Performance Benchmark Fundamentals

Performance benchmarks on components or designs seek to establish the capacity and limitations of the design under various conditions. This paper discusses the goals of benchmarking and the experimental design considerations for producing benchmarks.

Version 1.0

September 9, 2008

The Power of Now®

http://www.tibco.com

Global Headquarters
3303 Hillview Avenue
Palo Alto, CA 94304
Tel: +1 650-846-1000
Toll Free: 1 800-420-8450
Fax: +1 650-846-1005

Contents

1 Introduction

The goal of performance measurement is to understand the performance capabilities and limitations of an implemented component or design, which we will henceforth refer to as the *system under test* or *system* for short. Note that when we refer to the system under test, this includes all the software involved, the hardware it executes on, and the network infrastructure. Every system has finite limitations, and benchmarks seek to characterize the system in such a way that readers of the benchmarks can understand those limitations.

1.1 Benchmarks Reflect the Test Environment

The performance measurements you get reflect the combination of software, hardware, and networks being tested rather than the abilities of any given element of the design. Commonly, deploying the same components on different hardware and software often yields significantly different results. Thus when interpreting performance results, it is essential to understand the full design and deployment of the test environment and further to be able to relate your intended production environment to the test environment.

1.2 Benchmarking is not Simple

Benchmarks can be complicated if the system capabilities and limitations vary depending upon the nature of the demands being placed on the system. The capabilities and limitations can also vary with the resources that are available to the system (e.g. CPU, memory, network bandwidth, and disk bandwidth). The set of benchmark measurements must be carefully designed so that the impact of these factors can be clearly understood.

In order to design benchmark experiments you need a deep understanding of the system being tested. You need to know how various combinations of factors stress the design and then design a set of experiments that clearly illustrate the impact of each factor. You also need a deep understanding of the intended system usage so that your test results can be readily related to the real-world usage of the system.

Under ideal circumstances, the test and production environments are identical and the test cases reflect exactly the intended utilization. Under less than ideal circumstances, the test cases do not match the intended utilization and the test environment differs in meaningful ways from the production environment. Under such circumstances, you need to be able to determine whether there are any benchmark results that can be reasonably extrapolated to predict production performance. If not, you may have to design your own benchmark experiments to accurately predict the system's behavior in production. In either case, you need a deep understanding of the factors that influence performance in both environments in order to use benchmarks to reliably predict production performance.

2 Benchmark Basics

Each benchmark measurement is basically a data point on a performance curve. In order to understand the benchmark, you need to understand where that benchmark is on the performance curve.

2.1 Performance Curves

Figure 1 shows a basic performance curve. The X axis characterizes the demand that is being placed on the system expressed as a rate of some kind. While this is typically the rate at which inputs are being applied, it may be more appropriate to characterize an input data rate or some other rate measurement depending upon the nature of the component. The Y axis characterizes the rate at which the component is responding to the demand, again expressed as a rate. While the output rate is typically the rate at which the outputs (responses) are being delivered, once again it may be more appropriate to characterize an output data rate or some other measurement.

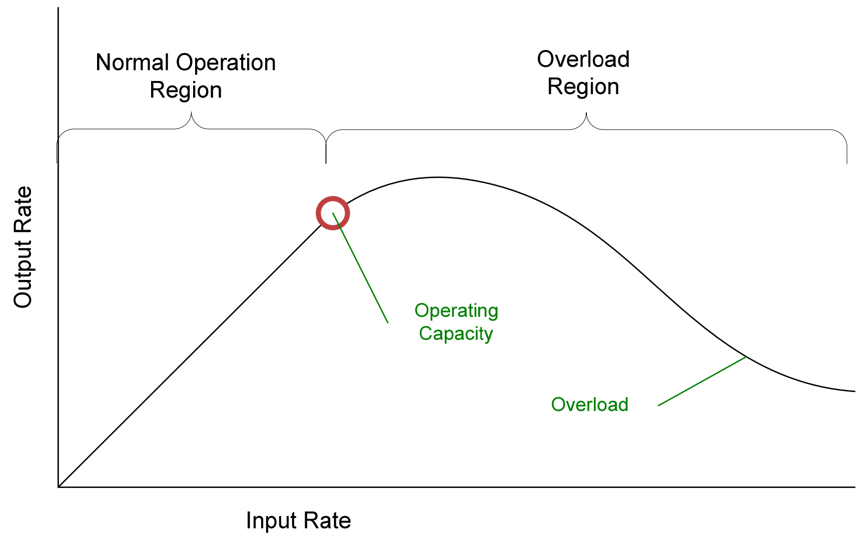

Figure 1: Basic Performance Curve

2.2 Benchmark Experiments

This curve is the result of a series of controlled experiments (Figure 2). In each experiment, the system is being exposed to demands at a controlled rate and operated steady-state for some period. After an initial stabilization period, both the input and output rates are measured, thus providing one data point for the performance curve. This, of course, assumes that all other factors are being held constant.

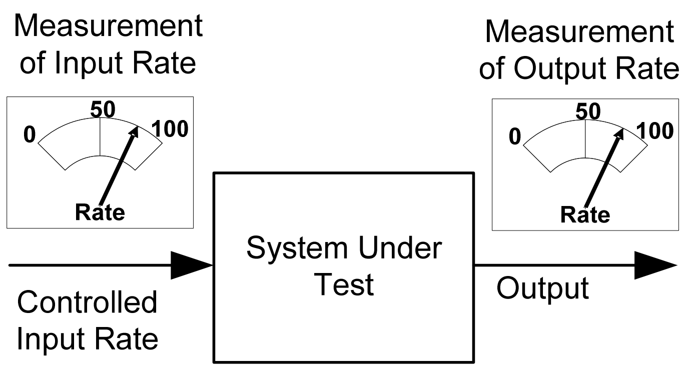

Figure 2: Experimental Test Setup

The shape of the performance curve tells us a lot about the system under test. If each input results in a single output, then over the normal operating range, the output rate will exactly match the input rate (within statistical bounds). This means that new demands are being placed on the component at the same rate that work is being completed. If each input results in more than one output or if you are measuring data rates instead of input and output rates, there may be a scale factor relating the two values. Regardless, over the normal operating range there should be a linear relationship between the two rates – i.e. the curve is a straight line.

The input rate that marks the end of this linear region marks the ***operating capacity*** of the system as deployed in the experiment. This may mark the true limits of the system design, or it may indicate that some type of resource limit has been hit: the available physical memory may have been exhausted, or the bandwidth available on the NIC card may have been exhausted, or the available CPU cycles may have been exhausted. Whatever the limit is, once it is hit it is important to determine the nature of the limit, as this may indicate ways to alter the environment and increase capacity either by tuning the system or adding resources.

Beyond the operating capacity further increases in the input rate exceed the capacity of the system to perform work (for whatever reason). Once this occurs, increasing the input rate will no longer produce the same level of increase in the output. What is happening here is that the inputs are piling up faster than the system is producing output. If the input rate continues to increase, a point will be reached at which the output rate actually begins to decline. The system is taking resources away from completing work and applying them to simply accepting the inputs.

Operation in the overload region is inherently unstable. Inputs are arriving faster than work is being completed, and the result is that inputs are piling up somewhere. Wherever this buffer is (memory, disk, messaging system, etc.), it is finite in capacity. When the capacity is reached, the system will ultimately fail. Thus systems can, at best, operate in the overload region for short periods of time.

2.2.1 Documenting the Experimental Design

The measurements being made reflect the design and configuration not only of the system being tested but also of the test harness being used to test it. In order to understand the measurements and their implication, you need to understand the test setup – well enough to duplicate the setup and run the experiments again, if necessary.

