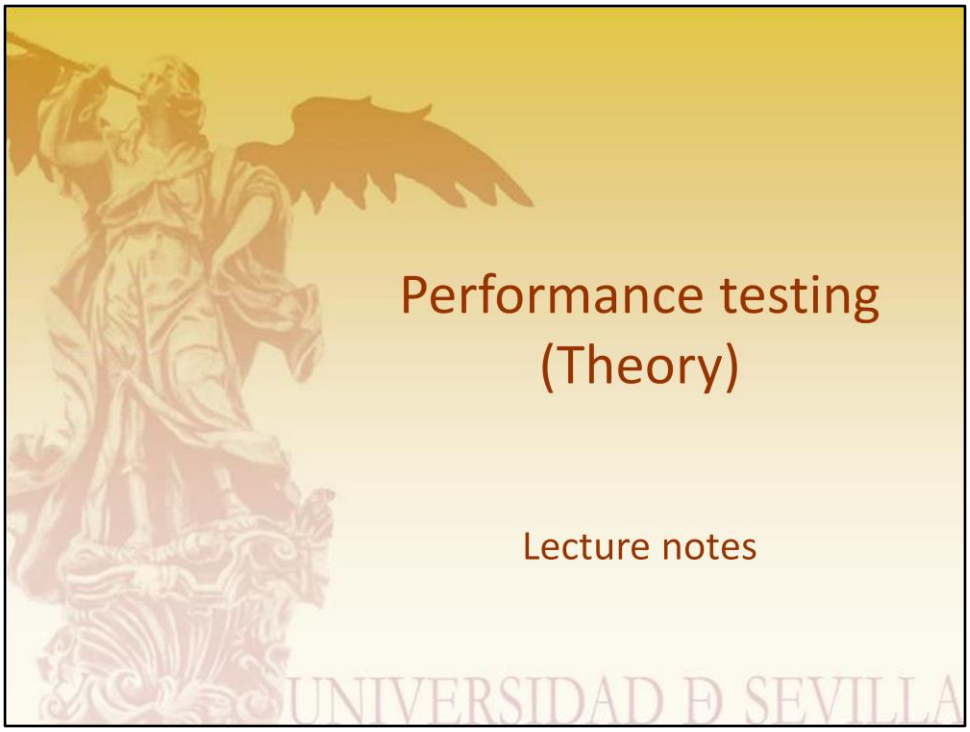


Welcome to this lecture! Today, we're going to work on performance testing.

--

What's performance testing about?

UNIVERSIDAD Ð SEVILLA

2

As usual, we start this lecture with a question: what's performance testing about?

This is a good definition

Performance testing refers to tests to determine how a system performs in terms of responsiveness and stability under a particular workload.

UNIVERSIDAD Ð SEVILLA

3

Performance testing, aka benchmarking, refers to a series of tests that are conducted to determine how a system performs in terms of responsiveness and stability when it's confronted with a particular workload.

Performance testing is very important, because this is the only way to guarantee that your system will be able to serve the requests with which it’s confronted gracefully. Unless you know what the limits of your system are, you’re very likely to produce a system that works well when it serves a dozen users but collapses when the number of users scales to one hundred, which is quite a modest workload.

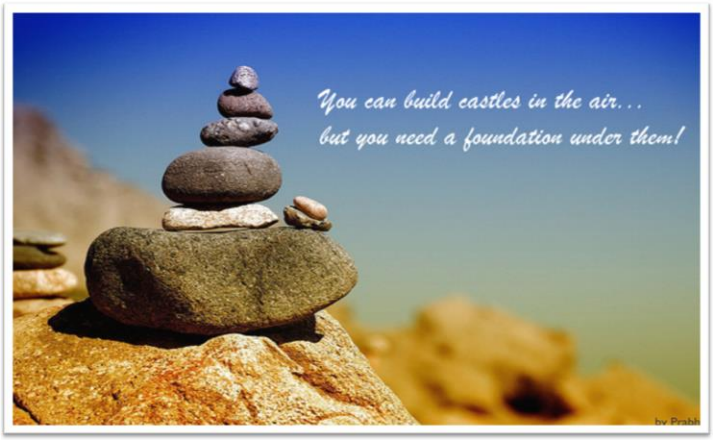
What do I have to know?

UNIVERSIDAD Ð SEVILLA

5

Let’s now report on what you have to know regarding performance testing.

Step 1: foundations

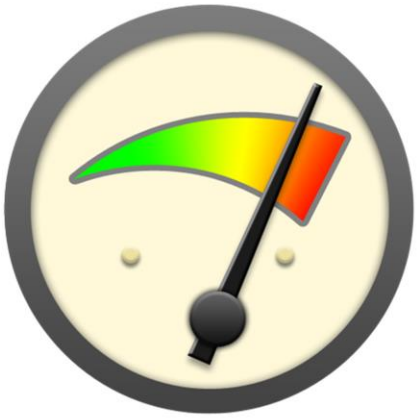


UNIVERSIDAD Ð SEVILLA

6

First, and foremost important, you need to know a few foundations about performance testing.

Step 2: the testing tool



UNIVERSIDAD Ð SEVILLA

7

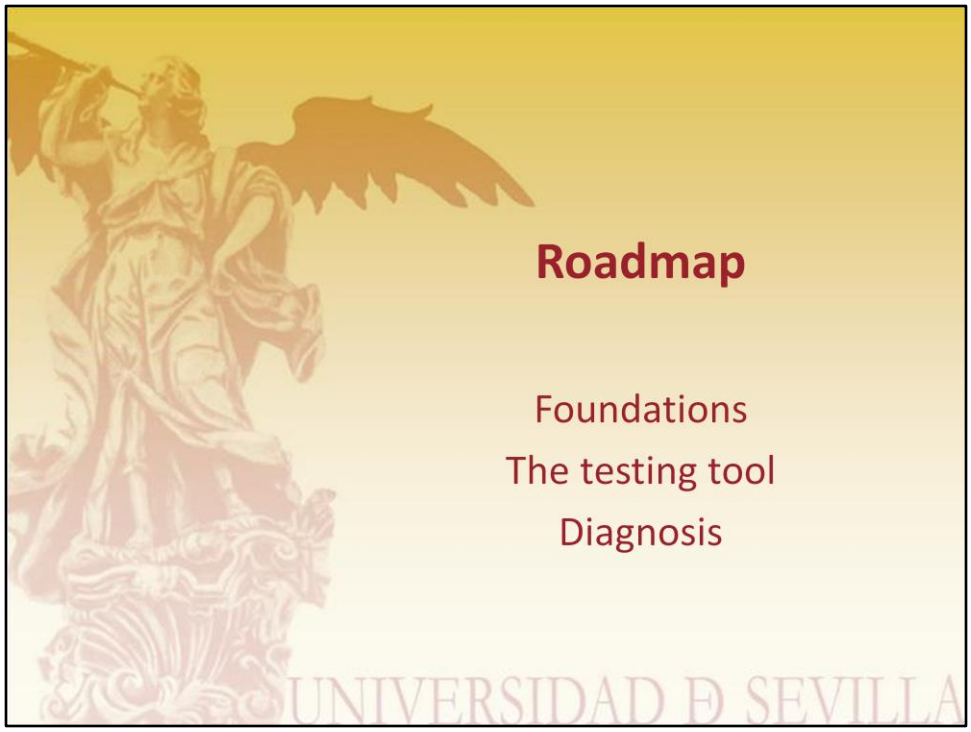
Then you need to know a testing tool that allows you to simulate a workload and measure how your system performs.

Step 3: diagnosis

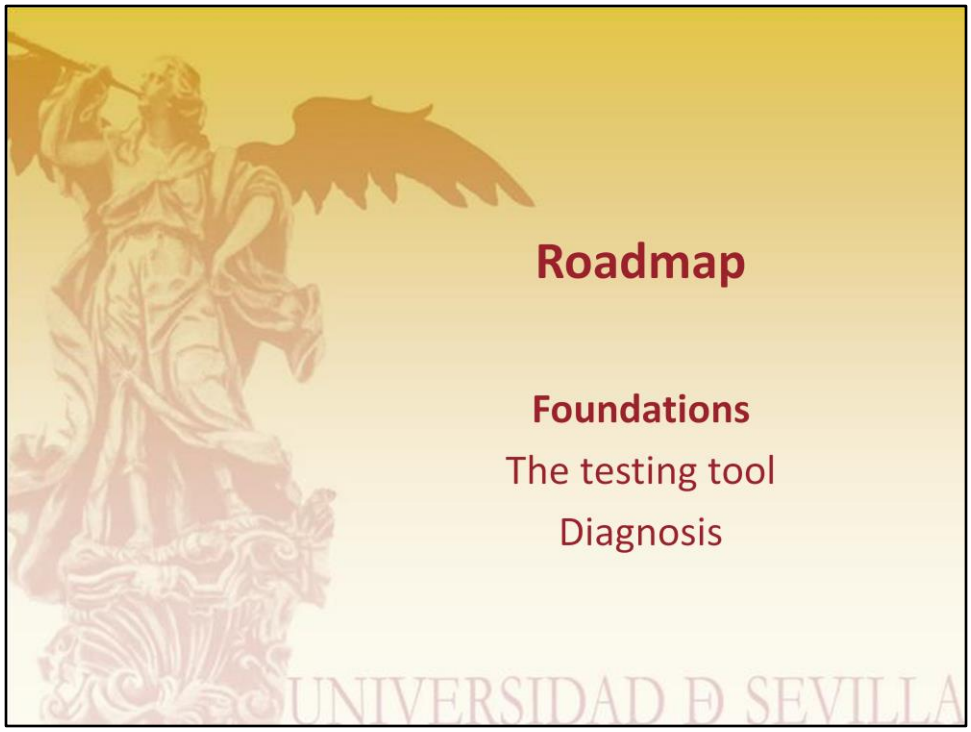
UNIVERSIDAD Ð SEVILLA

8

Finally, you need to know how to diagnose your system when it performs poorly.



It shouldn't then be surprising to you that this is our roadmap for today's lecture. We'll first provide a foundation, then will introduce the testing tool that we're going to use, and, finally, we'll report a little on diagnosis.



Let's start with a few fundamental concepts.

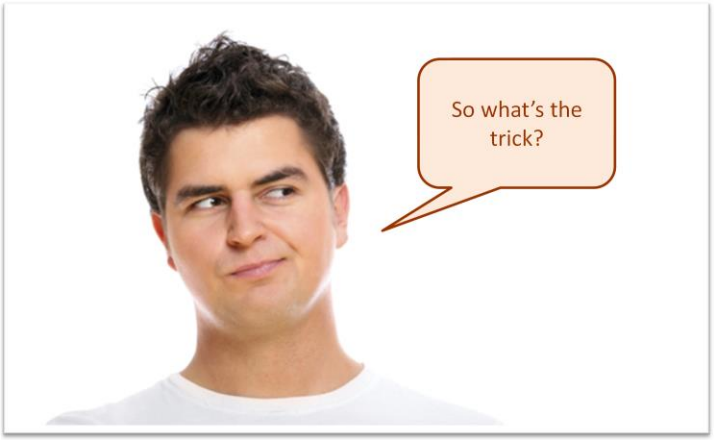
What happens in an Internet minute?

This slide shows a well-known chart by Intel that is entitled “What happens in an Internet minute?” In summary: 47,000 applications are downloaded from on-line app stores; Amazon sells goods that are worth $83,000; 100 LinkedIn accounts are created; 20,000,000 photos are viewed in Flickr; 100,000 messages are twitted; 6,000,000 facebook pages are viewed; 2,000,000 searches are served by Google; and 1,300,000 videos are viewed in YouTube. These are use cases that are executed by users world wide, and the corresponding systems must be able to handle this workload gracefully.

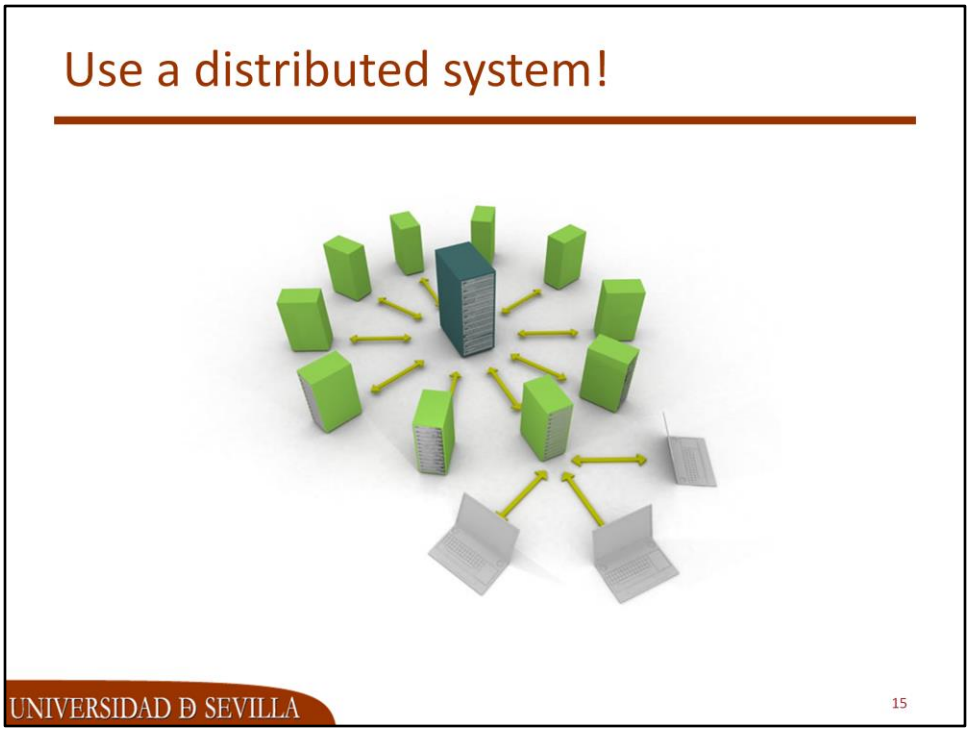
You may wonder which computer can handle such a workload. That's a good question.

