**Performance Test**

**[Performance Test](#_Toc393794526)** [1](#_Toc393794526)

[1 Performance Test Strategy 4](#_Toc393794527)

[1.1 Identify the Test Environment 5](#_Toc393794528)

[1.1.1 Development Environment 5](#_Toc393794529)

[1.1.2 Production Environment 7](#_Toc393794530)

[1.2 Identify Performance Acceptance Criteria 8](#_Toc393794531)

[1.3 Plan and Design Tests 8](#_Toc393794532)

[1.3.1 Performance requirement survey 8](#_Toc393794533)

[1.3.2 Key scenario & navigation path 9](#_Toc393794534)

[1.3.3 Delay Time 9](#_Toc393794535)

[1.3.4 Determining Test Data to complete Key scenarios 10](#_Toc393794536)

[1.4 Configure the Test Environment 12](#_Toc393794537)

[1.4.1 Prepare test environment 12](#_Toc393794538)

[1.4.2 Prepare data maintained by target system 12](#_Toc393794539)

[1.4.3 Prepare Tools 13](#_Toc393794540)

[1.5 Implement the Test Design 14](#_Toc393794541)

[1.6 Execute the Test 14](#_Toc393794542)

[1.7 Analyze Results, Report, and Retest 15](#_Toc393794543)

[1.8 Summary 15](#_Toc393794544)

[2 Jmeter User Manual 16](#_Toc393794545)

[2.1 Introduction 16](#_Toc393794546)

[2.1.1 Jmeter download 16](#_Toc393794547)

[2.1.2 Jmeter plugin 16](#_Toc393794548)

[2.2 Elements of a Test Plan 17](#_Toc393794549)

[2.2.1 Thread Group 17](#_Toc393794551)

[2.2.2 Test Fragments 18](#_Toc393794552)

[2.2.3 Transaction Controllers 18](#_Toc393794553)

[2.2.4 Listeners 18](#_Toc393794554)

[2.2.5 Timers 18](#_Toc393794555)

[2.2.6 Assertions 18](#_Toc393794556)

[2.2.7 BeanShell Sampler 18](#_Toc393794557)

[2.2.8 Http proxy server 18](#_Toc393794558)

[2.3 Building a Web Test Plan 19](#_Toc393794559)

[2.3.1 Add Thread Group 19](#_Toc393794560)

[2.3.2 Setting the Parameters of thread group 19](#_Toc393794561)

[2.3.3 Produce Scenario & Workload 20](#_Toc393794562)

[2.3.4 Make sure Construction of the scripts 22](#_Toc393794563)

[2.3.5 Record scripts 23](#_Toc393794564)

[2.4 Parameterization 27](#_Toc393794565)

[2.4.1 Three primary concern 27](#_Toc393794566)

[2.4.2 Duplication of data: 30](#_Toc393794567)

[2.4.3 Associated date range： 30](#_Toc393794568)

[2.4.4 Commonly used parameterization function – use Dynamic Data 30](#_Toc393794569)

[2.5 Add Listener 33](#_Toc393794570)

[2.5.1 Aggregate Report 33](#_Toc393794571)

[2.5.2 View Results Tree 34](#_Toc393794572)

[2.6 Run our JMeter Script 36](#_Toc393794573)

[2.6.1 Run Jmeter Script 36](#_Toc393794574)

[2.6.2 Validate Jmeter script 36](#_Toc393794575)

[2.7 View Results 38](#_Toc393794576)

[2.7.1 JMeter results 38](#_Toc393794577)

[2.7.1.1 The result of View Result Tree 39](#_Toc393794580)

[2.7.1.2 The result of View Result in Table 39](#_Toc393794581)

[2.7.2 JMeter Plugin Results 42](#_Toc393794582)

[2.7.2.1 Active Threads Over Time Listener 42](#_Toc393794583)

[2.7.2.2 Response Times vs Threads 42](#_Toc393794584)

[2.7.2.3 Graph Response Times vs Threads 43](#_Toc393794585)

[2.7.2.4 Transaction Throughput vs Threads 44](#_Toc393794586)

[2.7.2.5 Response Times Distribution 44](#_Toc393794587)

[2.7.2.6 Response Times Over Time 44](#_Toc393794588)

[2.7.2.7 Transactions Per second 46](#_Toc393794589)

[3 Performance Testing techniques 47](#_Toc393794592)

[3.1 Endurance Test 48](#_Toc393794593)

[3.1.1 Testing Goal 48](#_Toc393794594)

[3.1.2 Testing Approach 48](#_Toc393794595)

[3.1.3 Key Criteria to Monitor 49](#_Toc393794596)

[3.1.4 Expect Result: 49](#_Toc393794597)

[3.1.5 Analyze Result 49](#_Toc393794598)

[3.2 Load Test 52](#_Toc393794599)

[3.2.1 Normal load Test 52](#_Toc393794600)

[3.2.2 Increasing Load Test 56](#_Toc393794601)

[3.3 Stress Test 60](#_Toc393794603)

[3.3.1 Application stress testing 61](#_Toc393794604)

[3.3.2 Transactional stress testing 62](#_Toc393794605)

[3.4 Monitor Key Metrics 66](#_Toc393794606)

[3.4.1 PerfMon Metric Collector 68](#_Toc393794607)

[3.4.2 Command of Vmstat, nmon, topas 68](#_Toc393794608)

[3.4.3 Aggregated Report, Response Time Distribution, Graph Response Times vs Threads ,Transaction Throughput vs Threads 68](#_Toc393794609)

[3.4.4 Throughoutput 69](#_Toc393794610)

[3.4.5 WAS PMI 69](#_Toc393794611)

[3.4.6 JavaMelody 69](#_Toc393794612)

# Performance Test Strategy

Performance testing is the most efficient way to find serious performance issue before the application going to higher platforms. The performance testing approach used in this guide consists of the following activities:



## Identify the Test Environment

Identify the physical test environment and the production environment as well as the tools and resources available to the test team. Having a thorough understanding of the entire test environment at the outset enables more efficient test design and planning and helps you identify testing challenges early in the project. As below take Argus ZDEV system for example:

### Development Environment

The hardware and software setting of development environment are listed as below:

* 1. 20\*CPUs(PowerPC\_Power7), 3300 MHz CPU clock rate, 16GB memory
  2. OS： AIX 7.1.1.16 TL00
  3. Web sphere Application Server - ND, 7.0
  4. DB2 version: 10.1

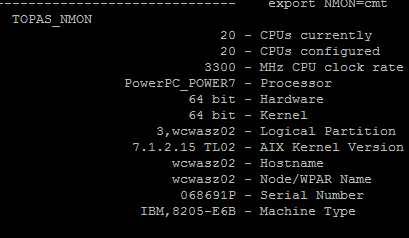




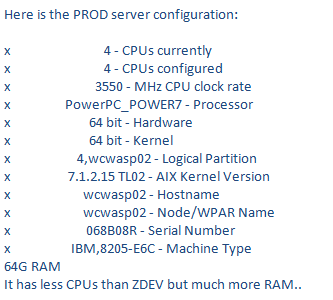
Figure 1 - Logical view of Argus Pharmacy-Network Development Environment



Figure 2 - Physical view of Argus Pharmacy-Network Development Environment

### Production Environment

The hardware and software setting of production environment are listed as below:



## Identify Performance Acceptance Criteria

Identify the response time, throughput, and resource utilization goals and constraints. In Argus system, no specific performance goals are provided from customers. So we need to determine it by our own.

In Argus, the current reference performance benchmark of response time is as below:

Excellent (<1s);

Good (1s-2s);

Acceptable (2s-5s), this is the performance acceptance criteria;

Fair (5s-10s);

Poor (>10s), in which Acceptable (2s-5s).

Our current performance acceptance criteria is as below, which will evolve as more test results can be referenced:

* Under the normal work load of concurrent user volume, the resource utilization(CPU, memory, Disk IO, etc..) should be less than 80%, the response time of 90% requests should be at "Acceptable" level or better.
* No key action is worse than "Acceptable" level.

Additionally, identify project success criteria that may not be captured by those goals and constraints; If test result is out of acceptance criteria, using performance tests, you can get to evaluate what combination of configuration settings will result in the most desirable performance characteristics. It helps us to config test environment as the settings to make test result up to acceprance criteria. The project is still successful.

## Plan and Design Tests

Identify key scenarios, determine variability among representative users and how to simulate that variability, define test data, and establish metrics to be collected. Consolidate this information into one or more models of system usage to be implemented, executed, and analyzed. It includes Key scenario & navigation path/User delay/…

### Performance requirement survey

Before performance testing, a performance requirement survey must be made for the team to get to know better about what is the volume of the user the application may have, how much data the database table may contain, general distribution of key business scenarios …etc, it is used to follow the template to facilitate the performance survey.



### Key scenario & navigation path

The following doc describes the key scenarios & workload characterization to be used for evaluating the performance of Argus Pricing Option system. The more accurately users are modeled, the more reliable performance test results will be. The workload is estimated by the Argus team, provided in form of above [Performance requirement survey](#_Performance_requirement_survey).



### Delay Time

Identify user delay time. The more accurately users are modeled, the more reliable performance test results will be. One frequently overlooked aspect of accurate user modeling is the modeling of user delays. Below are some modeld user delays, In practice we use item 3 in performance test:

1. Real delay time: In theory, it is more accurate using real delay time according to user actual behavior on production, but it is hard to collect this information by production log.
2. Generated delay time: Most of the record tools can generate delay time according to user real waiting time during record time. It will generate delay time automatically reference record delay time. If you want to have each thread pause for the same amount of time for the request, use this timer.
3. Gaussian Random delay time: This timer pauses each thread request for a random amount of time, with most of the time intervals ocurring near a particular value. The variation around constant offset has a gaussian curve distribution. It is more reliable as real world. (You can get the particular value through method 2)).

How to Use Jmeter to define delay time, refer to（[Think Time - Add Gaussian Random Timer](#_Think_Time_-)）.

### Determining Test Data to complete Key scenarios

Once you have a list of key scenarios, you will need to determine test data for users to actually accomplish the tasks or activities related to those scenarios. The more accurately test data are modeled, the more reliable performance test results will be. Design test data for each scenario according to performance requirement. Never overlooked aspect of accurate user modeling is the modeling of Test Data. Or you will get optimistic or pessimism result.

|  |  |  |  |
| --- | --- | --- | --- |
| Application | Scenario | Data size(general) | Data size(Max) |
| Drug List | Drug list search result |  |  |
| Drug list entry search result |  |  |
| Drug list details -- entries tab | 20 | 100 |
| Drug list mass change search result |  |  |
| Drug list mass change -- Entries tab |  |  |
| Drug list mass change -- Criteria tab |  |  |

As above, one drug list component has 20 entries generally, so design test drug list components with about 20 entries generally will lead to appropriate test result. In this case, if you design test data just with 2 or 3 entries, you might get more optimistic result.

One drug list component can have max 100 entries, so design several components with 100 entries, which will be used in stress test.

In practice, note below tips:

1. Create a new customer set, which will be used only for application performance test.

2. Create independent components for each scenario, view/update/delete/…

3. When recording script, you just select one data item for each scenario to record. Do not use the same data to isolate the impact.

4. I Argus project, we found add /delete script only run successfully when test data contain the same number of entries. So record Add entry operation with Delete entry operation, to make sure the number of entries stays same as original time. I There might have many reasons. One is the id of add/delete/save button get changed after 1 entries added successfully, another one is the validation cannot be fitted if adding another entry after 1 entries added successfully. So record Add operation with Delete operation to make sure the number of entries stays same as original time. In case your script can do add/delete operation no matter how many entries are contained, skip this notice.

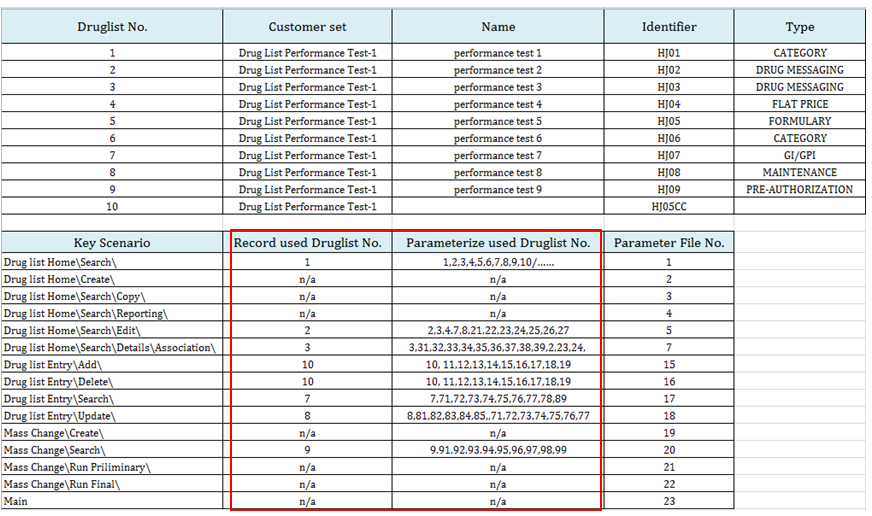
5. Use DB sql to add and remove test data. Because test data might become unuseful as a result of error/debug/concurrency. All of this might make your script failed during the next test. So test data should be inserted before test and removed after test.

6. During performance test, use dynamic test data, because the same data repeatedly will frequently lead to invalid performance test results. It means prepare more than one test data to accomplish the same tasks or activities related to the same key scenarios. It also means use random data as parameter during test. Refer to [Commonly used parameterization function – use Dynamic Data](#_Commonly_used_parameterization). And make sure your script run successfully on all of the test data.

8. Make sure some data are used for more than one scenario. It is realism that one user search for the drug list while another user is editing it. It is used to test concurrency performance issues on the same data.

9. Some scenarios do not need unique test data, like create/copy. But you will need random data to avoid duplication error.

Exp:



## Configure the Test Environment

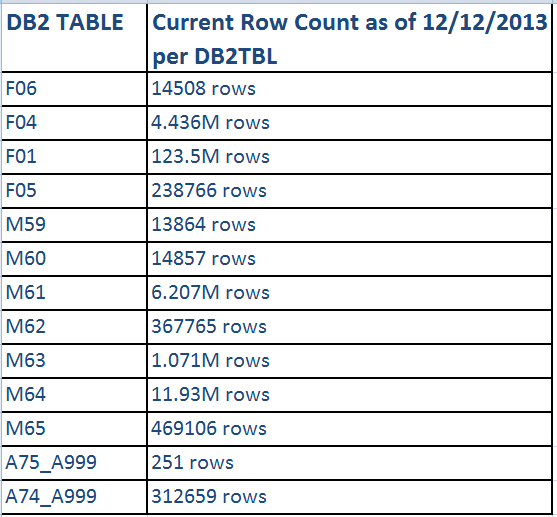
Prepare the test environment, tools, and resources necessary to execute each strategy as features and components become available for test. Ensure that the test environment is instrumented for resource monitoring as necessary.

### Prepare test environment

According to [Identify the Test Environment](#_Identify_the_Test)

### Prepare data maintained by target system

It generally including data in database or file server. The volume of the dataset is a key factor which affects the performance testing result, The inserted data should have business meaning. Number of the data records should be similar like production environment. Exp:



Note:

1. Relationship in “one-to –many should reflect real business requirement. Exp: in Drug List application, how many entries does one drug list contain generally or at most contain affects system performance. In Benefits application, how many data ranges one component contain generally or at most affects system performance. This information can be calculated generally from above tables or achieved from the feedback of Performance Requirement above.
2. Especially when designing and debugging performance tests, test databases can become dramatically overloaded with data. Periodically check to see if the data base is storing unrealistic volumes of data for the situation you are trying to simulate.