The test setup must first be described in terms of the components involved and the interfaces they use to interact with one another. Figure 3 shows two commonly used test setups.

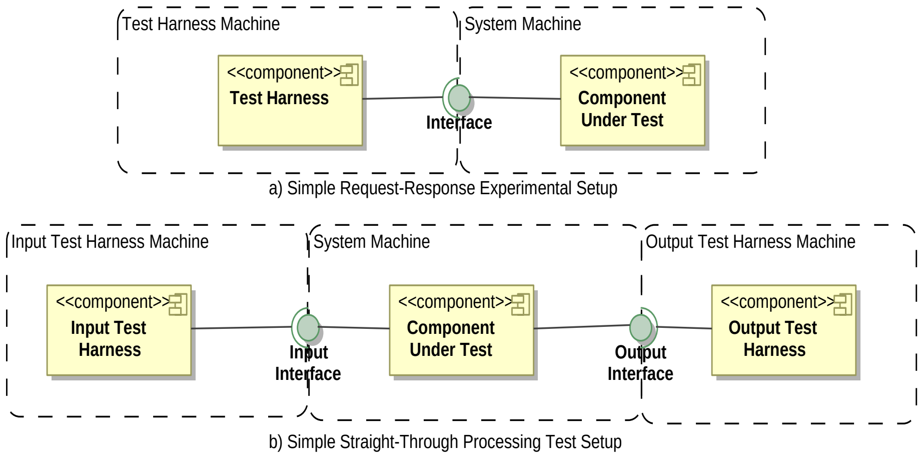

Figure 3: Test Setup Logical Design

Once the components and interfaces have been identified, the individual components must be described in terms of the software and software involved. For experiments in which more than one input will be processed simultaneously, the test harness documentation must identify the number of parallel threads being used, as this identifies the maximum possible number of concurrent inputs and/or outputs.

The machines on which the components are operating must also be documented. Figure 4 shows some typical documentation.

Performance Experiment Setup

System Under Test (repeat for each machine)

Processor	
CPU Type	x86
Number of CPUs	1
Cores/CPU	2
Clock Speed	2.66 GHZ
Memory	3GB
Network Interface Card	
Bandwidth (Mbit/second)	100
Half/Full Duplex	Full
Disk	
Number of Spindles	1
RPM	10,000
RAID Configuration	N/A
Access Bandwidth	10 MB/Sec
FT Ram Buffer?	No

Test Harness (repeat for each machine)

Processor	
CPU Type	x86
Number of CPUs	1
Cores/CPU	2
Clock Speed	2.66 GHZ
Memory	3GB
Network Interface Card	
Bandwidth (Mbit/second)	100
Half/Full Duplex	Full
Disk	
Number of Spindles	1
RPM	10,000
RAID Configuration	N/A
Access Bandwidth	10 MB/Sec
FT Ram Buffer?	No

Network

Backbone Bandwidth	1 Gbit

Figure 4: Machine and Network Documentation

2.2.2 Basic Data Collection

The test design should specify the data to be collected. This data, along with the values of any parameters that could impact the test results, need to be captured as part of the experiment. Some of the parameters, such as input message size, will reflect the setup of the test environment. Others, such as the number of worker threads, will reflect the tuning of the system being tested. Both must be understood in order to accurately interpret the data.

Measurements themselves are rarely perfect. The tools used require resources as well, and this resource demand can limit the resources available to the system under test and thus alter the measurements. Tools may also capture data using conveniently available interfaces – data that in some cases may not measure exactly what you'd like to measure. For these reasons, the tools used to take the measurements should be documented as part of the experiment.

Collected data is generally placed in a spreadsheet along with the key parameter values (Figure 5). Note that some of the data is calculated: the input data rate is the product of the measured input message rate and the size of the message.

The output data rate, depending upon the experiment, may be either an actual measurement or a calculation. Because of the difficulties in making direct measurements, disk access rate and bandwidth utilization are almost always computed based on the input and/or output rates and some knowledge of the actual system design.

Experimental Paramters

Message Size (	5
Worker Thread	10

Test Data

Input Message Rate (messages/ second)	Input Data Rate (KB/ second)	Output Message Rate (messages / second)	Output Data Rate (KB/second)	Latency (milliseconds)	CPU Utilization (% of available CPU)	RAM Utilization (MBytes)	Disk Access Rate (accesses/ second)	Disk Bandwidth Utilization (MB/ second)	Network Bandwidth Utilization (KBytes/ second)	Available CPU
1	5	1	5	2	0%	150	3	0	10	100%
2	10	2	10	2	0%	150	6	0	20	100%
5	25	5	25	3	0%	150	15	0	50	100%
10	50	10	50	3	1%	151	30	0	100	100%
20	100	20	100	4	1%	151	60	0	200	100%
50	250	50	250	4	3%	153	150	1	500	100%
100	500	100	500	5	5%	155	300	2	1,000	100%
200	1,000	200	1,000	5	10%	160	600	3	2,000	100%
500	2,500	500	2,500	6	25%	175	1,500	8	5,000	100%
1,000	5,000	1,000	5,000	6	50%	200	3,000	15	10,000	100%
2,000	10,000	1,250	6,250	10	100%	250	3,750	19	16,250	100%
5,000	25,000	800	4,000	38	100%	400	2,400	12	29,000	100%
10,000	50,000	500	2,500	120	100%	650	1,500	8	52,500	100%
20,000	100,000	350	1,750	343	100%	1,150	1,050	5	101,750	100%
25,000	125,000	1	5	150,000	100%	1,400	3	0	125,005	100%

Figure 5: Typical Experimental Data

The most useful representation of benchmark data is generally a graph with the input message rate on the horizontal axis and one or more of the other data sets being graphed on the vertical axis. For most of the data sets logarithmic scaling on the axes is appropriate.