Unfortunately, no computer in the world can handle such a workload! There are computers that are very powerful, but none of them can handle a million concurrent users.

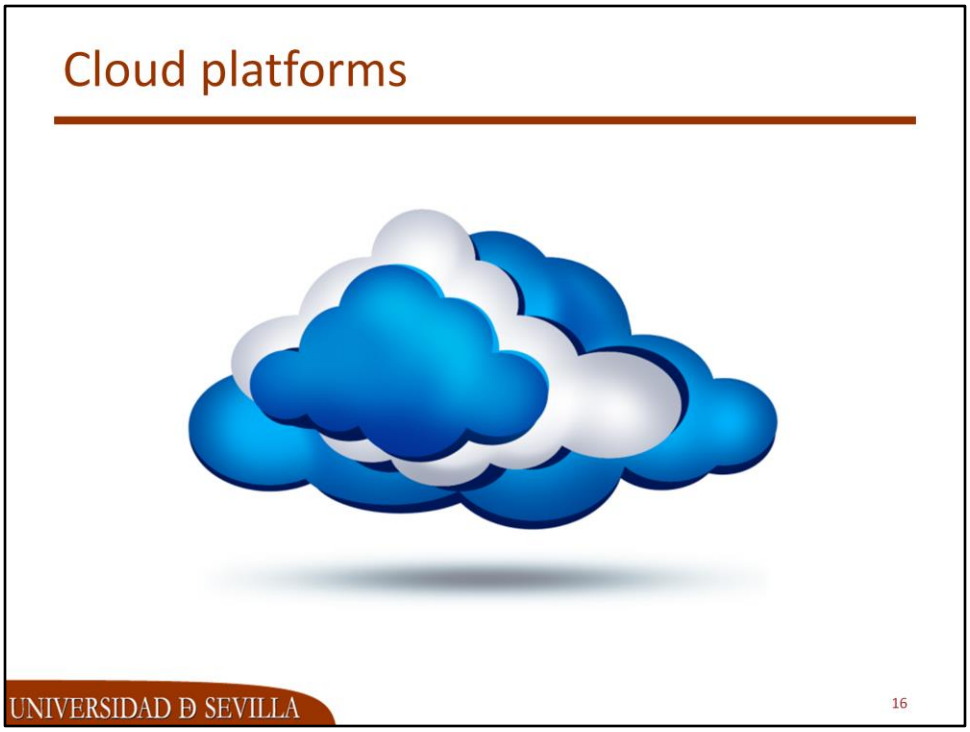
There's no trick



This seems contradictory, but there's no trick at all.



In cases in which a single computer cannot handle the workload, the solution is to add more computers: additional databases and application servers plus a so-called load balancer. A load balancer's a system that redirects the incoming requests to the computer that has the minimum workload. Obviously, a good load balancer must be a quick and responsive system or, otherwise, it'll easily become the bottleneck of the whole system.

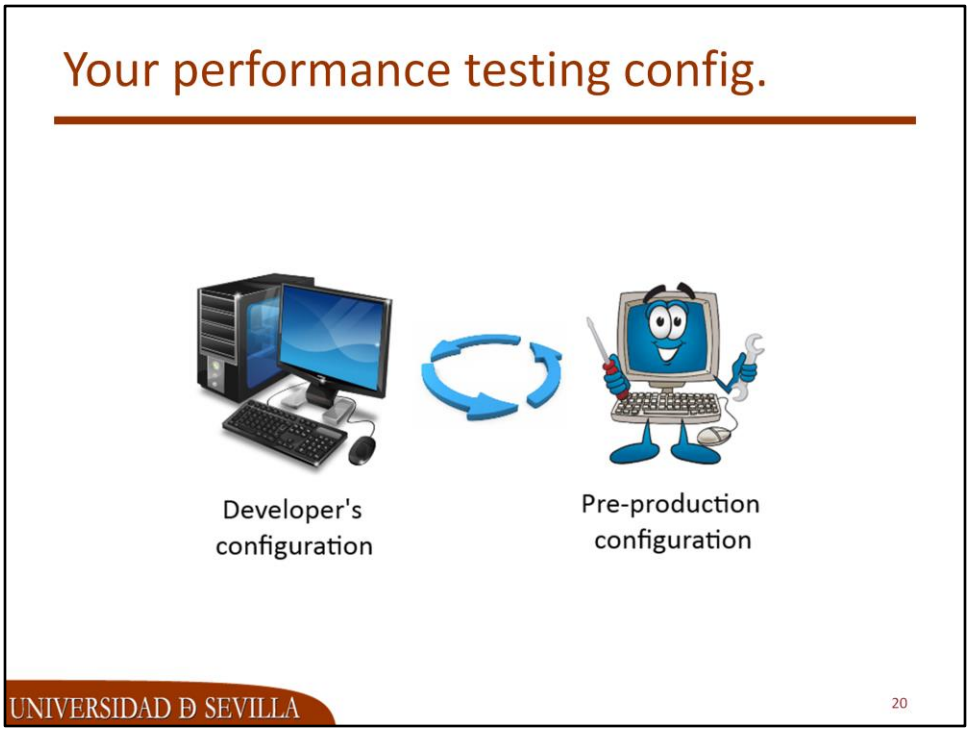


If you're interested in scaling your applications, we strongly recommend that you should take a look at cloud platforms like Jelastic, Heroku, Jellify, Microsoft Azure, or Google App Engine, to mention a few examples. In general, they are not difficult to command and they will save you from a lot of work when setting up a distributed system. We strongly recommend that you should explore these technologies.

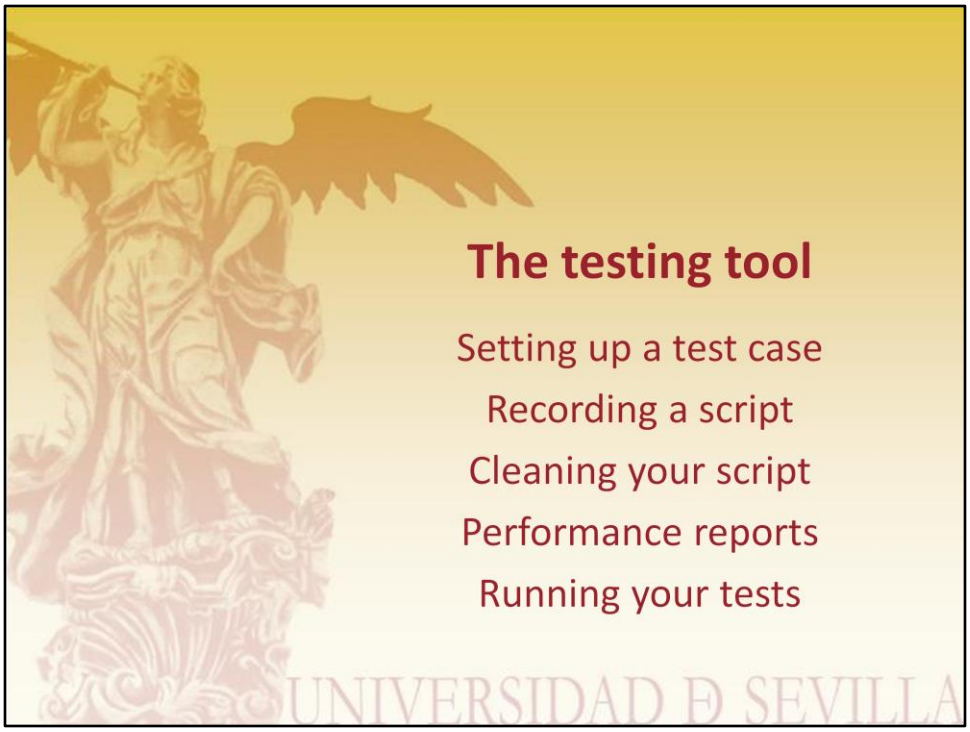
Unfortunately, there's not enough time in this subject to study this topic. So we'll restrict our attention to systems that work on a single machine.

Let's now report on the performance testing tool that we're going to use in this subject.

We're going to use Apache jMeter, which is quite a simple tool; but it's very popular because it helps create performance tests and run them very easily. We provided you with a copy of this tool months ago. If you don't have it, please, browse the tool repository that is available at ftp://practica:practica@postgrado.lsi.us.es/DT and download the version that is appropriate for your system. If you're using the developer's configuration that we provided when the course started, then jMeter is pre-installed.



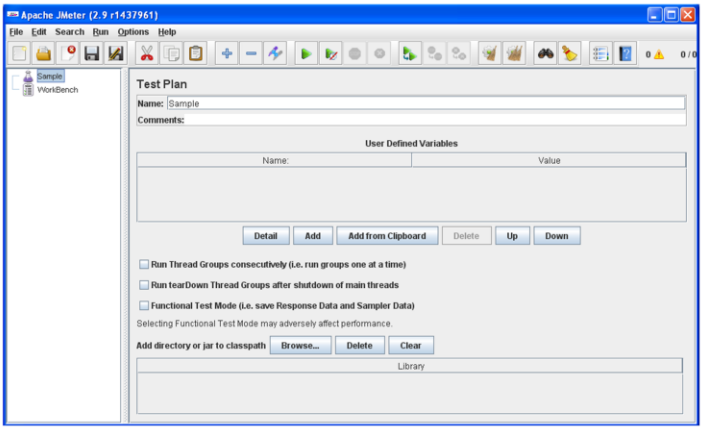
Please, recall that we have a developer's configuration and a pre-production configuration. The former provides every tool you need to develop your web information systems and the latter's a clone of your customer's server. You must use your pre-production configuration to learn how to deploy your web information systems to your customer's actual computer facilities and to conduct performance and acceptance tests.



This is the roadmap that we're going to explore regarding jMeter: first, we'll explore how to set up a test case, then how to record a script, how to clean it, how to create performance reports, and finally, we'll explore how to run your performance tests.

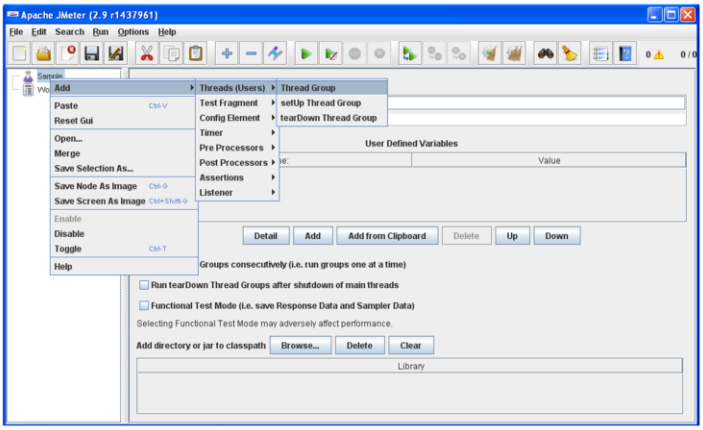
Let's start with how to set up a test case.

Start jMeter up



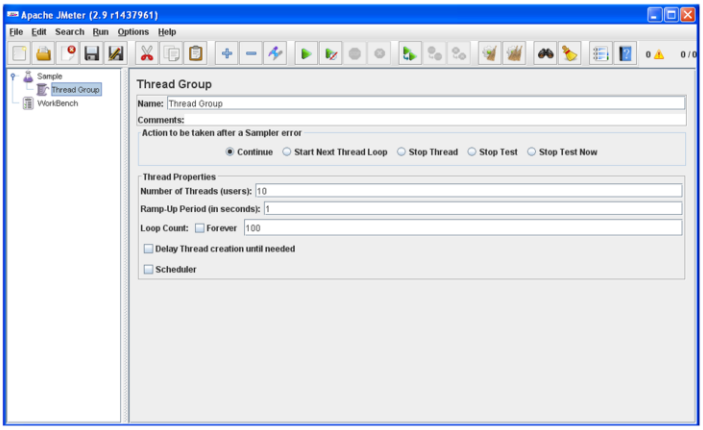
This is how jMeter looks like. By default, it starts up with a blank performance test called “Test Plan”. Change the name to an appropriate identifier, e.g., “Sample”. You don’t have to configure anything else in this screen.

Create a thread group



After changing the name, we need to create a so-called thread group. Simply put: a thread group will help us simulate a number of concurrent users who are executing typical use cases on our system.

