Load Testing

What is Load Testing?

Load Testing is a type of Performance Testing that determines the performance of a system, software product, or software application under real-life based load conditions. Basically, load testing determines the behavior of the application when multiple users use it at the same time. The goal of Load Testing is to improve performance bottlenecks and to ensure stability and smooth functioning of software applications before deployment.

Why is it Important?

The significance of load testing is explained in greater depth further down.

- **Load testing simulates real user scenarios:** When testing your website, app, or API endpoint under a load, we are actually simulating how it will perform when hundreds, thousands, or millions of users visit it in real life. Real people who will use the system or product should not be neglected. Understand, analyze, and fix errors, bugs, and bottlenecks before they happen.

- **The system performs differently under a load:** KPIs like response time, error rate, memory leakage, and CPU consumption might be top-notch when running functional tests, but when scaling to thousands of users from all over the globe, they could suddenly plummet and require-dev attention. A load test is required to learn where and when the system breaks, fix the problems, and avoid upset users and revenue loss.

- **Code can change the product in unexpected ways:** Let's assume a responsible developer who actually tested the system two months ago. The results showed that most of the API endpoints and services worked. Some small bottlenecks were detected, but they can be fixed. Some code changes were made since then; in fact, new versions are likely released. These modifications may have affected your system in unexpected ways. To avoid a crash or slow responsiveness, an automated load test should be run as part continuous integration cycle.

- **It helps to be fully agile:** Faster releases, lighter versions, and more automation create a better product and make for a better working environment for developers and engineers. Load testing with every commit is an inherent part of any continuous integration process, alongside issue management, code analysis, and other lifecycle activities.

- **Users are impatient and unforgiving:** Every time an important website crashes, headlines announce the blunder and angry users take to social media and condemn the website's owners for not taking proper measures to avoid the crash. Unsatisfied customers have a good memory and there is a limit to the number of times they will try to revisit a slow or unreliable website. Instead of desperately trying to get the

website up while under pressure, dealing with frantic PR managers, and investing money in re-branding, just load tests on time.

- **Load testing saves money:** Teaching developers to set up and automate load tests will take up some resources, but it will cost more to constantly fix website crashes and restore the brand reputation after scores of unhappy customers are fed up and move on to the competitors.

- **Big events are too important to disregard:** Think about major events like Black Friday and Cyber Monday, big sports days, music festivals, flash sales, all the many other exciting days that ignite a massive crowd's interest. Those events result in enormous traffic spikes and thus require special load testing beforehand, above and beyond regular load testing.

Types of Load Testing:

It can be noticed that the website/application and infrastructure behave differently depending on how to configure the load tests. That signifies that conducting a variety of tests provides a variety of results, and more data is generally always a good thing.

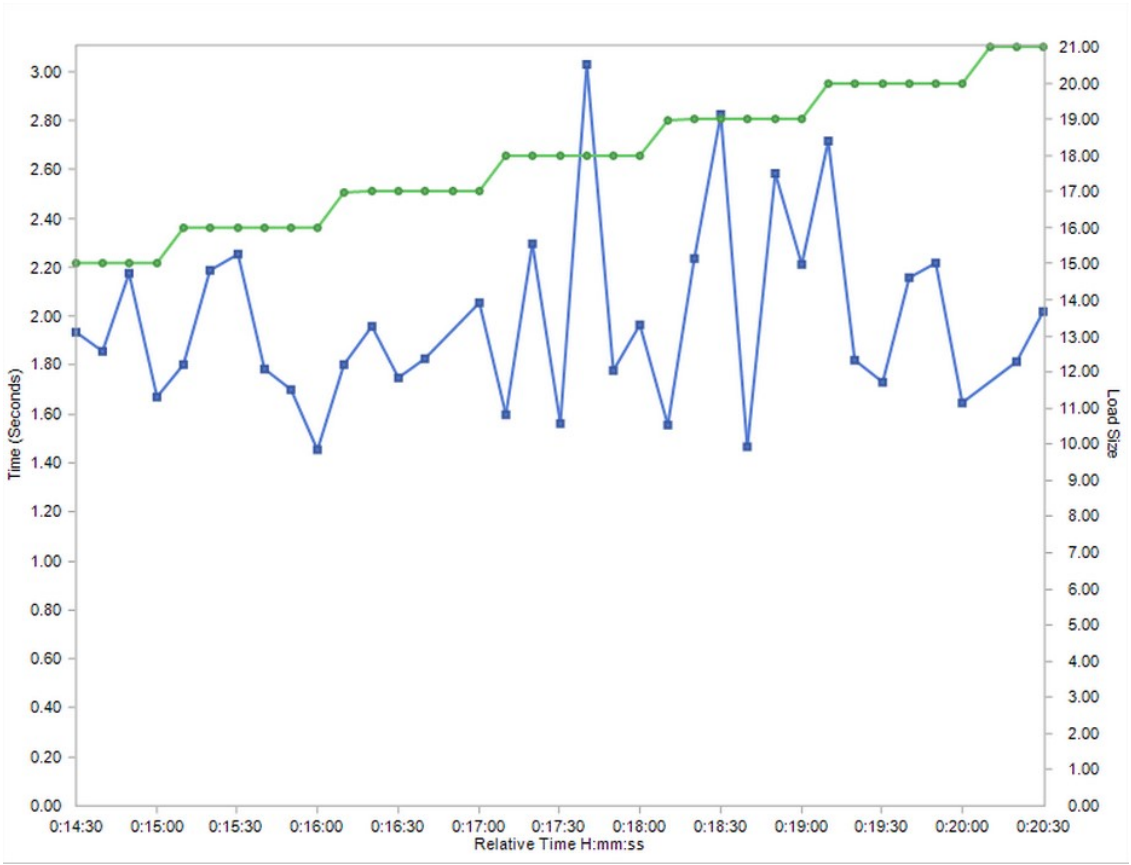
Here are four distinct load test options that will teach about web properties and provide the information that is needed to enhance the performance.