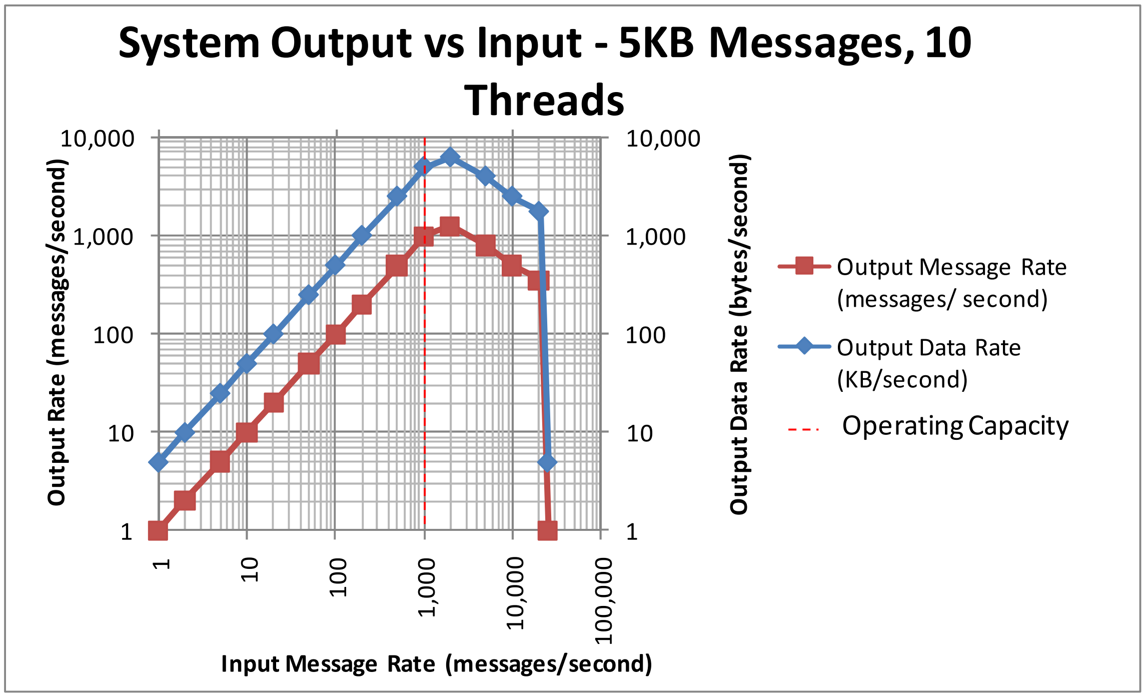

Figure 6: Basic Performance Results

2.2.3 Interpreting Benchmarks

Each benchmark measurement provides a single data point on the performance curve, but in order to meaningfully interpret that benchmark you need to know where you are on the curve. A failure to understand where you are on the curve can lead to significant misinterpretations of the data. Thus it is important that the benchmark measurements characterize the full range from no-load up to the design capacity. Measurements beyond design capacity may or may not be warranted depending upon the intended use of the benchmarks. In many cases it is important to understand the behavior of the system under overload

2.2.4 Potentially Misleading Experiments

2.2.4.1 Overload Tests

One of the most commonly run performance experiments is the overload (drain-the-bathtub) test. In this experiment, a large but fixed number of inputs are applied at the fastest possible rate, often by placing them all in an input queue and then turning the system on. The output rate of the component is then measured.

While this is a perfectly valid experiment, the results are often falsely misinterpreted as being indicative of the system’s operating capacity. If you look at the full performance curve for the system, it is highly likely that during this particular experiment the system is operating far into the overload region. In this region, the output rate can be significantly below the system’s true operating capacity. This is not to say that overload tests are unwarranted – it is important to understand the behavior of the system under overload conditions. However, overload tests usually do not reflect the system’s true capacity. This is especially true when it is possible to further configure or tune the system to limit the input rate so that it cannot exceed the operating capacity.

2.2.4.2 Low-End Performance Tests

Another type of testing, also misleading, involves running a few experiments at the low-end of the performance spectrum. While these experiments may be sufficient to establish the slope of the normal operation curve, they give no insight as to the actual capacity of the system. In fact, they can lead to false conclusions when comparing designs.

For example, when load distribution is added to the design, the addition of the load distribution mechanism adds some overhead. This translates into a requirement for more resources per input. Measurements at the low end of the performance curve will show only the increased resource utilization, giving the impression that the design without the load distribution mechanism is, in some sense, better. It is not until you reach the upper end of the performance curve that the real difference between the designs becomes apparent, with the distributed load design having a much higher design capacity than its non-distributed counterpart.

3 Latency Measurements

Latency measurements characterize the time delay between the application of each system input and the production of its corresponding output (Figure 7). Making latency measurements requires some design work, capturing the timestamps for both the input and output, correlating the two, and computing the difference. Since it is to be expected that latency will vary with input rate, it should be measured for each input rate. Furthermore, it is to be expected that latency will be much worse in the overload region than in the normal operating region. Thus, for any given latency measurement it is essential to know where in the operating region or overload region the latency measurement is being taken. It is good practice to show throughput and latency on the same graph, using separate Y-axis scales for each.

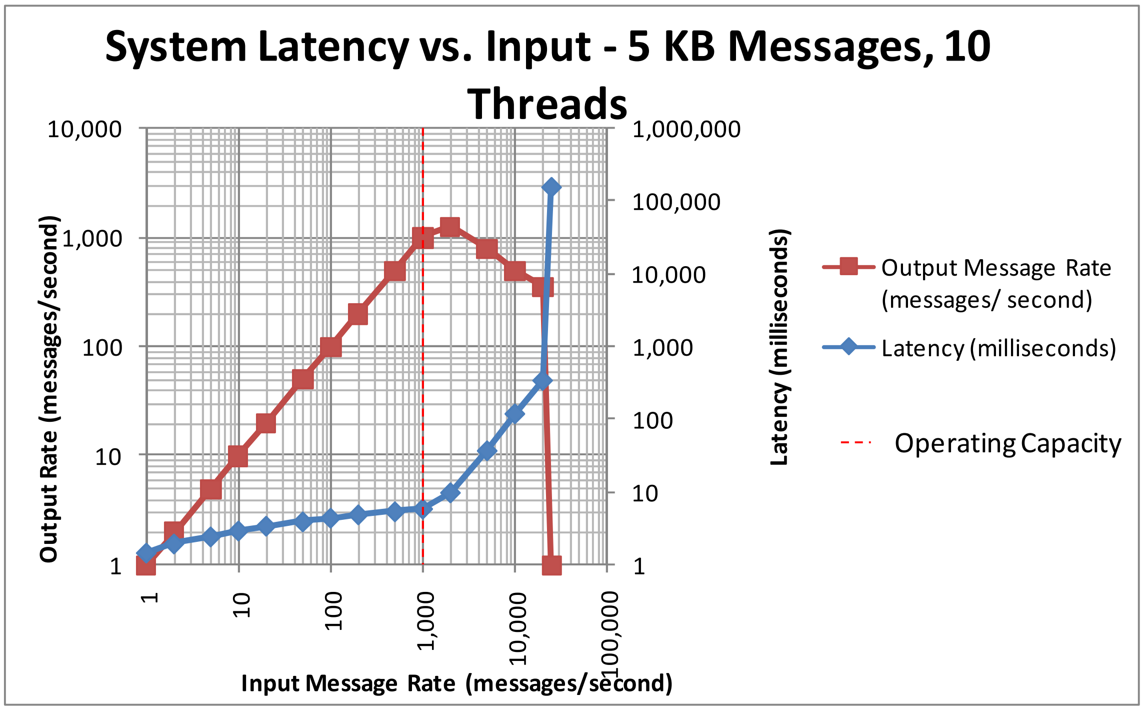

Figure 7: Latency Measurement

4 Resource Consumption Measurements

Systems require resources in execution, primarily the central processing unit (CPU), network bandwidth, disk, and memory. As the input rate increases, so does the demand for the resources until the available supply of that resource has been exhausted: the system has reached its operating capacity.

Measuring the resource consumption during the benchmark test will tell you the resource consumption, but it may or may not indicate which resource has been exhausted. If all of one available resource is being consumed and there is still available capacity on the other resources, one may reasonably assume that the resource that is fully consumed is the limiting factor determining the operating capacity. On the other hand, if the operating capacity is reached without exhausting all of the available resources, it may be that the internal design of the system is, itself, limiting the demand for resources. In such cases, it may be that the tuning the system will allow it to utilize additional resources and thus move the operating capacity.

4.1 CPU Utilization

The central processing unit (CPU) is the core engine of most systems. As the input rate increases and the system attempts to do more work, the amount of CPU that is consumed will increase. If the CPU is not the limiting factor in system performance, then you will obtain a graph similar to Figure 8 in which the exhaustion of the available CPU occurs after the system has reached its operating capacity – if it occurs at all. In some designs, the CPU capacity may never be exhausted as the system is incapable of placing that much demand on the CPU.