Configure your thread group

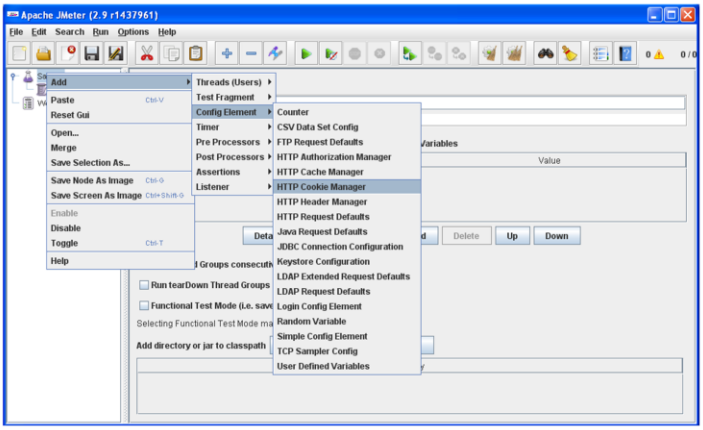


The minimum configuration that you must perform is to change the number of threads, which indicates the number of users that will be simulated, and the loop count, which indicates how many times each thread will execute the use case with which we're going to test our system (we'll provide additional details on this later). In our example, we set these parameters to 10 and 100, respectively; that means that jMeter will simulate 10 users who execute a use case 100 times each.

WARNING: please, start with 10 users. Don't try 1,000 or 10,000 users since this may render your computer totally unresponsive. It won't crash (we hope it doesn't); but it'll be so unresponsive that you'll have to reboot it.

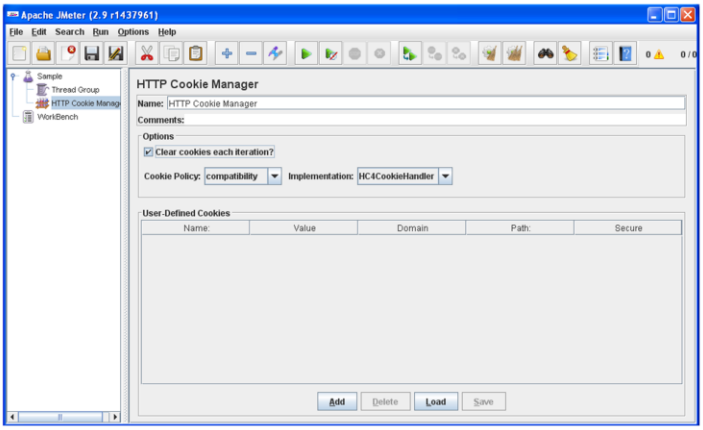
NOTE: there's a parameter called ramp-up period that is sometimes useful. It tells jMeter the total time that it has to start the threads in which it executes the test cases. If there are 10 threads and the ramp-up period is 100 seconds, then jMeter will take 100 seconds to get the 10 threads up and running. Each thread will start 100 / 10 = 10 seconds after the previous thread was started. If there are 30 threads and the ramp-up period of 120 seconds, then each successive thread will be delayed by 120 / 30 = 4 seconds.

Add an HTTP cookie manager

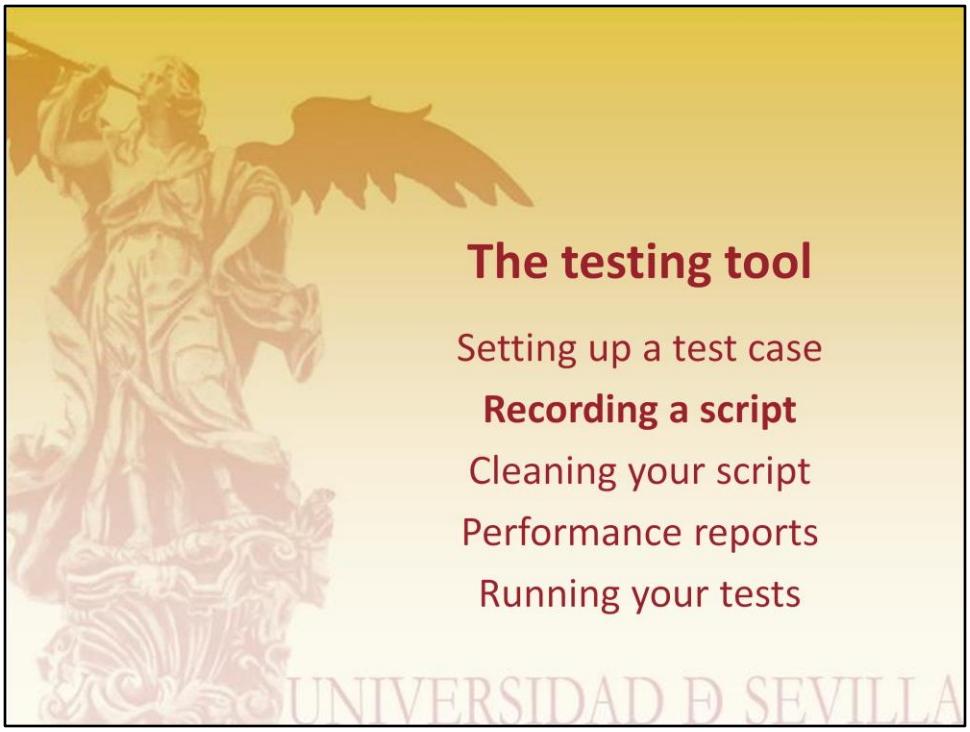


We also need to add an HTTP cookie manager to our benchmark. Please, recall that our application uses cookies to, e.g., persist the language in which the interface must be rendered. Thus, we need an HTTP cookie manager in jMeter to store them.

Configure the HTTP cookie manager



There's little to configure regarding the HTTP cookie manager: check that you wish the cookies to be cleared on each iteration, set the cookie policy to "compatibility", and set the implementation to "HC4CookieHandler", which is the newest one.



Setting up a test case, wasn't difficult at all, was it? Now we need to explore how to record a script.

What's a script?

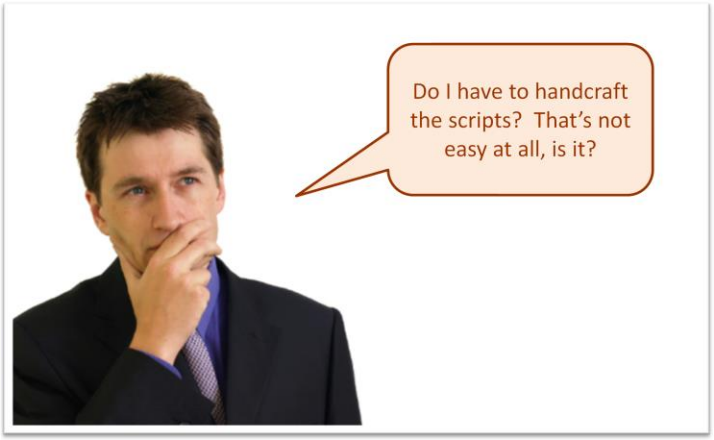
A script's a sequence of interactions that a user performs when he or she's executing a use case

UNIVERSIDAD Đ SEVILLA

29

A script is a sequence of interactions that a user performs when he or she's executing a use case. Simply put, a list of HTTP requests that he or she makes to the server. We need to add scripts to our test cases so that jMeter can reproduce them and simulate a number of users who are interacting with our system.

This is a good question



UNIVERSIDAD Ð SEVILLA

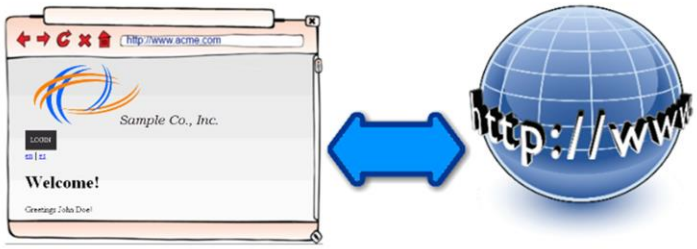
30

Handcrafting such a script is not easy at all; more than that: it's very error prone.

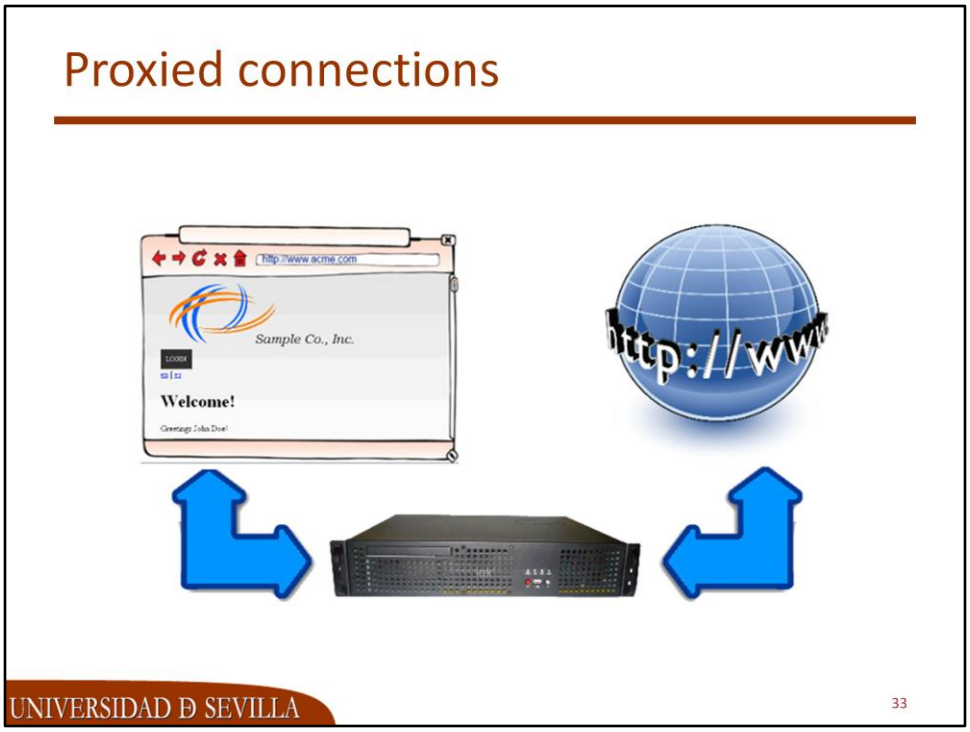
These are good news for you

Fortunately, jMeter can record them automatically while you interact with your systems. jMeter uses proxies to record your scripts, so we'd better learn a little theory about them.

Direct connections



Typically, when you make your browser for an address like “http://www.acme.com”, it sets up a network connection to the corresponding server and sends an HTTP GET request through it; the server then produces an answer, returns it, and the browser renders it on the screen. Pretty simple. That’s what experts call a direct connection. We explored this process in one of our earliest lectures.

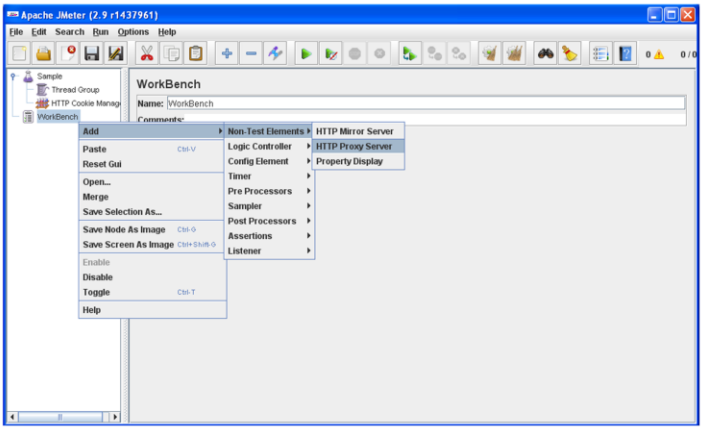


But there are also so-called proxied connections. Such a connection goes through a proxy, which is a machine that acts as an intermediary between your browser and the actual site to which you're sending a request. Having a proxy has a number of advantages; for instance, a proxy allows your machine to be in a private network to which other machines in the world may not have access, which helps your organisation be more secure; a proxy may also cache common requests that dozens of people make daily without hitting the actual server, which decreases the bandwidth that you consume. These are the reasons why proxies are very common in professional scenarios.

The question is: what can a proxy do to record a script? That's a good question. We're going to use a proxy to sniff the network and capture the requests that a user makes from his or her browser to the system we're going to benchmark.

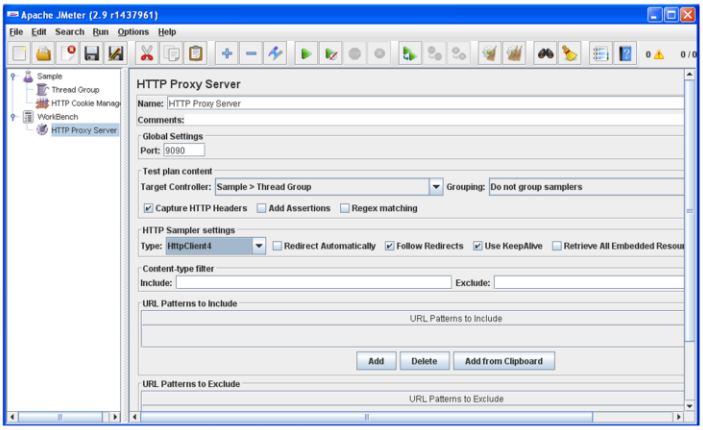
Didn't you get it? Ok, let's try a new picture: jMeter will be the proxy. The idea is that when you make your browser for "http://www.acme.com", a proxied connection will be established to that site. Since it's a proxied connection, it'll go through a proxy, that is, though jMeter, which will then be able to easily record the interactions of the user with the actual site. Let's try to put these ideas into practice so that you can understand how it all works.

