CHAPTER 5
Performance Test Analysis of Web Applications

5.1 Introduction
Performance testing [39,42] of a Web application is the process of understanding how the Web application and its operating environment respond at various user load levels. In general, we want to measure the latency, throughput, and utilization of the Web site while simulating attempts by virtual users to simultaneously access the application. Often the primary problems that projects report after field release are no system crashes or incorrect system responses, but rather system performance degradation or problems handling required system throughput. For a software system it is very common to go through the extensive functionality testing, but it was never really tested to assess its expected performance. The issues [38,40,47] that have to be addressed when doing performance testing differ in a number of ways from the issues that must be addressed when doing typical functionality testing. In this chapter we look at a representative performance testing approach and arrive at some key aspects that should be considered during performance testing. Also we examine performance-testing issues and describe an approach to address some of these issues for certain classes of software systems.

5.2 Performance Testing Approach
There are several steps involved in performance testing [48] – setting up the test environment, creation of test scripts using a COTS load testing tool, executing the scripts simulating a multi-user scenario and analyzing the test results. The following are the important steps for application performance analysis: (i) Work Load Modelling, (ii) Load test

54
the application by simulating the workload, (iii) Monitor and capture system performance, (iv) Analyze the system performance results to identify bottlenecks and (v) Optimize and Tune the system.

The challenges that a developer faces as a part of step (i) to step (v) are:

- Workload Modeling
- Load generator and simulation of the user behavior?
- Load generator and simulation of the required number of users?
- How do I ensure that my system can support more users?
- User Education

This chapter provides answers to these questions by reflecting on empirical validations of distributed applications and highlighting aspects that should be considered while answering these questions.

5.2.1 Workload Modeling

Workload modeling is performed to capture user behavior patterns, business productivity targets and workload growth. The Workload model identifies the transactions associated with the application, user loads and the concurrent occurrences of these transactions. It captures the variation of these user loads in normal and peak scenarios, different service levels and the workload growth over a period of time. The network, the protocols, the hardware configuration of the existing systems and current levels of utilization are also part of this model. This model describes all factors that have a bearing on performance and capacity. Identifying these parameters is very important as this model forms the basis for all
subsequent infrastructure decisions. The model should also capture the workload distribution on the system during a typical business day.

A Workload Distribution is a representation of the functions performed by a user community on a system. The target performance goals are worked collaboratively between performance team and all key stakeholders through mutual experience, conversations and workshops. Testing and tuning continues until all goals were met. While coming up with the workload pattern and model users are too pessimistic in providing the numbers. Also the think times provided by the users are too pessimistic. End users need education and looking at their historical data realistic workload model needs to be arrived at.

5.2.2 Load generator and simulation of the user behavior?

The performance test performed by using load generators should depict the difference in the load scenario while simulating users. Internet users log on to the site performs transactions and goes away. These users depict an open system where the rate of requests and the transactions performed by them is relevant – arrival rate of user requests and arrival distribution defines the workload on the system. Testing the application with a hundred, two hundred, or even a thousand users will not be able to mimic unlimited users. Here, “workload” refers to the simultaneous workload on the system. The same is empirically validated subsequently.

An online toys store application is implemented for purchase of toys. In addition to facilitating the purchase process, the application also caters to the toy store employees who use it for checking the toys available in the store and updating the inventory. Performance testing should simulate the load generated by both the Internet users and the toy store
employees. There is a difference in the usage of the application by Internet users and the toy store employees.

The toy store employees are limited – three hundred employees will be regularly using the application for short time intervals. On an average, the toy store employee would use the search toy inventory menu every five minutes. The toy store employees depict a closed system where the number of users, and the idle time spent between each user request (known as think time) are relevant for identifying the workload on the system. Hence, the load generators need to cater to the above two user behaviors, requiring to simulate Internet users (open system) by simulating the arrival rate – requests/sec or hits/sec and simulating toy store employees (closed system) by simulating users with some idle time between subsequent user requests. Most load generators simulate load in a manner that mimics a closed system.

