

# Performance Testing

Performance Testing is an overloaded term that is used to refer to several subcategories of performance related testing, each of which has different purpose.

A good description of overall performance testing is as follows:

Performance testing is a type of testing intended to determine the responsiveness, throughput, reliability, and/or scalability of a system under a given workload. [Performance Testing Guidance for Web Applications](#).

Before getting into the different subcategories of performance tests let us understand why performance testing is typically done.

## Why Performance Testing

Performance testing is commonly conducted to accomplish one or more the following:

- **Tune the system's performance**
  - Identifying bottlenecks and issues with the system at different load levels.
  - Comparing performance characteristics of the system for different system configurations.
  - Come up with a scaling strategy for the system.
- **Assist in capacity planning**
  - Capacity planning is the process of determining what type of hardware and software resources are required to run an application to support pre-defined performance goals.
  - Capacity planning involves identifying business expectations, the periodic fluctuations of application usage, considering the cost of running the hardware and software infrastructure.
- **Assess the system's readiness for release:**
  - Evaluating the system's performance characteristics (response time, throughput) in a production-like environment. The goal is to ensure that performance goals can be achieved upon release.

- Evaluate the **performance impact of application changes**
  - Comparing the performance characteristics of an application after a change to the values of performance characteristics during previous runs (or baseline values), can provide an indication of performance issues (performance regression) or enhancements introduced due to a change

## Key Performance Testing Categories

Performance testing is a broad topic. There are many areas where you can perform tests. In broad strokes you can perform tests on the backend and on the front end. You can test the performance of individual components as well as testing the end-to-end functionality.

There are several categories of tests as well:

### Load Testing

This is the subcategory of performance testing that focuses on **validating the performance characteristics of a system**, when the system faces the load volumes which are expected during production operation. An **Endurance Test** or a **Soak Test** is a load test carried over a long duration ranging from several hours to days.

### Stress Testing

**This is the subcategory of performance testing that focuses on validating the performance characteristics of a system when the system faces extreme load.** The goal is to evaluate how does the system handles being pressured to its limits, does it recover (i.e., scale-out) or does it just break and fail?

### Endurance Testing

The goal of endurance testing is to make sure that the system can maintain **good performance under extended periods of load**.

### Spike Testing

The goal of Spike testing is to validate that a software **system can respond well to large and sudden spikes**.

## Chaos Testing

Chaos testing or Chaos engineering is the practice of **experimenting on a system to build confidence that the system can withstand turbulent conditions in production**. Its goal is to identify weaknesses before they manifest system wide. Developers often implement fallback procedures for service failure. Chaos testing arbitrarily shuts down different parts of the system to validate that fallback procedures function correctly.

## Best Practices

Consider the following best practices for performance testing:

- **Make one change at a time.** Don't make multiple changes to the system between tests. If you do, you won't know which change caused the performance to improve or degrade.
- **Automate testing.** Strive to automate the setup and teardown of resources for a performance run as much as possible. Manual execution can lead to misconfigurations.
- **Use different IP addresses.** Some systems will throttle requests from a single IP address. If you are testing a system that has this type of restriction, you can use different IP addresses to simulate multiple users.

## Performance Monitor Metrics

When executing the various types of testing approaches, whether it is stress, endurance, spike, or chaos testing, it is important to capture various metrics to see how the system performs.

At the basic hardware level, there are four areas to consider.

- Physical disk
- Memory
- Processor
- Network

These four areas are **inextricably linked**, meaning that **poor performance in one area will lead to poor performance in another area**. Engineers concerned with understanding application performance, should focus on these four core areas.

The classic example of how performance in one area can affect performance in another area is memory pressure.

If an application's available memory is running low, the operating system will try to compensate for shortages in memory by transferring pages of data from memory to disk, thus freeing up memory. But this work requires help from the CPU and the physical disk.

This means that when you look at performance when there are low amounts of memory, you will also notice spikes in disk activity as well as CPU.

## Physical Disk

Almost all software systems are dependent on the **performance of the physical disk**. This is especially true for the **performance of databases**. More modern approaches to using **SSDs for physical disk storage can dramatically improve the performance of applications**. Here are some of the metrics that you can capture and analyze:

Counter	Description
Avg. Disk Queue Length	This value is derived using the $(\text{Disk Transfers/sec}) * (\text{Disk sec/Transfer})$ counters. This metric describes the disk queue over time, smoothing out any quick spikes. Having any physical disk with an average queue length over 2 for prolonged periods of time can be an indication that your disk is a bottleneck.
% Idle Time	This is a measure of the percentage of time that the disk was idle. i.e. there are no pending disk requests from the operating system waiting to be completed. A low number here is a positive sign that disk has excess capacity to service or write requests from the operating system.
Avg. Disk sec/Read and Avg. Disk sec/Write	These both measure the latency of your disks. Latency is defined as the average time it takes for a disk transfer to complete. You obviously want is low numbers as possible but need to be careful to account for inherent speed differences between SSD and traditional spinning disks. For this counter is important to define a baseline after the hardware is installed. Then use this value going forward to determine if you are experiencing any latency issues related to the hardware.

Counter	Description
Disk Reads/sec and Disk Writes/sec	These counters each measure the total number of IO requests completed per second. Similar to the latency counters, good and bad values for these counters depend on your disk hardware but values higher than your initial baseline don't normally point to a hardware issue in this case. This counter can be useful to identify spikes in disk I/O.

