Solving for X:
Reading behind the meters to find throughput limits and how much time is being wasted waiting

Bob Wescott

Performance Mentors

Article Summary

With a bit of simple math you can figure out how much work a lightly loaded resource can theoretically handle, how many jobs are in the system, and how much time those jobs are waiting for their turn to run. These numbers are very useful in capacity planning work and when double-checking numbers that don't seem quite right. To do this magic trick, I will first define a few technical terms and then we'll do a bit of simple math.

This article is excerpted (with modifications) from **The Every Computer Performance Book**.

Defining a Few Terms First

Response time is the total amount of time you waited for something you asked for. Here is an example:

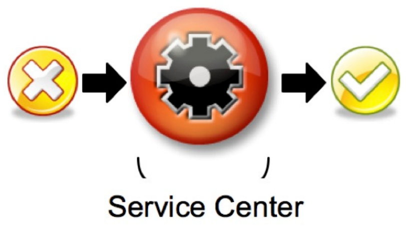
Time	Action
00:00	You say: "Can I have a cookie?"
00:02	Mom takes a cookie from the jar.
00:04	Mom hands you the cookie.
00:05	You insert cookie and begin chewing.

As far as your taste buds were concerned, the response time for your cookie request was five-seconds. If Mom had been busy doing other things, then you would have had to wait, and that would lengthen the response time.

Utilization is the technical term for "busy" and is typically expressed as a decimal fraction with a range between zero and one. A 45% busy resource has a utilization of 0.45. As utilization goes up, the response time also tends to go up.

Service Center and Service Time

A **service center** is where the work gets done. CPUs, processes, and disks are examples of service centers. To accomplish a given task, it is generally assumed that it takes a service center a fixed amount of time – the **service time**. In reality this assumption is usually false, but still very useful.



The crafty people who designed your hardware and software typically put a few optimizations in the design. If you could meter every job going through a service center, you'd find that the amount of time and effort to accomplish each "identical" job is somewhat variable. Having said this, it is still a useful abstraction to think about each identical task taking an identical amount of time to be serviced at the service center. Just as you don't require quantum mechanics to predict the flight path of a baseball, you can mostly ignore the individual variations and focus on the big picture.

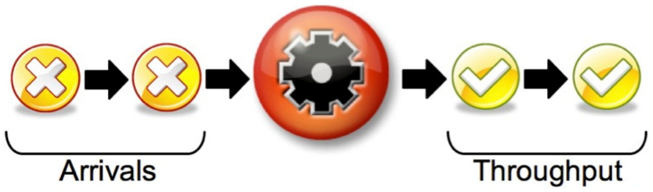
Averaged over time, a service center can have a utilization from zero to one or, if you prefer, 0% to 100% busy. You are always interested in the utilization averaged over a short period of time, i.e., seconds or minutes. You are never interested in the instantaneous utilization (it is always 0 or 1) and are rarely interested in the utilization averaged over long periods (hours, days, etc.)

Only a fool would plan for a service center to be 100% busy, as there is no margin for error and the incoming work usually does not arrive at a convenient pace.

You can set the boundaries for a service center anywhere you like. A service center can be a simple process, or the entire computer, or an entire array of computers. For that matter a service center can be an oven. The service time for baking bread = 30 minutes at 350℉. A service center is where work gets done, and you get to define the boundaries.

Arrivals and Throughput

Work arrives at a service center and, when processing is complete, it exits. The work is composed of discrete things to do that might be called transactions, jobs, packets, tasks, or IOs, depending on the context.



The rate at which tasks arrive at the service center is the arrival rate. The rate at which tasks exit a service center is called the throughput. In performance work, most of the time these values are measured over a period of a second or a minute, and occasionally over a longer period of up to an hour.

To stay out of trouble, be sure that you don't confuse these terms and keep your units of time straight. Arrivals are not the same as throughput, as anyone knows whose ever been stuck in a long airport security line. If you accidentally mix "per second" and "per minute" values in some calculation, then badness will ensue. Try not to do that.

Wait Time

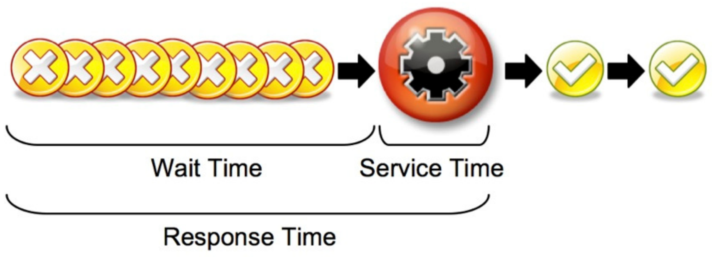
Unless you are reading this in a post-apocalyptic world where you are the only survivor, there will be times when tasks arrive at a faster rate than the service center can process them. Any task that arrives while the service center is busy has to wait before it can be serviced. The busier the service center is, the higher the likelihood that new jobs will have to wait.

The upper limit on wait time is controlled by two things: the maximum number of simultaneous arrivals and the service time. If ten tasks arrive simultaneously at an idle service center where the service time is 10 milliseconds, then the first task gets in with zero wait time, the last job will wait for 90 milliseconds. The average wait time for all these tasks is:

$$45\text{ms} = (0+10+20+30+40+50+60+70+80+90) / 10$$

The overall response time is what most people care about. It is the average amount of time it takes for a job (a.k.a. request, transaction, etc.) to make it through the service center and (typically) back to the user. For any given service center:

ResponseTime = WaitTime + ServiceTime



Finding What You Don't Have

Now that we have a few terms defined, let's look at some tools and rules that will help you find useful information, spot errors, and give you key insights about how systems behave when they get busy.