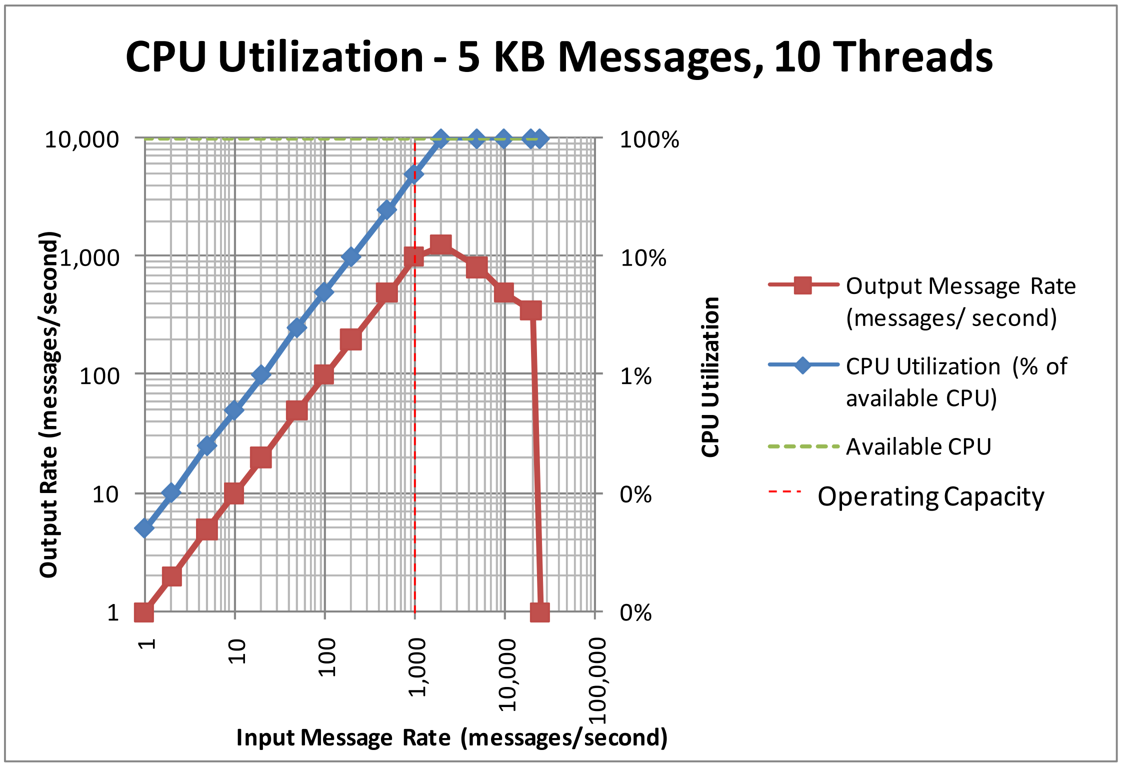

Figure 8: CPU Utilization without CPU as the Limiting Performance Factor

The inability to fully utilize the available CPU may be an indication that the exhaustion of some other resource is determining the operating capacity. On the other hand, it may be an indication that the internal design of the system is limiting its ability to utilize resources. For example, if the system design is single-threaded and you are running it on a two-CPU (or dual core) processor, the design will only be able to utilize 50% of the available CPU. ***Thus if the measurements indicate that none of the system resources are being fully utilized at operating capacity, it may be that tuning of the system (e.g. adding additional threads) will increase its ability to use those resources and thus increase the operating capacity.*** For this reason it is essential to determine exactly what the limiting factor is when performing benchmark tests.

On the other hand, if the CPU is the factor limiting the operating capacity, you will obtain a graph similar to that of Figure 9. Here you can see that the full utilization of the CPU coincides with the system reaching its operating capacity.

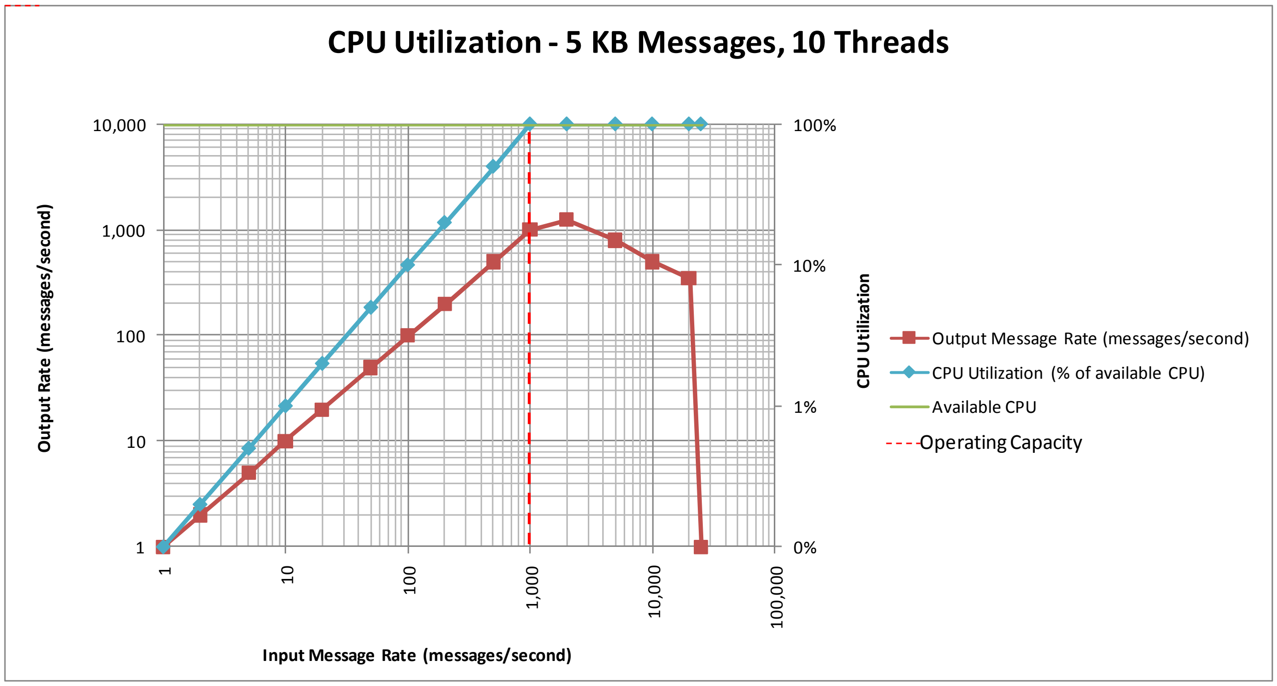

Figure 9: CPU Utilization with CPU as the Limiting Performance Factor

4.2 Network Bandwidth Utilization

Another resource whose availability might limit throughput is network bandwidth. This is particularly the case when a series of experiments is repeated with increasing message size on each experiment. The only way to be sure that the network bandwidth is not the limiting factor is to record (or compute) the network bandwidth utilization and graph it against the available bandwidth as shown in Figure 10. Typically the network bandwidth utilization will be the sum of the input data rate and the output data rate, although other communications (including disk I/O over the network) may come into play as well. In this example, disk I/O is occurring over a dedicated fiber channel connection and thus does not impact network bandwidth. Note that while the full network bandwidth is consumed, this only occurs well into the overload region – network bandwidth is not the limiting factor in system performance.

