What is Performance Testing?

Performance Testing also knows as "*Perf Testing*", is a type of testing performed to check how application or software performs under workload in terms of responsiveness and stability. The Performance Test goal is to identify and remove performance bottlenecks from an application. This test is mainly performed to check whether the software meets the expected requirements for **application speed, scalability,** and **stability**.

Performance testing is a type of non-functional testing in which the performance of the system is evaluated under expected or higher load. The various performance parameters evaluated during performance testing are:

- Response time,
- Reliability,
- Resource usage,
- Scalability, etc.

A well-performing application is one that lets the end user to carry out a given task without undue perceived delay or irritation.

Why is Performance Testing important?

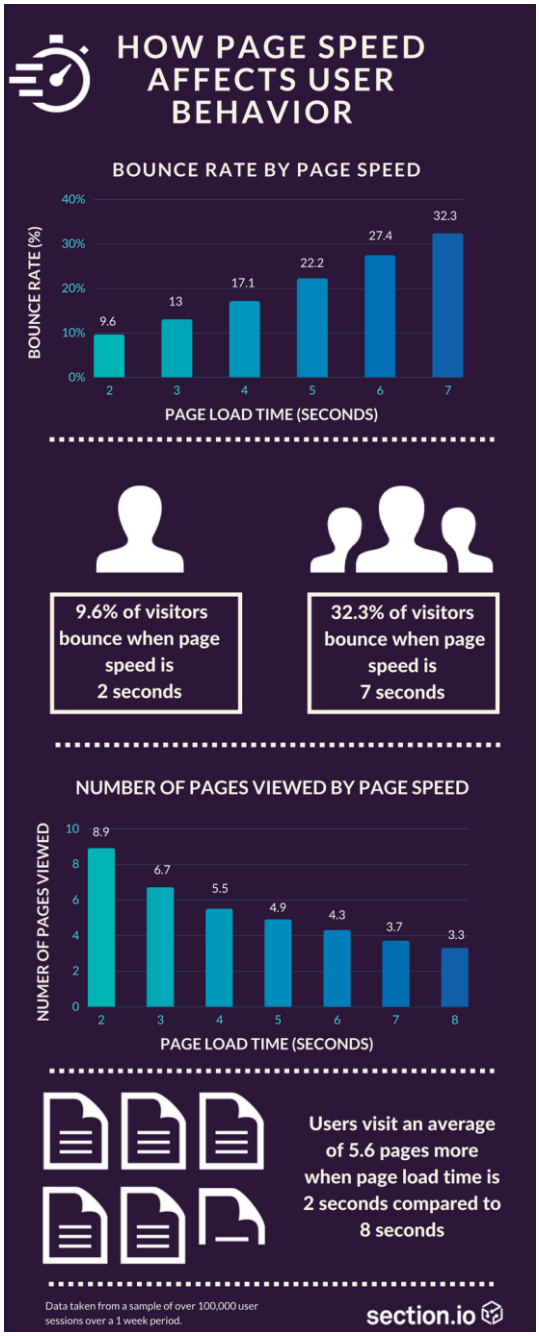
Performance standard doesn't exist, but there were some attempts to define for example a minimum page refresh standard. The original research was based largely on green-screen text applications, but its conclusions are still very relevant. The critical response-time barrier seems to be 2 seconds, which is where the expected application web page response time now sits.

EVERY SECOND COUNTS!!!

Figure XX: Page Abandonment and Page Load Correlation

The following list shows result of the research that attempted to map end-user productivity to response time. Response time should be appropriate to the task.

1. Longer response times (>15 seconds) are disruptive
2. Typing, cursor motion, mouse selection: 50-150 milliseconds
3. Simple frequent tasks: 1 second
4. Common tasks: 2-4 seconds
5. Complex tasks: 8-12 seconds



The next lines are taken from Ian Molyneaux book "*The Art of Application Performance Testing*". Performance problems have a nasty habit of turning up late in the application life cycle, and the later you discover them, the greater the cost and effort to resolve will be. Figure 2 illustrates that point. The solid line (planned) indicates the expected outcome when the carefully factored process of developing an application comes to fruition at the planned moment (black diamond). The application is deployed successfully on schedule and immediately starts to provide benefit to the business with little or no performance problems after deployment.