1. Load Testing
2. Capacity Testing
3. Stress Testing
4. Soak Testing

1. Load Testing

As predicted, this is every load tester's bread and butter, where you test how a system operates with a high number of users and what the response time is for different situations. You probably already have a good idea of how much traffic you get regularly, and you're doing load testing to ensure that you're meeting performance metrics as you add new features to your product.

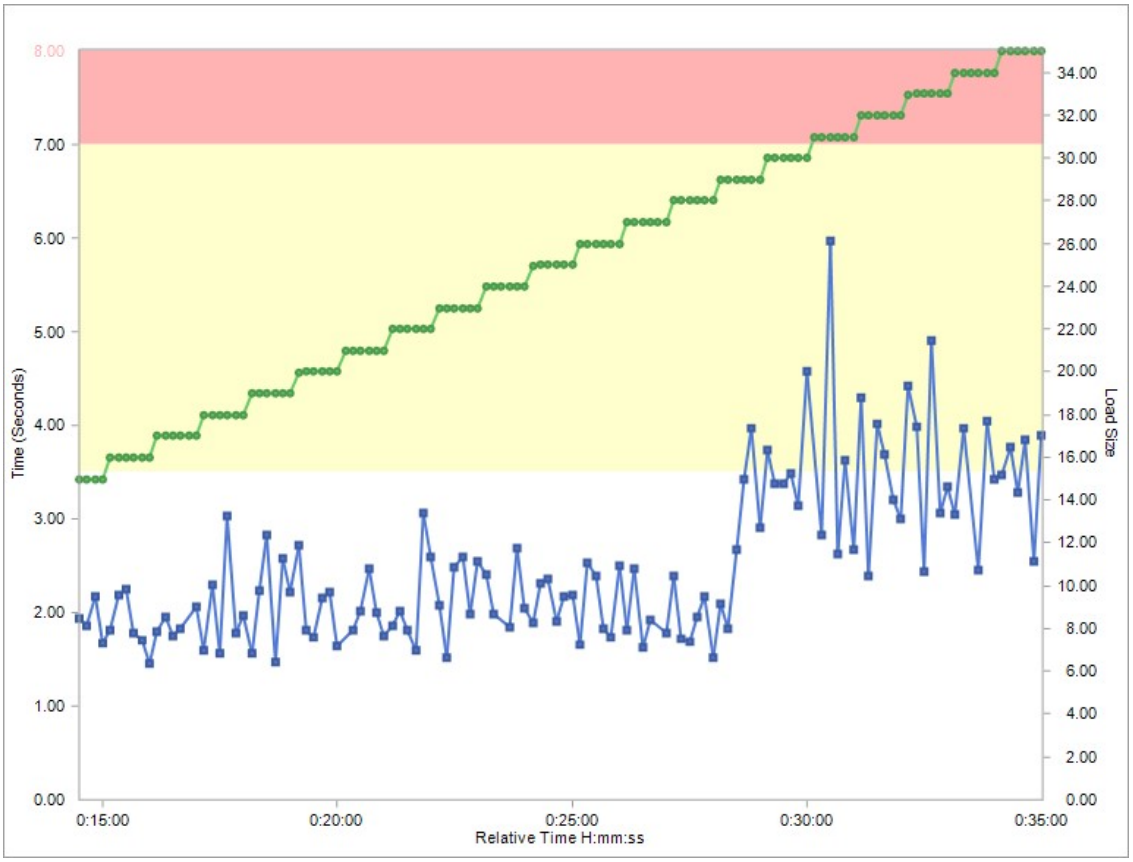
For example, in the graph below, you're running a load test of 20 users to see that the page time does not exceed 3.5 seconds.



