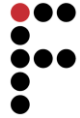


Ingenieurbuero David Fischer GmbH
Muehlemaattstrasse 61, CH-3007 Bern
Switzerland

<http://www.proxy-sniffer.com>
E-Mail: direct@d-fischer.com

INGENIEURBÜRO
DAVID FISCHER



A Guide to Getting Started with Successful Load Testing

English Edition



Proxy Sniffer™

Table of Contents

1	Introduction	3
2	First, a little theory	4
2.1	Application Overload – Point of Collapse	5
3	Choosing Test Cases	6
4	Performing Load Tests - the top 8 points to keep in mind.....	7
5	Poor Response Time – is it the Network?	8
6	Tuning Tips	8
6.1	Optimizing Response Times.....	8
6.2	Solving Stability Problems	9
6.3	Preventing Application Overload	9
7	Other Kinds of Testing.....	10
7.1	Duration or Endurance Testing.....	10
7.2	Starting under Load	10
7.3	Degradation Testing	10
7.4	Short-Circuit Testing	10
7.5	Stripped Test	11
8	Further Information and Manufacturer.....	11

1 Introduction

This short introductory guide is designed to help beginners quickly "get up to speed" with the demanding process of professional load testing.

A load test can be quickly defined and performed; however, when it comes time to interpret the results, it is often difficult to be sure what those results actually mean, and whether or not they reflect reality.

To help you reach your goal of performing successful load tests, a little Know-How, and a structured approach, are absolutely necessary. This information is presented in the following pages.

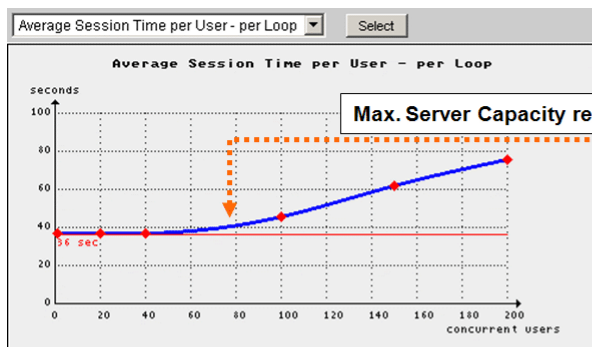
2 First, a little theory

The most important measurements from a load test are the so-called "Load Curves". These provide, at a single glance, an overview of the behaviour of the target system under various loads. Of all possible Load Curves, three types are of special interest:

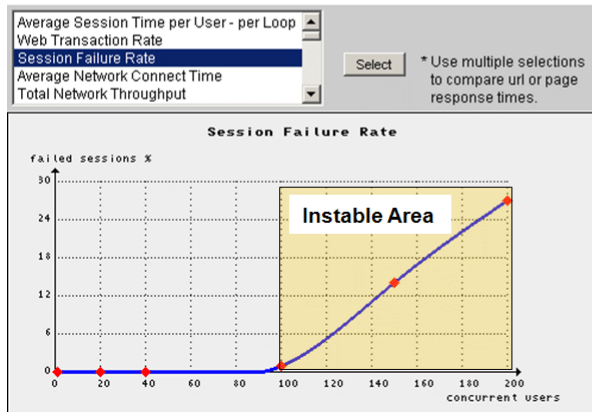
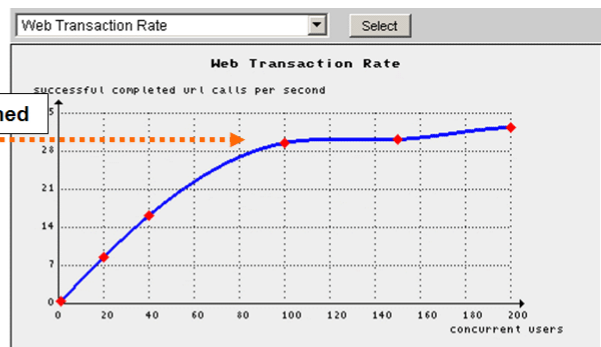
- The average duration of a Web Surfing Session, per User.
This curve represents the Response Time under load.
- The total number of successful URL calls, per second, measured over all Users.
This curve represents the Capacity of the Web Application.
- The percentage of failed Web Surf Sessions, measured over all Users.
This curve represents the Stability of the Web Application.