The broken line (actual) demonstrates the all-too-frequent reality when development and deployment targets slip (striped diamond) and significant time and cost is involved in trying to fix performance issues in production. This is bad news for the business because the application fails to deliver the expected benefit. "

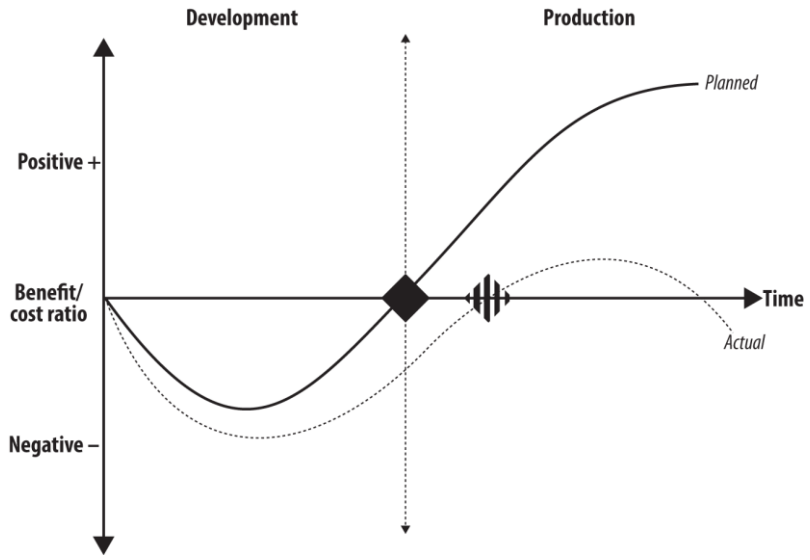


Figure XX: The IT business value curve [*The Art of Application Performance Testing*].

If you don't take performance considerations into account during application design, you are asking for trouble. A "performance by design" mindset lends itself to good performance, or at least the agility to change or reconfigure an application to cope with unexpected performance challenges. Design-related performance problems that remain undetected until late in the life cycle can be difficult to overcome completely and doing so is sometimes impossible without significant (and costly) application reworking. Most applications are built from software components that can be tested individually and may perform well in isolation, but it is equally important to consider the application as a whole. These components must interact in an efficient and scalable manner in order to achieve good performance.

A system's performance can be analyzed along three dimensions:

- Is it fast? (enough) — Given a level and pattern of load, are latency and throughput within an acceptable threshold?
- Does it scale? (how)— What effect will adding more servers (horizontal scaling) or getting more powerful hardware (vertical scaling) have on latency and throughput?
- Is it reliable? (enough) — At what amount of extra load (over projected levels) will the system degrade to the point of being unusable?

Some Important considerations during application design phase are:

- How many end users will the application need to support at release? After 6 months, 12 months, 2 years?
- Where will these users be located, and how will they connect to the application?
- How many of these users will be concurrent at release? After 6 months, 12 months, 2 years?

These answers then lead to other questions, such as the following:

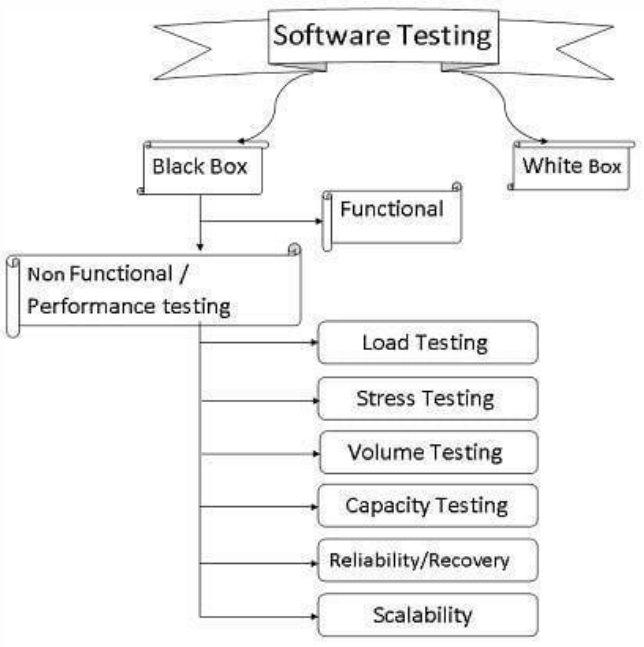
- How many and what specification of servers will I need for each application tier?
- Where these servers should be hosted?
- What sort of network infrastructure do I need to provide?