### Prepare Tools

For Jmeter tool, refer to [Jmeter User Manual](#_Generate_Script_with)

For tools to monitor key metrics, refer to [Monitor Key Metrics](#_Monitor_Key_Metrics)

## Implement the Test Design

Develop the performance tests in accordance with the test design. We use Jmter to implement the test design, refer to [Jmeter User Manual](#_Jmeter__User)

## Execute the Test

Validate and run your tests.

1) Validate the test environment and test script:

Validate the test environment through some smoke test to ensure no system error found.

Validate test scripts to check if correct metrics are being collected, and if the test script design is correctly simulating workload characteristics. Refer to [Validate Jmeter script](#_Validate_Jmeter_script)

2) Run tests:

Performance test is a generic term that can refer to many different types of performance-related testing. [Performance Test](#_Performance_Test) addresses several common types or categories of performance-related testing, each of which addresses a specific problem area and provides its own benefits, risks, and challenges. To know the detail approaches for different test techniques, refer to:

[Endurance Test](#_Endurance_Test)

[Load Test](#_Load_Test_1)

[Stress Test](#_Stress_Test)

3) Generate result， refer to:

[JMeter results](#_JMeter_results)

[JMeter-Plugin results](#_JMeter-Plugin_results)

[Monitor Key Metrics](#_Monitor_Key_Metrics)

4) Create Baseline or benchmark

## Analyze Results, Report, and Retest

Consolidate and share results data. Analyze the data to get useful information for performance analyze or tuning. [Performance Testing techniques](#_Performance_Test) addresses several common types or categories of performance-related testing. For each test, it introduces how to analyze result. If any potential issue is discovered, retest to collect more useful information.

Note:

Key Mathematic Principles for Performance Testers: Members of software development teams, developers, testers, administrators, and managers alike need to know how to apply mathematics and interpret statistical data in order to do their jobs effectively. It includes:

Average

Percentiles

Medians

Normal Values

Standard Deviations

Uniform Distributions

Normal Distributions

Performance Test Reporting Fundamentals(TODO later)

## Summary

Now you have a general view of performance test activities, following sections will describe performance test in detail separately, and focus on how to implements each segment with Jmeter tool. The doc will be used as user guide to do web application performance test with Jmeter.

In Chapter 2 [Jmeter User Manual](#_Generate_Script_with) , it introduces implementing test design with Jmeter tool, how to execute tests, how to analyze result.

In Chapter 3 [Performance Testing techniques](#_Performance_Testing_techniques), it introduces how to execute your performance test on different purpose and target on activity 6 and 7 in detail. The chapter addresses several common types or categories of performance-related testing. It is a generic term that can refer to many different types of performance-related testing: [Endurance Test](#_Endurance_Test), [Load Test](#_Load_Test_1), [Stress Test](#_Stress_Test). Each of them addresses a specific problem area and provides its own benefits, risks, and challenges.

# Jmeter User Manual

## Introduction

Apache JMeter is a 100% pure Java desktop application designed to load test client/server software (such as a [web application](http://jmeter.apache.org/usermanual/build-web-test-plan.html) ). It may be used to test performance both on static and dynamic resources such as static files, Java Servlets, CGI scripts, Java objects, [databases](http://jmeter.apache.org/usermanual/build-db-test-plan.html) , [FTP servers](http://jmeter.apache.org/usermanual/build-ftp-test-plan.html) , and more. JMeter can be used to simulate a heavy load on a server, network or object to test its strength or to analyze overall performance under different load types. Additionally, JMeter can help you regression test your application by letting you create test scripts with [assertions](http://jmeter.apache.org/usermanual/test_plan.html" \l "assertions) to validate that your application is returning the results you expect. For maximum flexibility, JMeter lets you create these assertions using regular expressions. But please note that JMeter is not a browser.

### Jmeter download

This project is a сustom set of plugins for Apache JMeter, not affiliated with Apache Software Foundation.( <http://code.google.com/p/jmeter-plugins/>)

### Jmeter plugin

You need to run JMeter 2.8 or above with a JRE 1.6 or above to use the plugins.

For zip files refer to http://172.16.5.12/svn/argus/common/Architect/Architecture Action List/5. Performance Tuning/Jmeter



Download latest version refer to [http://jmeter-plugins.org/#/](http://jmeter-plugins.org/" \l "/)

Unzip and put all jar under Jmeter Install directory\lib\ext, exp: D:\apache-jmeter-2.9\apache-jmeter-2.9\lib\ext. Then restart Jmeter.

JMeter-Plugins version 1.0.0 and newer are shipped via three distribution ZIPs:

JMeterPlugins-x.x.x.zip - contains plugins, must be unpacked into lib/ext directory inside JMeter

JMeterPlugins-libs-x.x.x.zip - contains additional JARs used by JSON/HBase/Hadoop/etc plugins, must be upnacked into lib directory inside JMeter

ServerAgent-x.x.x.zip - contains server resource monitoring agent to use with PerfMon Metrics Collector plugin, standalone utility

This project is a сustom set of plugins for Apache JMeter, not affiliated with Apache Software Foundation.( http://code.google.com/p/jmeter-plugins/)

## Elements of a Test Plan

The Test Plan object has a checkbox called "Functional Testing". If selected, it will cause JMeter to record the data returned from the server for each sample. If you have selected a file in your test listeners, this data will be written to file. This can be useful if you are doing a small run to ensure that JMeter is configured correctly, and that your server is returning the expected results. The consequence is that the file will grow huge quickly, and JMeter's performance will suffer. This option should be off if you are doing stress-testing (it is off by default).

If you are not recording the data to file, this option makes no difference.

You can also use the Configuration button on a listener to decide what fields to save.



### Thread Group

Reference to <http://jmeter.apache.org/usermanual/test_plan.html> ,4.1 Thread Group.

### Test Fragments

Reference to <http://jmeter.apache.org/usermanual/test_plan.html> ,4.2.3 Test Fragments.

### Transaction Controllers

Reference to

[http://jmeter.apache.org/usermanual/component\_reference.html#Transaction\_Controller](http://jmeter.apache.org/usermanual/component_reference.html" \l "Transaction_Controller) ,

### Listeners

Reference to [http://jmeter.apache.org/usermanual/component\_reference.html#listeners](http://jmeter.apache.org/usermanual/component_reference.html" \l "listeners)

Listeners.

### Timers

Reference to <http://jmeter.apache.org/usermanual/test_plan.html> ,4.4 Assertions.

### Assertions

Reference to <http://jmeter.apache.org/usermanual/test_plan.html> , 4.5 Assertions.

### BeanShell Sampler

Reference to

[http://jmeter.apache.org/usermanual/component\_reference.html#BeanShell\_Sampler](http://jmeter.apache.org/usermanual/component_reference.html" \l "BeanShell_Sampler) , 18.1.10 BeanShell Sampler.

### Http proxy server

Reference to

[http://jmeter.apache.org/usermanual/component\_reference.html#HTTP\_Proxy\_Server](http://jmeter.apache.org/usermanual/component_reference.html" \l "HTTP_Proxy_Server)

## Building a Web Test Plan

### Add Thread Group

The first step you want to do with every JMeter Test Plan is to add a Thread Group element. The Thread Group tells JMeter the number of users you want to simulate, how often the users should send requests, and the how many requests they should send.Go ahead and add the ThreadGroup element by first selecting the Test Plan, clicking your right mouse button to get the Add menu, and then select Add --> ThreadGroup.You should now see the Thread Group element under Test Plan. If you do not see the element, then "expand" the Test Plan tree by clicking on the Test Plan element.

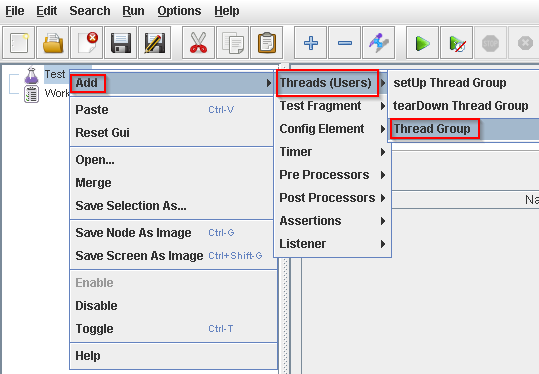


Figure 4

### Setting the Parameters of thread group

Start by providing a more descriptive name for our Thread Group. In the name field, enter JMeter Users.

Next, increase the number of users (called threads) to 1.

In the next field, the Ramp-Up Period, leave the the default value of 1 seconds. This property tells JMeter how long to delay between starting each user. For example, if you enter a Ramp-Up Period of 5 seconds, JMeter will finish starting all of your users by the end of the 5 seconds. So, if we have 5 users and a 5 second Ramp-Up Period, then the delay between starting users would be 1 second (5 users / 5 seconds = 1 user per second). If you set the value to 0, then JMeter will immediately start all of your users.

Finally enter a value of 2 in the Loop Count field. This property tells JMeter how many times to repeat your test. If you enter a loop count value of 1, then JMeter will run your test only once. To have JMeter repeatedly run your Test Plan, select the Forever checkbox.

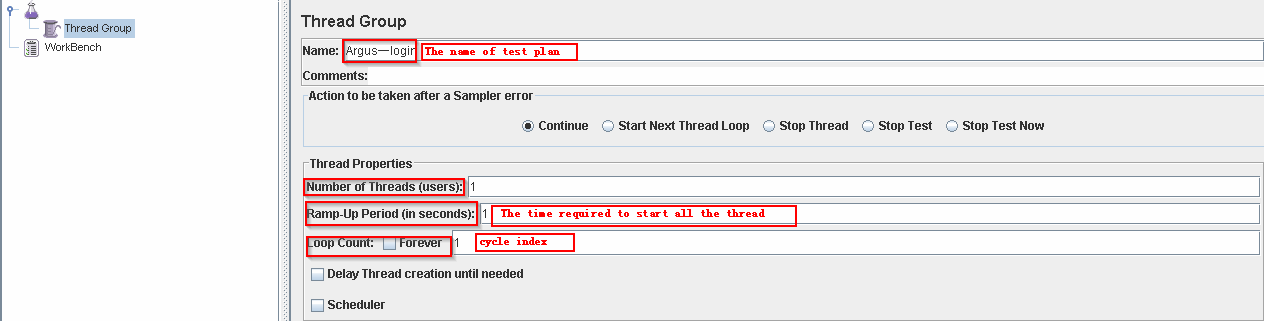
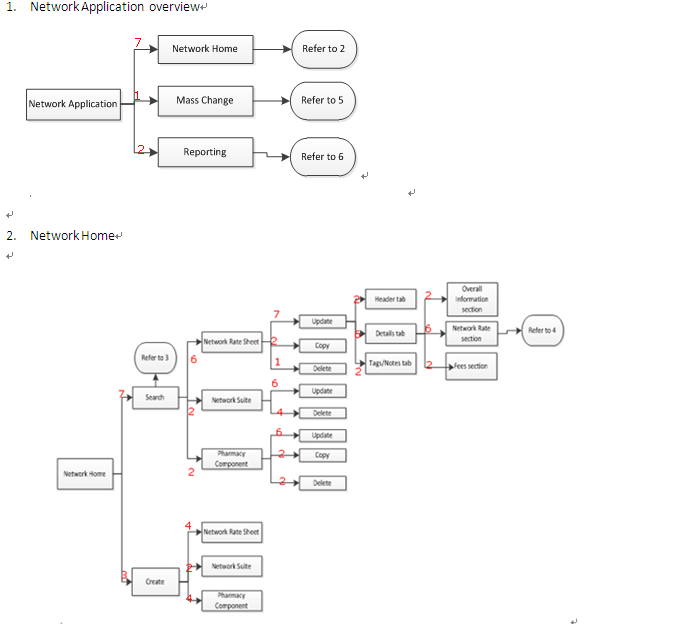


Figure 5

Notice：(In most applications, you have to manually accept changes you make in a Control Panel. However, in JMeter, the Control Panel automatically accepts your changes as you make them. If you change the name of an element, the tree will be updated with the new text after you leave the Control Panel (for example, when selecting another tree element).

### Produce Scenario & Workload

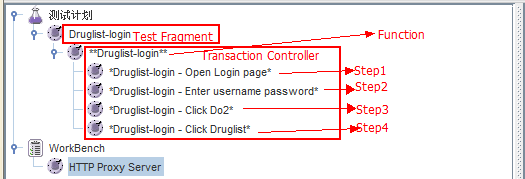


Using BeanShell Sampler and Switch Controller to simulate the user activities, first using Math.random() Randomly generated data between 1-10, and if the number <7, then execute the Network Home , if the number <8,then execute the Mass change, if the number <10, execute the Reporting.



### Make sure Construction of the scripts

Before recording, you should make sure what to Monitor and how many steps the test case should be divided into. Transaction Controller is used to Separate different steps which will make it easy to record. Note: give a typical name for each step to make report data clearly to see like below: [project name] – [Function name] – [Step name]. It is very important during test and analyze report period.



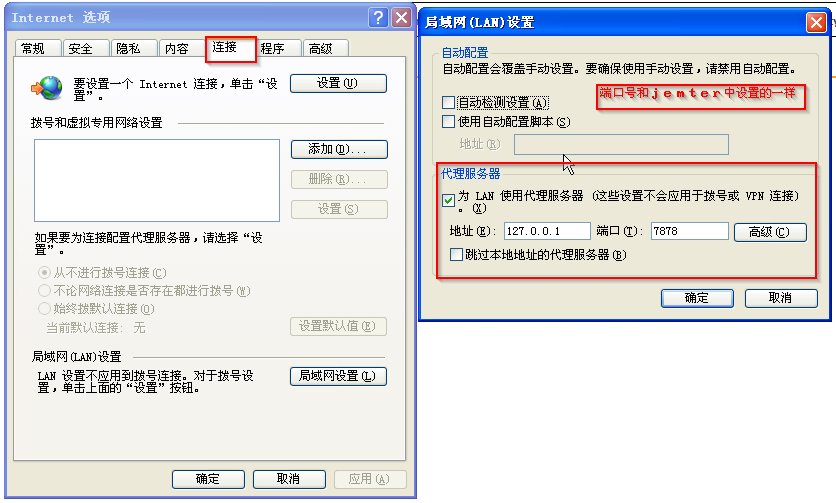
**Note:** Change Transaction Controller default configuration to below， Do not be include Think time into Step spent time.



### Record scripts

#### Set the browser proxy server

Open the browser, open Internet Options, the proxy server is set in the Local Area Network (LAN) Settings: localhost, port is set in the proxy server port: 7878.



