Throughput Capacity Planning and Application Saturation

Alfred J. Barchi

ajb@ajbinc.net
http://www.ajbinc.net/

Introduction

Applications have a tendency to be used more heavily by users over time, as the users get used to the application, and as the user base expands. Additionally, application performance tends to degrade somewhat over time as new features are added to the application.

It is therefore important to be able to determine the throughput capacity of an application through performance testing and by evaluating operational data, in order to establish growth trends and to ensure that adequate infrastructure is available in time to meet the growth, so that service level agreements continue to be met and performance of the application doesn't degrade.

This paper discusses how to assess the maximum throughput capacity of an application and how to identify the bottleneck devices in an application. As data on application utilization is collected over time, it can then be used to perform trending analysis to predict the future growth of the application.

The examples discussed deal with the Microsoft Windows operating systems, Microsoft IIS, and the Mercury Interactive LoadRunner tool. However, the approach itself is applicable to any application hosted on any operating system.

What Is Saturation?

Basically, as load increases, throughput increases, until maximum resource utilization on the bottleneck device is reached. At this point, maximum possible throughput is reached. Saturation occurs once maximum throughput is reached, and at this point, queuing occurs. Queuing typically manifests itself by degradation in response times.

This phenomenon is described by Little's Law:

$$Q = X * R$$

Where Q is equal to the number of transactions/operations/bytes/bits in the system, X is the throughput of the system, and R is the response time. As Q increases, X increases (R also increases slightly, because there is always some level of contention at the component level). At some point, X reaches X_{max} – the maximum throughput of the system. At this point, as Q continues to increase, the response time R increases in proportion. This is illustrated in the following graphs (taken from "The Art of Computer Systems Performance Analysis", by Raj Jain, 1991):

Copyright © 2007, Alfred J. Barchi, Inc. All rights reserved.

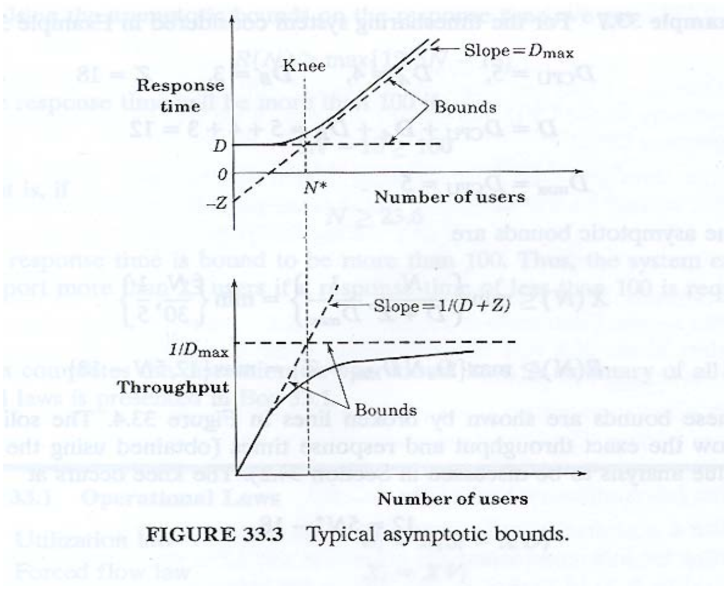
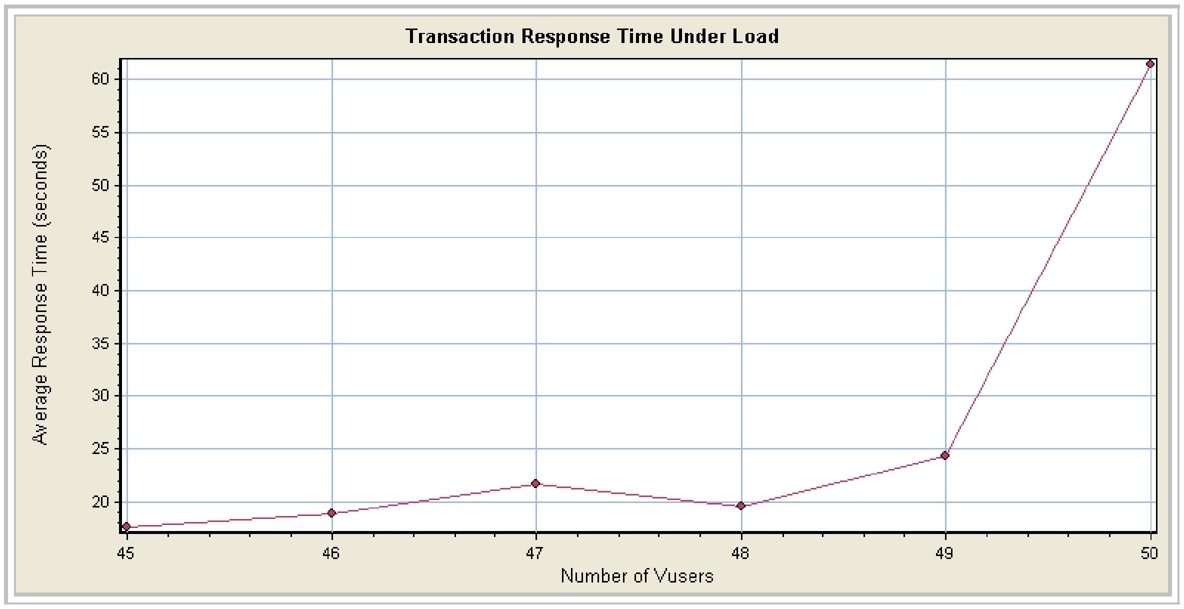