Examples of performance problems in production that could be avoided

Popular store could not handle the increased traffic generated by their advertising campaign resulting in loss of both marketing investment, and potential sales.

Encyclopedia Britannica declared free access to their online database as a promotional offer. They were not able to keep up with the onslaught of traffic for weeks.

Types of Performance Testing



Types of Performance Testing

Performance test classification is important, but the most important above any classification is to **USE YOUR IMAGINATION!!!** It is important to test expected system behavior, but equally important is to predict system usage that is not so obvious at first sight, such as dealing with requests that have body larger than usual, for example 1MB or even more. Then, how does the application deal with huge

number of malformed requests or unauthorized requests? Beside virtual clients, use at least one real client during load test, and check if major features are working as expected.

Load Testing

Load Testing is a type of performance test where the application is tested for its performance on normal and peak usage. The performance of an application is checked with respect to its response to the user request and its ability to respond consistently within an accepted tolerance on different user loads.

The key considerations are:

1. What is the maximum load the application is able to hold before the application starts behaving unexpectedly?
2. How much data the Database able to handle before the system slows or the crash is observed?
3. Are there any network related issues to be addressed?

Stress Testing

Stress Testing is used to find ways to break the system. The test also provides the range of maximum load the system can hold. Generally, Stress Testing has an incremental approach where the load is increased gradually. The test is started with a load for which the application has already been tested. Then, more load is added slowly to stress the system. The point at which we start seeing servers not responding to the requests is considered the breaking point.

It attempts to cause the application or some part of the supporting infrastructure to fail. The rationale for stress testing is that if our target concurrency is 1 000 users, but the infrastructure fails at only 1 005 users, then this is worth knowing because it clearly demonstrates that there is very little extra capacity available. The results of stress testing provide a measure of capacity as much as performance. It's important to know your upper limits, particularly if future growth of application traffic is hard to predict. For example, the scenario just described would be disastrous for something like an airport air-traffic control system, where downtime is not an option. Most prominent use of stress test is to determine the limit, at which the system or software or hardware breaks.

The following questions are to be addressed:

- What is the maximum load a system can sustain before it breaks down?
- How is the system break down?
- Is the system able to recover once it's crashed?
- In how many ways a system can break and which are the weak node while handling the unexpected load?

Volume Testing

Volume Testing is to verify that the performance of the application is not affected by the volume of data that is being handled by the application. In order to execute a Volume Test, a huge volume of data is entered into the database. This test can be an incremental or steady test. In the incremental test, the volume of data is increased gradually.

Generally, with the application usage, the database size grows, and it is necessary to test the application against a heavy database. A good example of this could be a website of a new school or a college having small amounts of data to store initially, but after 5-10 years, the data stores in the database of the website is much more.

Capacity Testing

Is the application capable of meeting business volume under both normal and peak load conditions?

Capacity Testing is generally done for future prospects. Capacity Testing addresses the following:

1. Will the application be able to support the future load?
2. Is the environment capable of standing for the upcoming increased load?
3. What are the additional resources required to make the environment capable enough?

Capacity Testing is used to determine how many users and/or transactions a given web application will support and still meet performance. During this testing, resources such as processor capacity, network bandwidth, memory usage, disk capacity, etc. are considered and altered to meet the goal.

Online Banking is a perfect example of where capacity testing could play a major role.

Reliability/Recovery Testing

Reliability Testing or Recovery Testing – is to verify whether or not the application is able to return back to its normal state after a failure or abnormal behavior and how long does it take for it to do so (in other words, time estimation).

If an online trading site experiences a failure where the users are not able to buy/sell shares at a certain point of the day (peak hours) but are able to do so after an hour or two, we can say the application is reliable or recovered from the abnormal behavior.