#### WorkBench-Non test Elements –HTTP proxy serve

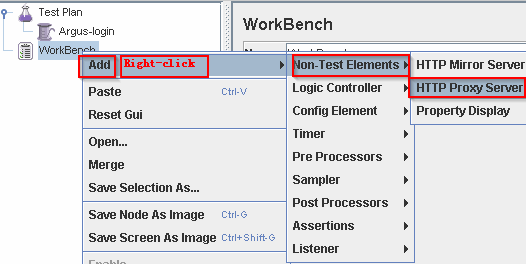


Figure 6

#### Port：The proxy server listening on this port，we set 7878，Target Controller：

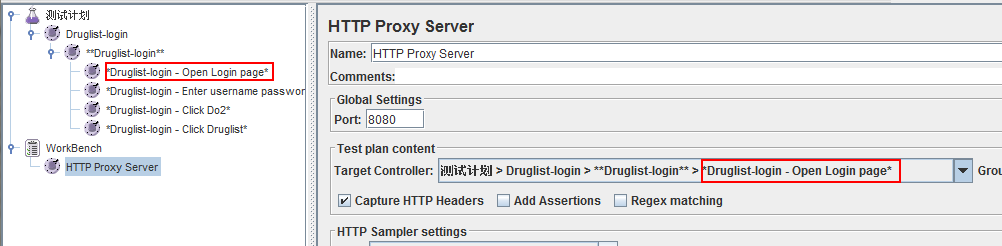


Figure 7

#### Grouping: Request packet. The concept of "grouping" is the number of requests summary packet, we regard the url as request and group an exhaustive list of all requests.

* + “Do not group samplers”: An exhaustive list of all requests
  + “Add separators between groups”: Add a virtual dividing line between different action
  + “Put each group in a new controller”: Output according to controller during the perform
  + “Store 1st sampler of each group only”: This option is good for a url request, the actual many times http request, because we often do not care about the latter of those requests.

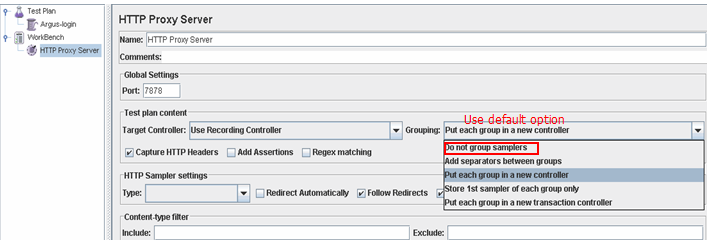


Figure 8

#### Think Time - Add Gaussian Random Timer

Add Gaussian Random Timer to the HTTP proxy server(Used to inform Jmeter to add a timer automatically when generate the HTTP request). The timer will cause the sampler to be delayed. The rule to Delay is Affected by the timer on an access request response delay a specified time, the next sampling access requests will be sent.

If Add Gaussian Random Timer to HTTP proxy server, It should be in one of the Constant Delay Offset（Constant Delay Offset）Add ${T} (For automatic reference record delay time).

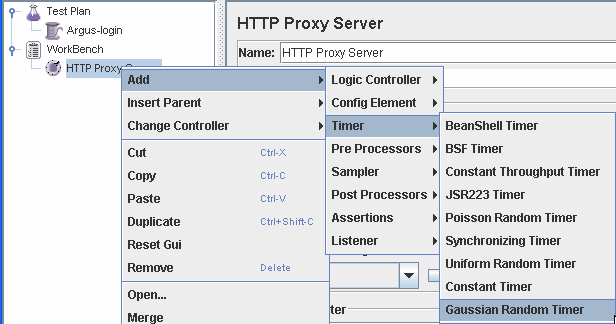


Figure 9

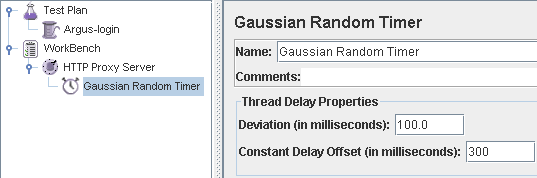


Figure 10

#### Set HTTP samples Setting

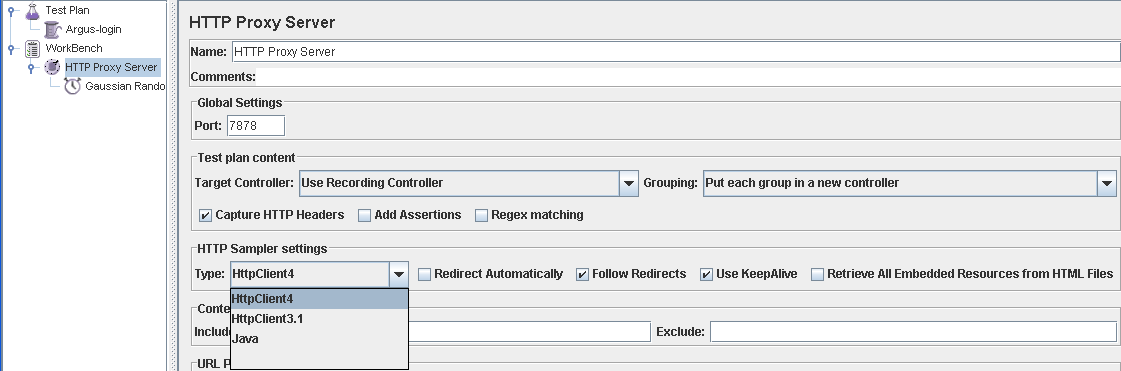


Figure 11

#### Start the HTTP proxy server

After HTTP access to the proxy server is configured, click start, the proxy server will start recording accepted the request.

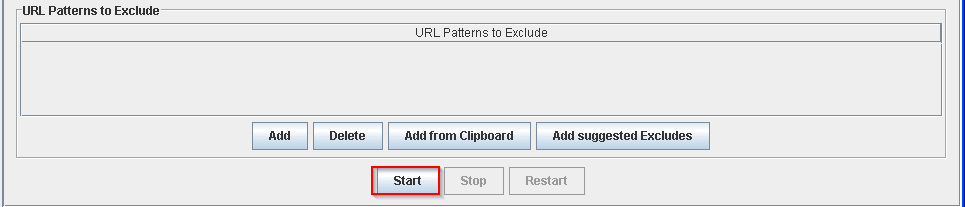


Figure 12

#### To browse for operation:

Enter the address in the browser address bar and related operations for recording, After the recording is completed, stop HTTP proxy server; components in the right-click will be recorded on the recording controller components are saved to a file for later reuse Also, do not forgot to restore the browser's proxy server settings.

#### When recording Https

Principle: We recorded the agent acts as the middleman, so to obtain the trust of both sides: that is, the browser and the server.  
For server party, we need to set the truststore, server certificate is imported into a keystore server-side keystore in system.property and JMeter.property set so, proxy will default certificate provided by the server-side.  
Then is the proxy need to provide a certificate to the browser, the need to include the public key, private key and certificate keystore. JMeterproxy.jks this the keystore to meet this requirement. Keystore related properties This keystore located.  
Note: The above use two different keystore need to set different attributes.  
Open JMeter proxy, this time browsing other HTTPS site JMeter will provide us with a certificate (can be found by looking at the certificate), I take Hengtian mailbox CMB site have tested, and can be recorded script  
Doubts exist: Although the recorded script, but the console will throw an exception, this exception we have recorded constant of days mailbox and Merchants Bank online banking when they met, but the results can be recorded. Still not sure the exception script recording situation may JMeter HTTPS proxy is not good to maintain the status of the connection to the remote host will take the initiative to disconnect (whether in order to prevent the Man in middle attack?), remote disconnect the browser will disconnect, because in inconsistent session JMeter need to be re-handshake.  
This case the process of recording influential, affecting only recorded will throw exception. Replay should be no problem, because only two parties in the playback process, only need to trust the server just fine, do not need to deceive customers end.

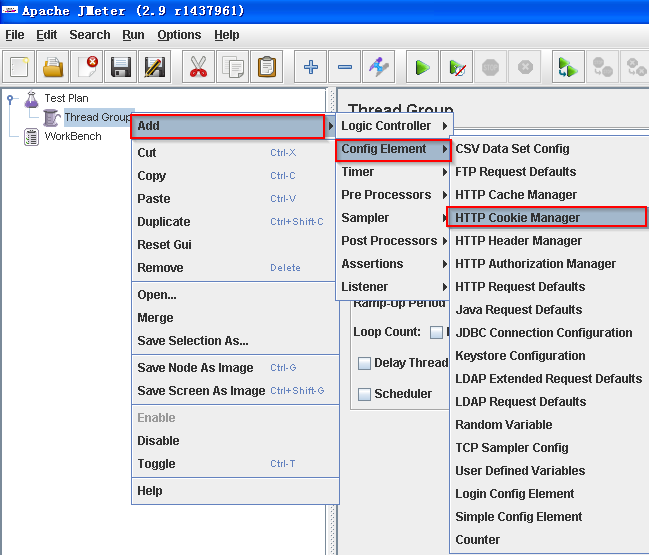
## Parameterization

The first Primary goal is to make sure your the scrip can run successfully, In Argus Application there are some Dynamic data that comes from the response, if you don’t do the parameterization work, the script can’t run successfully, so you must do the parameterization work to make sure the script run.

### Three primary concern

#### JSESSIONID:

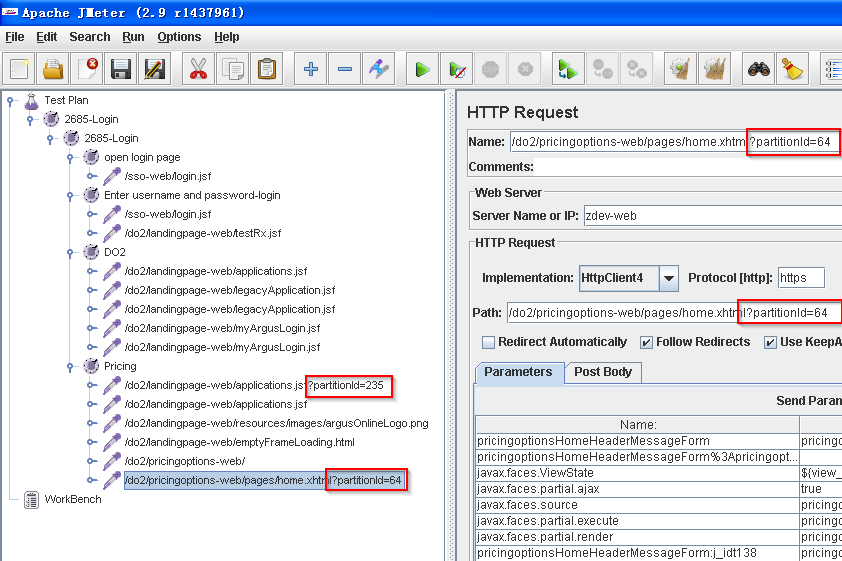
To identify JSESSIONID, you just need to add a HTTP Cookie Manager, please make sure the HTTP Cookie Manager is on the father level of the script which you want to run.



1. Figure 24

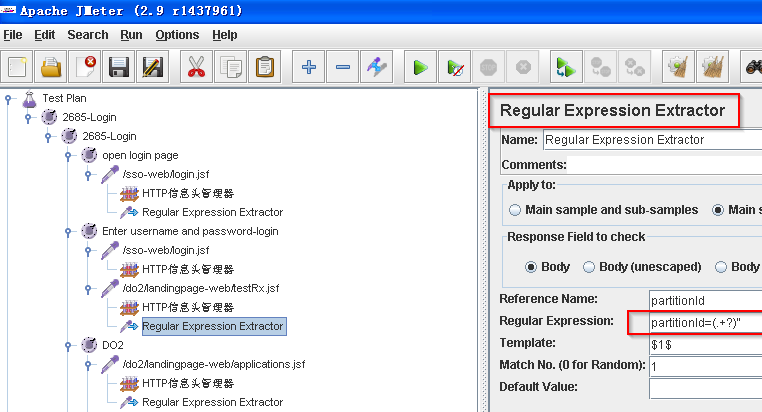
#### PartitionID:

PartitionID is the variant of JSESSIONID, different operation has different PartitionID, It make sure different operation independently of each other , as you can see in the figure：



1. Figure 25

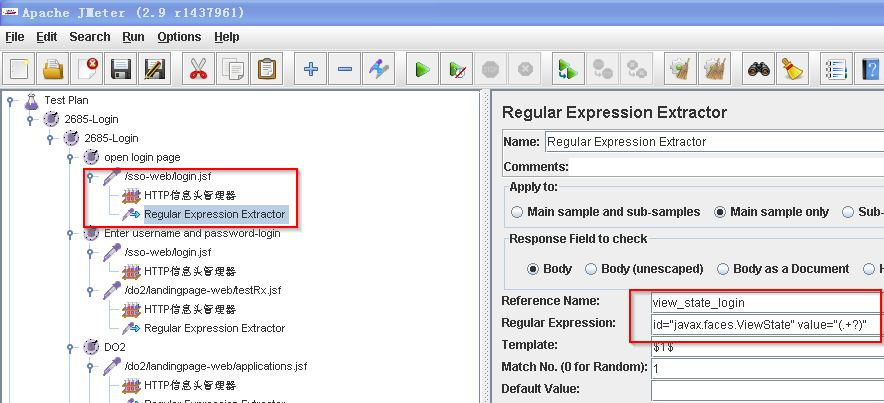
The partitionID comes from the response, a Regular Expression Extractor is needed to get it, the partitionID keeps you in the same session. The following picture shows a script that added the Regular Expression Extractor :



1. Figure 26

#### ViewStatus:

The status of Login, different page has different view statues. You must get the values of ViewStatus from the different responses, and set the ViewStatus in the following request which will make sure your status is Login.



1. Figure 27

### Duplication of data:

Errors pop up in Argus system when you fill information which is existed in the database into some textboxes, for example existing ID or Name. But JMeter won’t give any error message at this situation. You must make sure the data which to be filled in is not duplicated. To make sure whether you get correct response please go to see the detail information in View result tree.

### Associated date range：

If the Data range is associated, it is unable to pass the Verification of the system, if you are not carefully enough, you may make this mistake. To make sure whether you get correct response please go to see the detail information in View result tree.

### Commonly used parameterization function – use Dynamic Data

Parameterized data: Username/Password (login), Edit data, Search criteria…

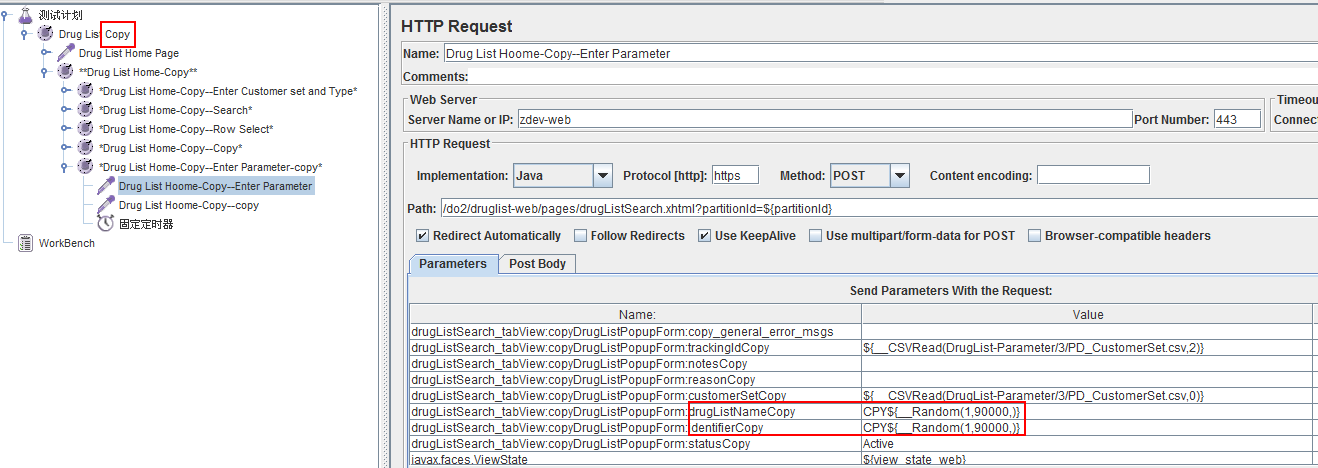
Unique data: component name, identifier…that should not be duplicate.