The following diagrams show examples of typical load curves. The number of simultaneous users, or load, is always shown on the horizontal X-axis:

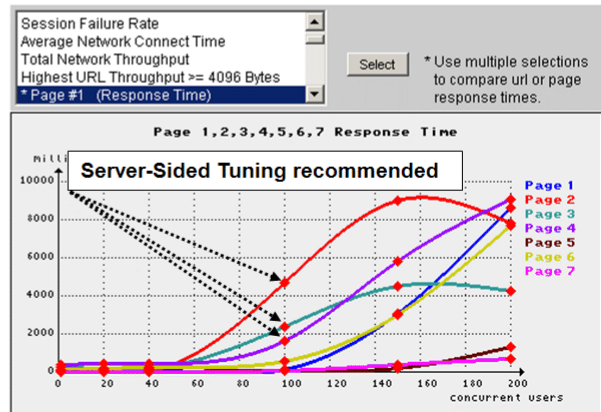
Average duration of a Web Surf Session per User



Number of successful URL calls per Second / Throughput



Percentage of failed Web Surf Sessions



Average Response Time per Web Page

In the diagram at lower right, additional load curves are shown. These curves display the Response Times of various individual Web Pages. From this diagram it is apparent that, starting at 100 simultaneous Users, the Response Times for Web Pages 2, 3, and 4 rise dramatically.

The most important concept is, however, the performance limit of the Web Application. As long as this limit is not reached - in this example it is approximately 80 simultaneous Users - the Response Times are relatively constant, irrespective of the number of Users (diagram at top left above). When the number of Users exceeds 80, the Response Times begin to rise in a linear fashion because the Web Application throughput can no longer increase to match the load (diagram at top right above).

The diagram at lower left above shows that, at approximately 100 Users, the first signs of instability appear, and that this instability rises thereafter in a linear fashion under increased load. This means

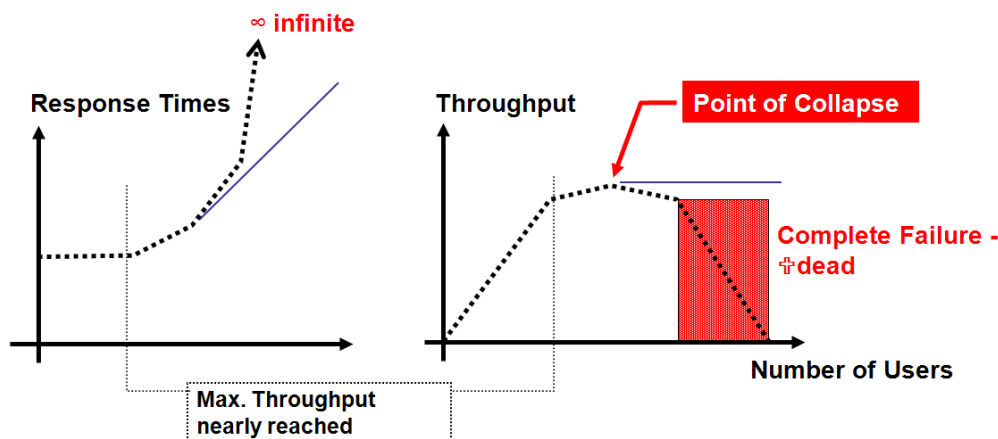
that this Web Application can support a maximum of 95 simultaneous users; in addition, you must also accept the increasingly long Response Times which will occur with more than 80 simultaneous Users.

2.1 Application Overload – Point of Collapse

Not every Web Application behaves as in the previous example, in which the Response Times merely increase in a linear fashion when the application's capacity is exceeded. Many applications also experience deteriorating throughput across the entire System when they are overloaded. This measurement, at the point where throughput starts to decrease under escalating load, is the "Point of Collapse".

If this point is surpassed, the Response Times will tend towards infinity; in practice, this means that the application is no longer available, or "dead". This will not, however, be apparent, and intrepid Web Users will hope that they will eventually receive a response from the Web Application if they continue to click using the mouse on the (unresponsive) Web Page hyperlinks, or if they use the browser Refresh button. In this situation, the application continues to be overloaded and unresponsive until the majority of Users recognize that there is something wrong, and that continued activity brings no result. At this point, Users will begin to leave the system.

If new Users constantly arrive - for example, with an E-Banking Application or an Internet concert ticket shop - the Web Application can remain beyond the Point of Collapse for several hours.



The reason an Application has a Point of Collapse can almost always be traced back to problems with internal parallel processes, indicating errors in the Architecture, Programming, or Configuration. Examples of these kinds of errors are: extensive synchronisation of program code, unoptimized database update operations, and a lack of Operating System resources.

3 Choosing Test Cases

The sponsor of a Web Application often specifies fundamental requirements which must be satisfied; for example:

1. The Web Application should be able to support 200 users at the same time
2. The Response Time for a Web Page should not be more than 3 seconds on average, and never more than 5 seconds.