FIGURE 33.3 Typical asymptotic bounds.

In this diagram, N^* is the number of users (an expression of load) at which the application begins to go into saturation. It is also sometimes referred to as the point of optimal throughput. D is the service demand, which will be discussed below, D_{max} is the maximum service demand of all of the devices in the application, again discussed below, and Z is user 'think time' (not important to this discussion).

In LoadRunner, we typically see saturation as a degradation in response times with increased load, defined as the number of simultaneous Vusers. For example:

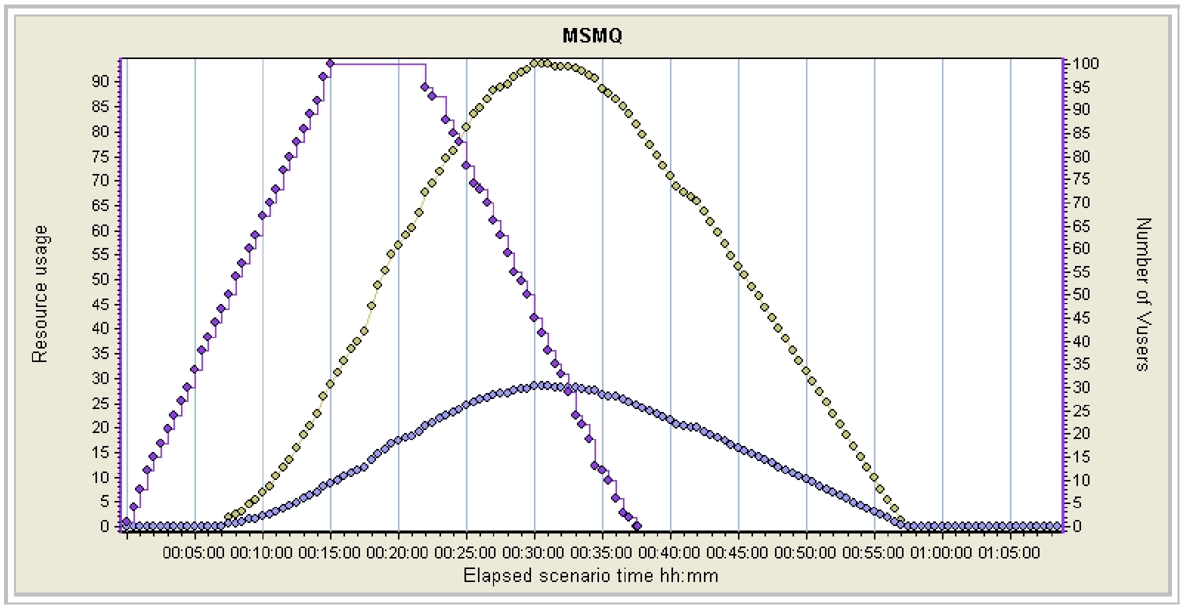


This chart was taken from an actual load test. It is important to always generate this chart in LoadRunner in order to diagnose whether or not the application is saturating. Here, the application goes into saturation at a load equivalent to 49 simultaneous virtual users,

Copyright © 2007, Alfred J. Barchi, Inc. All rights reserved.

where the operations performed be each virtual user are designed to mimic as closely as possible the mix of operations performed by actual users.