The load generators create the required number of users as threads, each thread sends a request to the server, gets a response, and sends a new request again. However, to simulate the load of an open system, the load generator would need to create required number of threads, take the arrival rate as an input, and ensure that the threads keep sending requests at the specified rate without waiting for a response from the server.

While all load generators successfully simulate closed systems, there are few load generators (such as Mercury Interactive's Load Runner) that support open system load generation where the arrival rate can be specified. In these load generators, the arrival distribution of requests cannot be specified. However, they can be used to simulate a constant arrival rate. This will not be a real-life scenario, but is good enough for verifying the performance of the system for a given arrival rate of requests. Hence, the load generator has to be configured for simulating the workload as arrival rate or number of users, based on the user behavior.
5.2.3 Is the load generator generating sufficient workload?

The load generator tool should not become a bottleneck by not being able to simulate the number of users or the arrival rate as required. Such possibilities do occur if either the load generator or the system resources (such as CPU and memory) have some limitations that deter generation of the required workload. Verifying the arrival rate for an open system is straightforward. The throughput (transactions completed per second) of the system should be equal to the arrival rate. This would mean that if the load generator is simulating an arrival rate of 30 requests/sec, the measured throughput of the system should be 30 requests/sec.

The throughput of the SUT will be less than the arrival rate only when the SUT has reached saturation – the resources of the system (CPU, Memory, Disk) have reached 100% utilization. Verifying the number of users for a closed system can be done using Little’s law. The response time and throughput measured during the performance test and the think time incorporated within test scripts can be used to arrive at the number of users being simulated by the load generator. If the number of users arrived by using Little’s law matches with the number of users configured in the load generator with a deviation of <10%, the testing is valid. If the results do not match, there is a bottleneck in the load generator that is preventing it from generating the required load.

5.2.4 Will the System Support Required Number of Users?

Little’s law helps in finding the number of users that need to be simulated using the load generator. This law relates think time and the number of users. Response time and throughput being the same, the number of users of the system would be proportional to the think time – inferring that the performance of the system being the same, the number of users that the
system can support is high if the idle time (think time) of the users between submissions of requests is high. The same is validated empirically:

A retail company has a thousand stores all over the country and each store has ten cashiers. A cashier enters all items purchased by a customer and hits the Enter button on the cash register. The order request comes to the retail application that is centrally located at XYZ. The performance of the retail application needs to be tested for 10,000 users. It is neither cost effective, nor practical to buy 10,000 user licenses of a load generator tool for simulating the load of 10,000 users as in the above situation. Each cashier spends about two minutes, scanning all the items bought by a purchaser before entering the order – the think time of the user is 180 seconds. The performance requirement of the retail application is a response time of less than ten seconds. The number of users that should be simulated to ensure sufficient load on the system can be found using these two inputs. The approach followed for this is given below.

- Maintain low think times in test scripts - the test scripts for the retail application have a think time of five seconds.
- Find the ratio of the sum response time and think time of real life users, and the sum of response time and think time of the simulated users - for the retail application, the ratio works out to be \((180+10)/(5+10) = 12.7\)
- Arrive at the number of users that need to be simulated by the load generator - this can be done by dividing the number of real-life users with the response time to think time ratio. In the retail application example, it will be \(10000/12.7 = 790\) users.

The simulated users can be arrived at using the following equation.
\[ N_{\text{load-generator}} = N_{\text{real-users}} \times \left( \frac{R + Z_{\text{load-generator}}}{R + Z_{\text{real-users}}} \right) \]  

(5.1)

Where, \( N_{\text{load-generator}} \) is the number of users to be simulated by load generator, \( N_{\text{real-users}} \) is the number of users of in real-life, \( Z_{\text{load-generator}} \) is the think time in the load generator scripts, \( Z_{\text{real-users}} \) is the think time of the real life users, and \( R \) is the acceptable response time. Hence, with lesser think time in the test scripts, sufficient load can be generated with lesser number of simulated users.