Finding Wait Time and Service Time

As you'll see shortly, the wait and the service time are wildly useful numbers to know, but the response time is the only number that most meters, if they provide that data at all, are likely to give you. So how do you dig out the wait and the service time if there are no meters for them?

The service time can be determined by metering the response time under a very light load when there are plenty of resources available. Specifically, when:

- Transactions are coming in slowly with no overlap
- There have been a few minutes of warm-up transactions
- The machines are almost idle

Under these conditions, the response time will equal the service time, as the wait time is approximately zero.

ServiceTime + WaitTime = ResponseTime
ServiceTime + 0 = ResponseTime
ServiceTime = ResponseTime

The wait time can be calculated under any load by simply subtracting the average service time from the average response time. This is a useful calculation to do, as it shows you how much better things could be if all the wait time was cleared up. Performance work, at some level, is all about time and money. If you know the wait time, you can show how much time a customer might save if your company spent the money to fix the problem(s) you've discovered.

Finding the Maximum Throughput

If you know the service time, you can find the maximum throughput because:

MaxThroughput ≤ 1 / AverageServiceTime

A service center with an average service time of 0.05 seconds has a maximum throughput of: 1 / 0.05 = 20 per second.

CAUTION: With this calculation you have to be a bit careful when you have a broadly defined service center. For example, a Google search with the word "cat" returned after 0.25 seconds. This value was reasonably constant when tested very early in the morning on a weekend so we can assume that the utilization of the Google servers is fairly low. Using the above formula, we can scientifically show that the maximum throughput for Google is four searches per second. Clearly that is not right. So, is this rule wrong? No, it was just used in the wrong place. Google has a massively parallel architecture, and so we are not looking at just one service center. Here we got a reasonable Average Service Time, did the calculation, and came up with a Max Throughput number that made no sense. With all these tools the most important things you bring to the party are common sense and a skeptical eye.

Finding The Mean Number of Jobs In The System

Little's Law shows the fixed relationship between three things: the mean number of jobs in the system that are either waiting or being serviced, the mean response time, and the mean throughput. If you know two of these three things you can figure out the third. This can be useful when doing a quick validation of metered data and for checking the soundness of proposals and plans.

Suppose you are capacity planning for an application that (as either a licensing or a configuration restriction) has an upper limit on the number of concurrent requests being processed, and you need to know how close you are to that limit according to Little's Law:

MeanNumberInSystem = MeanResponseTime * MeanThroughput

Imagine the application has a mean response time of 0.2 seconds and a throughput measured at 50/second. The mean (average) number in the system is: 10 = 0.2 * 50

Finding The Average Response Time

Here we can also use Little's Law when applications thoughtlessly do not have any easy way to meter response time. If all you can find is the mean number in the system and the throughput, you can rearrange the equation to find the response time info you need like so:

MeanResponseTime = MeanNumberInSystem / MeanThroughput

Now we'll look at the same application with different meters. Here we have a meter (like queue depth) that tells us the number of jobs waiting to be serviced. There are nine jobs in the queue and one being processed, so there is an average of ten jobs in the system. Another meter shows a mean throughput of 50 per second. You can calculate the mean response time as: 0.2 seconds = 10 / 50 per second

When exploring Little's law and learning to trust it, be aware of the common newbie mistakes of using arrivals when throughput is called for, and not keeping the units of your measurements the same.

Also Little's law expects the system to be in a steady state during the measurements, with no surges or drop-offs in demand, and that all the jobs are uniform in size. Given that in real life applications these restrictions are never fully met, Little's Law is mostly useful for reality checking (do the meters make sense) and ballpark estimations of missing performance numbers.

Finding How Busy a Resource Is

The Utilization Law, using the same definitions, limits, and restrictions of Little's Law, allows us to estimate the utilization of a service center if you only know the service time and the mean throughput like so:

MeanUtilization = ServiceTime * MeanThroughput

This is really helpful when you are studying a complex service center, such as a key process in the transaction path that burns a little CPU, does some disk I/O, and communicates with other processes. Typically processes do not have a utilization meter that encompasses all these activities. You can usually know the CPU utilization of a process, but a process can be bottlenecked long before it hits 100% CPU utilization.

You could, for example, monitor a process as it experienced a normal workload with a network sniffer. By picking through a couple of minutes worth of data, you could gauge the service time and mean throughput of the process, and that would allow you to calculate its overall utilization. With a utilization and a mean throughput number, you could then gauge how much more headroom this process has like so:

HeadRoom = 1 – (ServiceTime * MeanThroughput)

The key thing to remember here is that the service time of the process is not likely to improve as the load increases. Therefore, the max throughput number you generate is likely to be the most optimistic case.

Finding The Service Time a Different Way

The complex math of operations research that defines the Service Demand Law boils down to a simple idea. You can divide any utilization number the system gives you by the throughput rate to find the cost per transaction.

ServiceTime = MeanUtilization / Mean Throughput

For example: Imagine a system is processing 20 transactions per second with an overall CPU utilization of 0.8 or 80% busy. At one transaction per second the CPU utilization would be: .8/20= 0.04 or about 4% busy. If that system can provide 4000 milliseconds of CPU service per second (thanks to its four CPUs) then four percent of 4000 milliseconds = 160 milliseconds of CPU per transaction.

The Service Demand Law can be useful when you need to do a capacity plan where part of the overall transaction load is being moved to another system. It is easier to scale a number like 160 milliseconds CPU than a more complex number like 4% CPU busy.

In Closing:

You can use these basic performance laws to find a lot of useful, hard to meter information that can be quite helpful in performance work.