2. Capacity Testing

Capacity testing (sometimes called scalability testing) helps you identify the maximum capacity of users the system can support, while not exceeding the max page time you defined. Your higher-level objective is to identify the 'safety zone' of your system. To what extent can you 'stretch' it, without hurting the end-user experience?

The previous load testing example showed that the system easily served 20 users with a page time of 3.5. Now you want to find out the capacity of your system – or at which point it will not be able to keep the 3.5 seconds response time? Will it be at 21 users, 30, 40, or 50 users?



In the results graph that follows you'll notice that for a page time of 3.5 seconds, the system supports 28 users. But when the load size reaches 29, page time starts exceeding the threshold of 3.5 seconds.

3. Stress Testing

The objective of a stress test is to find how a system behaves in extreme conditions. You purposely try to break your system, using any set of extreme conditions – whether doubling the number of users, using a database server with much less memory, or a server with a weaker CPU.

What you're trying to find out is how will the system behave under stress and what will be the user experience? Will the system start throwing out errors? Will response time double? Or will the entire system get stuck and crash?

To continue with the previous example, instead of stopping at around 30 users (when the page time exceeds your goal), you'd continue to increase the user load on the system. Your stress test discovers that up to the load size of 41 users, the system functions, despite the increase in page time. But when the load size reaches 42 users, the system starts to deteriorate, with page time reaching 15-17 seconds.