Performance testing metrics

In the context of load testing, key performance indicators (KPIs) show user and traffic measurements for websites and applications, in order to best determine if they can withstand a certain amount of load to its backend servers. You need to identify the key performance indicators that should be monitored for your system. This information is vital to achieve the accurate root-cause analysis of any problem that may occur during performance test execution.

Here are some of the KPIs for load testing.

- Response KPIs
- Volume Measurements
- Server KPIs

Response KPIs

Response KIPs are:

- Connect Time
- Latency or Time to First Byte (TTFB)
- Response Time or Time to Last Byte (TTLB)

Connect time is the measurement of how long it takes the user to connect to the server, and the server to respond, including SSL handshake.

Latency is the time from sending the request, processing it on the server side, to the time the client received the first byte.

Response time is the amount of time from the first byte sent to the server to last byte received at the client side.

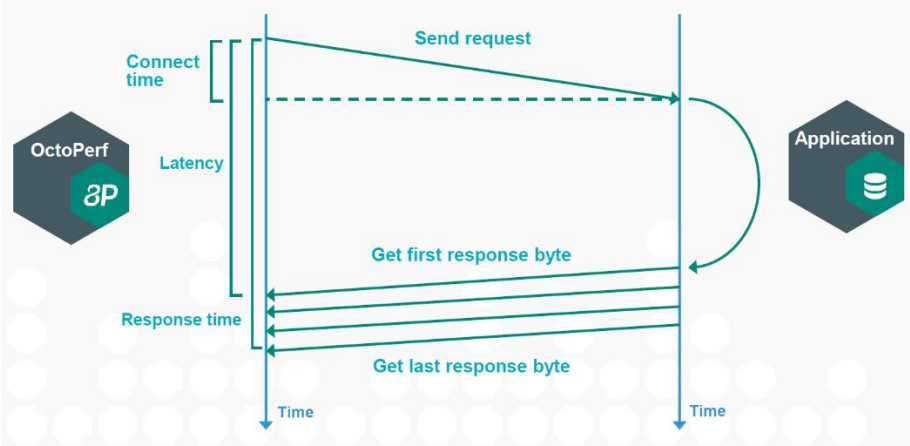


Figure X: Latency and Response Time

The ratio between the TTFB and TTLB is critical. This is also known as “how fast can you move a chunk of data from the server to the client” (HFCYMACODFTSTTC). If the time from TTFB to TTLB is short, your connection is good.

Beside **JMeter** that records latency and response time for each request sent during the load test, you can also use several tools available online, as Google Chrome and Google Analytics instruments. Once you get the TTFB value, you can evaluate the web server reactivity using the fallowing guidelines:

- 0,1-0,2 seconds: very good
- 0,3-0,5 seconds: good
- 0,6-0,9 seconds: average
- 1-1,5 seconds: above the average
- 1,6 seconds or slower: very bad

The key thing is that as an engineer, you can (and should) control the TTFB. You can optimize your server with Varnish (caching HTTP reverse proxy) and Redis (database cache) to make both static content and dynamic content load faster.

Volume measurements KPIs

Volume measurements KPIs are:

- Concurrent active users
- Hits per Second
- Errors per second
- Bytes per second

Concurrent active users is used to make sure websites can handle heavy loads created by the usage of many people, we simulate concurrent active users through virtual users. These virtual users act like "real" users on the website. By simulating real users, developers, DevOps and QA engineers can discover bottlenecks that could occur during real-time daily use or traffic spikes.