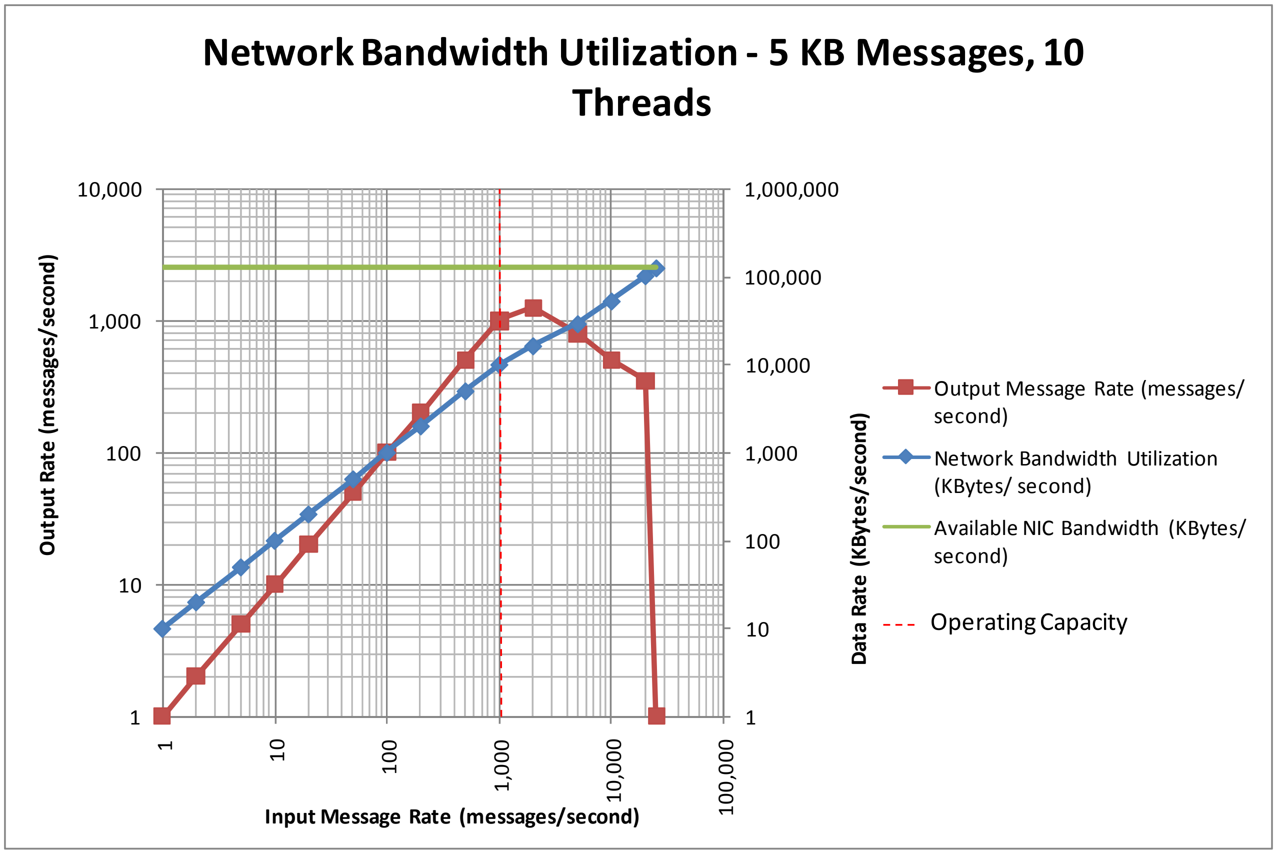

Figure 10: Network Bandwidth Utilization with Other Factors Limiting Performance

When the network does become the limiting factor (as in Figure 11), hitting the bandwidth limitation usually alters the experiment itself. Here the message size has been increased from 5KB in the earlier experiments to 100KB. This message rate is so high that the network cannot handle 1,000 inputs and outputs per second. In fact, as the graph indicates, the bandwidth becomes exhausted at 625 messages/second input rate. Beyond that, further increases in the input rate result in a decrease in the output rate since the system cannot obtain sufficient network resources to send the output. By the time the input rate has reached 1250 messages/second, the input is fully consuming the available network bandwidth and the output rate has dropped to zero. Experimentally, it is impossible to increase the input rate further since the network is fully loaded.

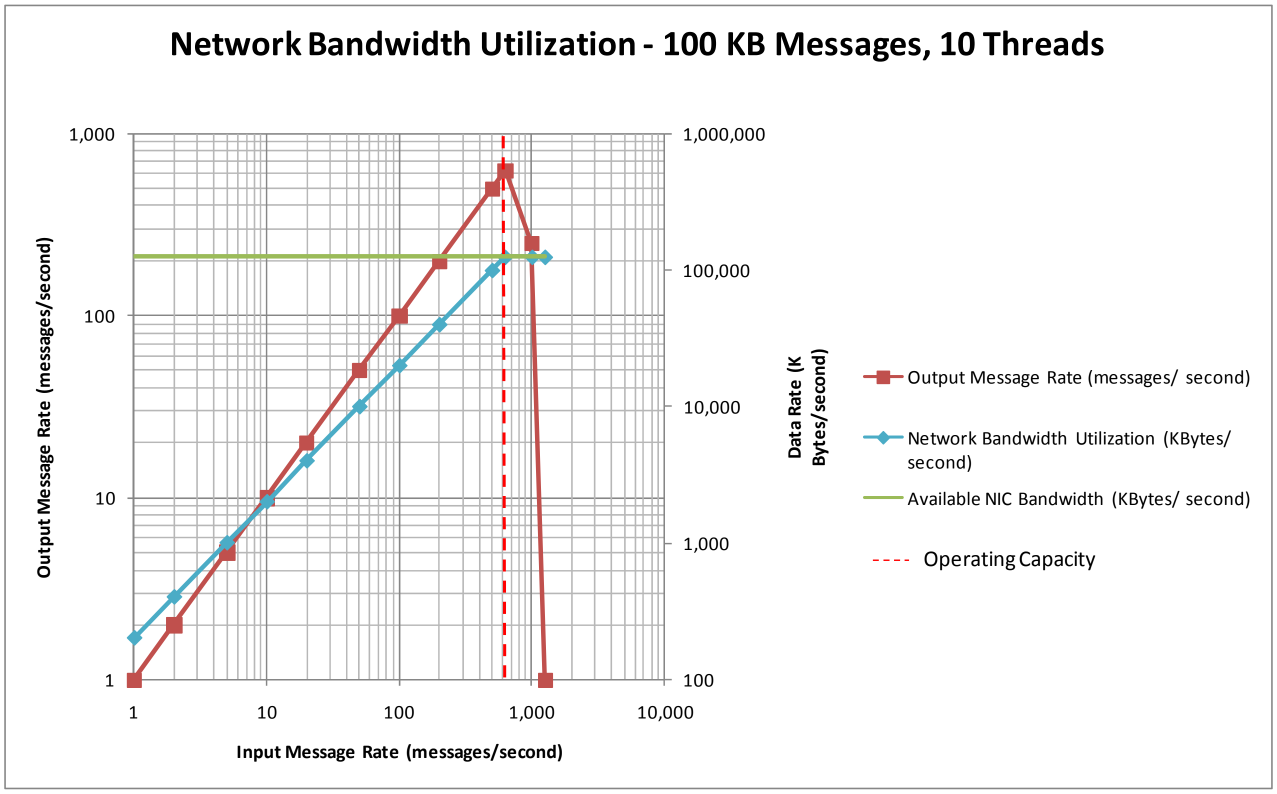

Figure 11: Network Bandwidth Utilization with Bandwidth Limiting Performance

4.3 Disk Utilization

For systems that utilize files and/or databases in their operation, disk performance rather than system performance is often the limiting factor. When disks are being utilized, you need to understand the nature of the disk subsystem, the manner in which the machines actually access the disk subsystem, and the synchronous/asynchronous manner in which the system employs the disk subsystem. The following provides a brief summary of the factors affecting disk performance. A more detailed discussion of these factors can be found in *Implementing SOA: Total Architecture in Practice*[1].