*Using the Jmeter function to obtain the parameter value，Specific calls the method is as follows:  
${\_\_Random(,,)}，${\_\_threadNum}，${\_\_CSVRead(,)}，${\_\_StringFromFile(,,,)}.  
Reference to jmeter use manual, Through the menu "Options" -> "function help dialog", can be found in the " function help" pop-up box on Jmeter function.*

The first parameter of ${\_\_Random(,,)} is lower limit of the random , the second parameter is the upper limit of the random, the third parameter storage the random variable names, The first parameter of ${\_\_CSVRead(,)} is filename , the second parameter is the file list (The column number starting from 0), The first parameter of ${\_\_StringFromFile(,,,)} is filename, ${\_\_StringFromFile (,,,}) method is not specified in the parameter which column in the file, So ${\_\_StringFromFile (,,,)} can read only contains a list of files.

#### ${\_\_Random(0,10,)}

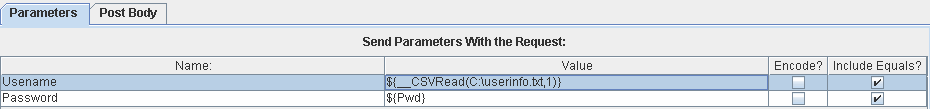
Exp: in Copy/Create scenarios, component name, identifier should not be duplicate, use random data like below.



1. Figure 20

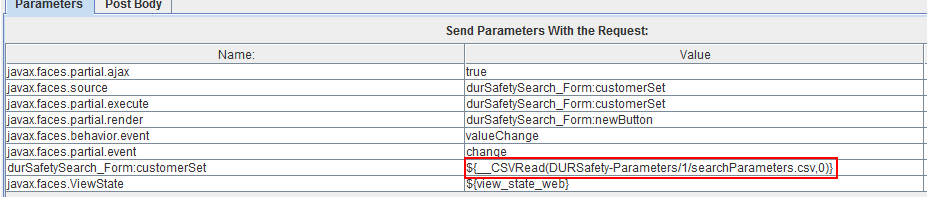
#### ${\_\_CSVRead(c:\userInfo.txt,1)}

Read data from txt file.



Save username and password into userinfo.txt as follows, Make sure it has **multiple rows** to use dynamic data.：  
oriana,123456  
admin,admin  
dandan,123456

#### ${\_\_CSVRead(searchParameters.csv,0)}



Read data from searchParameters.csv as below, all rows are read by sequence and loop by default. Make sure it has **multiple rows** to use dynamic data.

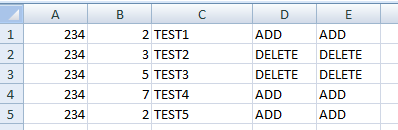


Figure 21

#### Add ”config Element”->“User Defined Variables”

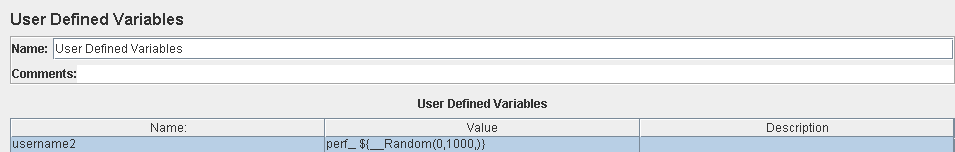


Figure 22

#### Add CSV Data Set Config

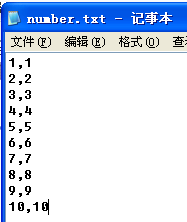
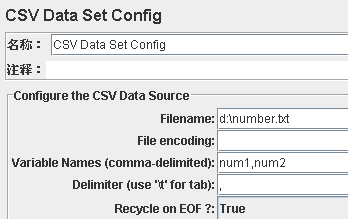


Figure 23

Filename ---Set the file path

File Encoding ---File compiling method, general is empty

Variable Names ---File all the columns in the item parameter; Use a comma between each

parameter; Parameters are the name of the item should be in accordance with the

parameters in the HTTP Request item.

Delimiter ---If file is using commas, fill out a comma; If using the TAB, fill in the \ t;

Recycle on EOF? ---True= When reading the file to the end, and then start reading the file False = when reading the file to the end, stop reading the file

Stop thread on EOF? ---when Recycle on EOF? Is False Works, True= When reading the file to knot the tail, Halt the process,

## Add Listener

A listener is a component that shows the results of the samples. The results can be shown in a tree, tables, graphs or simply written to a log file. To view the contents of a response from any given sampler, add either of the Listeners "View Results Tree" or "View Results in table" to a test plan. To view the response time graphically, add graph results, spline results or distribution graph. The listeners section of the components page has full descriptions of all the listeners.

### Aggregate Report

The aggregate report creates a table row for each differently named request in your test. For each request, it totals the response information and provides request count, min, max, average, error rate, approximate throughput (request/second) and Kilobytes per second throughput. Once the test is done, the throughput is the actual through for the duration of the entire test.

The thoughput is calculated from the point of view of the sampler target (e.g. the remote server in the case of HTTP samples). JMeter takes into account the total time over which the requests have been generated. If other samplers and timers are in the same thread, these will increase the total time, and therefore reduce the throughput value. So two identical samplers with different names will have half the throughput of two samplers with the same name. It is important to choose the sampler names correctly to get the best results from the Aggregate Report.



### View Results Tree

You can see the response of every request，and check out whether it has the right response, and you can also find which request encountered an Error.

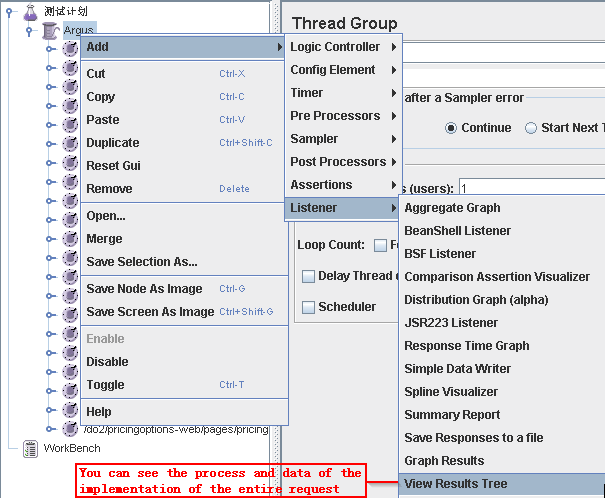
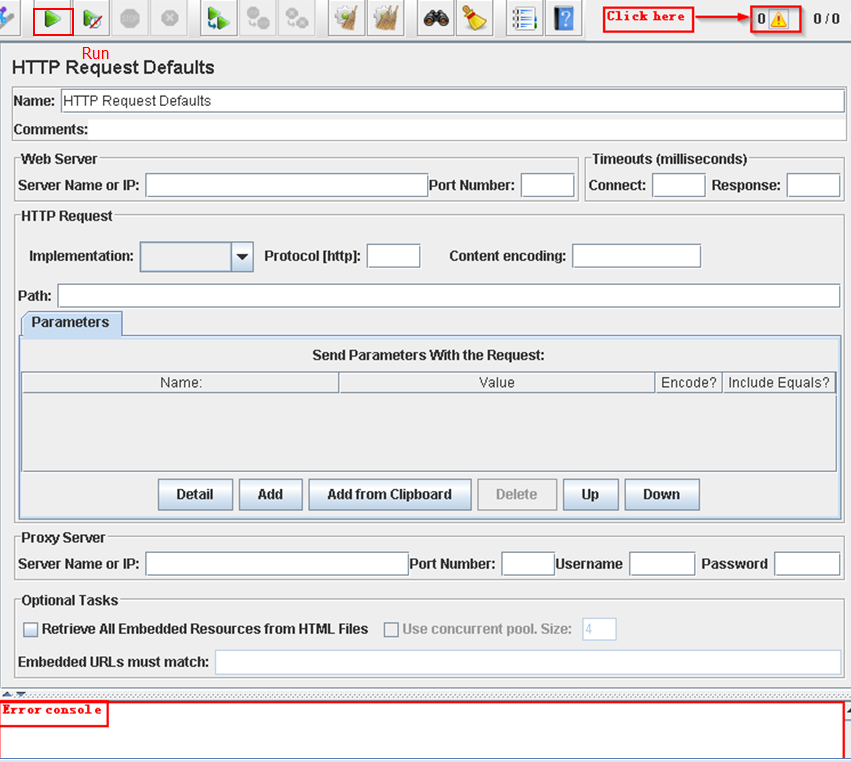


Figure 16

## Run our JMeter Script

### Run Jmeter Script

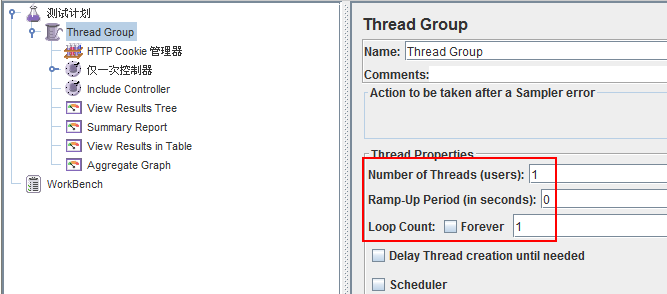
After script are validated, you run your script, you had better to open the Error page, and check out whether the script running correctly.



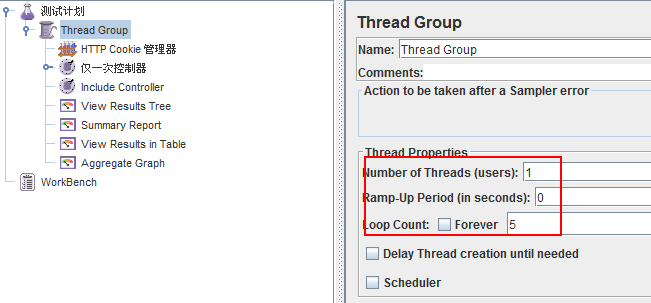
### Validate Jmeter script

In performance test, poor load simulations can render all previous work useless. So you must validate all scripts before real performance test.

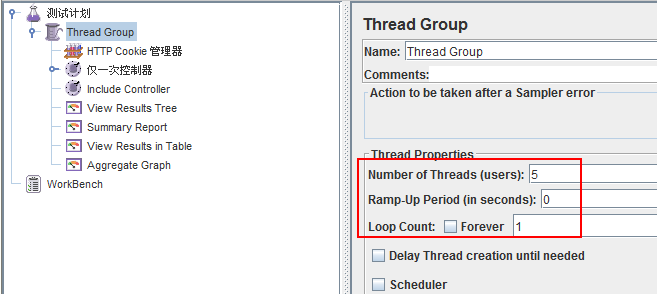
• **Test design implementation.** Run the test with 1 user loop for 1 time.To validate that you have implemented your test design accurately (using whatever method you have chosen), you will need to run the test and examine exactly what the test does.



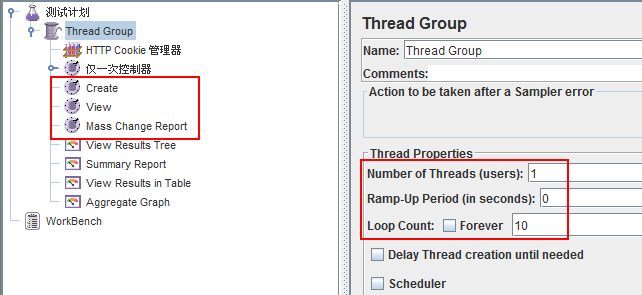
• **Continuity.** Run the test with 1 user loop for several times.After you have validated that your test conforms to the test design when run one time, run the test for several times. Ensure that each time is seeded with unique data.



• **Concurrency.** Run the test with 5 users loop for 1 time.After you have validated that your test conforms to the test design when run with a single user, run the test with several users. Ensure that each user is seeded with unique data, and that users begin their activity within a few seconds of one another — not all at the same second.



• **Combinations of tests.** Run the test combined with other tests. Having validated that a test runs as intended with a single user and with multiple users, the next logical step is to validate that the test runs accurately in combination with other tests.



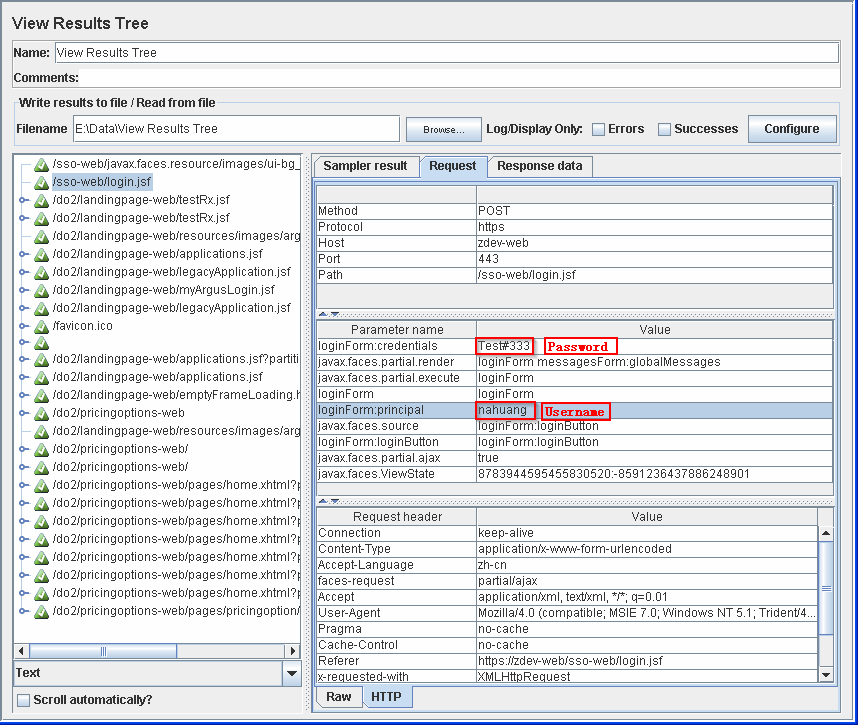
• **Test data validation.** Once you are satisfied that your tests are running properly, the last critical validation step is to validate your test data. Performance testing can utilize and/or consume large volumes of test data, thereby increasing the likelihood of errors in your dataset. In addition to the data used by your tests, it is important to validate that your tests share that data as intended, and that the application under test is seeded with the correct data to enable your tests.