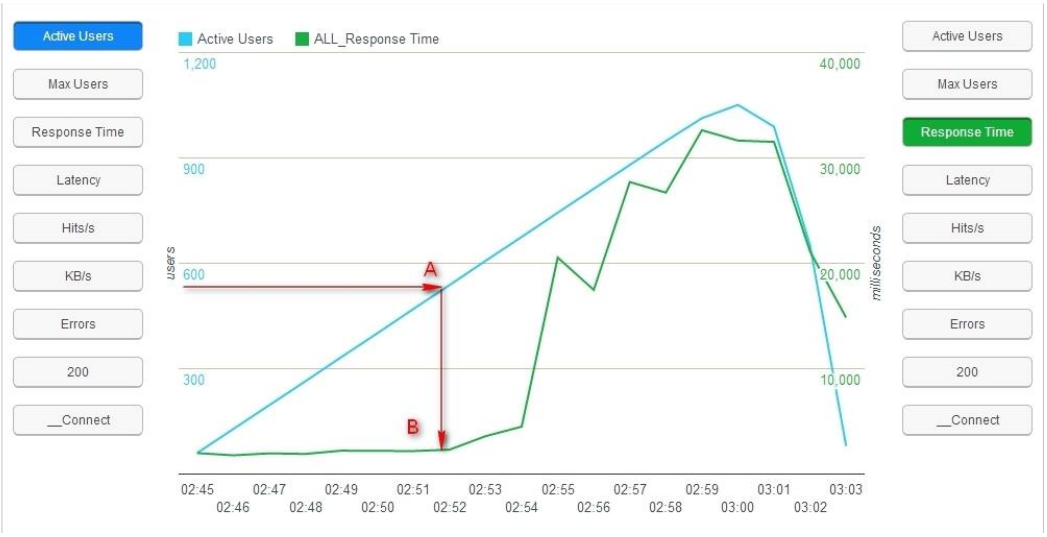
Hits per second KPI represents how many requests are generated to the target server from users' actions. Simulating the number of users can give us a good idea of the load, but it's not by itself an accurate enough measurement. By understanding the correlation between the Number of Users and Hits per Second, we can accurately simulate and measure the types and loads of usage of the website, thus validating that the site's performance abilities are adequate.

Errors per second is self-explanatory. A high Errors per Second rate can indicate a bottleneck that needs fixing before going live.

Bytes per second measures the amount of data flowing to and from the servers. This measurement shows the average bandwidth consumption, that is generated by the load test, per second. Isolating this KPI is important for ensuring your network interface controllers are performing properly.

Example 1: Link

We will now analyze one load testing report based on response time, latency, requests/s, number of active (concurrent) users and errors.