Add an HTTP proxy



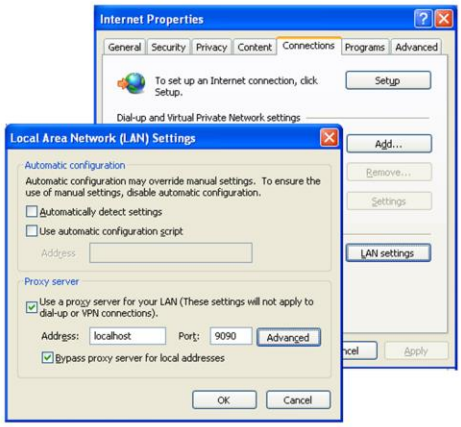
First of all, right click on “Workbench” and add an HTTP proxy server.

Configure the HTTP proxy



You only need to change the port to 9090 in order to configure it. You can keep 8080 if you wish, but we recommend to use 9090 to avoid confusion with the HTTP port that we use in development configurations. That's all you need to configure.

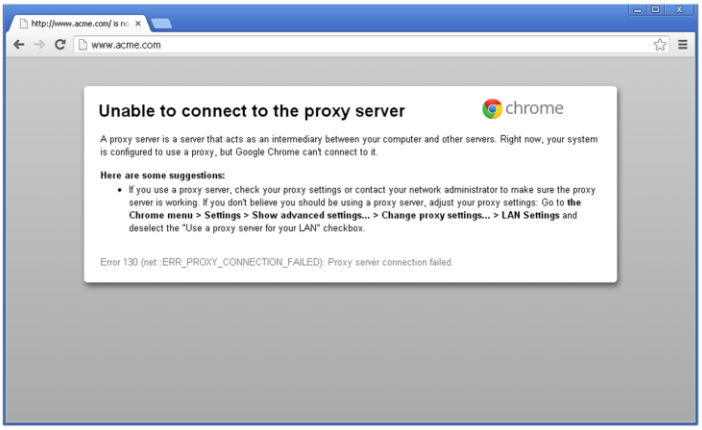
Change your system's proxy



The next step is to change your operating system's proxy. To do so, you must open your local area network settings by clicking on Start > Control Panel > Network and Internet Connections > Internet Options > Connections > LAN settings. You'll get a dialog like the one in this slide, where you must check that your proxy server is at address "localhost" and port "9090". Please, don't forget to check that you wish to bypass the proxy server for local addresses; that means that if you make your browser for "http://www.acme.com", the connection will be proxied, but if you make it for "http://localhost" the connection will be a direct one. The reason why local addresses must be bypassed will become clear in a few slides.

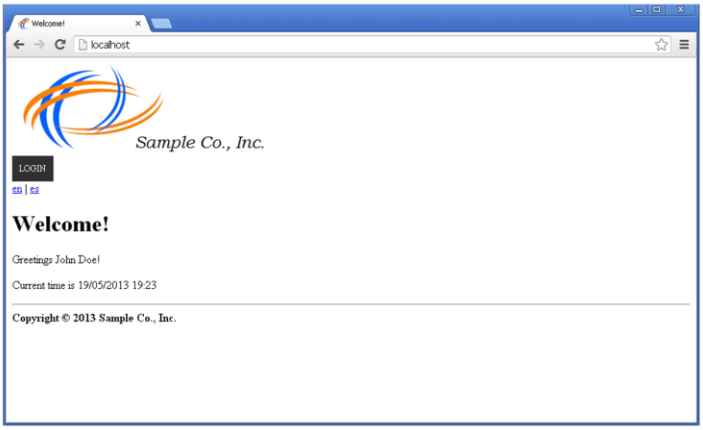
NOTE: please, consult your operating system's documentation to find out how proxies are configured. Unfortunately, there's no a standard way to configure your operating system's proxy. That's why we can't provide you with a guideline.

Check that … nothing works



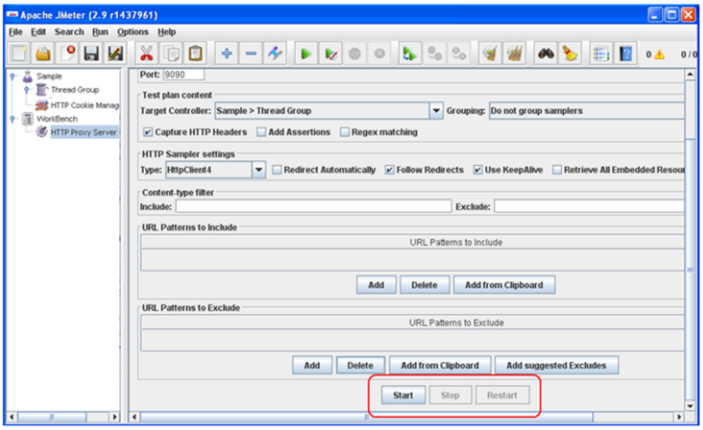
Let's try the proxy. Make your browser for "http://www.acme.com". You should get this error message, which indicates that your browser's attempting to establish a proxied connection to the site, but the proxy's not responding. That shouldn't be surprising: before, we configured the proxy, but we didn't start it.

Except for localhost requests



Before starting the proxy, please, make your browser for “http://localhost”. That should work because this is a local address and you’ve instructed your operating system not to proxy local addresses.

Let's start jMeter's proxy up

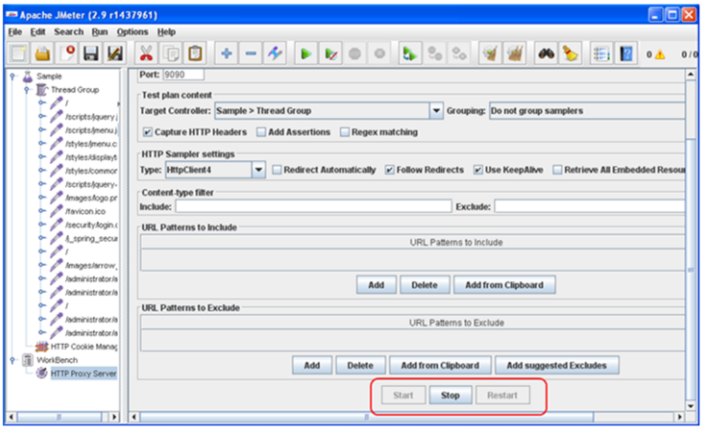


Ok, it's now time to start jMeter's proxy. Get back to jMeter, scroll a little down, and press the "Start" button. Apparently, nothing happens, but get back to your browser.

Let's record a simple script

Make your browser for "http://www.acme.com", log in to the system, and execute a few options from the main menu. Can you see that jMeter's recording your interactions? Every time you make a request to the server, jMeter intercepts it and records it.

Stop the proxy



Ok, play for a little and record a small script. Press the “Stop” button when you’re done.

Let's now talk a little about cleaning your scripts.

Cleaning your scripts

UNIVERSIDAD Ð SEVILLA

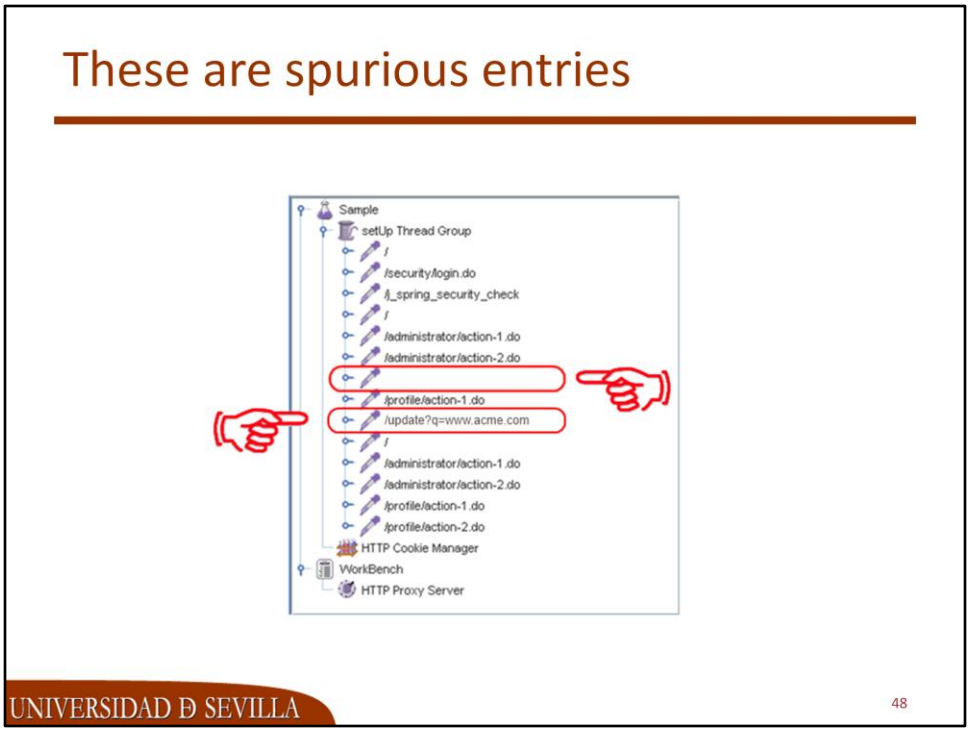
45

Unfortunately, the scripts that you record are not usually ready to be executed. They need to be cleaned a little: you need to remove spurious entries and add sensible human delays.

Let's talk about spurious entries

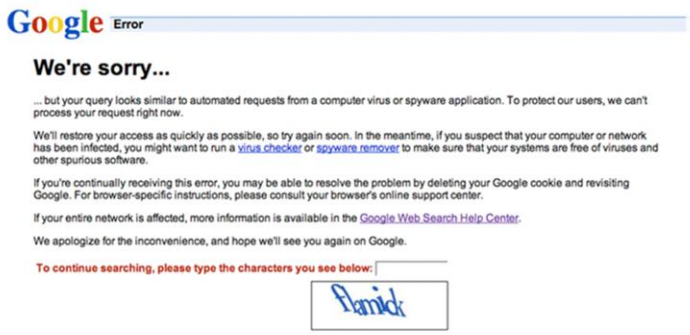
Let's first talk a little about spurious entries.

Please, review your script carefully when you finish recording it. You must check that every entry in your script corresponds to your web information system; that is, you must check that there aren't any spurious entries.



Spurious entries are entries like the ones that we have highlighted in this picture. Note that there is an entry without a path and an additional entry that corresponds to a path that doesn't exist in our system. These entries typically correspond to background requests that your browser performs. In this case, we were using Chrome and the spurious requests were sent to Google's servers. Do you know what happens if you simulate a workload of 10 users running the script in this slide 100 times each? That means that you're performing at least 2,000 requests to Google's servers in a very short period of time.

Be careful with such entries



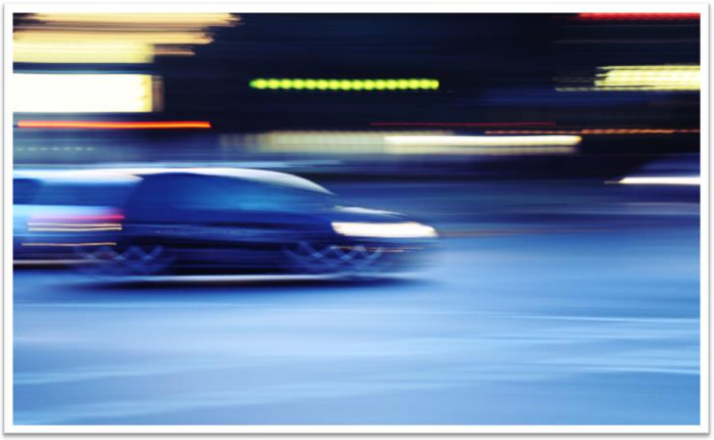
And this is how Google's servers protect themselves from your script: Google will kick you out! The first few times, Google allows you to recover the service by typing in a captcha; but if you persist, Google will definitely ban your IP address from having access to their services. So you'd better review your script and remove entries that involve other companies' servers.

Please, note that the IP addresses in our laboratories are limited. Before running a script, you must make sure that it does not include any calls to any companies' servers or otherwise you might ban a University IP address and prevent other students from using these companies' services. **Please, do check your scripts twice or three times before running them!**

Let's talk about adding delays

Let's now talk about adding sensible human delays.

The scripts run at full speed!