Queuing doesn't always manifest itself as a degradation in response times, however. In the same application from which the above chart was taken, a different configuration was tested using a message queuing component (Microsoft MSMQ) to de-couple the user presentation layer from the back-end processing by queuing user transaction requests and responding to the user immediately. The back-end database would then pull transaction requests off of the queue and process them as quickly as it could. The chart below illustrates this:



The purple line shows the number of Vusers. The yellow line shows the number of transactions in the MSMQ queue awaiting execution. The gray line shows another processing component of MSMQ. It is interesting to note that queuing occurs at the same level of load in both configurations, i.e., 49 simultaneous Vusers. At this point, the application is saturated.

Identifying the Bottleneck Device and Predicting X_{max}

Another important concept is the Utilization Law:

$$U_i = X * D_i$$

Where U_i is the percentage of utilization of a device in the application, X is the application throughput, and D_i is the service demand of the application device. The maximum throughput of an application X_{max} is limited by the maximum service demand of all of the devices in the application.

Copyright © 2007, Alfred J. Barchi, Inc. All rights reserved.

Typical application devices include: CPU, memory, disk and network interfaces. We get application throughput from LoadRunner. We get device utilization from Perfmon counters. For example, in a load test where LoadRunner reports 200 kb/sec average throughput:

CPU_{avg} = 80%
$Memory_{avg}$ = 30%
$Disk_{avg}$ = 8%
Network I/O_{avg} = 40%

We use the average values rather than the maximum for two reasons. First, if we used maximum values, our results would be based on outliers. By using average values, our calculations are based on a large number of samples over time. Second, LoadRunner gives us actual average throughput, but only give ‘graph’ maximum throughput. The averages should be filtered from the point where the load is first applied until the point where the load is removed.

In this example, the service demands are:

D_{cpu} =	0.8 / 200 kb/sec	= 0.004 sec/kb
D_{memory} =	0.3 / 200 kb/sec	= 0.0015 sec/kb
D_{disk} =	0.08 / 200 kb/sec	= 0.0004 sec/kb
$D_{network\ I/O}$ =	0.4 / 200 kb/sec	= 0.002 sec/kb

In this case, D_{max} corresponds to the CPU. So, the CPU is the bottleneck device. We can use this to predict the maximum throughput of the application by setting the CPU utilization to 100% and dividing by D_{cpu}. In other words, for this example:

$Xmax = 1 / D_{cpu}$ = 250 kb/sec

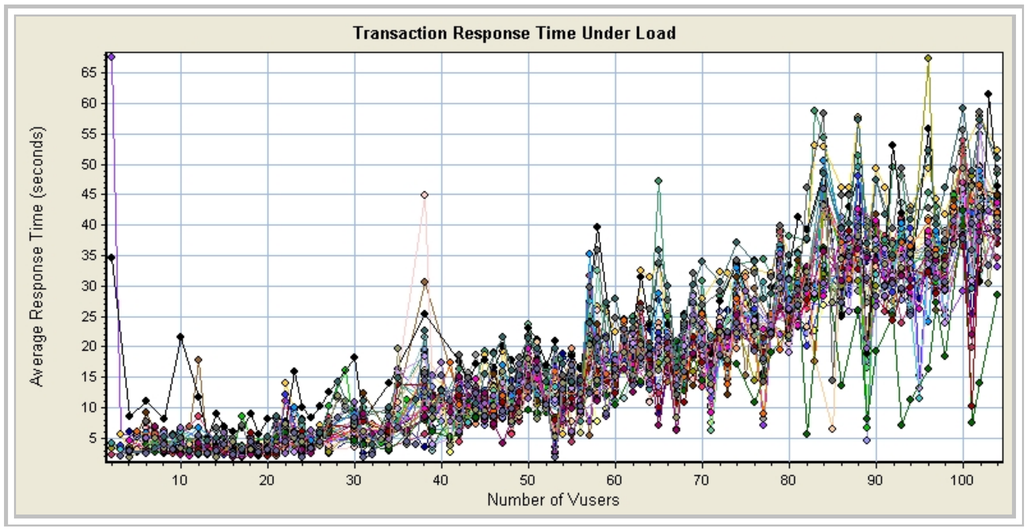
In order to increase the capacity of this application, it would first be necessary to increase CPU capacity. Increasing memory, network capacity or disk capacity would have little or no effect on performance until after CPU capacity has been increased sufficiently.