**Understanding Little's law**

Little's law is one of the most important laws of operational analysis. The law states that the number of requests in a system is a product of the throughput and response time

\[ N = X \cdot R \]  

(5.2)

Where \( N \), Average Number of request in the system and \( R \), Average response time per request \( X \), Throughput.

**FIGURE 5.1: LITTLE’S LAW FOR A CLOSED SYSTEM**

The manifestation of Little’s law for a closed system can be explained using the Figure 5.1. Level 1 indicates the Little’s law applied to the system without the users where the number of
requests in the system would be the product of response time and throughput. Level 2 indicates Little’s law applied to the entire system including the users. Here time by the request at Level 2 would be the time spent by the users thinking (Z) and the response time of the system processing the request. The Little’s law applied here would result as

\[ N = X(R+Z) \]  \hspace{1cm} (5.3)

One of Xyz Corporation’s primary business goals is to provide a broad range of services that meet the diversifying demands of the modern financial marketplace. These services range from traditional commercial and retail banking to financial consulting, investment counseling, and brokerage. The SUT is a Web-based loan service system aimed at an audience of 100,000 users, who will be able to perform basic account activities: view account status, make personal information changes, submit loan application, etc. Business analysts estimate the average number of concurrent users to be around 1% of the total base of customers. The architecture of the Web service consists of six Web servers, installed on Windows systems and two application servers running Linux with an off-the-shelf application system acquired from an external vendor. A large and powerful DB server, running Oracle, is also part of the infrastructure.

When analyzing the workload, the system analyst identified the view account status as the most frequent and resource intensive user action. The test workload is generated by virtual users created by the automated testing tool. In the stress test mode, as soon as a virtual user receives a response, it immediately submits the next request. In other words, the virtual user think time equals zero (Z = 0). In defining the test workload, one has to specify the number of concurrent virtual users. This number can be estimated as follows: Let R denote the average
application server response time, X0 the server throughput, Nr the number of simultaneous real customers, and Nv the number of virtual users. Considering that the throughput and the response time should be the same during the tests and the live operation, one can write the following:

\[
X_0 = \frac{Nr}{R + Z} \quad (5.4)
\]
\[
X_0 = \frac{Nv}{R + 0} \quad (5.5)
\]

From the two above equations, we have \( \frac{Nv}{Nr} = \frac{R}{R + Z} \). \( (5.6) \)

Let us assume that we have set \( R = 2 \) seconds as the average response time goal and \( Z = 20 \) seconds for actual users. Then,

\[
Nv = \frac{Nr \times 2}{2 + 20} = 1,000 \times \frac{1}{11} = 90.9 = 91. \quad (5.7)
\]

So, the tests should use up to 91 virtual users to generate the requests.

5.2.5 End User Education:

Generally end users want to have very quick response times without understanding that the limitations of the bandwidth. For example user wants a response time of less than 6 seconds for every page on a bandwidth of 128kbps. For a page size exceeding 60KB theoretically a response time of 6 seconds is not possible because of bandwidth constraint. Network Time Calculation on 128kbps, assuming 70% utilization of bandwidth and 0 processing time:

\[
\text{Network Time} = \frac{(60 \times 8)}{(70\% \times 128)} = 6 \text{ Seconds} \quad (5.8)
\]

62
Based on the calculation above a 60 KB page size may take 6 seconds on the network even if we have practically 0 second processing time for the request on the server. So users need to be educated about the limitation of the bandwidth and at some places rationalization of User Interface needs to be done.

5.3 Summary

Software performance testing means all the activities involved in the evaluation of how the system can be expected to perform in the field. This is considered from user’s perspective and is typically assessed in terms of throughput, stimulus response time, or some combination of the two. An important issue to consider when doing performance testing is scalability: the ability of the system to handle significantly heavier workloads than are currently required. This necessity might be due to such things as an increase in the customer base or an increase in the system’s functionality. Either of these two changes would typically cause the system to have to be able to provide a significantly increased throughput. If there has not been appropriate scalability testing, unacceptable levels of denial of service or unacceptable response times might occur as workload increases. This is likely to have very negative impact on customer satisfaction and therefore retention. Application performance can be validated and verified adequately using simple techniques.