To meet these requirements, we first need to define which Web Pages should be included in the test. Also needed is an idea of how long a single (actual) User will look at a Web Page before clicking on the next Hyperlink - the user's "think time".

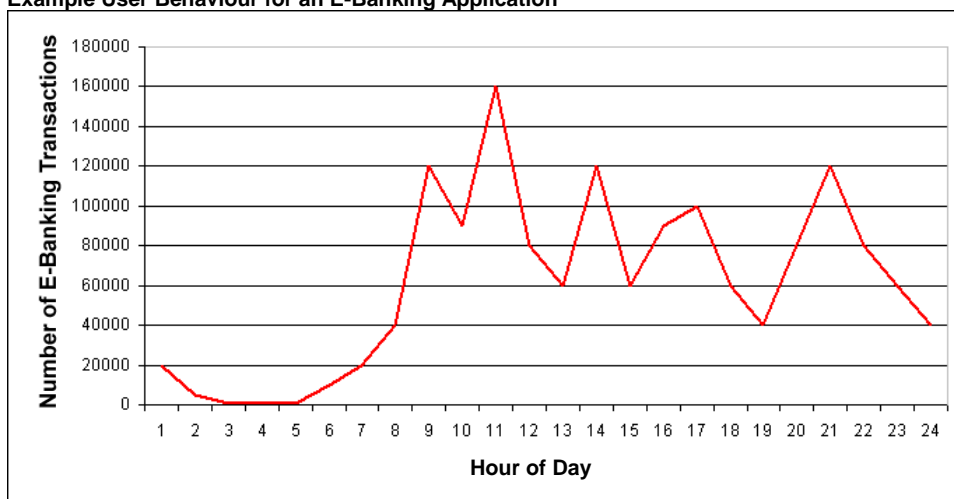
If the Web Application is already in production use, some of the missing requirements can be gleaned from an analysis of the application, or Web Server, log files. For example, the 15 Web Pages called most often can be identified from the access log files, and then a Load Test can be defined to use exactly those Pages.

In some Web Applications, key Menu Options will not be among the Web Pages called most often - as, for example, in the case of an Online Shop. With these applications, Users often browse for an extended time in the product catalogue before finally ordering and paying. Even though the order and payment Web Pages will be relatively less often used than catalogue pages, this order/payment process must also comply with any Response Time requirements. In this case, it is often more convenient to define two simultaneous Load Tests instead of one. As an example, one Load Test, with a large number of simultaneous users, can be executed to create a basic "background" load, and a second test - run at the same time as the first - can be executed with fewer Users to cover the most important special cases.

If you have only general summary information on how the Web Application is used, or no information at all, you will have to fill in the missing data by extrapolation or estimation in order to arrive at a reasonable approximation of actual User behaviour.

For example, if the only known fact for an E-Banking Application is that it processes 1.5 million financial transactions per day, you must take into consideration that these transactions will likely not occur evenly distributed over the entire day:

Example User Behaviour for an E-Banking Application



If only the average value for the desired load is known, this average load value must be multiplied by some factor for estimating peak-time activity. From experience with E-Banking applications, a realistic peak load factor multiplier is somewhere between 3 and 12, with 5 being the most common value used.

Continuing with the E-Banking example, knowing only the number of financial transactions is not enough to arrive at how many Users will be required for the Load Test. However, by evaluating the Load Curves produced from Load Tests, you can discover the Web Application's maximum capacity, and thereby also the maximum number of financial transactions it can support. Because Load Curves

highlight the direct relationship between throughput and Response Times, you will then also know the Web Application's expected Response Times under light, medium, and heavy loads.