A disk subsystem can range in complexity from a dedicated disk directly mounted within the machine to a stand-alone storage subsystem providing logical disks to machines. In general, you need to know enough about the disk subsystem to determine the *latency* involved in responding to individual requests and the *rate* at which data can be transferred during each request. For most disk subsystems, these parameters are determined by the physical characteristics of each disk, the number of disks in the RAID array (if there is one), and the configuration of the RAID array. Here an interaction with the physical disk is required to actually persist the data in a manner that will survive a system restart. However, some high-end disk subsystems have fault-tolerant battery-backed buffers. With these subsystems, only an interaction with the buffer is required to persist the data, and interactions can be several orders of magnitude faster than with physical disks.

Key questions to ask about the storage subsystem include:

- Is the disk part of the machine using it, or is it in a separate storage subsystem?

[1] Paul C. Brown, *Implementing SOA: Total Architecture in Practice*, Addison-Wesley, Boston, MA (2008) pp 605-609

- If it is in a storage subsystem, does the subsystem have a fault tolerant buffer?
- Is the disk a single physical disk, or a RAID array? If it is a raid array, how many spindles are in the array, and what RAID configuration is being used?
- What is the rotational speed of each disk?
- What is the average seek time of the disk head?
- What is average data transfer rate for the disk?
- What is the size of the buffer cache for the disk?

For a single spindle, the average rotational latency of the disk is:

$$avgRotationalLatency_{\mathrm{sec}} = \frac{1}{(diskSpeed_{RPM} / 60) \times 2} = \frac{1}{diskSpeed_{RPM} / 30}$$

The time it will take to transfer the data is:

$$dataTransferTime_{\mathrm{sec}} = \frac{messageSize_{bytes}}{transferRate_{bytes/\mathrm{sec}}}$$

For synchronous writes to the disk (the most demanding requirement, but one that is fundamental to avoiding data loss) the average rate at which the disk can be accessed is thus:

$$avgAccessRate_{\mathrm{sec}} = \frac{1}{\max(avgSeekTime + avgRotationalLatency) + dataTransferTime}$$

For the disk used in this example, this works out to 212 accesses/second. Figure 12 shows what the consequences would be on system performance if this access rate were to become the constraining factor: the operating capacity would be reduced from 1,000 messages/second down to approximately 65 messages/second.

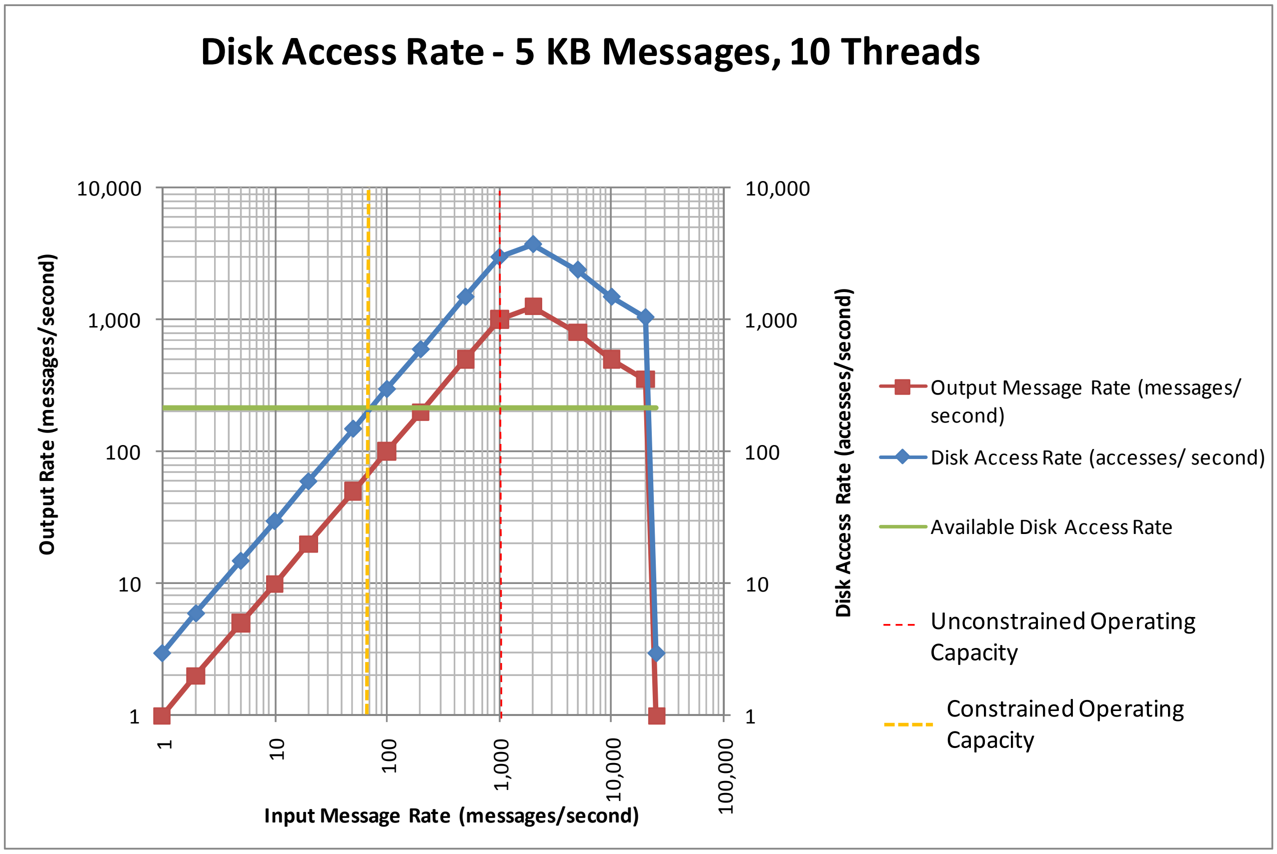

Figure 12: Disk Utilization Showing Impact of Access Rate Constraints

Another important consideration is the manner in which the application writes to the disk: synchronously or asynchronously. Synchronous writes (which must be used for all critical data storage) require the application to wait for the disk write operation to complete before proceeding. With asynchronous writes, the application can continue on to the next task while the disk write operation takes place. Thus asynchronous write operations generally yield higher throughput numbers. However, asynchronous writes also run the danger of losing data in the event of a system and/or disk subsystem failure. When examining performance numbers related to disk utilization, be sure you understand whether synchronous or asynchronous writes were being used for the test!

4.4 Memory Utilization

As with other limited resources, it is important to understand whether the availability of physical memory is a limiting factor in performance. To gather memory data, it is necessary to operate the system at each input rate long enough that the system's use of memory reaches a steady-state value. A failure of the system to reach steady state is usually an indication of a "memory leak" – the failure of the system to make memory available for reuse once it is done with that memory.

Figure 13 shows a typical memory utilization curve in a situation in which the availability of memory is not constraining system performance. Each system typically has some base level of memory requirement (150 MB in this case) plus an additional increment for each input being processed. As the input rate increases, so does the memory demand. Most systems will scale linearly with the input rate as long as the total memory demand does not exceed the amount of physical memory available to the system. Even in a system with virtual memory, once the physical memory threshold is exceeded, performance will begin to deteriorate. The implication is that the amount of physical memory available should be at least as great as the memory demand when the system is running at its operating capacity.

