average transaction response time.
c	is the number of servers in the system.
L	is the load on each server.
T_q	is the average time that the transaction waits in the transaction queue.
T_s	is the average service time of the server.

Note that L is the load on *each* server. The load carried by the system is cL and is the transaction arrival rate multiplied by the servers' service time. If the transaction arrival rate is r, then

$$L = \frac{rT_s}{c}$$

Thus, if each server is running 50% busy (L = 0.5), and if there are five servers in the system (c = 5), the system is carrying a transaction load of 2.5. Put another way, if the average service time is 10 milliseconds, and if the system is handling 250 transactions per second, the system load is 2.5. If there are five servers, each server is handling 50 transactions per second; and its load is 0.5.

The Analysis

The solution for the multiserver system is not pretty[2] and is presented here without proof. The formal derivation has only been made for exponential service times (k =1), but simulation studies show that the queuing time, T_q, varies in about the same way as single server systems with respect to the Khintchine-Pollaczek distribution coefficient, k.[3] Noting from Equation (3) that T_q varies directly with k, we apply that observation here.

Given that, the average time that a transaction waits in the transaction queue, T_q, is

$$T_q = \frac{k(cL)^c}{c(c!)(1-L)^2} p_0 T_s \qquad (4)$$

and the average transaction response time, T_r, is

$$T_r = T_q + T_s \qquad (5)$$

The term p_0 is a new term. It is the probability that there are no transactions waiting for service – that is, the server complex is idle. p_0 is given by

$$\frac{1}{p_0} = \sum_{n=0}^{c-1} \frac{(cL)^n}{n!} + \frac{(cL)^c}{c!(1-L)} \qquad (6)$$

As complex as these expressions are, we can glean some insight from them. Let us take the case of a single server (c = 1). Then Equation (6) evaluates to p_0 = (1 - L), which is what we would expect (L is the probability that the server is busy, so (1 – L) is the probability that it is idle). Substituting this into Equation (4) for c = 1, we have

$$T_q = \frac{kL}{(1-L)^2}(1-L)T_s = \frac{kL}{(1-L)}T_s \qquad \text{for } c = 1$$

which is Equation (3).

For c = 2 and k = 1,

$$p_0 = \frac{1-L}{1+L}$$

and

$$T_q = \frac{L^2}{1-L^2}T_s$$

[2] T. L. Saaty, *Elements of Queuing Theory*, pg. 116, McGraw-Hill;1961.
W, H. Highleyman, *Performance Analysis of Transaction-Processing Systems*, pg. 126, Prentice-Hall; 1989.
[3] J. Martin, *Systems Analysis for Data Transmission*, pg. 461, Prentice-Hall; 1972.

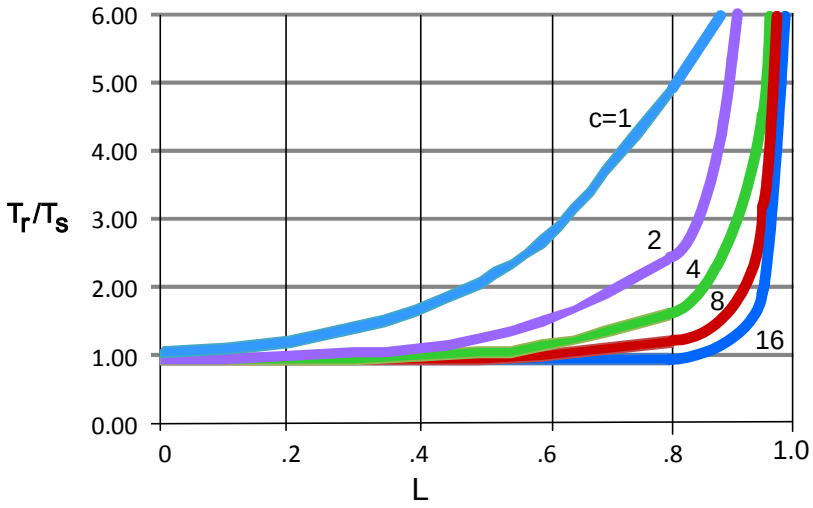
$$T_r = T_q + T_s = \frac{T_s}{1 - L^2} \qquad \text{for } c = 2$$

Compared to Equation (2) for a single server, two servers can process twice the load with a faster response time. For instance, if T_s were 10 msec., a single server could handle 50 transactions per second and be 50% loaded. Its response time would be 20 msec. However, a two-server system could handle 100 transactions per second with each server being loaded 50%; and its response time would be 13.3 msec.

This is the power of multiserver configurations.

The Results

Figure 2 shows transaction response time normalized to service time (T_r/T_s) as a function of load for multiserver systems using 1, 2, 4, 8, and 16 servers. The improvement in response time is clearly shown. The response time curve flattens out as the number of servers is increased and



Multiserver Transaction Time
Figure 2

breaks much more rapidly as the servers become fully loaded at 100%.

This leads to a caveat when using multiserver systems. They can be run at much higher loads, but small increases in transaction rates at high load levels will break the system much more quickly. An often used rule-of-thumb is that the load on a single server should not exceed 60%, and the load on a multiserver system should not exceed 80%.

Another caveat is that the transaction response time shown in Figure 2 applies to the number of servers involved in the application, not the number of servers in the system. For instance, in a sixteen-processor NonStop system, if a server class for an application is spread among four processors, the performance advantage for that application is that of a four-processor system, not a sixteen-processor system.

Summary

The use of multiserver systems can greatly improve transaction response times. In addition, availability is significantly increased since should a server fail, the other servers in the system will simply continue to process the common transaction queue. However, care must be taken not to run the multiserver system at too high a load. Though the transaction response time may be

acceptable, response time may rapidly increase with a small increase in load, perhaps bringing the system to its knees.

An Excel spread sheet that is useful in making these response time calculations can be found at http://www.availabilitydigest.com/public_articles/0401/performance_sla_2.xls.

So far, the analyses that we have presented in Parts 1 and 2 have dealt with average response times. But in order to answer the SLA question of what size server do we need to guarantee that a certain percentage of transactions will be completed within a specified time, we need to know the distribution of response times. We attack this problem in our upcoming Part 3 for servers with exponential service times and in Part 4 for servers with arbitrary service time distributions.