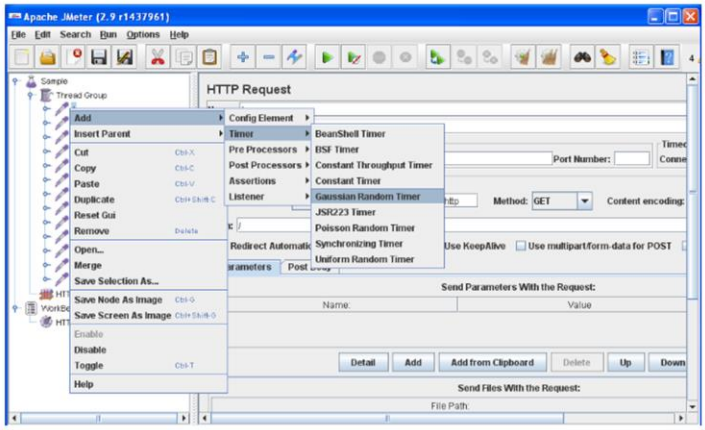


UNIVERSIDAD Ð SEVILLA

52

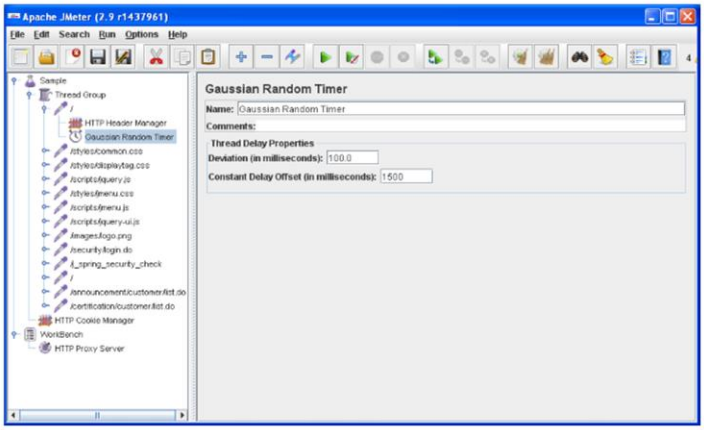
If you try to execute a script that has just been recorded, you'll easily find that it runs at full speed. Unfortunately, the proxy records the interactions with your web information system, but not the delays.

Add timers to your script



In order to introduce a delay in your scripts, you need to add timers to your entries. Click on every entry and select “Add > Timer”. There are a variety of timers available, but we recommend that you should use Gaussian timers.

Configure your timers



Configuring a Gaussian timer's very easy: you just need to provide a deviation and an offset. The terminology is a little odd, but it isn't that difficult to understand: a Gaussian timer with deviation *a* and offset *b*, delays the execution of a script for a period of time that is distributed according to a normal distribution with mean *b* and deviation *a*. Finding the appropriate values is not easy at all, but we recommend that you should set the deviation to 100 milliseconds and the offset to 1500 milliseconds; these figures typically work quite well.

NOTE: please, note that you need to introduce a delay only in the entries that are related to user interactions. For instance, after a user requests a page, he or she typically waits for a little until he or she interacts again. You don't have to introduce a delay when the interaction is performed automatically by the browser, e.g., to retrieve scripts, CSS files, or other resources.

It’s now time to learn how to add performance reports to your test cases.

Performance reports

UNIVERSIDAD Ð SEVILLA

56

A performance report's a report that provides a variety of performance measures that are computed while your test cases run. jMeter provides a variety of reports, but we're going to focus on two of them exclusively, namely: graph reports and aggregate reports.

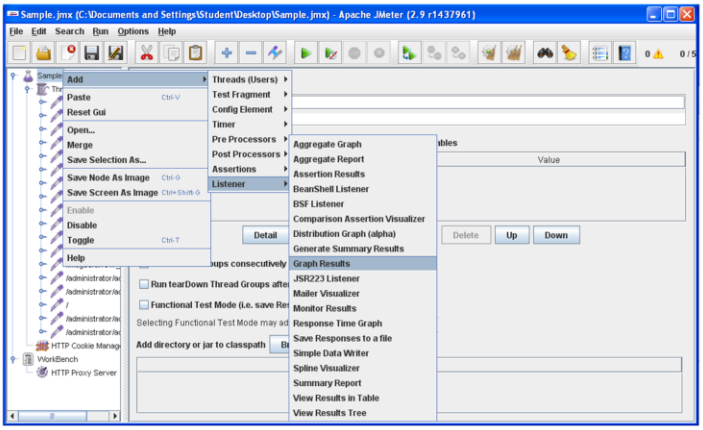
Performance reports

UNIVERSIDAD Ð SEVILLA

57

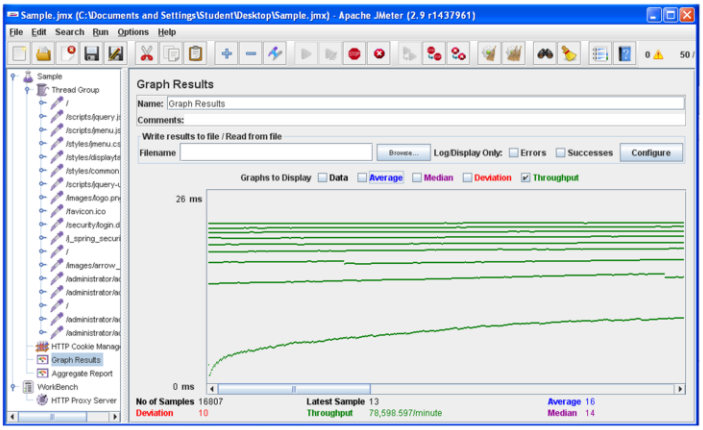
Let’s first report on graph reports.

Add a graph report



Reports are introduced by means of so-called listeners. A listener's an object that listens to the execution of your test cases, collects measures, and creates reports. Please, add a "Graph results" listener to your test case so that you can have a graph report.

A sample graph report

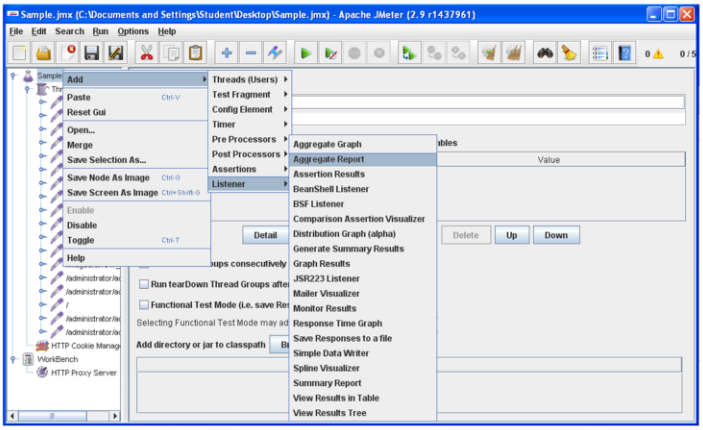


When you run your test case, you'll get something similar to the screenshot in this slide when you click on your graph report. The green line shows the throughput, that is, the average number of times per second, minute, or hour that jMeter is able to execute the script that you've recorded. In this chart, the throughput is roughly 78,500 executions per minute. Wow, that's quite an impressive figure! The benchmark was run on a virtual machine that was running on a commodity desktop computer. Each line represents an iteration; the line at the bottom clearly shows that the initial throughput is small because the threads are starting, and it increases as the workload increases and system caches start working. You can also show the average time to run the script (in blue), the median time (in magenta), and the deviation time (in red).

Performance reports

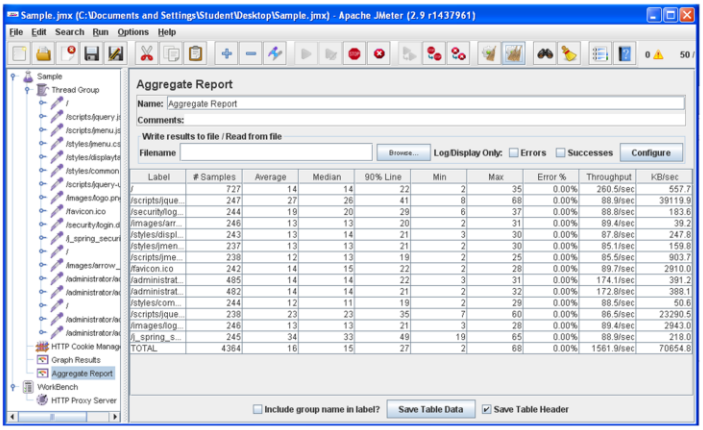
Let's now report on aggregate reports.

Add an aggregate report



Graph reports are very good to have an overall picture of your throughput, but you also need an aggregate report so that you can inspect the details. To add such a report click on “Add > Listener” and select “Aggregate Report”.

A sample aggregate report



UNIVERSIDAD D SEVILLA

62

An aggregate report shows the following information per URL requested to your system (times are measured in milliseconds):

- Label: it refers to a URL.
- Samples: it's the number of times that the corresponding URL's been requested.
- Average: it's the average time taken to serve a request to the corresponding URL.
- Median: it's the median time taken to serve a request.
- 90% line: it's the 90th percentile of the time taken to serve a request.
- Min: it's the minimum time that a request took.
- Max: it's the maximum time that a request took.
- Error %: it's the percentage of HTTP errors caught while requesting a URL. (An HTTP error is a response with a code different from 200 ...
- Throughput: it's the average number or request to a URL that are served per second.
- KB/sec: it's the average number of KiB/second that are sent to the client when serving a request to a URL.

It's important that you understand these measures, since they can provide an accurate overall picture of how your system's performing while you run your tests.

Finally, it's time to run your test cases.

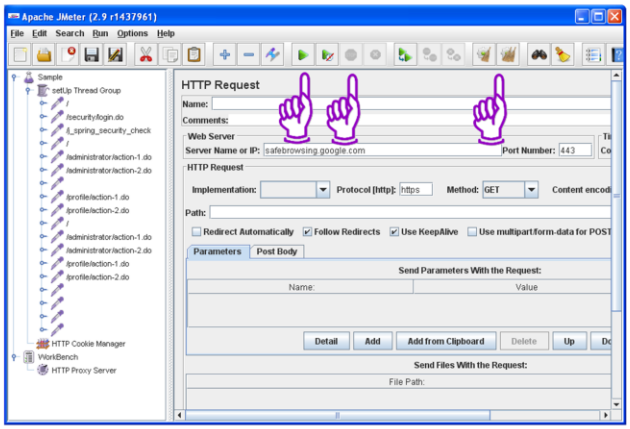
Let's try your test case

UNIVERSIDAD Ð SEVILLA

64

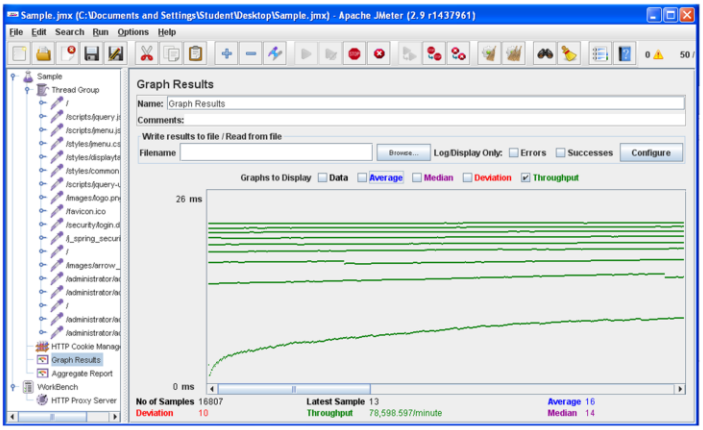
Cross your fingers! Very likely, this is the first time that your system's serving several concurrent users!

Learn these buttons



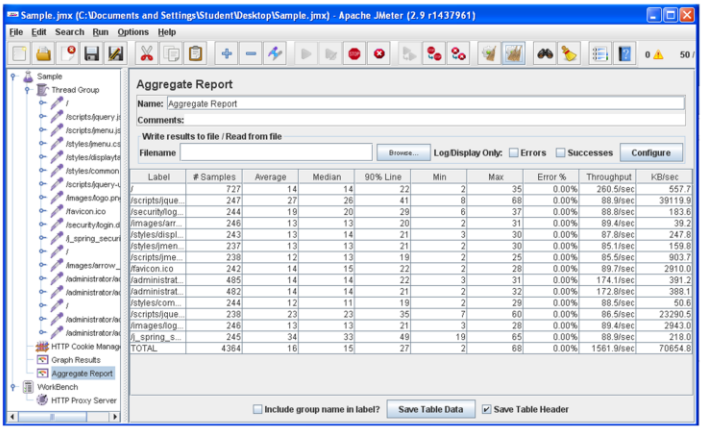
jMeter provides the typical green triangle button to start running a benchmark, a red stop button to stop it, and a clear button to clear the reports. It's important that you clear your reports whenever you restart a test case since, otherwise, the new results will be added to the previous ones.

Check the graph report



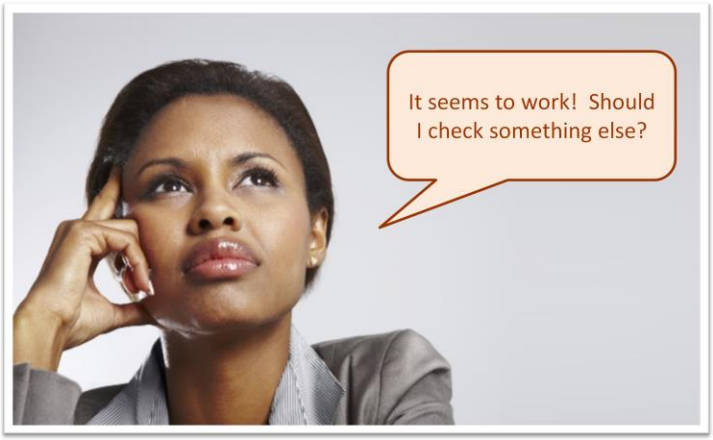
Click on the “Graph Results” report and you should see something very similar to this slide.

And also the aggregate report



Take also a look at your aggregate report.