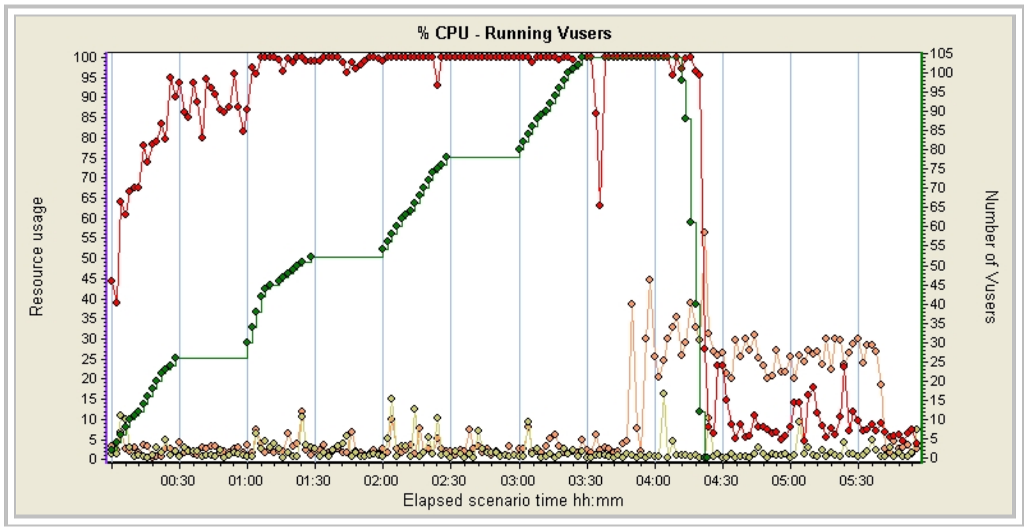
As a practical matter, the CPU is usually the bottleneck device. Sometimes, memory can *appear* to be the bottleneck device because the available memory on a server is artificially constrained, either through pre-allocation of memory or through the use of a garbage collection mechanism that doesn’t free up memory until some threshold is reached. One needs to have an understanding of how the application works in order to properly evaluate this.

Typically, all four service demands are automatically calculated for each host in the application, and then a decision is made as to whether or not to count the memory service demand for each specific host.

Copyright © 2007, Alfred J. Barchi, Inc. All rights reserved.

In the following application, these LoadRunner charts were obtained:



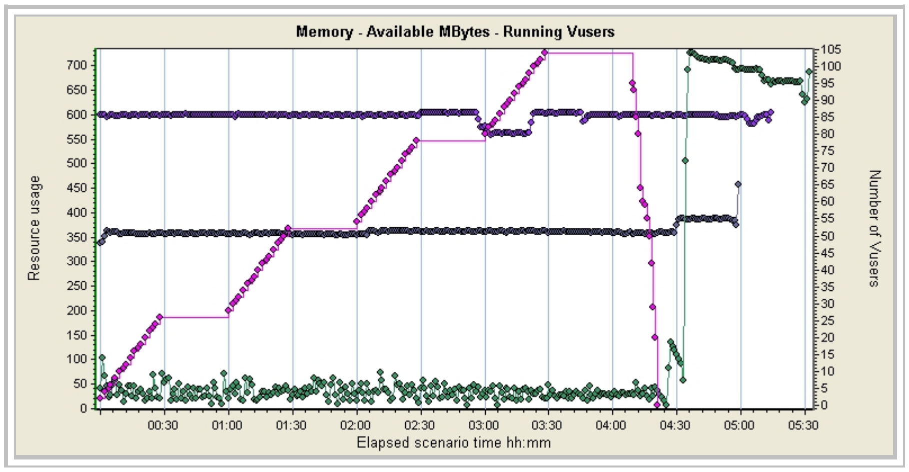


The first chart shows the application going into saturation at a load of about 30 Vusers. The second chart shows CPU utilization. The red line is the CPU utilization of the application server and the green line is the number of Vusers. This chart shows the CPU utilization on the application server going to 100% at a load of 30 Vusers.