## View Results

### JMeter results



#### The result of View Result Tree



#### **The result of View Result in Table**

#### The Results of Aggregate Report



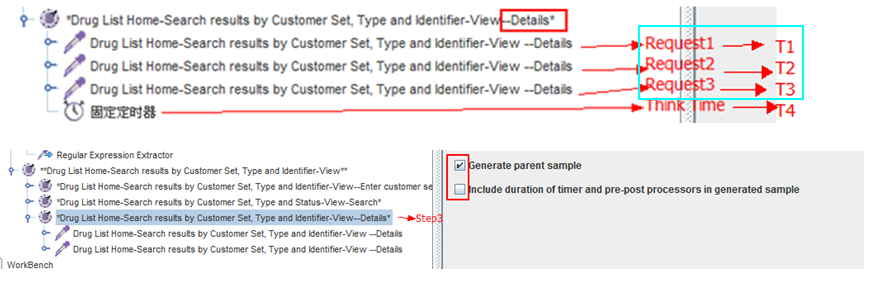
Figure 19

**Response time of a request**=*T1+T2+T3+T4*

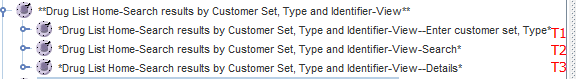


**Response time of an activity=** *T1+T2+T3*

*Note: “Transaction controller” must be configured like below, do not checke the second checkbox“”*



**Response time of a Function=** *T1+T2+T3*



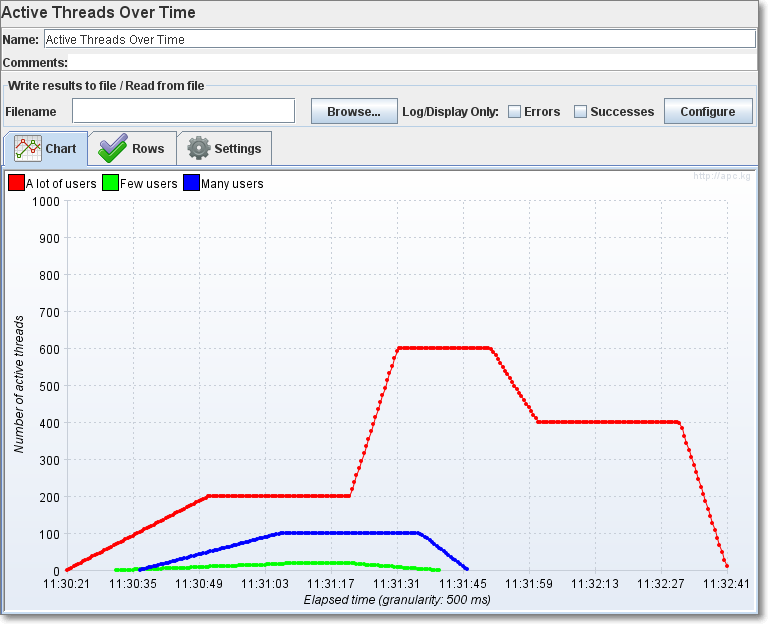
Notice:(

* Label - The label of the sample. If "Include group name in label?" is selected, then the name of the thread group is added as a prefix. This allows identical labels from different thread groups to be collated separately if required.
* # Samples - The number of samples with the same label
* Average - The average time of a set of results
* Median - The median is the time in the middle of a set of results. 50% of the samples took no more than this time; the remainder took at least as long.
* 90% Line - 90% of the samples took no more than this time. The remaining samples at least as long as this. (90 th percentile )
* Min - The shortest time for the samples with the same label
* Max - The longest time for the samples with the same label
* Error % - Percent of requests with errors
* Throughput - calculated as requests/unit of time. The time is calculated from the start of the first sample to the end of the last sample. This includes any intervals between samples, as it is supposed to represent the load on the server. The formula is: Throughput = (number of requests) / (total time).
* Kb/sec - The throughput measured in Kilobytes per second)

### JMeter Plugin Results

#### Active Threads Over Time Listener

Active Threads Over Time is a simple listener showing how many active threads are there in each thread group during test run. The plugin produces graph like shown below:



#### Response Times vs Threads

This graph shows how Response Time changes with amount of parallel threads. Naturally, server takes longer to respond when a lot of users requests it simultaneously. This graph visualizes such dependencies.

Overall



Detail:

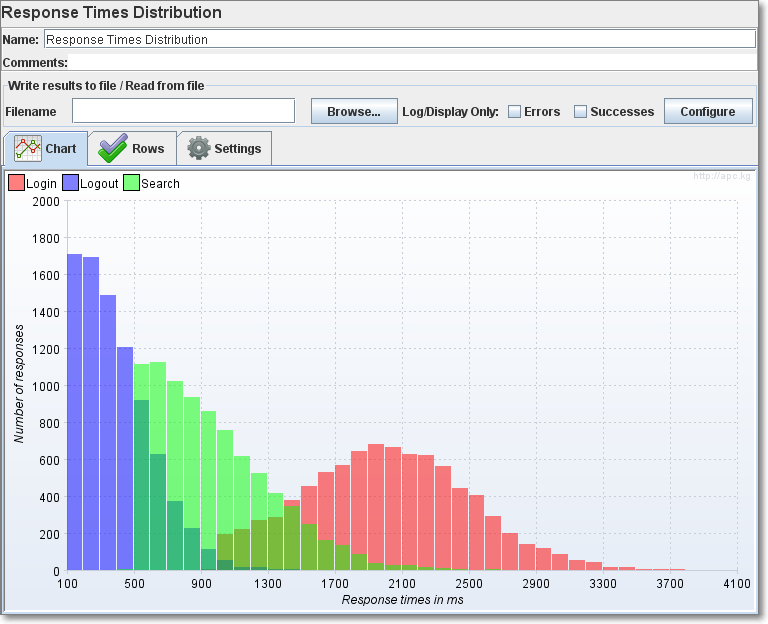


#### Graph Response Times vs Threads

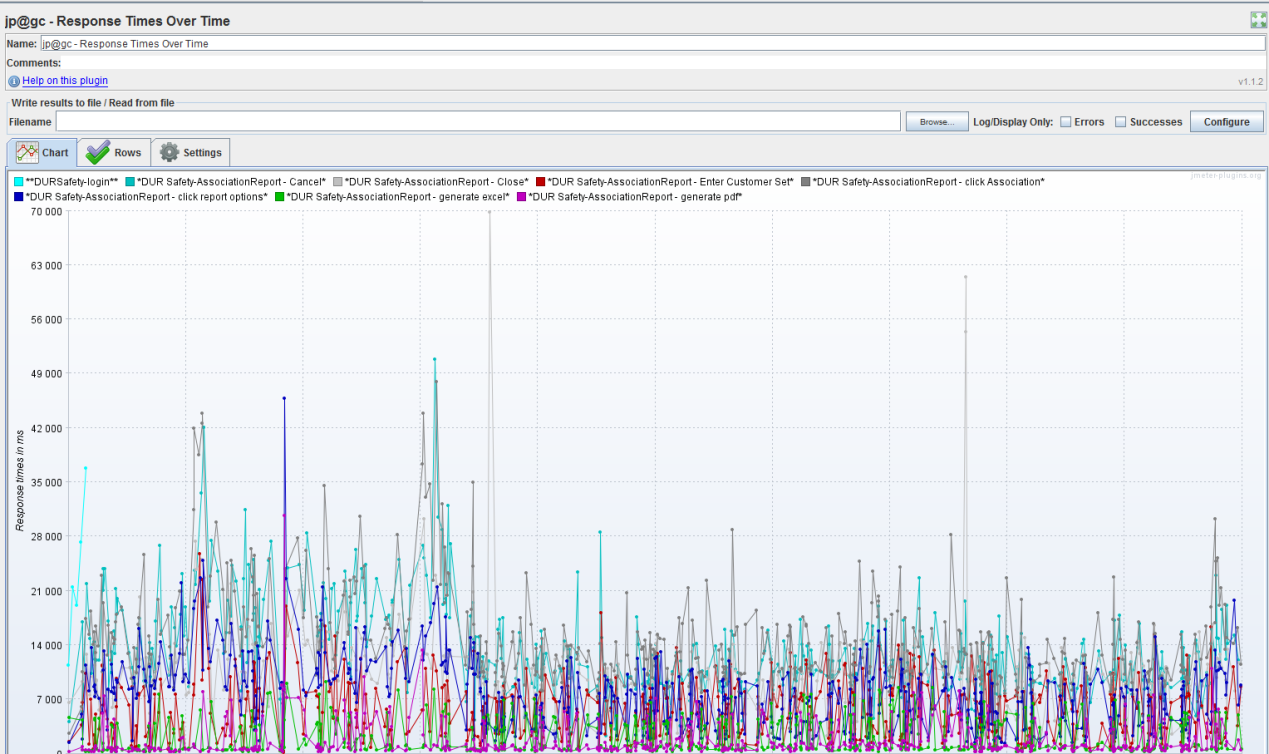
#### Transaction Throughput vs Threads

#### Response Times Distribution

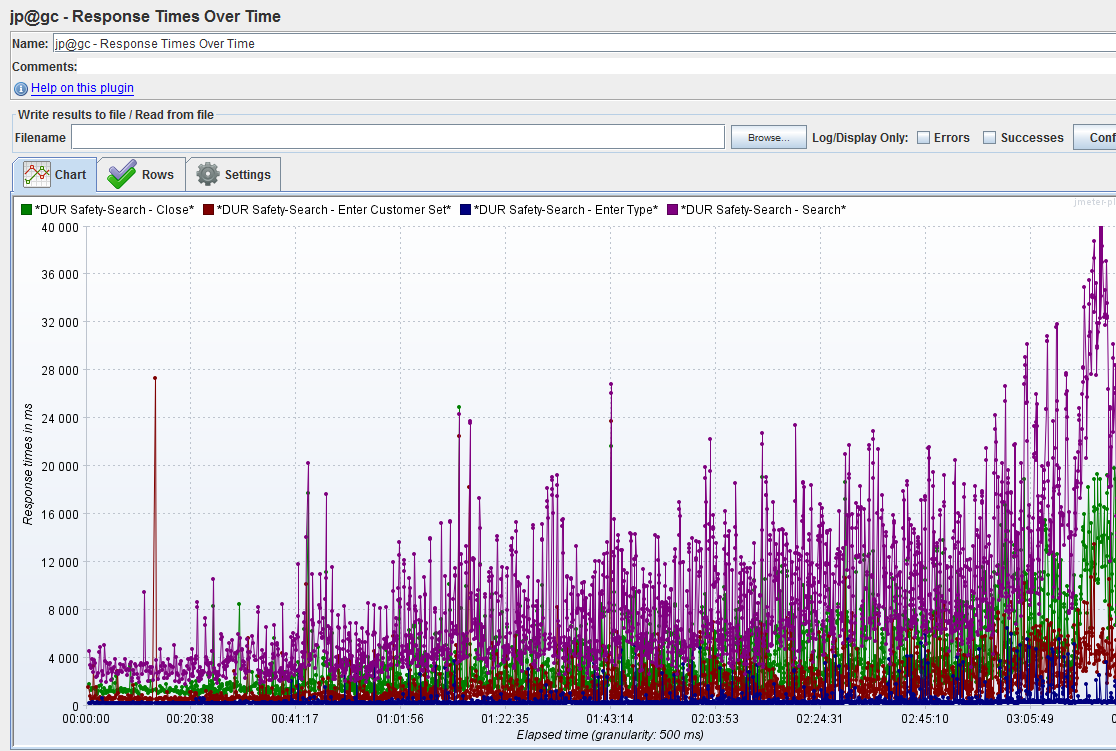
This graph will display the response time distribution of the test. The X axis shows the response times grouped by interval, and the Y axis the number of samples which are contained in each interval.



#### Response Times Over Time

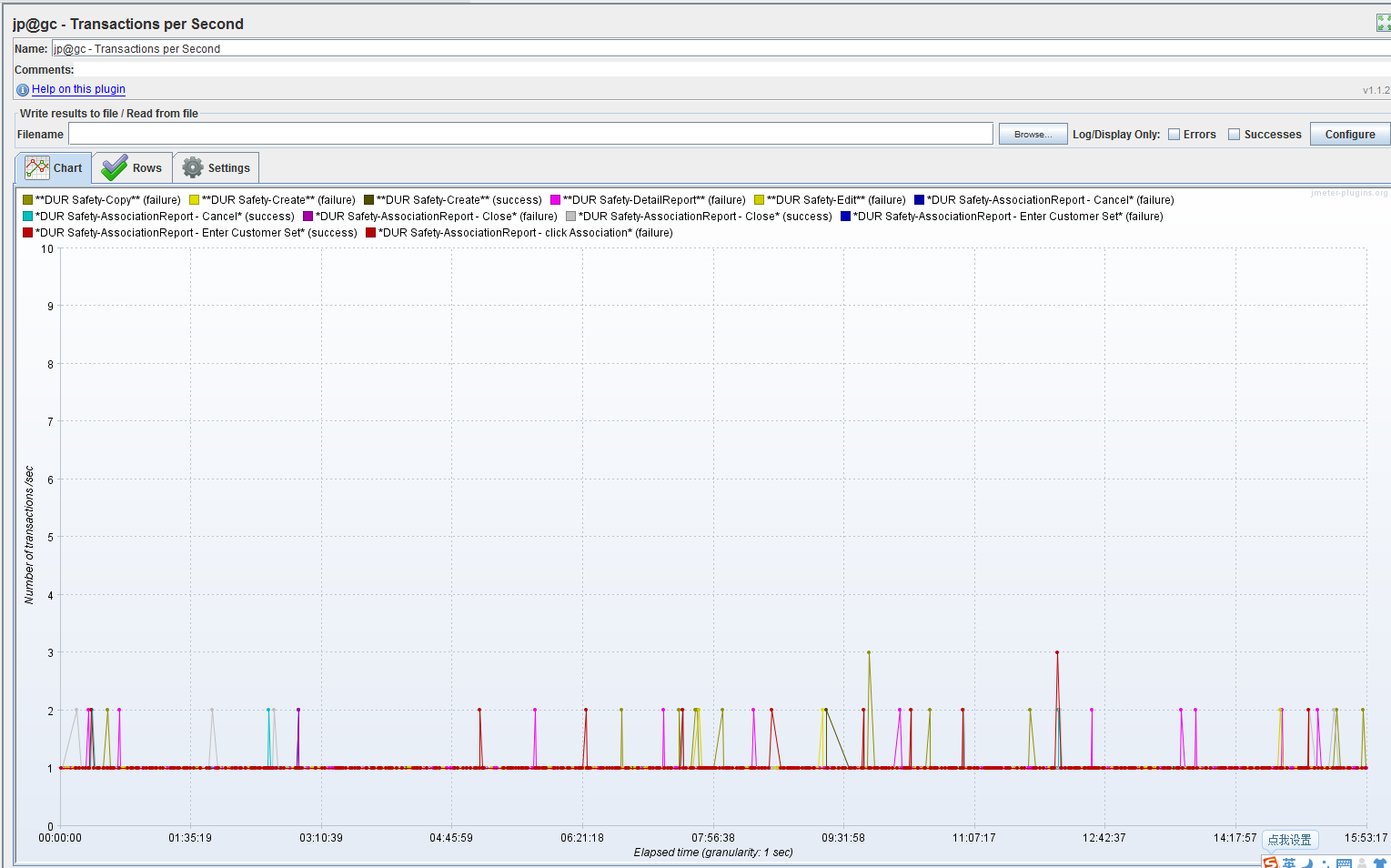


Response time decreased and then stays stable.