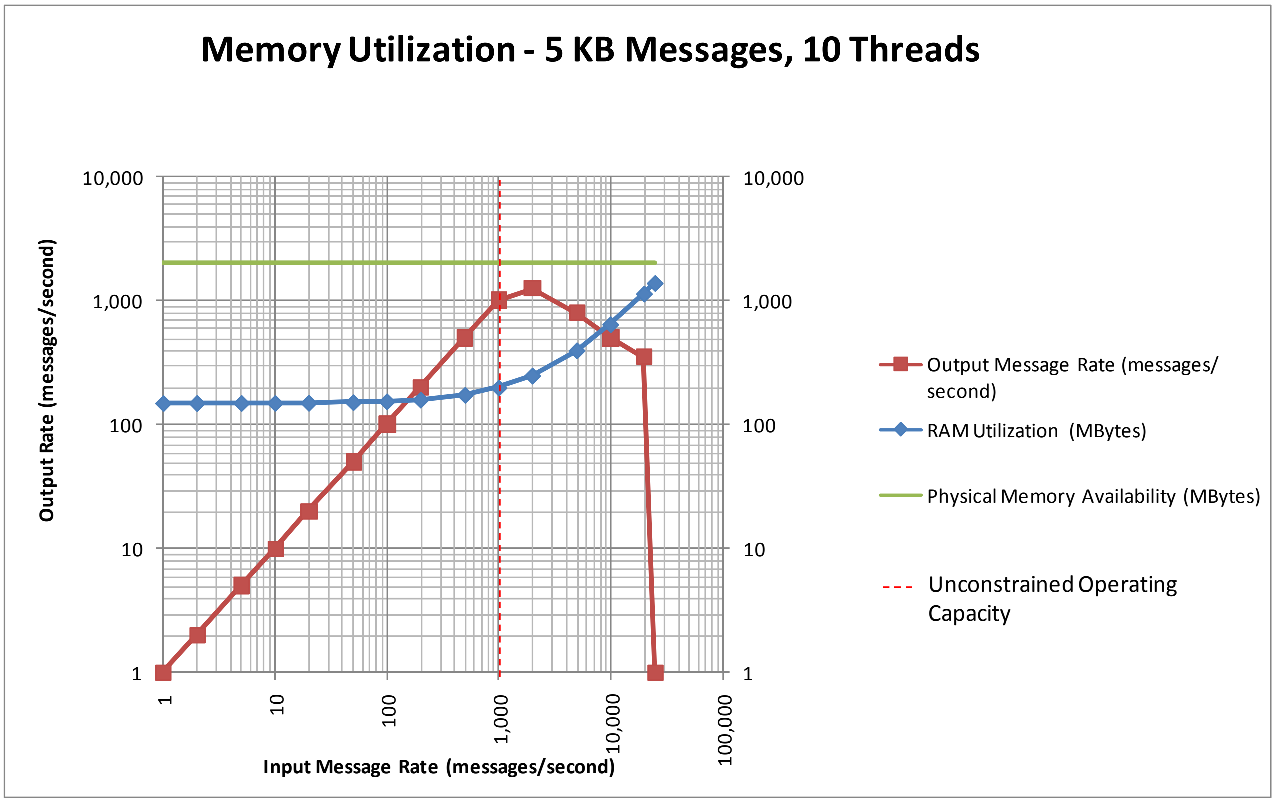

Figure 13: Memory Utilization

5 Experimental Variables (Parameters)

Besides the limited availability of resources, there are often other factors that can influence the capacity of a system. When these factors can be altered by setting parameter values, it is important to document these values for each experiment. Typical parameters include (but are not limited to):

- The number of threads in each system component
- The sizes of buffers
- Constraints on the number of activities that can occur in parallel
- The number of allowed connections (there may be different numbers for different connection types)
- The number of concurrent sessions

In addition to recording these values, it should be recognized that each value represents a potentially artificial constraint on the system capacity. When it appears that the capacity of a system has been reached without identifying a resource constraint whose limit has been reached, it is prudent to examine the parameter values to determine whether altering one of these values might increase the capacity of the system. Such tuning experiments should be a normal part of any capacity measurement experiment.

6 Test Harness Limitations

When the apparent capacity of a system is reached without having exhausted any of the available resources, it is prudent to consider whether the limiting factor might be the test harness rather than the system under test. A test harness with a limited number of threads may not be capable of providing inputs or receiving outputs at the rate needed to drive the system to its full capacity. This is particularly true when testing messaging systems or any other system that does a relatively small amount of work for each input received. Ideally, the test harness will be configurable through its own parameters. In some cases, it may be necessary to run multiple copies of the test harness, each on a different machine, in order to drive the system under test to capacity.

7 Performance Measurements with Multiple Components

The preceding discussion assumes that the system being tested consists of a single component as in Figure 14a. Additional considerations come into play when the system consists of multiple components or when comparing multi-component systems with single-component systems.

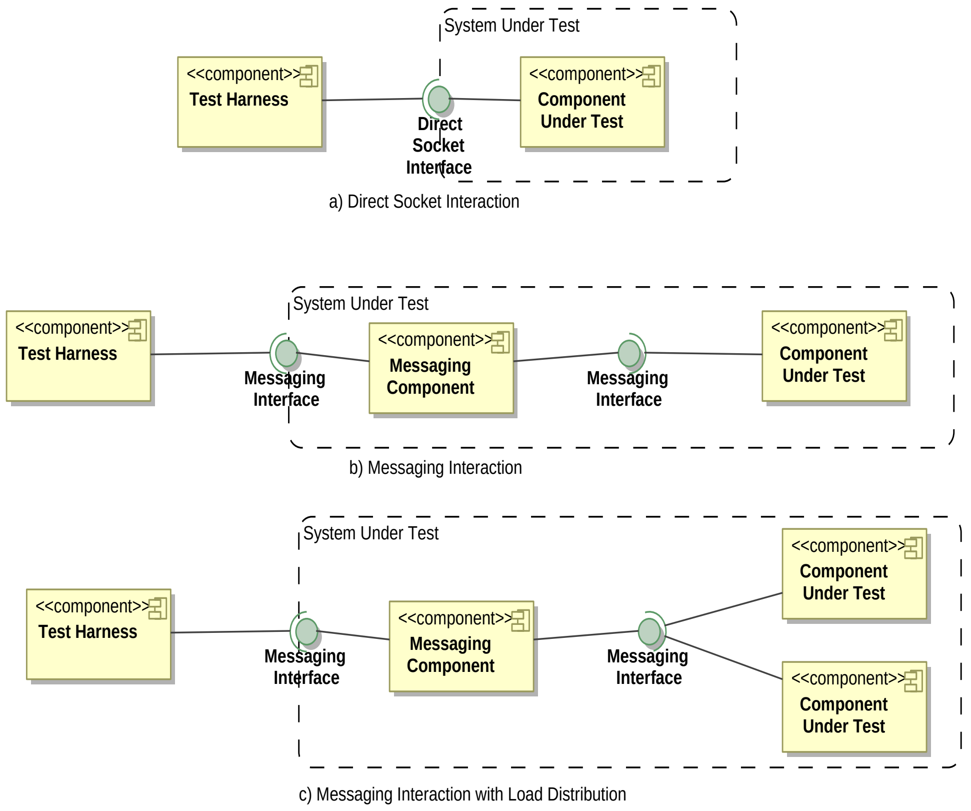

Figure 14: Multi-Component Test Setup Variations

7.1 Comparing Alternate Designs with Different Communications

Consider, for example, two alternative component designs, one that offers a direct socket connection (as in Figure 14a) and the other that offers a message-based interface (as in Figure 14b). A test harness is prepared that is capable of using either interface, and a series of performance experiments is run on both configurations. The measurements gathered by the experiments seem to indicate that the direct socket connection has a higher capacity than the messaging-based experiment. While it is entirely possible that the message-based design does, indeed, have a lower capacity, there is another explanation that must be considered and eliminated before this conclusion can be drawn.