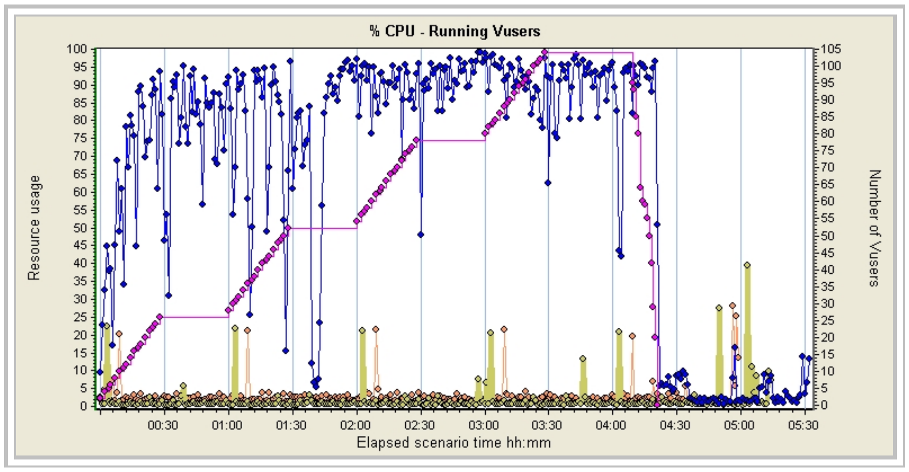
As an additional check, we can look at the processor queue length. Microsoft indicates that a processor queue length consistently greater than 2 indicates a bottleneck (the acceptable queue length is somewhat higher for virtual machines, such as those hosted under VMWare). If we see that the processor queue length is consistently high, we can confirm that the application is in saturation and insufficient CPU is the cause.

Copyright © 2007, Alfred J. Barchi, Inc. All rights reserved.

In this next example, the following LoadRunner charts were obtained:

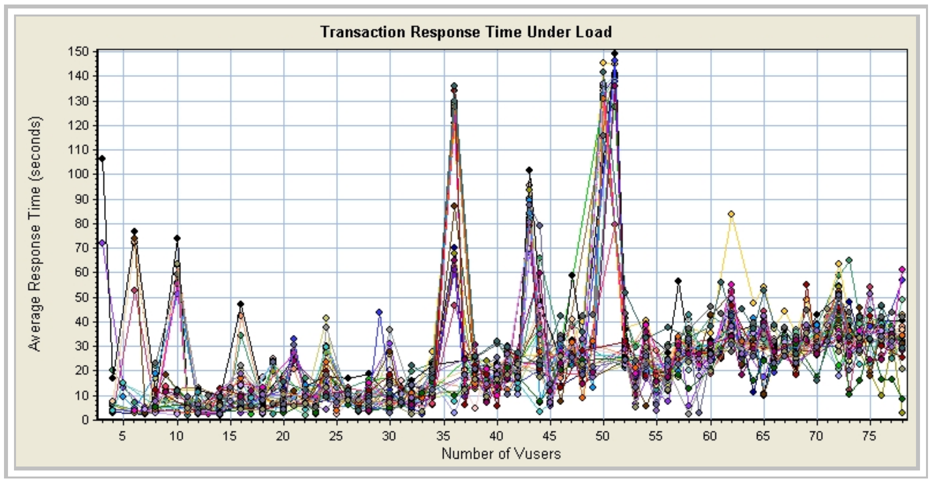


Color	Scale	Measurement	Min.	Ave.	Max.	SD
🟦	1	Memory (Available MBytes):usatrame0378	336	361.55	463	9.241
🟪	1	Memory (Available MBytes):usatrame9500	534	596.883	659	10.465
🟩	1	Memory (Available MBytes):usatrame9501	2	143.297	732	244.255
🟪	1	Run	N/A	N/A	N/A	N/A