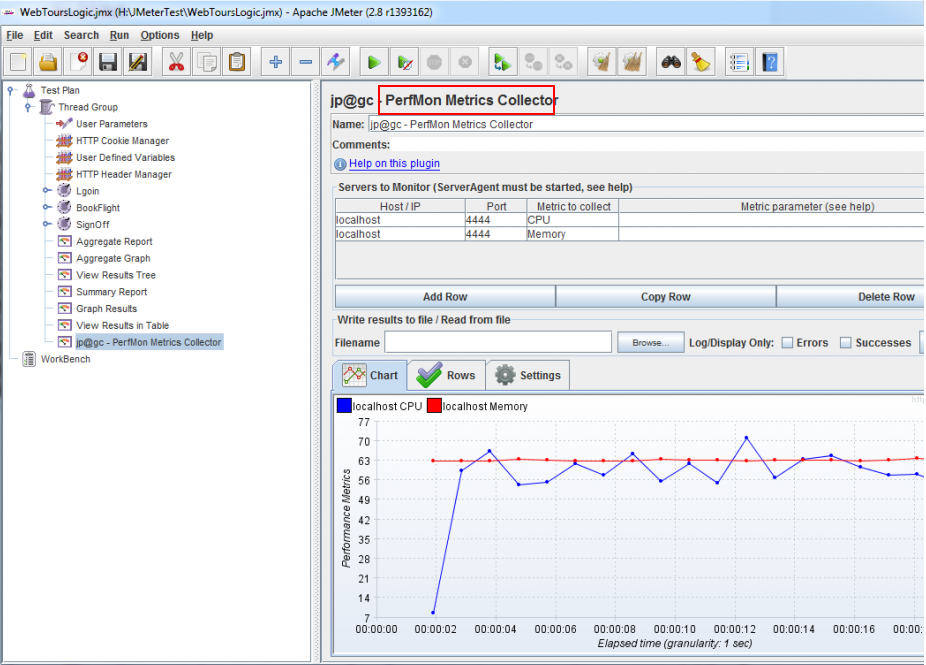
Response time keeps growing.

#### Transactions Per second



#### PerfMon Metric Collector

It is used monitor CPU, Memory, Disk, Network (on research, it need port [authority](javascript:void(0);) of ZDEV)








# Performance Testing techniques

Performance test execution is the activity that occurs after developing test scripts. In reality, this activity is significantly more complex than just clicking a button and monitoring machines. Performance testing is a generic term that can refer to many different types of performance-related testing. This chapter addresses several common types or categories of performance-related testing: Endurance test, load test and stress test, each of which addresses a specific problem area and provides its own benefits, risks, and challenges.

Before each test, you must validate your test environment and test scripts.

1. **Validate [Test Environment](#_Identify_the_Test)**

The goal is for the test environment to mirror your production environment as closely as possible. Typically, any differences between the test and production environments are noted and accounted for while designing tests. Before running your tests, it is important to validate that the test environment matches the configuration that you were expecting and/or designed your test for. If the test environment is even slightly different from the environment you designed your tests to be run against, there is a high probability that your tests might not work at all, or worse, that they will work but will provide misleading data.

1. **Validate [data maintained by target system](#_Prepare_data_maintained)**

Periodically check to see if the data base is storing unrealistic volumes of data for the situation you are trying to simulate. Especially when designing and debugging performance tests, test databases can become dramatically overloaded with data.

1. **Validate [Jmeter script](#_Validate_Jmeter_script)**

Poor load simulations can render all previous work useless. Before performance test, you must validate your test according to [Validate Jmeter script](#_Validate_Jmeter_script)

## Endurance Test

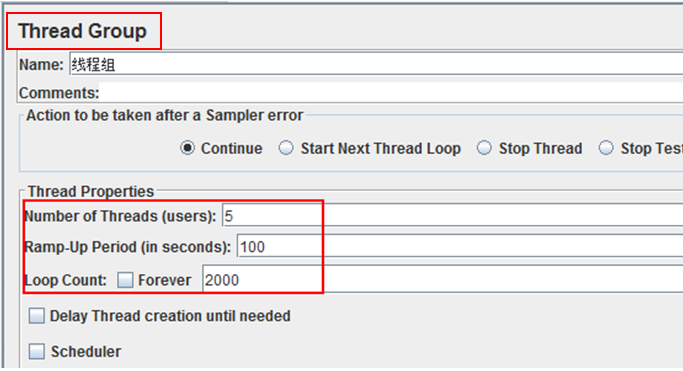
It is usually done to determine if the system can sustain the continuous expected load. During tests, memory utilization is monitored to detect potential leaks. Also important, but often overlooked is performance degradation. That is, to ensure that the throughput and/or response times after some long period of sustained activity are as good as or better than at the beginning of the test. It essentially involves applying a significant load to a system for an extended, significant period of time. The goal is to discover how the system behaves under sustained use.

### Testing Goal

Discover how the system behaves under sustained use. And to detect potential memory leaks.

### Testing Approach

Simulate appropriate virtual users run the test case for much more times. The test case should cover most of the key scenario, and the number of the dataset should be in the same level of real production. During running, monitor memory status to see if any memory leak happens. And to ensure that the throughput and/or response times after some long period of sustained activity are as good as or better than at the beginning of the test.



### Key Criteria to Monitor

[Response Times Over Time](#_Response_Times_Over), [Transactions Per second](#_Transactions_Per_second), Free memory, Used memory. If memory leak happens, monitor detail metrics: GC frequency, GC overhead, GC duration, JVM free heap size after GC …etc related to memory

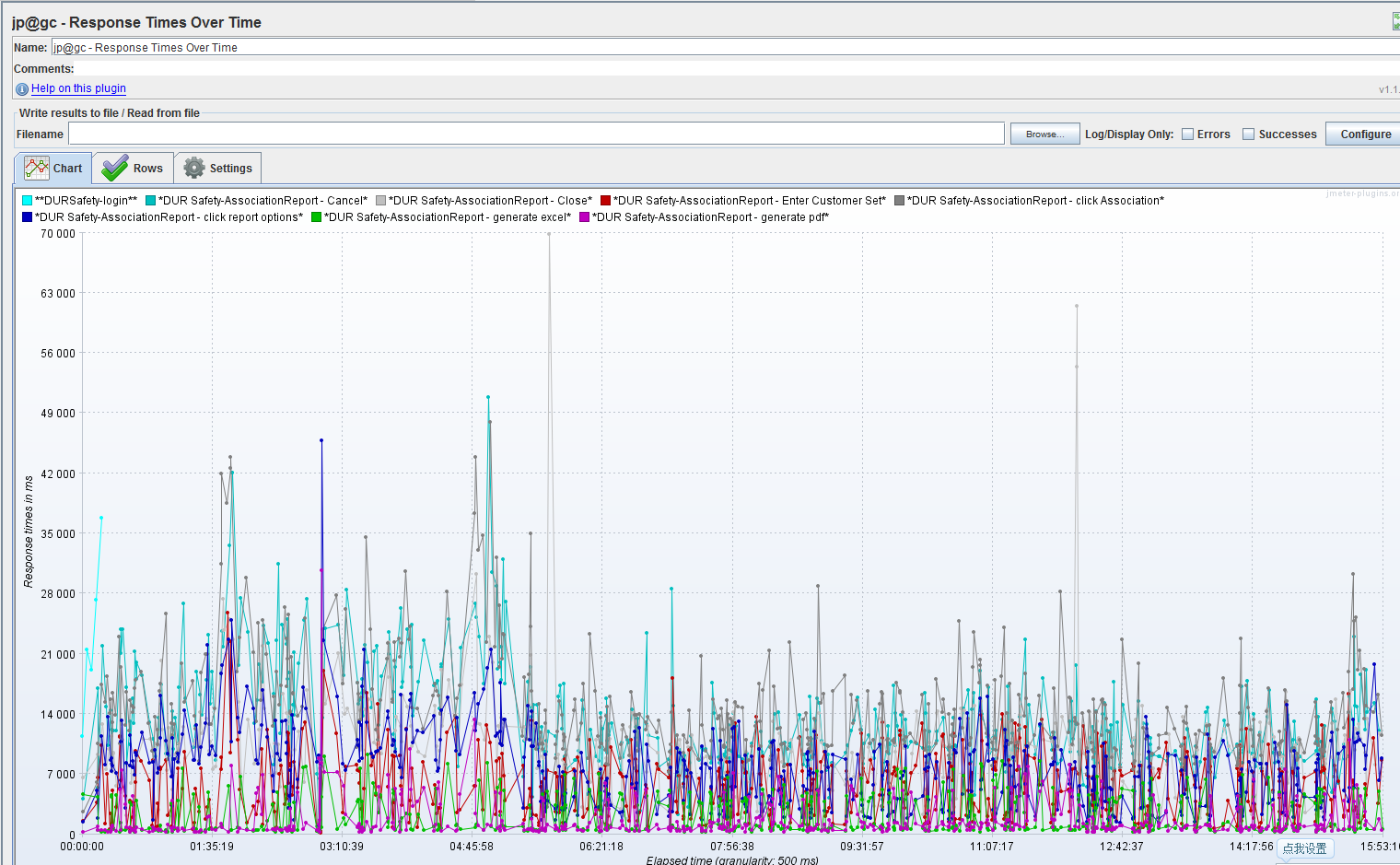
### Expect Result:

1. Does Response time stay stable during long period process?
2. Is memory used in normal level?
3. Potential bottlenecks that need to be analyzed in the white-box testing phase

### Analyze Result

1. Response time over time

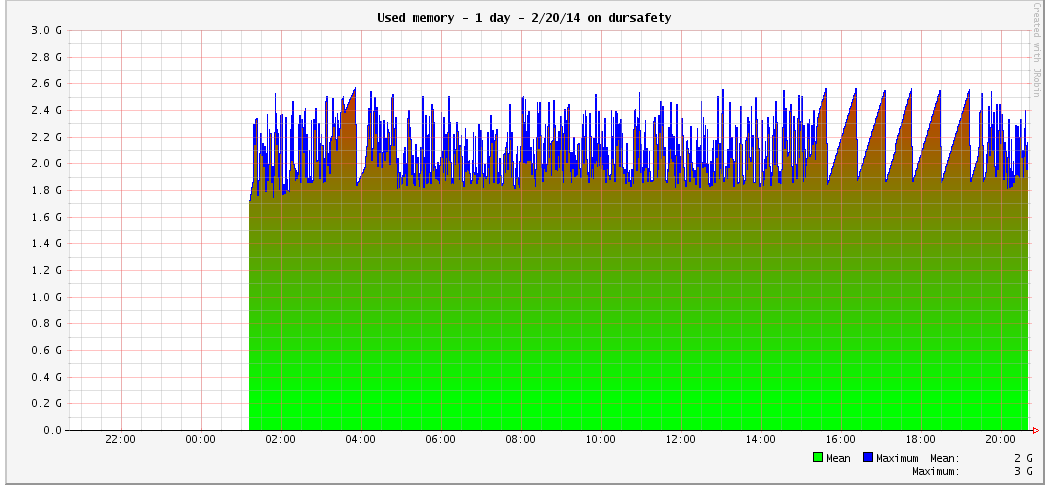
Below is an example, the system performances generally stable over time.



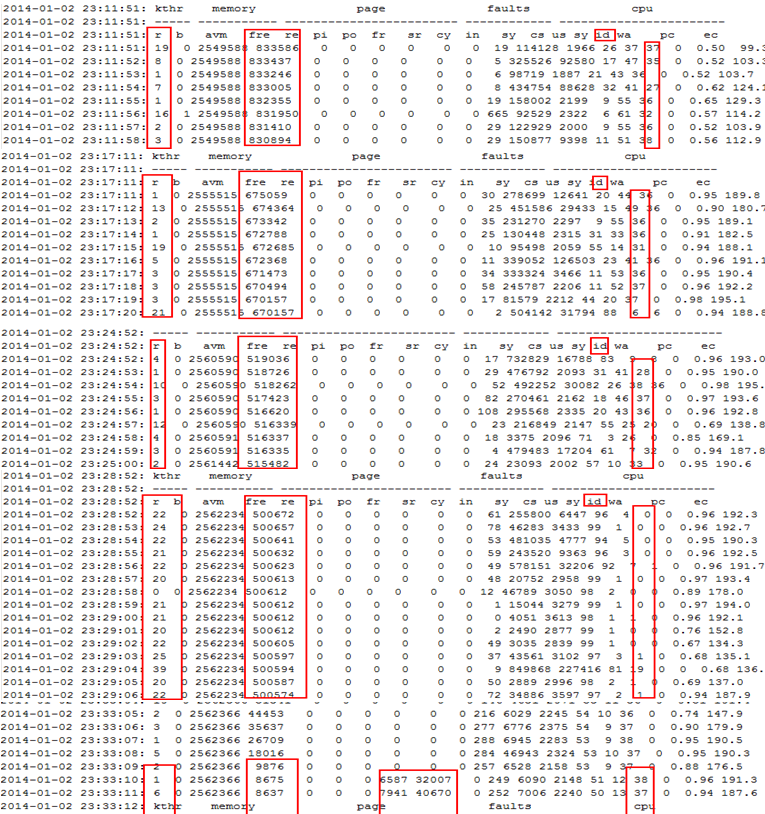
1. Transaction Per second
2. Memory usage

Analyze the measured metrics to diagnose potential bottlenecks. Based on the analysis, if required, capture additional metrics in subsequent test cycles. For example, suppose that during the first iteration of load tests, the process shows a marked increase in memory consumption, indicating a possible memory leak. In the subsequent iterations, additional memory counters related to generations can be captured to study the memory allocation pattern for the application. During running, monitor memory status to see if any memory leak happens. Key metrics to monitor are Free memory, used memory, GC frequency, GC overhead, GC duration, JVM free heap size after GC …etc related to memory.

Below is an example of memory used in normal:



Below is an example of free memory keeps decreasing. The report is generated through vmstat.



## Load Test

The most basic type of load testing is used to determine the Web application’s behavior under both normal and anticipated peak load conditions. Load testing helps to identify the maximum operating capacity of an application as well as any bottlenecks that might interfere with its operating at capacity.

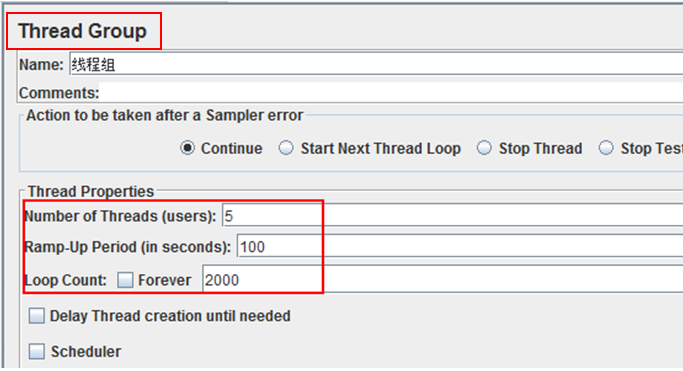
### Normal load Test

#### Testing Goal

Test how system performs under normal level, including response time, system stability under normal load. It helps to identifies mismatches between performance-related expectations and reality. It also provides tuning tips for reported performance issues, such like activities those response slow, scenarios that cause system instability.