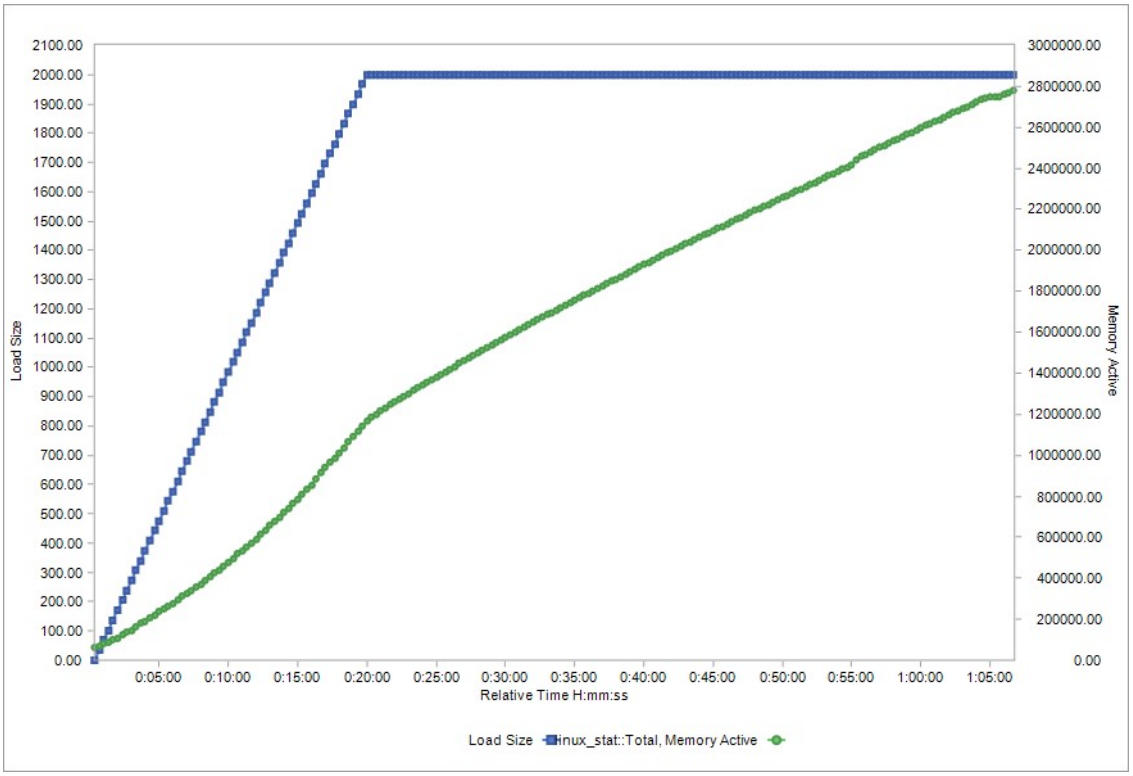
4. Soak Testing

Quite often, the 'standard' load testing, which is run for a short limited time will not succeed in uncovering all problems. A production system typically runs for days, weeks, and months. For example, an eCommerce application must be available 24×7 and a stock exchange application will run continuously on workdays. The duration of a soak test should have some correlation to the operational mode of the system under test.

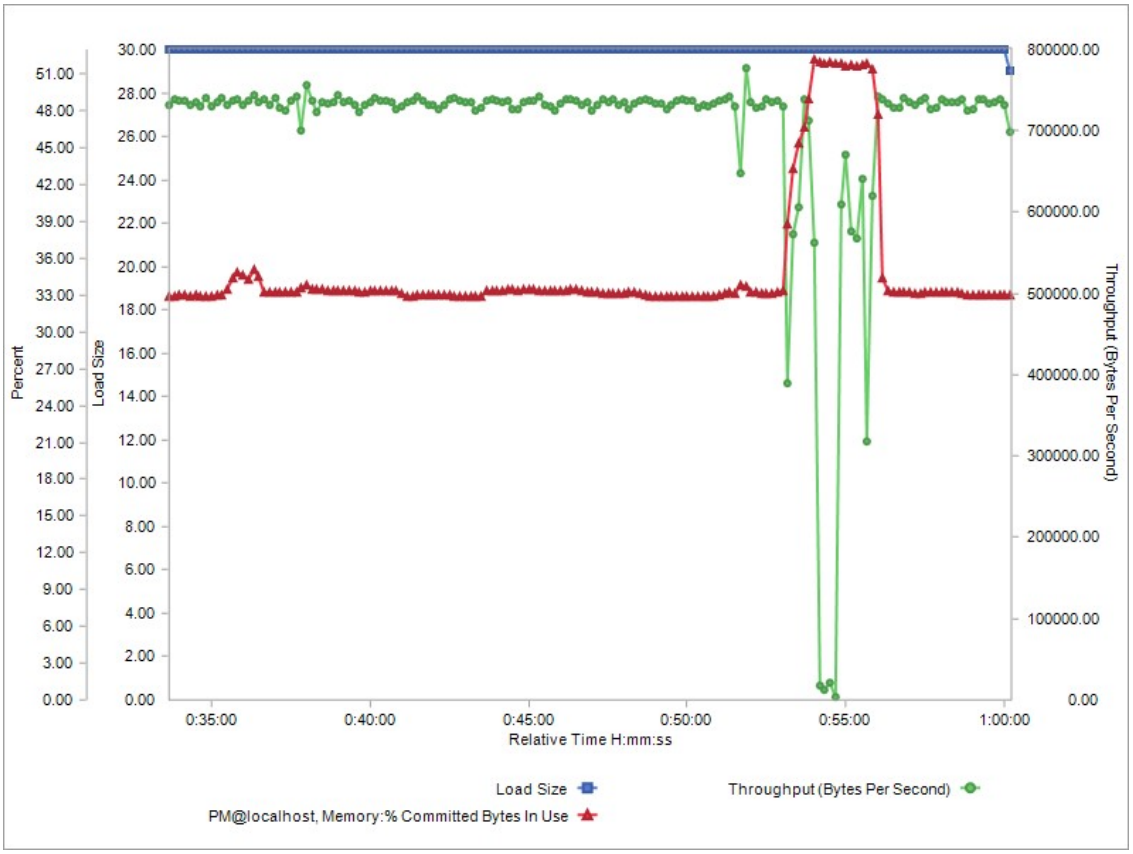
Soak testing aims to uncover performance problems that surface only during a long duration of time. During soak testing, you're asking questions such as:

- Is there a constant degradation in response time when the system is run overtime?
- Are there any degradations in system resources that are not apparent during short runs, but will surface when a test is run for a longer time? For example, memory consumption, free disk space, machine handles, etc.
- Is there any periodical process that may affect the performance of the system, but which can only be detected during a prolonged system run? For example, a backup process that runs once a day, exporting of data into a 3rd party system, etc.