Color	Scale	Measurement	Min.	Ave.	Max.	SD
🟧	1	Processor(_Total) (% Processor Time):usatrame0378	0.0	2.592	100	8.636
🟨	1	Processor(_Total) (% Processor Time):usatrame9500	0.0	1.545	100	7.865
🟦	1	Processor(_Total) (% Processor Time):usatrame9501	0.0	66.161	100	42.288
🟪	1	Run	N/A	N/A	N/A	N/A

Copyright © 2007, Alfred J. Barchi, Inc. All rights reserved.



In this particular example, the available memory on the application server was extremely low, reaching a minimum of 2 MB available out of a total of 1 GB. The CPU utilization on the application server was also extremely high. Once again, we can see that the application is starting to go into saturation at about 30 Vusers.

In this case, the memory is the bottleneck device, followed closely by the CPU. When memory was increased from 1 GB to 6 GB, the CPU became the bottleneck device (see the previous example). As a well-known expert in performance analysis observed, "You can never really eliminate a bottleneck device, you can only shuffle the deck."

Measuring and Evaluating Capacity in Operational Applications

In order to monitor an operational application and make predictions for capacity planning purposes, we need to obtain periodic measurements of application throughput, CPU utilization, memory utilization and disk utilization for all servers involved in the application. In certain instances, it may also be important to monitor back-end communications loads between the application servers and the database server, if we suspect that this portion of the application may be approaching saturation.

Measuring Application Throughput

For web applications, application throughput is measured at the web server, i.e., at the interface between the users and the applications. This is done by recording throughput in the web logs. For IIS 6.0 applications, web logs are maintained in "W3C Extended Log File Format," which is described at the following web page:

http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/bea506fd-38bc-4850-a4fb-e3a0379d321f.mspx?mfr=true

This page describes the values that are recorded in the log each time the application is accessed by a client. It should be noted that the throughput fields, i.e., 'Bytes Sent',

Copyright © 2007, Alfred J. Barchi, Inc. All rights reserved.

‘Bytes Received’ and ‘Time Taken’, are not enabled by default. *For capacity planning purposes, they need to be enabled.*

For other web servers, such as Apache, different log file formats are used. However, in every case, it is necessary to record information on the amount of data sent and received by the application.

A log file analyzer, either commercial, shareware or freeware, is used to reduce the data and provide information about average throughput over time, peak throughput periods by hour day, week and month, and average peak throughput values during the peak periods. As a first approximation, we would look at the average throughput during the peak hour. In addition to using this value to obtain current application utilization, we need to track it over time in order to identify growth trends.

Measuring Application Performance

For applications hosted on a Microsoft Windows operating system, application performance is measured by collecting Perfmon data on every host machine utilized by the application. At a minimum, it is necessary to collect ‘% CPU Utilization’, ‘Memory Available’, and ‘% Disk Available’ and ‘Bytes Total/sec’ for each interface on each host machine. Other operating systems provide similar data. The data should be sampled at a rate of 1 sample per minute, 24 hours a day and reported weekly. For archival purposes, it may be necessary to reduce the data even further by averaging it over larger time intervals.

A tool can be easily written in a scripting language, such as Perl, that reads in performance logs, averages data over larger time intervals, and regenerates the logs using the averaged data. That way, running the reduced logs through the analyzer will produce the same results. It should be noted that it is not adequate to simply reduce the sampling rate, because the data obtained can be highly skewed, as periods of either very high or very low activity are missed.

The average performance data for the peak hour of the day is then used along with the average throughput during the same time interval to compute service demands for the application. The process is repeated for each daily peak period, and the throughput and percentage of application utilization can then be plotted to detect growth trends.

Sudden increases in utilization without accompanying increases in throughput can also be used to detect degraded performance modes caused by software ‘aging’ issues, such as memory leaks.

Lastly, actual operational throughput capacities can be compared to throughput capacities predicted by load tests to validate the scenarios used in the load testing.

Copyright © 2007, Alfred J. Barchi, Inc. All rights reserved.