4 Performing Load Tests - the top 8 points to keep in mind

- Load Curves are produced by repeating the same Load Test with varying numbers of Users; for example, 1, 2, 5, 10, 20, 50, 100, 200, 500, and so on. It is recommended that the load be increased step-by-step, increasing the number of Users logarithmically. This is the fastest way to gain an overview of performance.
- Do not run Load Tests with too short a duration. Starting with 50 Users, the test should last at least 5 minutes; as a rule of thumb, divide the number of Users by 10 to arrive at the minimum test duration in minutes. With a very large number of Users, ensure that Users are not all involved immediately at the start of the test, and then remember to take into account this "ramp up" time, waiting until the required load is achieved before starting the test duration "clock".
- Sometimes a Web Application stops working during a Load Test, and does not recover by itself after the end of the test. Make sure that someone is available, during Load Testing, to restart the Web Application so you can continue with testing after an application failure.
- A Load Test is not valid if the load-releasing system - that is, the system creating the load - is itself overloaded. In this case the measured Response Times will not be valid. Always watch the CPU activity on the load-releasing system during the test for signs of overloading. If the load-releasing system does not have enough CPU power for the Load Test, additional systems can be added, and combined, to form a Load-Injector Cluster whereby the Load Test program is distributed across many computer systems.
- If the Web Application is a secure one, requiring authentication, each Load Test User should have its own login account during the test; otherwise, the test will not be "fair" to the person or company delivering the application. If many Users share a single login account during the test, errors may occur, and be measured, which would not occur in real life.
- During the Load Test, arrange for someone to observe the CPU activity on the Web or Application Server machine on which the Web Application is running. This information will be needed later to see if the server's CPU is 100% used at the time the maximum throughput (or performance limit) of the Web Application is reached. If the server's CPU usage at maximum load is below 70%, this indicates that the Web Application is not able to fully use server resources, and presents an opportunity for performance tuning.
- Ensure that the network capacity between the load-releasing system and the Web Application Server is sufficient; that is, at least 100 MBit/second. Do not perform Load-Tests over slow DSL connections.
- Ideally the System Administrator, the Database Administrator, and a Web Application Developer will be present during the Load Tests in order that you can analyze the Load Test results together, and immediately decide on and implement any necessary performance tuning measures. This test, analyze, and tune process should be repeated until the Web Application is performing satisfactorily. It is important to note that having all involved parties present and immediately available will save a large amount of time, and help bring you quickly to your goal.

5 Poor Response Time – is it the Network?

When the results of a Load Test show unacceptable Response Times, it is often assumed that the "Network" is the cause of the problem. In reality, this is seldom the case.

Fortunately, it is quite easy to check this using test measurement data. Simply compare the Response Time for a static image (such as JPEG, GIF, or PNG), with the Response Time for a dynamically-created HTML page of approximately the same size.

Consider this example:

- The Response Time for a static GIF image of 20,000 bytes is measured at 60 milliseconds.
- The Response Time for a dynamically-created HTML page with a size of 40,000 bytes is measured at 3 seconds.

Assuming the Response Time of the static image depends solely on the network bandwidth (which is not really true, but is useful for our purposes in this example), the dynamically-created HTML page should have a Response Time of 120 milliseconds since it is "only" twice the size of the image. Since the actual Response Time of the HTML page is 3 seconds, the Web Application uses at least 2880 milliseconds (3000 - 120) to produce and send the page.

6 Tuning Tips

The first step is always to try to establish the reason for poor Response Times or stability problems, based on an analysis of the measurement data. Optimize your Web Application infrastructure solely on the basis of facts ascertained by this kind of analysis:

6.1 Optimizing Response Times

- If, under increasing load, the Response Times for both static images and dynamically-created HTML pages rise in tandem, there may be a network problem or, more likely, an error in the configuration of the Web or Application Server hosting the Web Application. Examples of configuration errors would be not enough worker threads, or too few worker processes.
- If, under increasing load, some URL calls become disproportionately slower than others in relation to the rise in load, the error will be in the programming of the Web Application, including perhaps inefficient database queries.
 1. If the Web or Application Server uses a database connection pool, check first if this pool is big enough. For a single Web Page access, each User will usually need one connection out of the pool, unless the Web Application itself has implemented some form of caching.
 2. Check next the database engine itself, by arranging with the Database Administrator (DBA) to collect statistics on the slowest SQL queries and their respective Response Times.
 3. If these first two investigations do not uncover the cause of the problem(s), the cause will be application programming errors. The mistake made most often is inappropriate use of code synchronization in inner loops, resulting in code blocks needed by all users being executed sequentially instead of in parallel. A dead giveaway for this situation is when the CPU of the Web or Application Server is only lightly used under heavy load, because waiting for synchronized code to be released does not use CPU time.

6.2 Solving Stability Problems

Analyze the most frequent errors, and try to decide if the cause is inside the Web Application itself, or in the surrounding environment.

- If network errors or timeouts occur relatively evenly over all URL calls, the problem will not be with the Web Application. In this case, try first to tune the TCP/IP stack of the Web or Application Server machine using Operating System network and system parameters. Other possible sources of these errors are firewalls, load balancers, and reverse proxy servers. Perform tests from various different network locations, proceeding always from inner areas to outer areas. For example, test first directly and as close as possible to the Web or Application Server – from the same LAN - and then move progressively outwards to test indirect access via reverse proxies and/or load balancers.
- If stability problems occur only with specific URL calls, then the problem lies in the Web Application itself. This also applies if HTML pages are received which contain error messages, or partially incorrect or unexpected content.