This is a good question

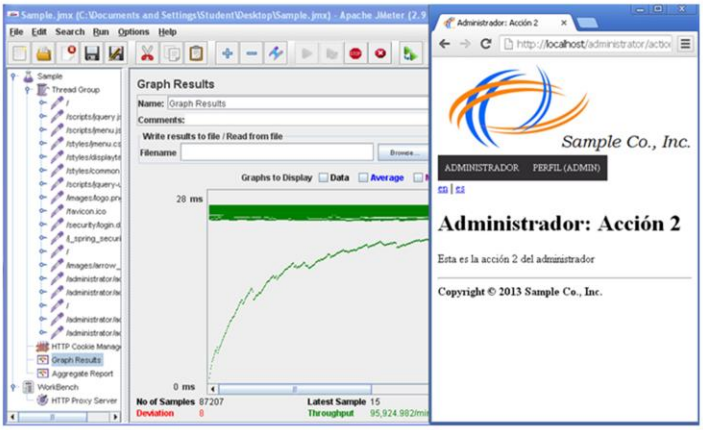


UNIVERSIDAD Ð SEVILLA

68

Usually, running your performance tests will take a little time. While they run, it's very important to check a couple of things that we're going to explore in the following slides.

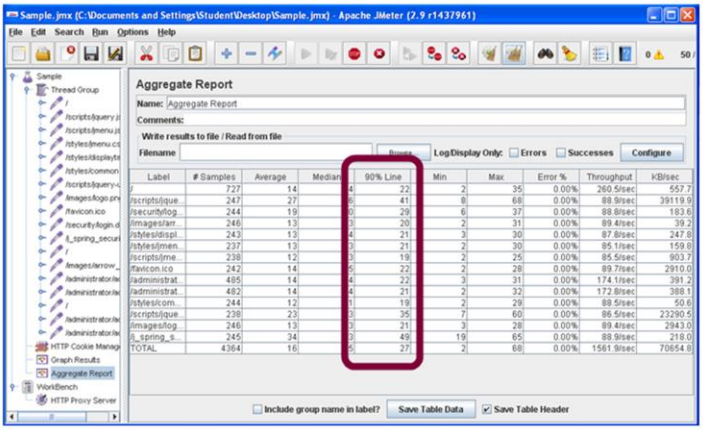
Check that the system's responsive



The first thing that you have to check is that your system keeps responsive while your test cases are running. You need to check that the system actually performs responsively, that it: you don't have to wait for one, two, or three seconds before it returns a response to an interaction. If that happens, then the workload is too high for your system.

NOTE: please, note that you need to make your browser for "http://localhost" to check that the system's responsive. If you make it for "http://www.acme.com", the browser will proxy the connection and it won't work because jMeter's proxy's off. Can you understand now why we configured the operating system not to proxy local addresses? Right!

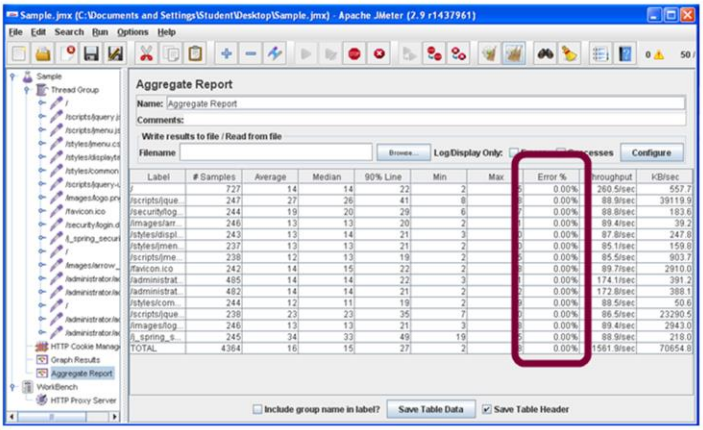
Check the 90% line



Next you should check that the average response time is not too high for the majority of users. Take a look at a measure called "90% Line" in your aggregate report; it measures the average time that a request to a URL takes to be served for 90% of your clients. For instance, the report in this slide shows that 90% of your clients have to wait for 22 ms to retrieve the index page of your web information system, 41 ms to retrieve a script, and so on. In total, the average time is 22 + 41 + 29 + … + 49 = 340 ms, that is, roughly 0.4 seconds. That's not a bad response time at all! That means that, very likely, this system can accept more users working concurrently without degrading its performance dramatically.

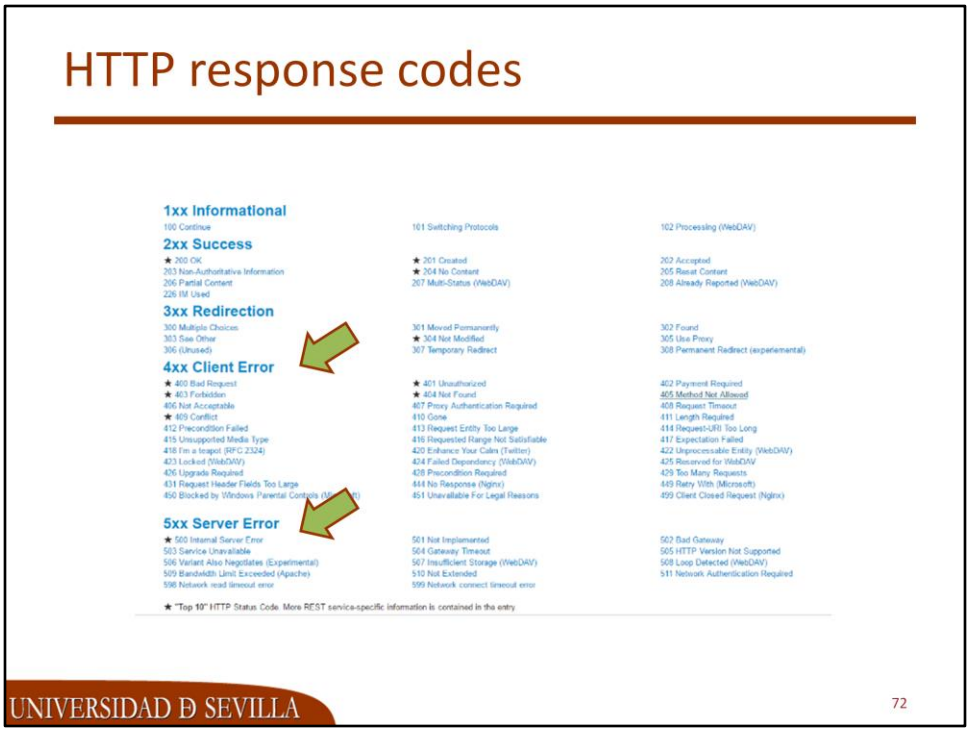
NOTE: note that the last line of the report seems to aggregate the sum of timings, but it is not computed correctly. Obviously, 22 + 41 + 29 + ... + 49 is not 27 ms. Sorry, you know that technology is far from perfect and jMeter's not an exception!

Check the HTTP errors



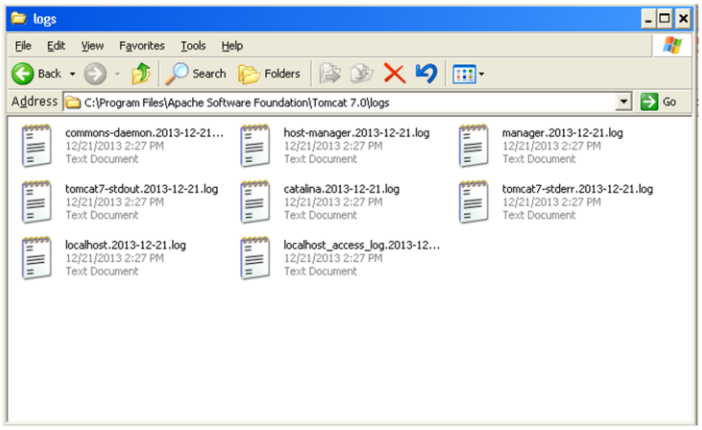
Finally, we strongly recommend that you should keep an eye on a measure called "Error %", which measures the number of HTTP errors that are caught by jMeter. An HTTP error happens when your system is far too overloaded to serve more requests. It's very important that your system doesn't return any HTTP errors during your tests! If you get them, then it'll be very difficult to debug them!

NOTE: if you have followed our guidelines, you should not get in trouble, and your system should be able to serve relatively high workloads. If you haven't, then it is very likely that your code's not ready to work in a concurrent environment, and then you're likely to get many HTTP errors. For instance, we are puzzled why many students use static variables. Have you used them? In that case, get ready for many concurrency problems that result in HTTP errors. We are also puzzled that many students load the whole database in memory several times to perform a task that can be very easily accomplished by means of a simple JPQL query. Have you done something similar? In that case, get ready, too, for many HTTP errors due to memory-related exceptions.

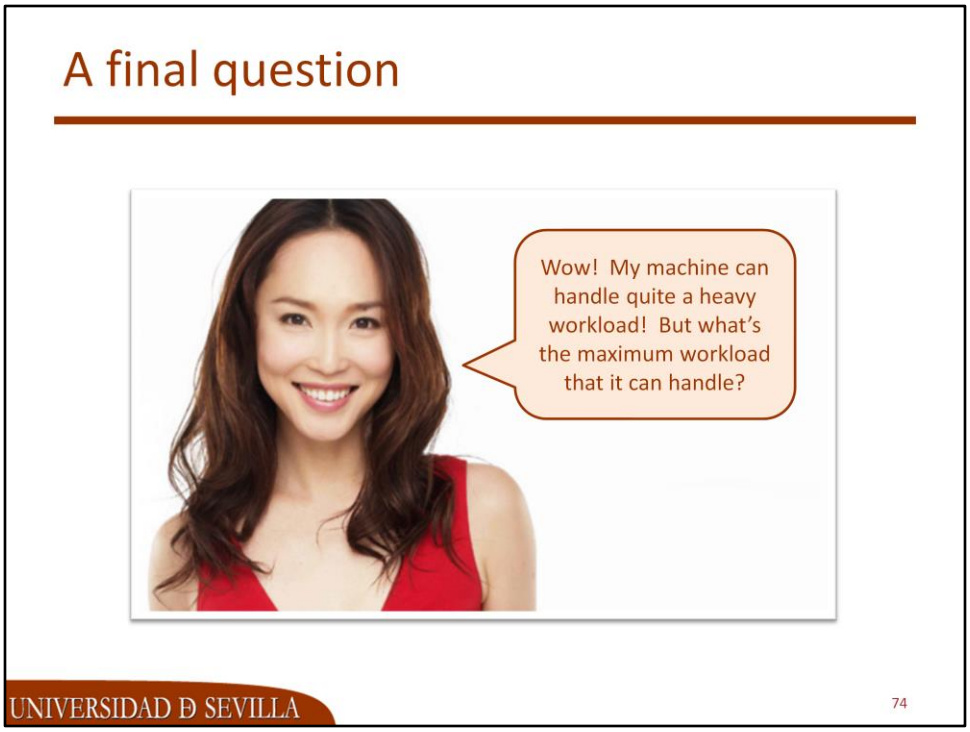


Every request to a web application results in a response whose meaning is encoded in an HTTP response code. The codes are grouped into the 1xx series (informational), the 2xx series (success), the 3xx series (redirection), the 4xx series (errors due to the client), and the 5xx series (errors due to the server). jMeter counts an error when it gets an HTTP response code in the 4xx or the 5xx series.

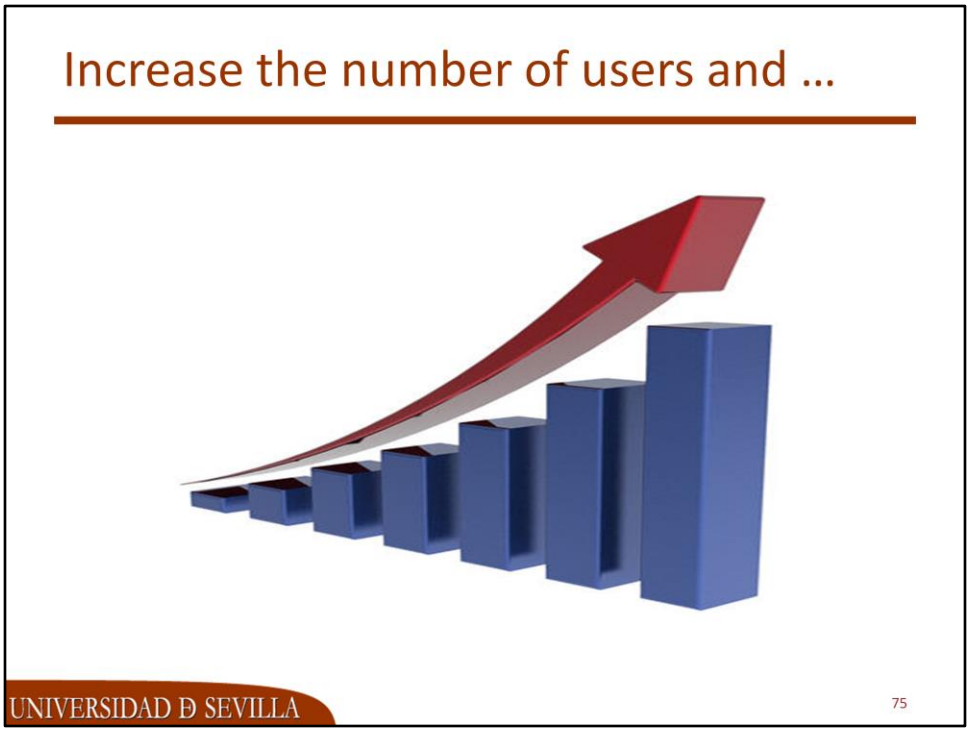
Peek at Tomcat's logs



If you get an HTTP error, you must find the reason why. Please, recall that there's only a way to diagnose such errors: you have to search for Tomcat's logs and go through them to find the problem. Recall that there's a folder called "logs" in Tomcat's installation directory. Tomcat's service logs information and error messages to the files in this folder. Please, search for the log files that correspond to the day on which you're working and analyse them.