When running soak testing, you're looking for trends and changes in system behavior. In the following example, notice how the load size is constant for a while, but still, there is a memory leak (green line). At first, this memory leak may not affect the system, but after a while, the system may crash. (The example below is shorter than a typical soak test, which would last hours, days, and even weeks.)



Another use case, seen below, is a constant load size, but then, after 50 minutes of running the test, there's an increase in memory (red line), which hurts throughput. This is due to a periodical, background process that affects the system when it starts running. (Again, in a real soak test, this would take much more time).



If there's one lesson from these four types of load testing, it is this: 'performance engineers must dedicate significant thought as to what are the objectives of each test run what are their expectations from the system under test. Before you invest in building complex, time-consuming scenarios, make sure you have clear goals for your tests.'

Is there any relation between the SDLC stage and Load testing?

Testing is an essential part of the software development life cycle (SDLC), since it helps to enhance the system's quality, reliability, and performance by checking what all functions the program is expected to do and making sure it isn't doing anything it isn't supposed to do. Similarly, the act of simulating demand on software, an application, or a website in order to evaluate or illustrate its performance under varied situations is known as load testing.

From the perspective of assuring performance, it is obvious that the SDLC testing phase and load testing both have identical goals. As a result, we may conclude that they are closely connected.

How do you perform Load testing?

The load testing process can be briefly described as below –

1) Identify the Load test Acceptance Criteria

For Example:

1. The response time of the Login page shouldn't be more than 5 sec even during the max load conditions.
2. CPU utilization should not be more than 80%.
3. The throughput of the system should be 100 transactions per sec.

2) Identify the Business scenarios that need to be tested.

Don't test all the flows, try to understand the main business flows which are expected to happen in production. If it is an existing application we can get his information from the server logs of the production environment.

If it is a newly built application then we need to work with the business teams to understand the flow patterns, application usage, etc. Sometimes the project team will conduct workshops to give an overview or details about each component of the application.

We need to attend the application workshop and note all the required information to conduct our load test.

3) Workload Modelling

Once we have the details about the business flows, user access patterns and the number of users, we need to design the workload in such a way in which it mimics the actual user navigation in production or as expected to be in the future once the application will be in production.

The key point to remember while designing a workload model is to see how much time a particular business flow will take to complete. Here we need to assign the think time in such a way that the user will navigate across the application in a more realistic way.

The Workload Pattern will usually be with a Ramp up, Ramp down, and a steady-state. We should slowly load the system and thus ramp up and ramp down are used. The steady-state will usually be a one-hour Load test with a Ramp-up of 15 min and a Ram down of 15 min.

Let us take an Example of the Workload Model:

Overview of the application – Let's assume online shopping, where the users will log into the application and have a wide variety of dresses to shop, and they can navigate across each product.

To view the details about each product, they need to click on the product. If they like the cost and make of the product, then they can add to the cart and buy the product by checking out and making the payment.

Given below is a list of scenarios:

1. **Browse** – Here, the user launches the application, Logs into the application, Browses through different categories, and Logs out of the application.
2. **Browse, Product View, Add to Cart** – Here, the user logs into the application, Browses through different categories, views product details, adds the product to the cart, and Logs out.
3. **Browse, Product View, Add to Cart and Check out** – In this scenario, the user logs into the application, Browses through different categories, views product details, adds the product to the cart, does check out, and Logs out.
4. **Browse, Product view, Add to cart Check out and Makes Payment** – Here, the user logs into the application, Browses through different categories, views product details, adds the product to the cart, does check out, makes Payment and Logs out.

S.No	Business Flow	Number of Transactions	Virtual User Load	Response Time (sec)	% Failure rate allowed	Transactions per hour
1	Browse	17	1600	3	Less than 2%	96000
2	Browse, Product View, Add to Cart	17	200	3	Less than 2%	12000

3	Browse, Product View, Add to Cart and Check out	18	120	3	Less than 2%	7200
4	Browse, Product view, Add to cart Check out and Makes Payment	20	80	3	Less than 2%	4800

The above values were derived based on the following calculations:

- Transactions per hour = Number of users*Transactions made by a single user in one hour.
- The number of users = 1600.
- The total number of transactions in the Browse scenario = 17.
- Response Time for each transaction = 3.
- Total time for a single user to complete 17 transactions = 17*3 = 51 rounded to 60 sec (1 min).
- Transactions per hour = 1600*60 = 96000 Transactions.