To solve stability problems with specific URLs, you will need the support of the Web Application developers. Describe the problems to them, and motivate them to analyze their Web Application source code with a view to understanding the observed behaviour. To help locate the source of the problems, application code changes will be necessary in order to insert log messages and/or checkpoints, with timestamps. This process is sometimes known as "White Box Testing". Internal checkpoints are first inserted at a high level in the application code, and then iteratively refined and moved to lower level suspect code areas as the tests zero-in on the cause. This process reduces the number of test iterations required to identify the source of the problems. "White Box Tests" can, of course, also be used in optimizing Response Times.

6.3 Preventing Application Overload

First, try to tune the Web Application to the point where it can provide acceptable Response Times under a load which is higher than that expected in actual production. If this is achieved, the overload point of the application would never be reached in actual use.

If this is not possible, the alternative is to insert a reverse proxy server - Apache for example - in front of the Web or Application Server, and configure the reverse proxy to act as a buffer to throttle the load, such that the overload point would never be reached. In order for this to work properly, repeated measurements of, and adjustments to, the reverse proxy must be made for the proxy to have exactly the same performance capacity as the Web Application. In addition, the TCP/IP stack on the reverse proxy must be carefully adjusted.

7 Other Kinds of Testing

Load Curve measurements are involved in all types of testing commonly known as "Load Tests" and "Stress Tests". There are other kinds of testing, which can produce additional useful information, and these are described in this section.

7.1 Duration or Endurance Testing

An endurance test provides data on the long-term stability of the Web Application; that is, it answers the question of whether the application is stable when running for many days. Normally, such a test is started in the evening and left to run overnight (for 12 hours) at 50% of the application's maximum capacity.

An endurance test usually makes many more URL calls than is normally seen in actual production for a single day. This acts as a sort of "time-lapse photography", in that many days of "production activity" can be simulated in a single night of testing.

7.2 Starting under Load

When the Web Application is brand new, and has never been in production before, plans should be made for this type of test. Leave the Web Application turned off, and start a Load Test which would establish a load on the application at near 100% capacity. Naturally, you will receive only error measurements, because the Web Application is not yet online. Now, start the Web Application during the on-going load test and observe the test results in real-time. The Web Application must be able to stabilize by itself under these conditions within a few minutes.

The idea behind this kind of test is that a new Web Application will become usable on the network before it has finished its internal initialization, and the first URL calls could cause it to immediately fail. This should not happen, and a "Starting under Load" test will allow you to check this.

7.3 Degradation Testing

If the Web Application is running in an environment equipped with components - such as a load balancer or a Web Application Cluster - which provide redundancy and high-availability, perform a few degradation tests whereby the high-availability components are individually turned off and then turned back on; for example, stop a branch of the load balancer, or turn off a computer in the Web Application Cluster. In this way, the effect of partial system failures in the application environment on the availability, performance, and stability of the Web Application can be tested.

7.4 Short-Circuit Testing

A "short-circuit test" is a type of test where the performance of the infrastructure is measured **without** the Web Application running. A few large, static image files are put on the Web or Application Server, and the Load Curves are produced using only these images. In this way, the maximum throughput of the system in the absence of the overhead associated with producing dynamic HTML pages can be determined. In addition, this test will indicate if there is a performance or stability problem even before the Web Application is running.

Another obvious test of this type is to measure the performance of direct accesses to the Web Application, and then to compare these results to measurements made while accessing the application via a reverse proxy. This test measures the performance loss associated with using a reverse proxy server.

7.5 Stripped Test

In a "stripped test", all URL calls which produce static content are removed from the Load Test configuration; for example, removing static images, style sheets, and javascript files. Only pure HTML content produced by the Web Application are involved in the test.

This kind of test is especially useful for developers who need to optimize specific dynamically-generated Web Pages. Removing unnecessary - from the point of view of the test - static content such as images reduces the network bandwidth required and lightens the CPU load on the Load-Releasing systems, enabling a larger number of simultaneous users than would otherwise be possible on systems with limited resources. This also reduces the number of measurements to be analyzed, allowing a clearer picture of the situation.

8 Further Information and Manufacturer

Ingenieurbuero David Fischer GmbH
Muehle mattstrasse 21
3007 Bern
Switzerland

Web Site: <http://www.proxy-sniffer.com>

E-Mail: direct@d-fischer.com

All rights reserved.