## Processor

It is important to understand the amount of time spent in kernel or privileged mode. In general, if code is spending too much time executing operating system calls, that could be an area of concern because it will not allow you to run your user mode applications, such as your databases, Web servers/services, etc.

The guideline is that the CPU should only spend about 20% of the total processor time running in kernel mode.

Counter	Description
% Processor time	This is the percentage of total elapsed time that the processor was busy executing. This counter can either be too high or too low. If your processor time is consistently below 40%, then there is a question as to whether you have over provisioned your CPU. 70% is generally considered a good target number and if you start going higher than 70%, you may want to explore why there is high CPU pressure.
% Privileged (Kernel Mode) time	This measures the percentage of elapsed time the processor spent executing in kernel mode. Since this counter takes into account only kernel operations a high percentage of privileged time (greater than 25%) may indicate driver or hardware issue that should be investigated.
% User time	The percentage of elapsed time the processor spent executing in user mode (your application code). A good guideline is to be consistently below 65% as you want to have some buffer for both the kernel operations mentioned above as well as any other bursts of CPU required by other applications.

Counter	Description
Queue Length	This is the number of threads that are ready to execute but waiting for a core to become available. On single core machines a sustained value greater than 2-3 can mean that you have some CPU pressure. Similarly, for a multicore machine divide the queue length by the number of cores and if that is continuously greater than 2-3 there might be CPU pressure.

## Network Adapter

Network speed is often a hidden culprit of poor performance. Finding the root cause to poor network performance is often difficult. The source of issues can originate from bandwidth hogs such as videoconferencing, transaction data, network backups, recreational videos.

In fact, the three most common reasons for a network slow down are:

- Congestion
- Data corruption
- Collisions

Some of the tools that can help include:

- ifconfig
- netstat
- iperf
- tcpretrans
- tcpdump
- Wireshark

Troubleshooting network performance usually begins with checking the hardware. Typical things to explore is whether there are any loose wires or checking that all routers are powered up. It is not always possible to do so, but sometimes a simple case of power recycling of the modem or router can solve many problems.

Network specialists often perform the following sequence of troubleshooting steps:

- Check the hardware
- Use IP config

- Use ping and tracert
- Perform DNS Check

More advanced approaches often involve looking at some of the networking performance counters, as explained below.

## Network Counters

The table above gives you some reference points to better understand what you can expect out of your network. Here are some counters that can help you understand where the bottlenecks might exist:

Counter	Description
Bytes Received/sec	The rate at which bytes are received over each network adapter.
Bytes Sent/sec	The rate at which bytes are sent over each network adapter.
Bytes Total/sec	The number of bytes sent and received over the network.
Segments Received/sec	The rate at which segments are received for the protocol
Segments Sent/sec	The rate at which segments are sent.
% Interrupt Time	The percentage of time the processor spends receiving and servicing hardware interrupts. This value is an indirect indicator of the activity of devices that generate interrupts, such as network adapters.

There is an important distinction between **latency** and **throughput**. **Latency** measures the time it takes for a packet to be transferred across the network, either in terms of a one-way transmission or a round-trip transmission. **Throughput** is different and attempts to measure the quantity of data being sent and received within a unit of time.

## Memory

Counter	Description
Available MBs	This counter represents the amount of memory that is available to applications that are executing. Low memory can trigger Page Faults, whereby additional pressure is put on the CPU to swap memory to and from the disk. If the amount of available memory dips below 10%, more memory should be obtained.
Pages/sec	This is actually the sum of "Pages Input/sec" and "Pages Output/sec" counters which is the rate at which pages are being read and written as a result of page faults. Small spikes with this value do not mean there is an issue but sustained values of greater than 50 can mean that system memory is a bottleneck.
Paging File(_Total)\% Usage	The percentage of the system page file that is currently in use. This is not directly related to performance, but you can run into serious application issues if the page file does become completely full and additional memory is still being requested by applications.

## Key Performance Testing Activities

Performance testing activities vary depending on the subcategory of performance testing and the system's requirements and constraints. For specific guidance you can follow the link to the subcategory of performance tests listed above. The following activities might be included depending on the performance test subcategory:

### → Identify the Acceptance Criteria for the Tests

This will generally include identifying the goals and constraints for the performance characteristics of the system

### → Plan and Design the Tests

In general we need to consider the following points:

- Defining the load the application should be tested with
- Establishing the metrics to be collected

- Establish what tools will be used for the tests
- Establish the performance test frequency: whether the performance tests be done as a part of the feature development sprints, or only prior to release to a major environment?

## Implementation

- Implement the performance tests according to the designed approach.
- Instrument the system and ensure that it is emitting the needed performance metrics.

## Test Execution

- Execute the tests and collect performance metrics.

## Result Analysis and Re-testing

- Analyze the results/performance metrics from the tests.
- Identify needed changes to tweak the system (i.e., code, infrastructure) to better accommodate the test objectives.
- Then test again. This cycle continues until the test objective is achieved.

The [Iterative Performance Test Template](#) can be used to capture details about the test result for every iterations.

## Resources

- [Patterns and Practices: Performance Testing Guidance for Web Applications](#)

---

Last update: August 22, 2024