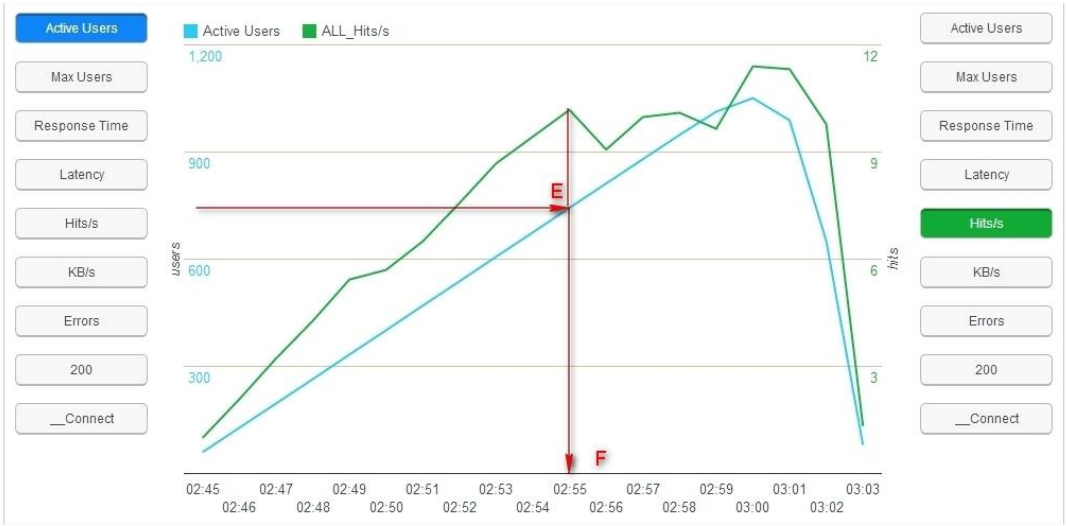


A) Active users vs response time during toad test

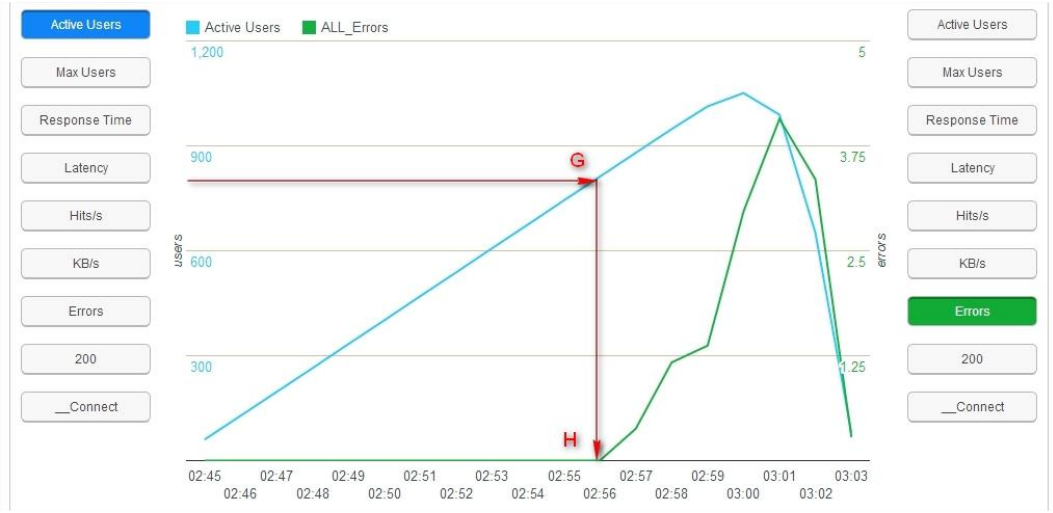
B) Active users vs latency during load test

1. From figure A, we can see that until minute 02:52 response time was stable, but at that point it starts to grow. This means that the site that was undergoing stress testing was able to handle about 580 users without issue. Then we have response time that grows up to ~32s. 2.

2. In the figure B, the Latency graph has the same exact base points as response time and it grows up to ~20s. Let's recall now, latency definition, **Latency** is the time from sending the request, processing it on the server side, to the time the client received the **first byte**. Latency = Connect time + server processing + time travel of first bite. Since difference between latency and response time is "10s, we can conclude that the rest of the bytes were traveling ~10s, which can indicate that we definitely have a network issue.



C) Active users vs Hits per Second



D) Active users vs errors

3. From Figures C and D, we can see what is the critical point of hits (requests)/s and concurrent users that are generating those requests when we start to get errors.

4. Next step would be to go to network monitoring tool and check for amount of traffic that has been sent and received and for network errors. Possible solution would be to increase network bandwidth and repeat the test.

Server KPIs

Some of the most common used server KPIs are shown on these blogs: blog and blog2.

We won't get into details on each of these parameters, but we will give some explanation on CPU usage, CPU run queue and context switching, because understanding how well a CPU is performing is a matter of interpreting these three parameters. In my practice, I interpret CPU performance based on CPU usage and CPU run queue and if necessary, I check context switching.

CPU utilization

CPU utilization is defined as the percentage of usage of a CPU. How a CPU is utilized is an important metric for measuring system. Most performance monitoring tools categorize CPU utilization into the following categories:

- **User Time** - The percentage of time a CPU spends executing process threads in the user space.
- **System Time** - The percentage of time the CPU spends executing kernel threads and interrupts.
- **Wait IO** - The percentage of time a CPU spends idle because all process threads are blacked waiting for 1O requests to complete.

CPU utilization should be max ~70%, that way we have spare space, for potential unexpected CPL usage peaks. If you have system with predictable traffic (your product is based on licenses and you have information that there will be no new customers in near future), then CPU usage above 70%, even close to 100% is tolerable, as long as CPU run queue value is close to the number of CPU cares on the machine. If a CPU is fully utilized, then the following balance of utilization should be achieved.