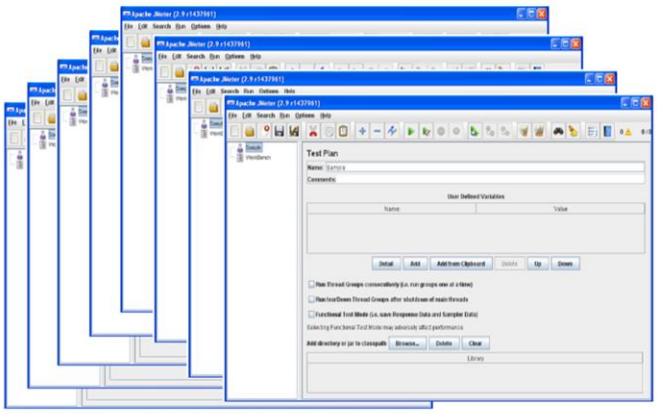
Before concluding this section, we think that it makes sense to raise a final question: how can you find the maximum workload that your system can handle? Answering this question is not difficult at all.



Repeat your test case, but increase the number of users little by little until the system starts becoming unresponsive, the 90% line measure is too high, or there are HTTP errors. That's the maximum workload that your system can handle regarding a test case.

NOTE: be very careful when increasing the number of users. If you increase it too much, that may collapse your system and render it totally unresponsive. In such cases, there's little you can do except for waiting for hours until it finishes serving the requests or rebooting it.

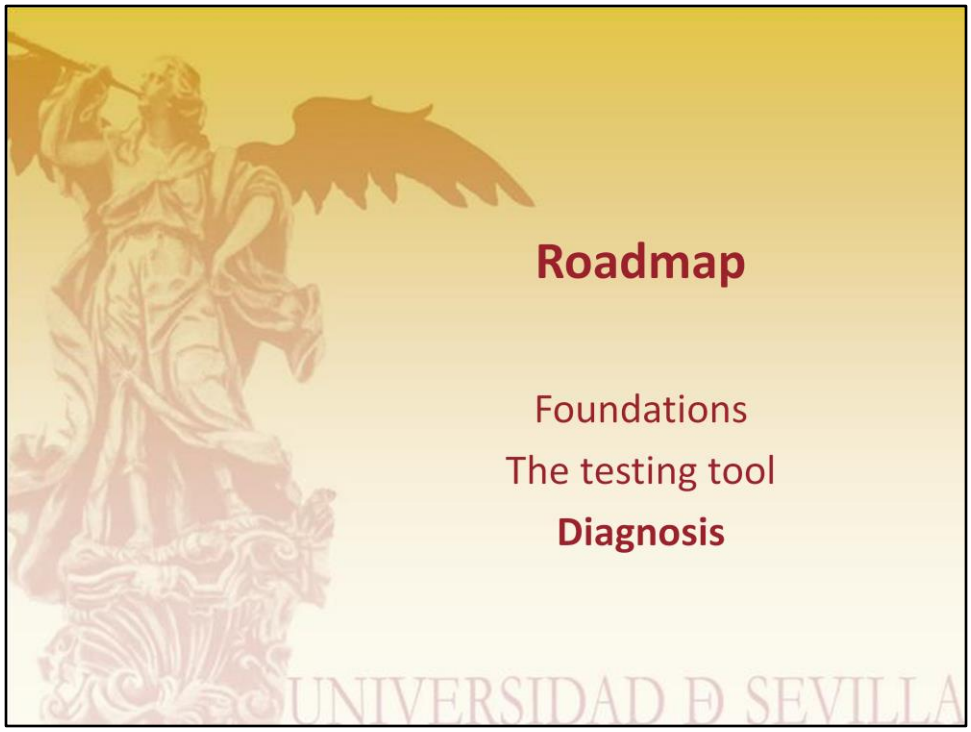
… and create new benchmarks



UNIVERSIDAD Ð SEVILLA

76

Just one benchmark isn't usually enough to measure the scalability of your system. You obviously need to create several benchmarks and each of them should focus on simulating the behaviour of a number of users who are executing typical use cases. The maximum workload that your system will be able to handle is the minimum workload it can handle across your different benchmarks.



Let’s finally report a little on diagnosis.

What's diagnosis about?

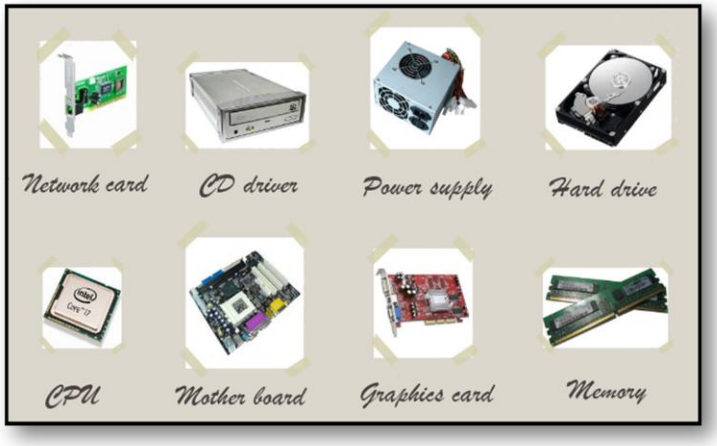
Diagnosis is about finding the reason why your system can't handle more workload

UNIVERSIDAD Ð SEVILLA

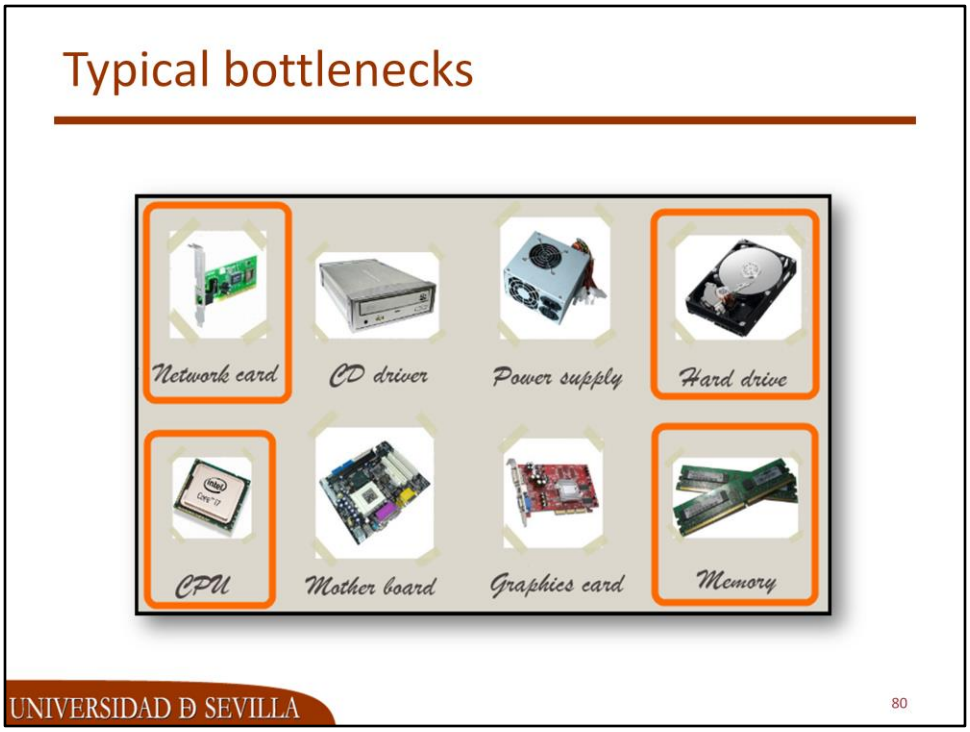
78

In the context of performance testing, diagnosis is about finding the reason why your system can't handle more workload. In other words, assuming that your application's been designed carefully so that it doesn't have any inefficiencies, the goal's to find out bottleneck components.

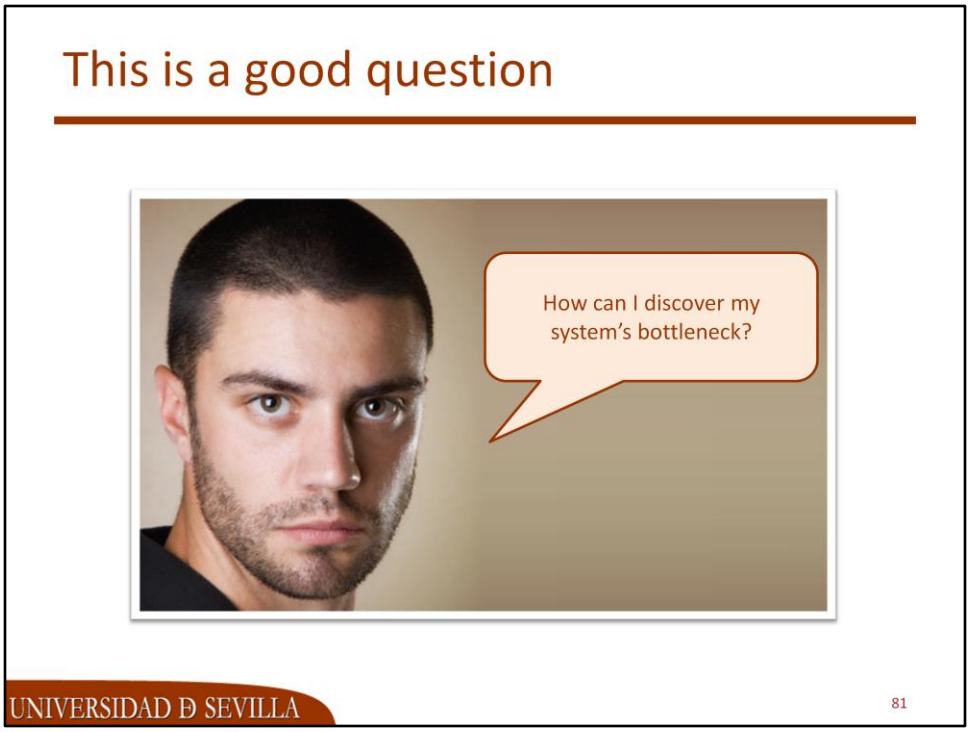
Components of a computer



This slide shows the typical components of which a typical computer is composed: a network card, a CD driver, a power supply, a hard drive, a CPU, a mother board, a graphics card, some memory chips… OK, the list isn't complete. It's just an summary of some key components.

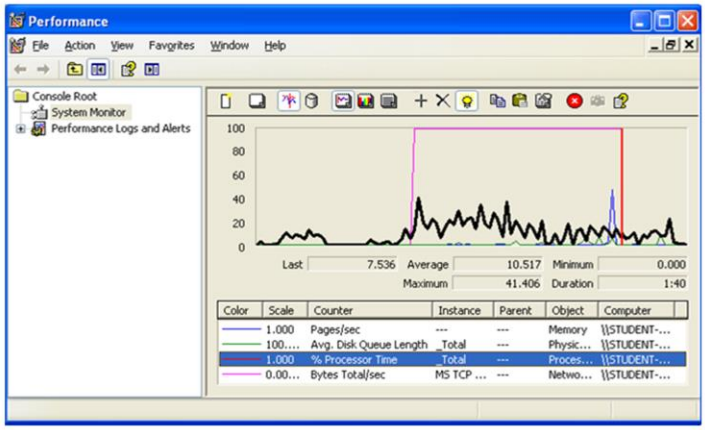


When a computer runs a web information system, there are a number of components that become typical bottlenecks. A bottleneck's a component that doesn't work fast enough; that is, it typically collapses while the other components are not fully used or even idle most of the time.



This is a good question: how can you discover which components of your system are bottlenecks.

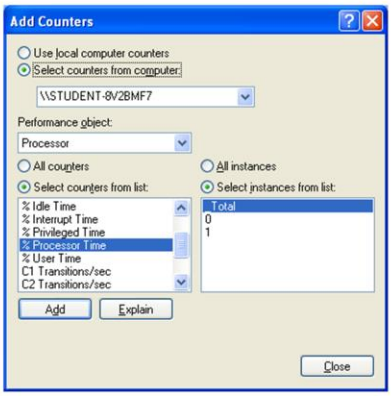
Use a performance meter



The answer's very simple: use a performance meter. In our pre-production configuration, the performance meter's an application that looks like in this slide. Open an administrator's shell and execute command "perfmon.exe". You need to add so-called performance counters to the performance meter so that you can monitor your system's activity while a performance test case is executed. In order to add performance counters, please, click on the "+" button in the button bar; unfortunately, there's no option in the main menu.

NOTE: there's not a universal performance meter. Please, find the most appropriate for your system.

Monitor your processor time



First, add a performance counter called “% Processor Time”. This counter measures the percentage of time that the processor spends to execute a user thread. It is the primary indicator of processor activity, and displays the average percentage of busy time observed during the sample interval.