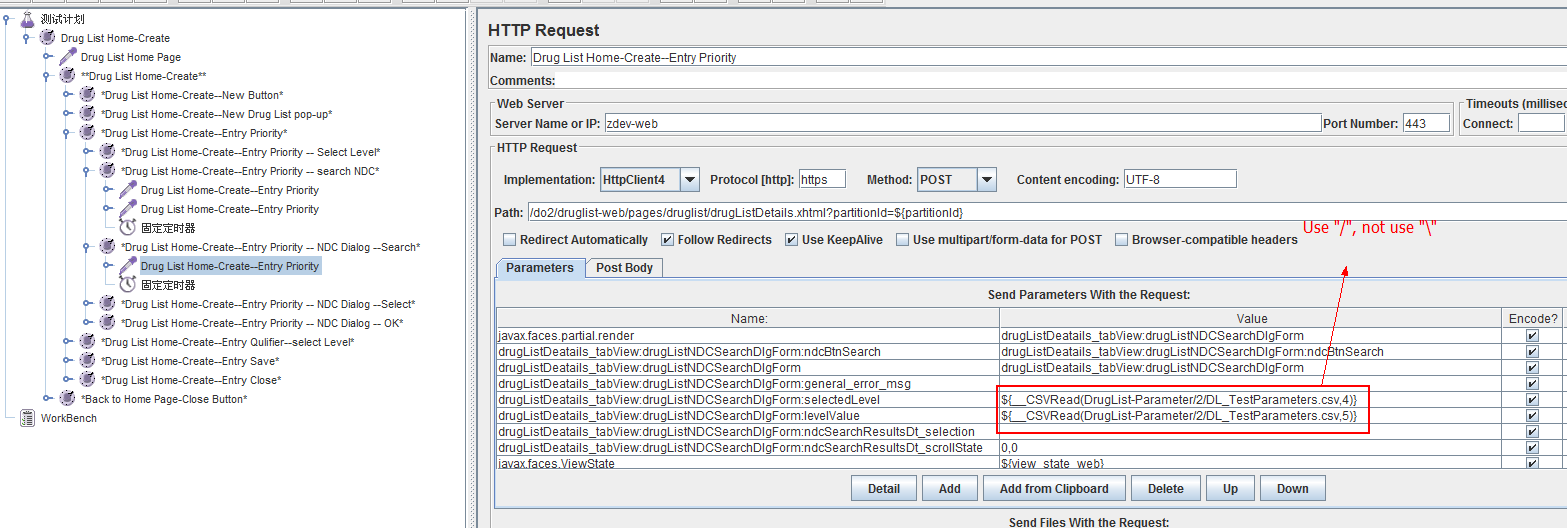
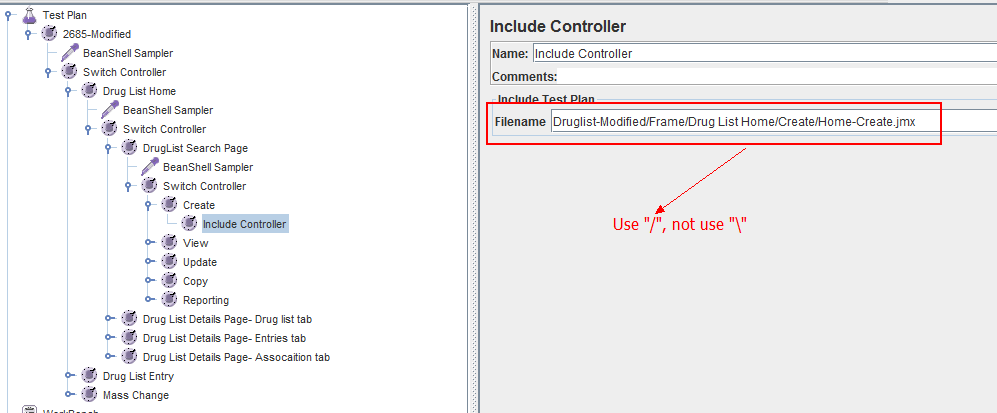
#### Testing Approach

Simulate appropriate virtual users run the test case for much more times under normal load. The test case should cover most of the key scenario, and the number of the dataset should be in the same level of real production. During running, monitor various performance measures to see if it matches expectation. Exp:



**Run script at local or from ZDEV(recommend) as below, in this way it reduce much time of network transfer.**

1. Put all script to server host, Make sure all script use “/”.



1. Set java path and run test from putty command console as below, as a result, report is generated into file *[Jmeter install path]/*DrugList-Result/report-0118.jtl. You can analyze the report in local Jmeter.

cd apache-jmeter-2.9/apache-jmeter-2.9/bin

export JAVA\_HOME=/usr/java6

export PATH=$PATH:$JAVA\_HOME/bin

jmeter -n -t Druglist-Modified/Frame/druglist\_ok\_zdev.jmx -l DrugList-Result/report-0118.jtl

#### Key Criteria to Monitor

[The Results of Aggregate Report](#_The_Results_of): Speed, Average, Percentiles, Medium, Max, Min, throughput

[Response Times Over Time](#_Response_Times_Over): Scalability and stability characteristics,

[Response Times Distribution](#_Response_Times_Distribution): Response time distribution

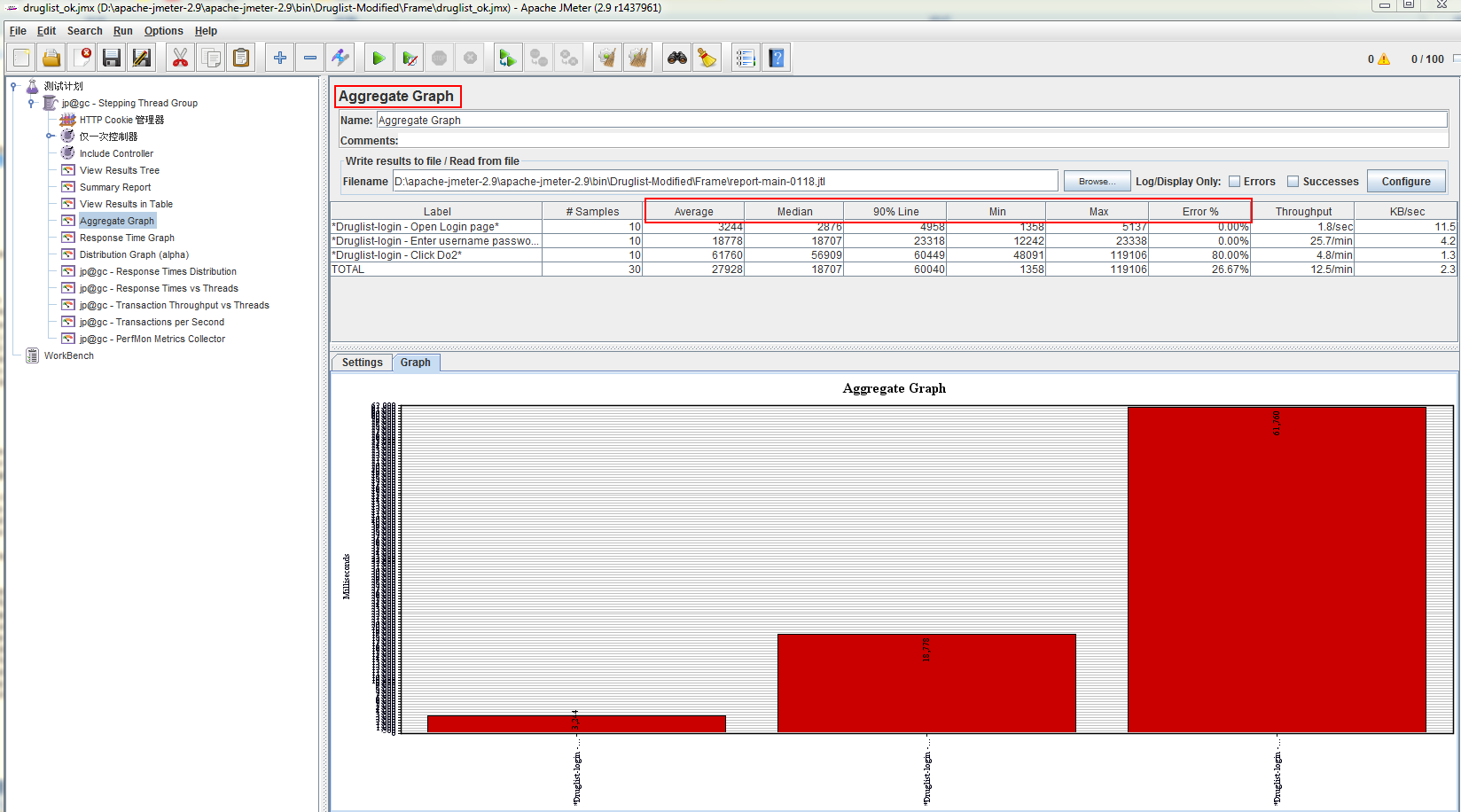
[JavaMelody](#_JavaMelody): Resource utilization

#### Expect Result

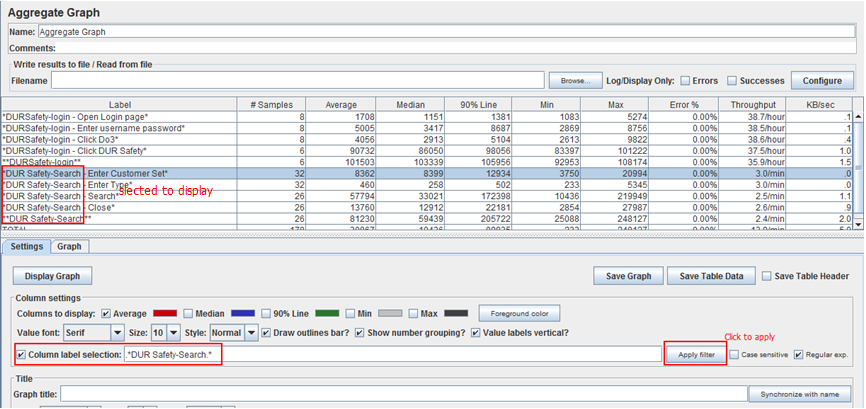
1. Various performance measures such as throughput, response time, and resource utilization, Speed, scalability, and/or stability.
2. Slowest scenarios, slowest activities (compared to requirement doc or bench mark).
3. If the performance meets expectations, will user be satisfied?

#### Analyze Result

1. Find out the slowest operation and slowest steps of the system. Analyze [The Results of Aggregate Report](#_The_Results_of) and [Response Times Distribution](#_Response_Times_Distribution) results:



Use “label selection” to filter special samples:



1. Stability: analyze [Response Times Over Time](#_Response_Times_Over) result.
2. Resource utilization : analyze [JavaMelody](#_JavaMelody) result.

### Increasing Load Test

Load testing helps to identify the maximum operating capacity of an application as well as any bottlenecks that might interfere with its operating at capacity. There are many reasons for load-testing a Web application. As you begin load testing, it is recommended that you start with a small number of virtual users and then incrementally increase the load from normal to peak. You can then observe how your application performs during this gradually increasing load condition. Eventually, you will cross a threshold limit for your performance objectives. For example, you might continue to increase the load until the server processor utilization reaches 75 percent, or when end-user response times exceed 8 seconds.



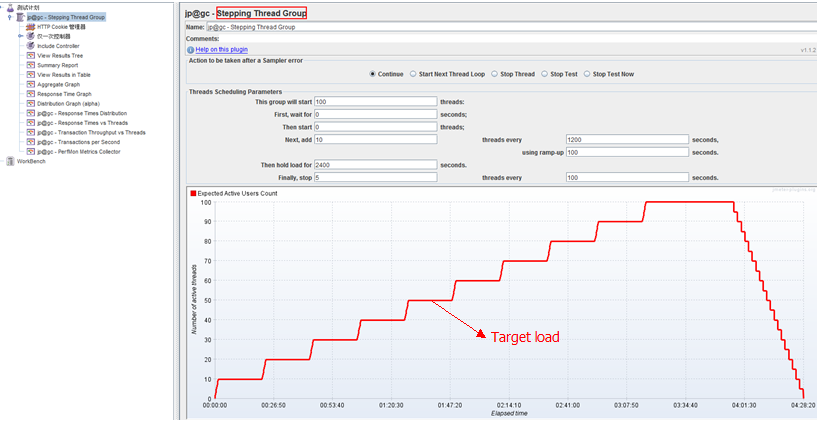
#### Test Goal

To verify application behavior under normal and peak load conditions. It is meaningful to find application behavior based on target load levels (production HTTP LOG \* 200% throughput). The purpose of identifying target load levels is to ensure that your tests can be used to predict or compare a variety of production load conditions. The main outcomes that load testing helps you to accomplish are:

1. Updated test plans and test designs for load and performance testing
2. Various performance measures such as throughput, response time, and resource utilization
3. Potential bottlenecks that need to be analyzed in the white-box testing phase, such like: Memory leak, too much slow scenarios/ activities (compared to requirement doc or bench mark), activities using too much resource.
4. Maximum operating capacity of an application.
5. The behavior of the application at various load level.

#### Test Approach

As you begin load testing, it is recommended that you start with a small number of virtual users and then incrementally increase the load from normal to peak. You can then observe how your application performs during this gradually increasing load condition. Eventually, you will cross a threshold limit for your performance objectives. For example, you might continue to increase the load until the server processor utilization reaches 75 percent, or when end-user response times exceed 8 seconds. You can use Stepping Thread Group like below, target load=50.



#### Key Criteria to Monitor

**[Response Times vs Threads](#_Response_Times_vs),**

**[Response Times Over Time](#_Response_Times_Over),**

**[The Results of Aggregate Report](#_The_Results_of),**

**[Transactions Per second](#_Transactions_Per_second),**

**[JavaMelody](#_JavaMelody):** CPU, Memory, disk, network

**[WAS PMI](#_WAS_PMI):** thread pool usage in WAS, connection pool usage in WAS, JVM status, system memory usage, CPU usage, Object Remote Broker thread pool …etc.

#### Expect Result

1. The behavior of the application at various load level. By testing a variety of scenarios, including what the team determines to be “best,” “worst,” and “expected” cases in terms of the measurements being collected.
2. System performance based on target load level will be satisfied by end user.
3. System capacity: Maximum operating capacity of an application. How many users and/or transactions a given system will support and still meet performance goals.

#### Analyze Result

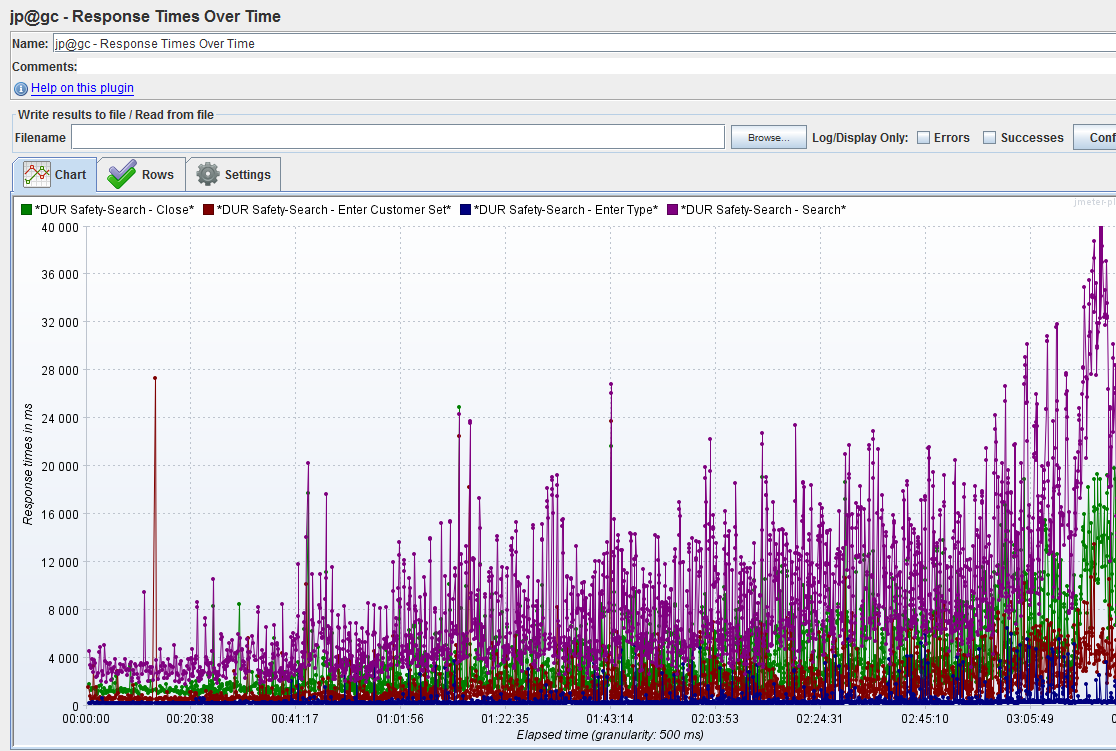
In this graph, a dashed line represents the performance goal. The 3 scenarios performances different under increasing load. The three curves represent the results from the worst-case (most performance-intensive), best-case (least performance-intensive), and expected-case user community models. Observing where these curves cross the horizontal line, one can see how many users can access the system in each case while still meeting the stated performance goal.