Let us assume that the test harness has exactly enough threads to drive the socket-based interface component to its full capacity. Each thread sends an input to the system under test and waits for the reply. As soon as the reply is received, it sends another input. What happens to this test harness when it is connected to a messaging system? Well, the messaging system will add some additional latency. This, in turn, will cause each thread of the test harness to provide inputs at a lower rate. If you are measuring only the output rate, then the messaging-based design will apparently have a lower capacity than the socket-based design. Here the test harness itself is the limiting factor in the test. Adding additional threads to the test harness (or an additional instance of the test harness) will increase the rate at which the test harness is capable of providing inputs, and thus allow it to drive the system to its full capacity.

This example illustrates why it is important to measure both input and output rate and to vary the input rate until the true operational capacity of the system has been identified.

7.2 Comparing Alternate Designs with Load Distribution

Assume, for the moment, that the experiment described in the previous section actually does show that the messaging-based design has a lower capacity than the socket-based design, at least with the messaging component and component under test running on the same machine. Does this necessarily indicate that the socket-based design is better? No.

The messaging-based design has the ability to deliver messages to more than one instance of the component under test (Figure 14c). If the two instance of the component under test are run on different machines, it is likely that the total capacity of the messaging server and the two instances of the component under test will be significantly greater than the capacity of a single instance of the component under test – regardless of the communications mechanism. Once again we see that true performance comparisons require driving the system to full operational capacity in order to obtain meaningful results.

8 Performance Measurements with Complex Components

Measuring the performance of complex components is … well … complex! There are different kinds of tasks that the component can perform, and the performance depends upon the mix of these tasks being executed. Complex components often involve multiple sub-components, such as an application running on top of a database. Sometimes they offer deployment alternatives in which different sub-components can be deployed on different machines.

Despite these complexities, there is a testing strategy that can be employed to shed practical insight on the component's behavior. This begins with identifying the "simplest" (lightest weight) tasks a component can execute with the simplest (smallest) inputs and outputs. You then characterize the component's ability to execute these tasks, tuning the component to produce the highest throughput with minimal latency (or whatever other optimization is appropriate for the component). This information serves as a baseline that represents the highest throughputs and shortest latencies that are possible with the component. In other words, this is the best performance you can expect from the component (at least on the present platform).

Next, you want to explore how the baseline changes as the inputs and outputs increase in size. Once again, you are trying to determine the basic overhead associated with varying data set sizes. This also affords an opportunity to explore differences in tuning the component to optimally handle inputs of various sizes. You also want to explore how other factors such as the number of records in an underlying database impact performance.

Third, you introduce, individually, more complex tasks, characterizing the performance of each over a range of input and output sizes. The difference between these measurements and the baseline represent the incremental work involved in performing each complex task. To make these tests meaningful, you need to vary the demand over a range from the trivial to a reasonable maximum that you would expect to see in practice. Often, at this point, you end up having to “tune” both the test harness (i.e. increase number of concurrent requests, etc.) and the component itself to get maximum performance. This again provides an opportunity to explore and demonstrate tuning. From this information, you can now begin to build formulas that predict component capacity based on the number of each kind of task, the rate at which they execute, and the size of the data sets involved.

Finally, it is a good idea to explore modifying a few of the variables in combination, particularly where there is interaction between the two. Some typical mixtures of task types, data set sizes, and demand rates should be explored. The intent here is to identify non-linear interactions between these variables – differences in performance that could not be predicted from the earlier individual measurements. In identifying these test cases there is no substitute for understanding the internal architecture of the component as a means of identifying potential non-linear interactions. You also have to be somewhat judicious in your choice of test cases because of the combinatorics: the number of possible test cases is much greater than any practical testing program could ever explore. It is good practice to pick combinations that are commonly found in practice.

When there are deployment options for the component, spend a lot of time characterizing a “basic” installation (one machine, perhaps a second for a related database, whatever the minimum is that you would recommend for a reasonably performant application). The emphasis should be on characterizing the maximum capability of that installation. Then explore one or two common larger variations, with the intent of helping the reader understand the benefits of the multiple deployment and illustrating those benefits through specific performance tests.

9 Interpreting Benchmarks

Benchmarks reflect the optimal, idealized use of resources. As such, their use in predicting actual production behavior requires a somewhat conservative interpretation.

9.1 Contention for Resources

Benchmark experiments generally characterize a system’s ability to fully utilize the available resources: CPU, network, disk, and memory. In real-world applications, however, it is rarely the case that these resources are fully dedicated to just these system components. Other processes run on the same machines, sharing CPU and memory. Multiple machines reside on the same network, sharing network and disk. Predicting system behavior under these circumstances requires an understanding of the actual resource availability and thus require an understanding of the other components that contend for these resources.

9.2 Resource Availability under Complex Demands

Benchmarks, by their nature, tend to be somewhat simplistic in their design. This simplicity often leads to their ability to consume resources at higher rates than may be possible under realistic deployment situations. Networks, for example, may allow a benchmark to consume nearly 100% of the available network bandwidth, but when there are complex communications from many sources it may be difficult in practice to effectively utilize more than 30% of that bandwidth. Consequently, it is common for conservative designs to plan for a maximum resource utilization that is less than 50% of the nominally available resource, with extremely conservative designs running closer to a planned 25% utilization. The same thinking should be applied to the usage of CPU, disks, and memory.

9.3 Extrapolation of Results

One of the biggest challenges you will face with benchmarks is extrapolating the results. If the benchmark was run on a machine with the same processor but a slower clock rate than your production machine, you would like to be able to extrapolate the results and predict what the performance would be on your production machine by scaling up the numbers by the ratio of the clock rates. While this may work in some cases, in others the limitation of network bandwidth, disk bandwidth, or memory speed may keep the performance from reaching the predicted levels. Once again, it becomes important to understand what factors limit performance and how close each benchmark result is to reaching each of the limits.

Extrapolating to different types of CPUs (RISC vs. CISC processors, for example) or single to multi-core processors is risky at best. Sometimes you can find benchmarks comparing one to the other and use these benchmarks to help extrapolate the benchmarks for your design to the different platform. Such extrapolations are, however, somewhat risky.

9.4 When in Doubt – Do Your Own Benchmark!

If you are unsure that you can extrapolate the benchmark you have to predict your system's performance under production circumstances, your best course of action is to develop your own benchmark that reflects your intended usage on your intended platform. However, you can't be naïve about this – you need to understand the design, the factors that are likely to be its limitations, and the tuning parameters available to you in order to do this effectively. This is not an exercise for amateurs!

10 Summary

For best results when performing benchmark tests, adhere to the following best practices:

1. Always document the test design in sufficient detail to allow others to accurately reproduce your results.
2. Always range demand until the operating capacity of the system under test has been reached (i.e. further increases in input rate do not result in proportional increases in output rate)
3. Always document measured or estimated resource availability and consumption
4. Once an apparent operational limit has been reached, investigate to determine whether a true resource constraint has been reached. Consider adding resources (i.e. adding memory, changing to a higher network bandwidth).
5. If an apparent operational limit has been reached without exhausting available resources:

a. Consider whether tuning the system under test might further increase the operational capacity.

b. Consider whether the design or configuration of the test harness might be the true limiting factor in the experiment.

Appendix A: Spreadsheet

The embedded spreadsheet was used to create the figures in this document and may be used as a starting template for performance benchmark experiments.

PerformanceExperimentData.xls