4) Design the Load Tests

The Load Test should be designed with the data that we collected so far i.e the Business flows, Number of users, user patterns, Metrics to be collected and analyzed. Moreover, the tests should be designed in a much more realistic way.

5) Execute Load Test

Before we execute the Load test, make sure that the application is up and running. The Load test environment is ready. The application is functionally tested and is stable.

Check the configuration settings of the load test environment. It should be the same as the production environment. Ensure all the test data is available. Make sure to add the necessary counters to monitor the system performance during test execution.

Always start with a low load and gradually increase the load. Never start with the full load and break the system.

6) Analyse the Load Test Results

Have a baseline test to always compare with the other test runs. Gather the metrics and server logs after the test run to find the bottlenecks.

Some projects use Application Performance Monitoring Tools to monitor the system during the test run, these APM tools help to identify the root cause more easily and save a lot of time. These tools are very easy to find the root cause of the bottleneck as they have a wide view to pinpoint where the issue is.

Some of the APM tools in the market include DynaTrace, Wily Introscope, App Dynamics, etc.

7) Reporting

Once the Test Run is complete, gather all the metrics and send the test summary report to the concerned team with your observations and recommendations.

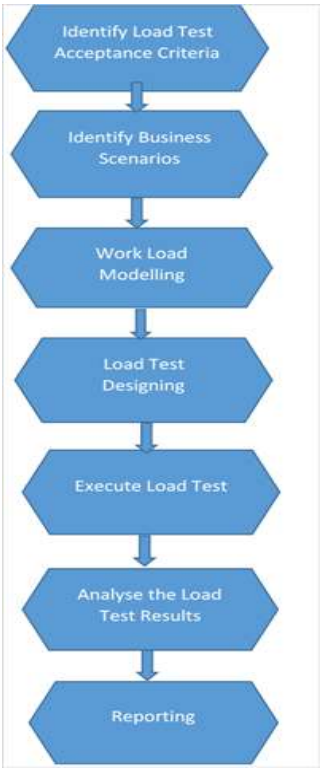


Figure: Execution steps of load testing

Automated tools to perform Load Testing

There are some automated tools available to perform Load Testing. A list of some of the automated tools is given below:

- **NeoLoad:** NeoLoad provides testers and developers automatic test design and maintenance, the most realistic simulation of user behavior, fast root cause analysis, and built-in integrations with the entire SDLC toolchain.
- **LoadComplete:** LoadComplete enables the creation and execution of realistic load tests for websites and web apps. It automates creating realistic load tests by recording user interactions and simulating these actions with hundreds of virtual users either from your local computers or from the cloud.
- **Loadster:** Loadster is a desktop-based advanced HTTP load testing tool. The web browser can be used to record scripts that are easy to use and record.
- **LoadView:** LoadView is a fully managed, on-demand load testing tool that allows complete hassle-free load and stress testing.
- **LoadNinja:** LoadNinja by SmartBear allows you to quickly create scriptless sophisticated load tests, reduces testing time by 50%, replaces load emulators with real browsers, and gets actionable, browser-based metrics, all at ninja speed.
- **LoadRunner:** This is a Micro Focus product that can be used as a Performance Testing tool. It can create and handle thousands of users at the same time.
- **WebLOAD:** WebLOAD is a tool of choice for enterprises with heavy user load and complex testing requirements. It allows you to perform load and stress testing on any internet application by generating load from the cloud and on-premises machines.
- **JMeter:** It is a Java platform application. It is mainly considered as a performance testing tool that can also be integrated with the test plan.
- **Locust:** Locust is a simple-to-use, distributed, user load testing tool that can help capture response times.
- **nGrinder:** It was developed to make stress testing easy and to provide a platform that allows you to create, execute, and monitor tests.
- **StormForge:** It is the only platform that combines performance testing with machine-learning-powered optimization which allows users to both understand the performance and automatically identify the ideal configurations of the application for performance and resource utilization.
- **The Grinder:** The Grinder is a Java-based framework. It provides users with easy-to-run and creates distributed testing solutions using many load generator machines to capture your end-users response times.
- **Gatling:** It allows to test and measure the application's end-to-end performance and easily scale up the virtual users.
- **K6:** k6 is a modern open-source load testing tool that provides an outstanding developer experience to test the performance of APIs and websites.
- **Tsung:** Tsung is an open-source, multi-protocol distributed load testing tool that can monitor a client's CPU, memory, and network traffic.