**Below is Dursafety application on DO3 test result:**



Users >9, response time of “Search” becomes greater than 10s.



## Stress Test

Stress testing is a type of performance testing focused on determining an application’s robustness, availability, and reliability under extreme conditions. The goal of stress testing is to identify application issues that arise or become apparent only under extreme conditions. These conditions can include heavy loads, high concurrency, or limited computational resources. Proper stress testing is useful in finding synchronization and timing bugs, interlock problems, priority problems, and resource loss bugs. The idea is to stress a system to the breaking point in order to find bugs that will make that break potentially harmful. The system is not expected to process the overload without adequate resources, but to behave (e.g., fail) in an acceptable manner (e.g., not corrupting or losing data).

**Design your test under different stress.** Examples of stress conditions include:

1) Excessive volume in terms of either users or data;

2) Resource reduction such as a disk drive failure or limited computational resources.

3) Unexpected sequencing.

4) Unexpected outages/outage recovery.

### Application stress testing

This type of test typically focuses on the whole application under stress, without the isolation of components/scenarios.

#### Test Goal

With application stress testing, you are likely to uncover defects related to data locking and blocking, network congestion, and performance bottlenecks on different components or methods across the entire application.

#### Test Approach

**Scenario:** Run Main script, which includes all test scenarios in proper possibility.

**Workload:** Exp: 1,000 simultaneous users. After loadtest, you can find system capacity, use more users than most user system can contain.

**Manual:** 1 user tests all scenarios to see response content directly.

**Think time:** Use a random think time between 1 and 10 seconds in the test script after each operation.

**Test Duration:** Run the test for two days.

#### Key Criteria to Monitor

Response time, Response content, CPU usage, Memory usage, free memory, Transaction error percentage

#### Expect Result

1) Application hosting process should not recycle because of deadlock or memory consumption.

2) Throughput should not fall below 35 requests per second.

Response time should not be greater than 7 seconds for 95 percent of total transactions completed.

4) “Server busy” errors should not be more than 10 percent of the total response because of contention-related issues.

Order transactions should not fail during test execution. Database entries should match the “Transactions succeeded” count.

#### Test Result

Mark “Pass/Fail” to each item in expects result.

### Transactional stress testing

Transactional stress tests aim at working at a transactional level (like: search/order/import/…) with load volumes that go beyond those of the anticipated production operations.

#### Test Goal

These tests are focused on validating transaction behavior under stressful conditions.

#### Test Approach

Design your test according to performance special requirement.

#### Key Criteria to Monitor

#### Expect Result

#### Analyze Result

#### Test case

The following is some examples of test case based on the special requirement.

##### Test case 1

**Test goal:** Order function does not fail under high concurrency.

**Test Design:** A lot of users will place order at the same time. It will be implemented by running script. In this case excessive volume is in terms of users;

**Scenario:** Order script

**Workload:** 1,000 simultaneous users.

**Think time:** Use a random think time between 1 and 10 seconds in the test script after each operation.

**Test Duration:** Run the test for two days.

**Expected results:**

1) Application hosting process should not recycle because of deadlock or memory consumption.

2) Throughput should not fall below 35 requests per second.

3) Response time should not be greater than 7 seconds for 95 percent of total transactions completed.

4) “Server busy” errors should not be more than 10 percent of the total response because of contention-related issues.

5) “Server Order transactions should not fail during test

**Test Result:**

Mark “Pass/Fail” to each item in expects result.

##### Test case 2

**Test Goal:** Search function does not fail under stress in terms of excessive volume of data;

**Test Design:** A user specifies a particularly wide query. It means much more records will match criteria. there could be a large impact on memory utilization. For example, memory utilization could be affected if a query returns an entire data table. It can be implemented by DEV/QA manual. In this case excessive volume is in terms of data;

**Workload:** 1 simultaneous user.

**Think time:** Use a random think time between 1 and 10 seconds in the test script after each operation.

**Test Duration:** Run the test for 1 hour.

**Expected results:**

1. Application hosting process should not recycle because of deadlock or memory consumption.

2) Response time should not be greater than 10 seconds for 95 percent of total transactions completed.

3) “Server busy” errors should not be more than 10 percent of the total response because of contention-related issues.

Search transactions should not fail during test

**Test Result:**

Mark “Pass/Fail” to each item in expects result.

##### Test case 3

**Test Goal:** Import function does not fail under stress in terms of excessive volume of data;

**Test Design:** Import data includes 10000 records. It can be implemented by DEV/QA manual. In this case excessive volume is in terms of data;

**Workload:** 1 simultaneous user.

**Test Duration:** Run the test for 1 day.

**Expected results:**

Import transactions should not fail during test

Application hosting process should not recycle because of deadlock or memory consumption.

Response time should not be greater than 10 minutes for 95 percent of total transactions completed.

**Test Result:**

Mark “Pass/Fail” to each item in expects result.

##### Test case 4

**Test Goal:** Test if system performances well when there are more than 100 entries or date ranges.

**Test Design:**

**Workload:**

**Test Duration:**

**Expected results:**

Update transactions should not fail during test

Application hosting process should not recycle because of deadlock or memory consumption.

Response time should not be greater than 10 minutes for 95 percent of total transactions completed.

**Test Result:**

Mark “Pass/Fail” to each item in expects result

## Monitor

Reevaluate the metrics to be collected regularly. Goals, priorities, risks, and current issues are bound to change over the course of a project. With each of these changes, different metrics may provide more value than the ones that have previously been identified.

### Key Metrics

|  |  |  |
| --- | --- | --- |
| **Category** | **Performance metrics** | **Tool** |
| **Network-specific metrics** | | |
| Network | Information about the overall health and efficiency of your network, including routers, switches, and gateways. |  |
|
|
|
|
|
| **System-related metrics** | | |
| Processor | Processor utilization | PerfMon Metric Collector  JavaMelody Vmstat Topas nmon (JavaMelody is usually used in practice.  Gragh display, choose PerfMon Metric Collector/JavaMelody. And PerfMon Metric Collector is monitored from Jmeter. Vmstat/topas/nmon will use command line.) |
|
|
|
|
|
| Process |  | Topas |
| Memory consumption |
| Processor utilization |
| Process recycles |
|  |
| Memory | Memory available | As tools to monitor "Processor" |
| Memory utilization |
| GC frequency | GC Log Dump file |
| GC overhead |
| GC duration |
| JVM free heap size after GC …etc related to memory |
| Disk I/O | Disk utilization | As tools to monitor "Processor" |
|
|
| Network I/O | Network usage | As tools to monitor "Processor" |
| **Platform-specific metrics** | | |
| Pool usage | Thread pool usage in WAS | WAS PMI |
| Connection pool usage in WAS |
| Object Remote Broker thread pool …etc |
| **Application specific statistics** | | |
| Response times | Transactions times | Jmeter: Aggregate Report |
| Average |
| Medium |
| 90% Line |
| Max |
| Min |
| Response time distribution | Jmeter: Response time distribution |
| Response Times vs Threads | Jmeter: Active Threads Over Time Listener Jmeter: Response Times vs Threads |
| Transactions/business metrics | Transactions/sec | Jmeter: Transactions per Second |
| Transactions & thread | Jmeter: Transaction Throughput vs Threads |
| Transactions succeeded | Jmeter: Aggregate Report |
| Transactions failed |
| Operation succeeded | Jmeter: View result in table |
| Operation failed |
| Response time at various levels | Web Action/Action Listner | JavaMelody |
| EJB service invocation |
| SQL execution |
| **Business metrics** | | |
| Business-related information | Such as the number of orders placed in a given timeframe. |  |

System throughput is the sum of workload unit that are processed by server within a unit time. It is an importance indicator of system efficiency. Many factors may influence system throughput, such as workload, software architecture, hardware and system configurations.

Throughput refers to the yield or the amount of service which business system provides within a unit time. Throughput is usually the overall evaluation of the capacity of a system and its components processing transmission data request's ability. Throughput of web application system is usually measured in requests per second or pages per second.

Theoretically, we can get the request sequence and calculate the windowed throughput over time series.

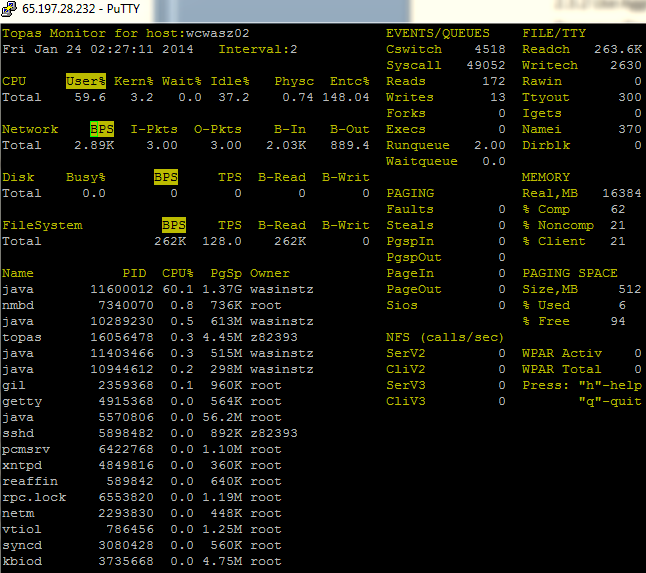
### Monitor Tools

#### PerfMon Metric Collector

[PerfMon Metric Collector](#_PerfMon_Metric_Collector)

#### Command of Vmstat, nmon, topas

It is used monitor CPU, Memory, Disk, Network as above, but has no gragh report.



#### Aggregated Report, Response Time Distribution, Graph Response Times vs Threads ,Transaction Throughput vs Threads

Refer to [View Results](#_View_Results)

#### WAS PMI

It is used to monitor Threading/Web container thread pool size/EJB ORB/EJB default/

#### JavaMelody

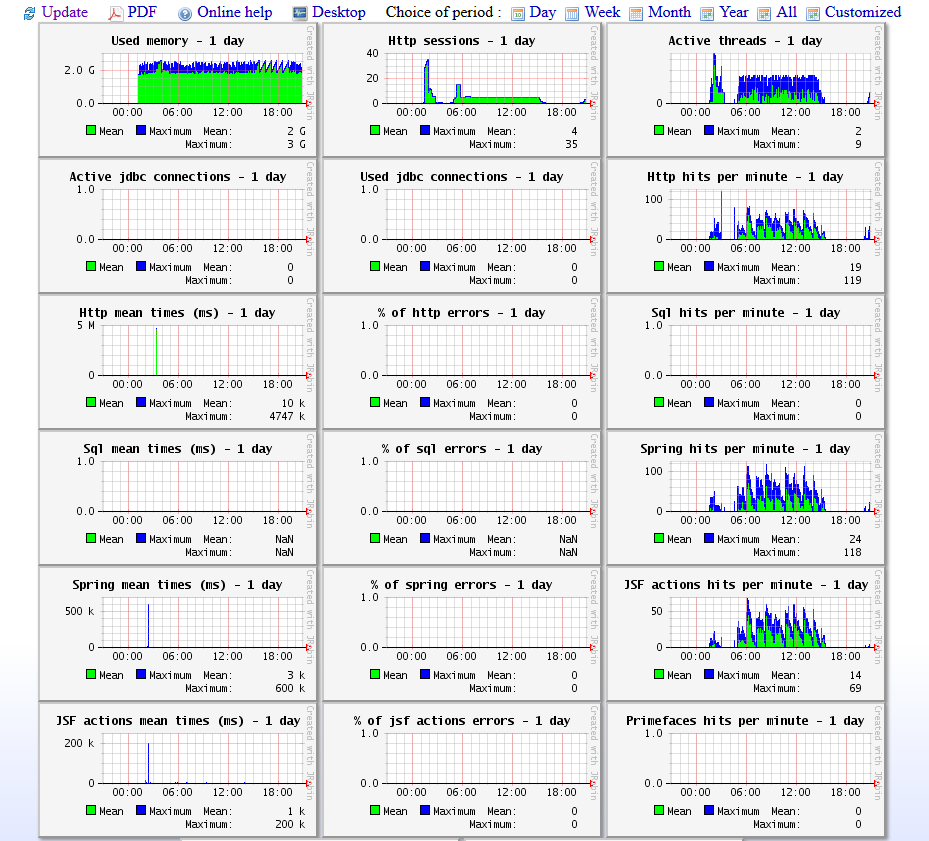
Wiki:

<http://dstconnect.dstcorp.net/display/Argus/Monitoring+Java+Web+Application+with+Javamelody>

Code source:

<http://svn/commons/com/argushealth/enterprise/javamelody-core/trunk>

Report:



It is a JVM based performance monitoring tool and used to monitor the performance of application:

1) Server level statistics: memory, CPU, thread status, etc

2) Web specific statistics: HTTP session usage, response time, errors, etc

3) Application specific statistics: response time at various levels such as web code, ejb service invocation, SQL execution, etc

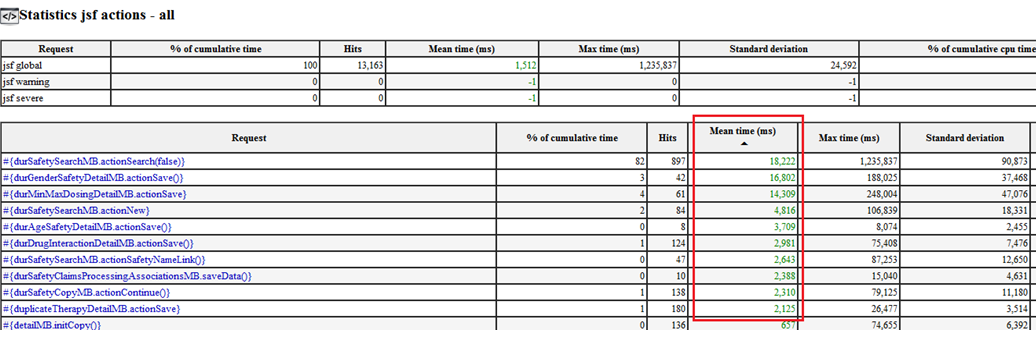
Some highlights of the report:

1) Statistics jsf actions: reflects the performance of web code, see snapshot below

2) Statistics spring: reflects the performance of ejb service invocation

3) Statistics sql: reflects the performance of SQL execution (still in researching)

Detail report:



It is proved that above statistics can help us to track performance issues and identify the root cause efficiently. Below are the major issues that we have found so far, most of which are common across similar functionalities within the application.