- 65% - 70% User Time
- 30% - 35% System Time
- 0% - 5% Idle Time

The Run Queue

Each CPU maintains a run queue of threads. Ideally, the scheduler should be constantly running and executing threads. Process threads are either in a sleep state (blocked and waiting on IO) or they are runnable. If the CPU sub-system is heavily utilized, then it is possible that the kernel scheduler can't keep up with the demand of the system. As a result, runnable processes start to fill up a run queue. The larger the run queue, the longer it will take for process threads to execute. A very popular term called "load" is often used to describe the state of the run queue. The system load is a combination of the amount of process threads currently executing along with the amount of threads in the CPU run queue. If two threads were executing on a dual core system and 4 were in the run queue, then the load would be 6. On a multicore system, your load should not exceed the number of cores available. Utilities such as top report load averages over the course of 1, 5, and 15 minutes. Command *vmstat* gives run queue values in real time ("*vmstat* 1" shows run queue values on 1s).

Context switching

Each thread has an allotted time quantum to spend on the processor. Once a thread has either passed the time quantum or has been preempted by something with a higher priority (a hardware interrupt, for example), that thread is place back into a queue while the higher priority thread is placed on the processor. This switching of threads is referred to as a context switch. Every time the kernel conducts a context switch, resources are devoted to moving that thread off of the CPU registers and into a queue. The higher the volume of context switches on a system, the more work the kernel has to do in order to manage the scheduling of processes.

Performance Test Process

Here are all the activities performed in this testing:

#1) Requirement Analysis/Gathering

The performance team interacts with the client for identification and gathering of requirements – technical and business. This includes getting information on the application's architecture, technologies, and database used, intended users, functionality, application usage, test requirement, hardware & software requirements, etc.

#2) POC/Tool selection

Once the key functionality is identified, POC (Proof of Concept – which is a sort of demonstration of the real-time activity but in a limited sense) is done with the available tools.

The list of available tools depends on the cost of the tool, protocol that application is using, the technologies used to build the application, the number of users we are simulating for the test, etc. During POC, scripts are created for the identified key functionality and executed with 10-15 virtual users.

#3) Performance Test Plan & Design

Depending on the information collected in the preceding stages, test planning and designing are conducted.

Test Planning involves information on how the performance test is going to take place – test environment, workload, hardware, etc.

#4) Performance Test Development

- Use cases are created for the functionality identified in the test plan as the scope of PT.
- These use cases are shared with the client for their approval. This is to make sure the script will be recorded with the correct steps.
- Once approved, script development starts with a recording of the steps in use cases with the performance test tool selected during the POC (Proof of Concepts) and enhanced by performing Correlation (for handling dynamic value), Parameterization (value substitution) and custom functions as per the situation or need. More on these techniques in our video tutorials.
- The Scripts are then validated against different users.
- Parallel to script creation, the performance team also keeps working on setting up the test environment (Software and hardware).
- The performance team will also take care of Metadata (back-end) through scripts if this activity is not taken up by the client.

#5) Performance Test Modeling

Performance Load Model is created for the test execution. The main aim of this step is to validate whether the given Performance metrics (provided by clients) are achieved during the test or not. There are different approaches to create a Load model. “Little’s Law” is used in most cases.

#6) Test Execution

The scenario is designed according to the Load Model in Controller or Performance Center, but the initial tests are not executed with maximum users that are in the Load model.

Test Execution is done incrementally. **For example,** if the maximum number of users is 100, the scenarios are first run with 10, 25, 50 users and so on, eventually moving on to 100 users.

#7) Test Results Analysis

Test results are the most important deliverable for the performance tester. This is where we can prove the ROI (Return on Investment) and productivity that a performance testing effort can provide.

Some of the best practices that help the Result Analysis process:

1. A unique and meaningful name to every test result – this helps in understanding the purpose of the test.
2. Include the following information in the test result summary:
    - Reason for the failure/s
    - Change in the performance of the application compared to the previous test run
    - Changes made in the test from the point of application build or test environment.
    - It's a good practice to make a result summary after each test run so that analysis results are not compiled every time test results are referred.
    - PT generally requires many test runs to reach the correct conclusion.
    - It is good to have the following points in result summary:
        - Purpose of test
        - Number of virtual users
        - Scenario summary
        - Duration of test
        - Throughput
        - Graphs
        - Graphs comparison
        - Response Time
        - Error occurred
        - Recommendations

#8) Report

Test results should be simplified so the conclusion is clearer and should not need any derivation. Development Team needs more information on analysis, comparison of results, and details of how the results were obtained.

The test report is considered to be good if it is brief, descriptive and to the point.