- **Siege:** Siege is a command-line HTTP load testing and benchmarking utility. It was designed to help developers measure their code under stress.
- **Fortio:** Fortio is a load testing library, command-line tool, advanced echo server, and web UI in go (golang). This tool allows to specify a set query-per-second load and record latency histograms and other useful stats.
- **Bees with Machine Guns:** This tool allows load testing on a site that needs to handle high traffic.

Similarities of the Load testing with other Testing

As both stress testing and load testing are the parts of performance testing there are some similarities among the three of them. They all are non-functional testing techniques.

When load testing tries to minimize the risks related to system downtime by maximizing the operating capacity; stress testing tries to minimize it by ensuring it will work appropriately in normal as well as abnormal conditions and case of failures, it will recover as soon as possible. In this sense, they both work on reliability as well as the robustness of the software system.

Strengths and weaknesses of load testing

Being part of the performance testing, load testing is essential to establish a robust system that can ensure effective implementation and successful functioning in real-life scenarios. It helps to shape the experience users receive by defining the parameters under which it performs adequately. With proper strategy, the load testing can consider all potential pitfalls and challenges that can create bottlenecks or hinder performance in a software application. It might seem that load stress has no demerits but that is not the case. The importance and challenges associated with load testing are described below.

Strength of Load Testing:

1. Load testing simulates real user scenarios. Like creating scripts that make different requests to the server, just like real-world users with various configurations.
2. Discovers performance bottlenecks before production deployment. It successfully tests out the system to have the ability to serve the users without any performance degradation.
3. Identifies an ideal infrastructure for deployment. It helps in configuring the most optimal infrastructure for the setup. Infrastructure costs can be saved by terminating extra machines. Also, the additional machine can be added in the case of suboptimal infrastructure.
4. Reduces the risk of the system going down due to an unexpected traffic spike and minimizes the risk of downtime by identifying and isolating the requests whose performance needs to be improved.
5. It provides a sense of confidence and reliability with proof of the application's performance.

Weakness of Load Testing:

1. Not programmed to principally concentrate on speed of response.
2. Load testing can be a costly mistake if done haphazardly, leading to inaccurate results and conclusions.
3. It should be carried out on multiple devices in different locations to check whether a user faces difficulties. Hence this testing can be costly.
4. Many of the load testing tools are licensed and charge a good amount of money for the license.
5. Even in the case of free and open-source tools like JMeter, an environment is required which should be as close to the production environment setup as possible. This again leads to additional resources and costs.
6. Load test script creation requires scripting knowledge of the language supported by the tool.

Opinion on how Load Testing can be more efficient in the future:

Load testing is to test the system by constantly and steadily increasing the load on the system until it reaches the threshold limit. Being part of the performance testing, load testing can be easily done by employing any of the suitable automation tools available in the market. Like JMeter and LoadRunner and many more, famous tools that aid in load testing with the sole purpose of load testing by assigning the SUT the largest job, it can possibly handle to test the endurance of the system and monitor the results and compare. Sometimes, the load testing involves the system being fed with an empty task to simulate the zero-load situation.

Now all possible loads are simulated with automated tools, but the used tools are mostly bought with a cost that cannot be included in the small system budget. The future of automated testing with load will give the new silver lining to budgeted systems to measure and predict their performance in real-life scenarios. So more automated tools can certainly make the load testing process for any system more efficient in terms of cost and time.

And another part to look forward to for load testing's future is related to AI. This might seem odd but AI-related automation to explore the application can predict real-time scenarios. Increase productivity and effectiveness of load testing while properly controlling and measuring software metrics. AI can be used to predict the timeline when the system might go under the overloading threat like the other similar software and test it according to that sort of threshold.

And lastly, self-adaptive performance engineering and infrastructure are related to software systems is a topic that might go beyond any kind of tested system. Smarter remediation actions are based on better data and better automation scripts. The system knows how to deal with problems. We help build more resilient systems that are smart, self-healing, smart scaling, with failover and redundancy.