Predicting Future Capacity Requirements

By tracking and analyzing application utilization over time, we can predict the point at which the application will become saturated, and hardware upgrades are needed. Changes in customer demand, customer usage patterns, changes in the size and complexity of the database and changes in hardware are all incorporated implicitly into the application utilization values.

However, changes to the software that add or modify functionality or that affect the efficiency with which the code performs should be evaluated before being made operational. Unfortunately, neither web logs nor performance counters are oriented toward measuring individual user transactions. So, a different approach is required.

Specifically, the application should be load-tested with a performance testing tool such as LoadRunner. In the case where functionality has been added to the application, the application should be load-tested with scenarios for both pre-change application usage and post-change application usage. The delta in application utilization can then be calculated and applied to future capacity predictions.

Ideally, we would want to drive the application into saturation with both the pre- and post-change scenarios.

Application Scalability

Application scalability is a significant issue, both for the application as a whole and for its component parts, e.g., web servers, application servers and database servers. While web servers, and potentially application servers, should scale well, since each user transaction is a separate, discrete and independent event, the back-end database server is a different matter.

In order to evaluate scalability, it is necessary to evaluate capacity of the application in a minimum of four different configurations in order to obtain sufficient data to perform the required statistical analysis (see, for example, "Guerilla Capacity Planning," Neil J. Gunther, the section on "Software Scalability" of a more complete discussion of this topic).

The calculations for this are not straightforward. They involve using regression analysis to calculate two parameters: contention and coherence. 'Contention' refers to when multiple processes attempt to utilize the same resource simultaneously. 'Coherence' refers to when a resource is not in contention, but it is not currently available either. For example, a process may try to access a row in a database. The row is not currently locked by another process, but it isn't yet available because the updated values of the row haven't yet been transferred from the cache of the process that last modified it.

Typically, this scalability model exhibits retrograde behavior as the amount of hardware is increased, vs. models, such as the Amdahl model, that are based on contention alone, and which approach an upper limit asymptotically.

Copyright © 2007, Alfred J. Barchi, Inc. All rights reserved.

Effects of Virtualization

The effects of virtualization need to be carefully considered before adopting this approach as a cost-saving measure. Preliminary indications suggest that while applications can be consolidated onto fewer host machines (at least temporarily, until their utilization grows sufficiently), the resource utilization of the host machines is considerably less than 100% efficient.

This is known as the 'lost MIPS' effect, and it is present whenever virtualization is used, whether on a micro- scale, as with hyper-threading, a meso- scale, as with VMWare and other virtual machine managers, or on a macro- scale, as with GRIDs or P2P networks.

The cause of the degradation is the polling rate of the virtualization manager. For example, with hyper-threading, Intel estimates that the performance gains are at best only 10% to 30%. The effects of polling rate on virtualization efficiency are, however, beyond the scope of this paper.

Additionally, response times tend to be considerably longer in a virtual environment, due to the amount of time that the components of the application spend in a wait state, awaiting their execution time-slice. The reason for this is that when you partition a host machine into several virtual machines, each virtual machine in effect runs at only a fraction of the speed of the host machine. And slower machines do everything more slowly.

Caveats

With LoadRunner, load is measured as the number of Vusers executing a specific scenario. The closer this scenario mimics actual use of the application, the more accurately saturation can be predicted.

Throughput is measured by LoadRunner in bytes/sec. using the average throughput value. Perfmon counters measure statistics on various hardware devices. Average values are used. However, none of these measures can be easily correlated to user transactions. IIS logs show actions originating from user browsers, but it is nearly impossible to correlate a sequence of GETs to a particular user action, particularly when anonymous access is used through NATed firewalls. So it is difficult to accurately profile actual user load. Other third-party solutions, such as Omniture, may be helpful in this regard.

In addition, whenever a new capability is added to the application, it may be similarly difficult to predict its impact without understanding the current user load profile in a quantitative way, and how the new capability will affect the profile.

Applications are complex. The closer an application gets to saturation, the more accurate will be the prediction of the saturation point. Best of all is driving the application into saturation and measuring the saturation point directly.

Copyright © 2007, Alfred J. Barchi, Inc. All rights reserved.