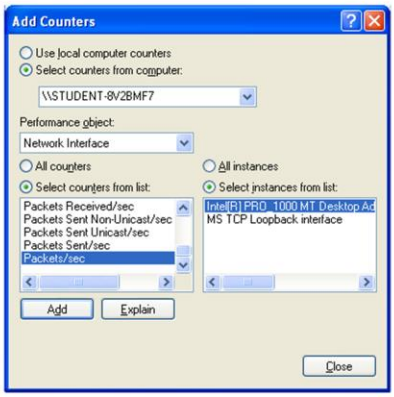
Monitor your disk activity

Now, add a performance counter called “Avg. Disk Queue Length”. It measures the average number of both read and write requests that were queued for the selected disk during the sample interval.

Monitor your memory

Next add a performance counter called “Pages/sec”. This performance counter measures the rate at which pages are read from or written to disk to resolve hard page faults. This counter is a primary indicator of the kinds of faults that cause system-wide delays. It includes pages retrieved to satisfy faults in the file system cache (usually requested by applications) and non-cached mapped memory files.

Monitor your network card



Finally, add a performance counter called “Packets/sec”. This counter provides an accurate estimate of the bandwidth that the network interface is delivering.

NOTE: there’s a counter called “Current bandwidth” that should provide a better estimate, but it doesn’t work well in a virtual machine. Note, too, that there are two instances of this counter: one that refers to the physical network card (“Intel ® Pro 1000MT” in this example), and another that refers to the loopback interface. Intuitively, you should use the instance that corresponds to the loopback interface if you run your tests on the pre-production configuration, but it doesn’t work in the virtual machine. Use the physical network interface all the time.

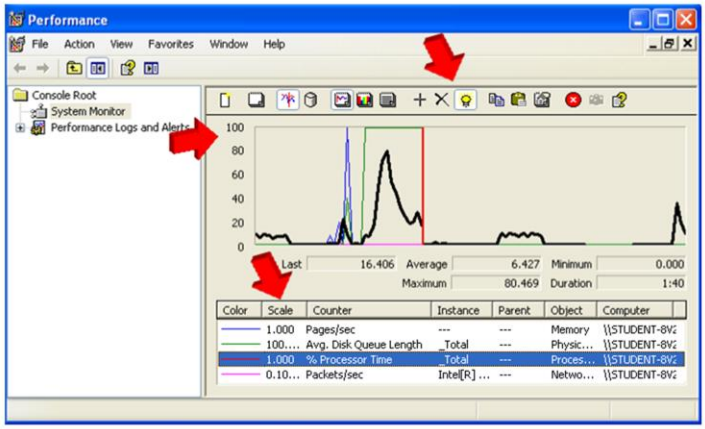
Nice to hear that!

UNIVERSIDAD Ð SEVILLA

87

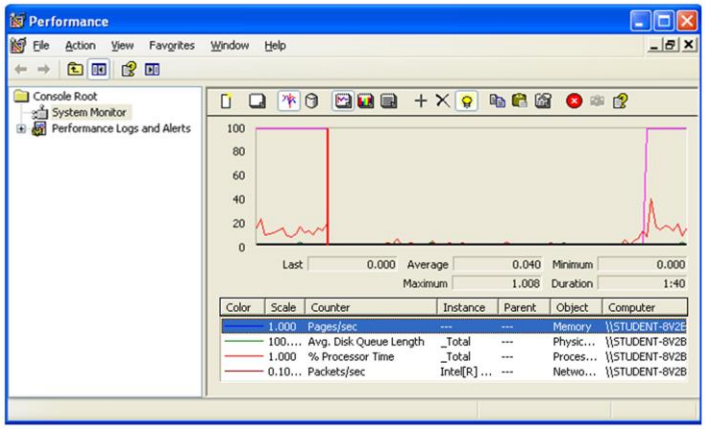
Typically, the previous performance counters are enough to diagnose a system and find its bottleneck. Let's now report on how to interpret the results that are provided by the performance meter.

The scale and the lamp



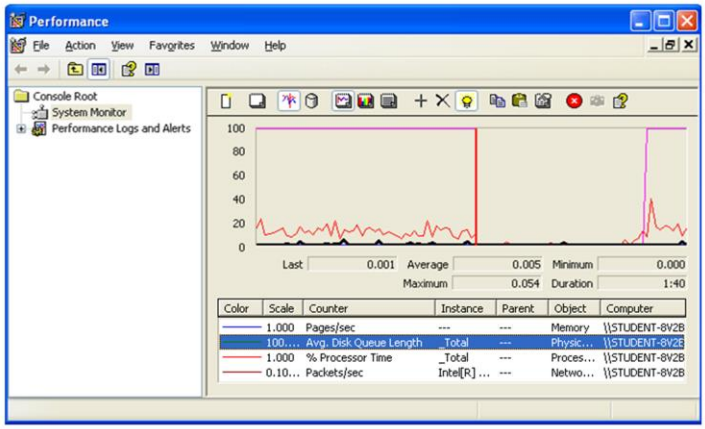
Interpreting the results of the performance meter is very easy. Realise that every performance counter has a scale, which is computed automatically; the scale is computed so that value 0 means that the corresponding component is idle and value 100 means that it's overloaded. In order to identify the counters easily, there's a lamp button that we suggest that you should activate always; when you activate this button, the performance counter that you select is represented as a bold black line.

Where's the problem? (I)



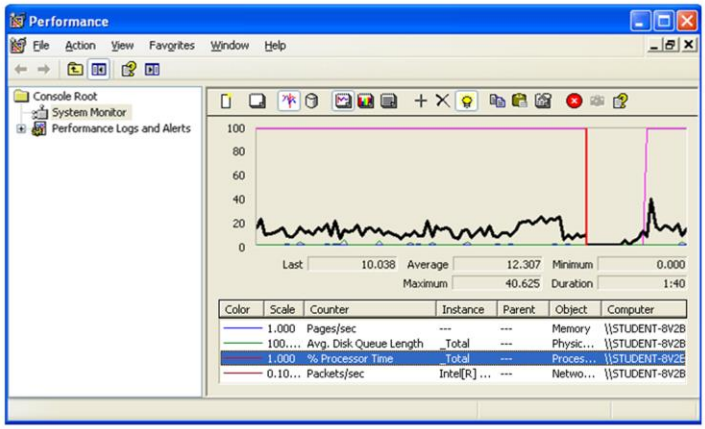
Assume that you're running your performance test cases and that you monitor your pre-production configuration. Is there a problem with your memory? Obviously not. Note that the "Pages/sec" counter is roughly zero all the time, which means that your memory is very, very far from becoming a bottleneck.

Where's the problem? (II)



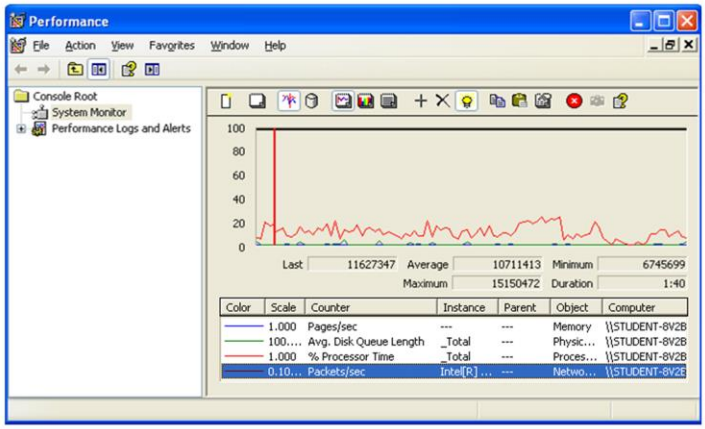
Is there a problem with your disk? No, not at all. Note that the "Avg. Disk Queue Length" performance counter is very close to zero all the time. That means that, other things equal, your disk might be able to serve a lot more requests.

Where's the problem? (III)

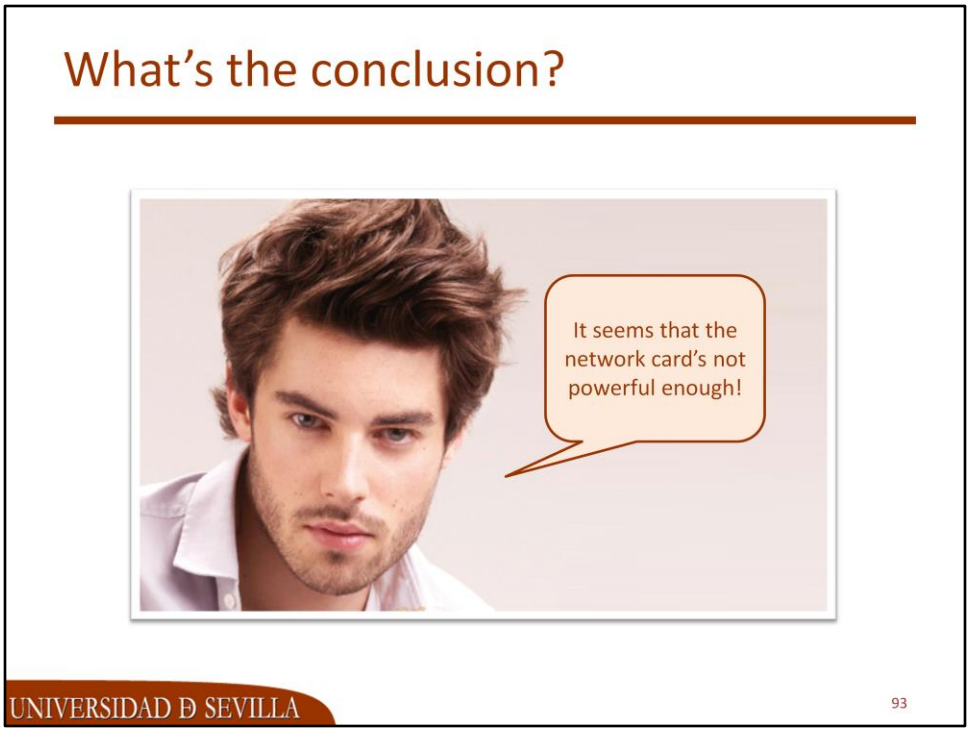


Neither seems it to be a problem with the CPU. Note that its workload's about 20%; in other words, it's very far from becoming a bottleneck.

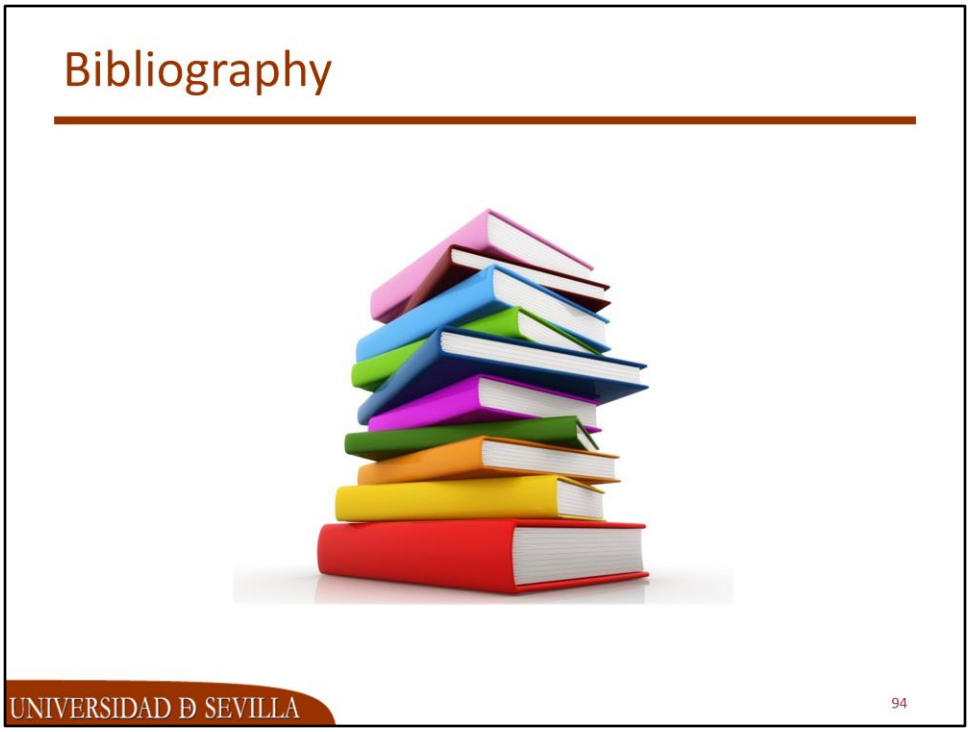
Where's the problem? (IV)



What about the network interface? Note that it's been running at 100% its bandwidth all the time; that means that this component can't handle the current workload.



Right, that's a conclusion: it seems that the bottleneck's your network card. In other words: replacing it with a more powerful one might boost your computer, which might help it serve a lot more requests per second. You don't have to update the CPU, the disks, or the memory, which are costly components, only your network card.



Unfortunately, there are not any electronic resources on performance testing available at the USE's library. For those of you who are seriously interested in this topic, we recommend that you should take a look at the following book:

Performance Testing With JMeter 2.9
Bayo Erinle
Packt Publishing, 2013

The official web site of jMeter is available at http://jmeter.apache.org/; there you can find additional free learning resources, including a user manual and best practices. If you need some help with the HTTP codes, please, take a look at the list that is available at the official source https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html; if you don't think that source is easy to grasp, then take a look at http://www.restapitutorial.com/httpstatuscodes.html.

Thanks for attending this lecture!