Markov4JMeter Tutorial

André van Hoorn

June 17, 2008

Abstract

By extending Apache JMeter with Markov4JMeter you're able to execute *Test Plans* based on configurable probabilistic behavior models: given a currently visited Web site, the next site to be requested is chosen based on configurable probability values. Moreover, Markov4JMeter allows to define how to vary the number of active user threads during test execution.

Our paper describing the the probabilistic and intensity-varying workload generation approach has been published in the proceedings of the *SPEC International Performance Evaluation Workshop 2008 (SIPEW '08)*:

> André van Hoorn, Matthias Rohr, and Wilhelm Hasselbring. *Generating Probabilistic and Intensity-varying Workload for Web-based Software Systems*. In Performance Evaluation – Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop (SIPEW '08), vol. 5119 of *Lecture Notes in Computer Science (LNCS)*, pages 124–143. Heidelberg: Springer (June 2008).

Throughout this tutorial, we will give installation instructions and show how to create an executable *Test Plan* for the iBATIS JPetStore sample application. The JMX file of the Test Plan and the associated files can be found in the directory *examples/jpetstore/tutorial* of the Markov4JMeter release. Extended example JMX file which are not covered by this tutorial are included as well.

We assume that you installed Apache JMeter version 2.2 or higher and know how to use it. Markov4JMeter requires the Java Runtime Environment versioned 1.5 or higher.

1 Downloading and Installing Markov4JMeter

From the Markov4JMeter homepage `http://markov4jmeter.sourceforge.net` you can download two types of archives:

Source Archives: The source archive contains the Markov4JMeter sources. These archive names follow the pattern *markov4jmeter-<version>-_src.{zip|tgz}*. This archive is required when Markov4JMeter shall be modified and compiled. It includes further instructions to do so.

Runtime Archives: The runtime archive contains a runnable version of Markov4JMeter which can be used with JMeter. These archive names follow the pattern *markov4jmeter-<version>.{zip|tgz}*.

In order to install a runnable version of Markov4JMeter, the following steps need to be performed:

1. Download and extract the Markov4JMeter runtime archive in the desired file type (tgz- or zip-format).
2. Copy the *Markov4JMeter.jar*, which resides inside the *dist/* directory, to the directory *lib/ext/* of the JMeter installation.
3. After restarting JMeter, two new entries within the Logic Controllers menu exist: Markov State and Markov Session Controller (see Figure 1).

Figure 1: After installing Markov4JMeter, the Logic Controller menu shows two new entries: Markov State and Markov Session Controller.

2 Using Markov4JMeter

This section contains a step-by-step description how a simple probabilistic Test Plan for the JPetStore is created. We use the publicly accessable JPetStore instance hosted by `http://www.jwebhosting.net/`, such that you're not required to set up your own JPetStore in order just to follow the instructions given in this tutorial. Hence, please keep the number of parallel users emulated by your first Markov4JMeter experiments small in order not to cause too much load on the provided server. A Markov4JMeter Test Plan can then be executed just like any ordinary JMeter Test Plan.

Preparing the Test Plan

By performing the following steps, the basic Test Plan shown in the left-hand side of Figure 2 is created.

1. Add a *Thread Group* to the empty *Test Plan* and select to stop the test when a sampler error occurs. Set the number of threads to 5 and the loop count to 5 without using the *Scheduler*.
2. Add an *HTTP Cookie Manager* from the *Config Element* menu to the *Thread Group* and select the cookies to be deleted after each iteration.
3. Add the *HTTP Request Defaults* from the *Config Element* menu and insert the data shown in Figure 2.
4. Add a *View Results Tree* for debugging purposes and select "Save Response Data".

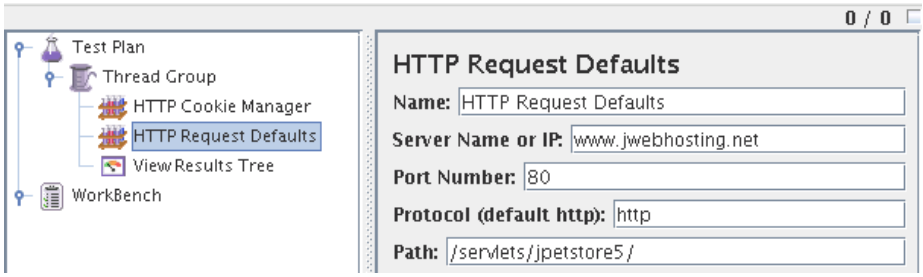

Figure 2: A prepared JMeter Test Plan.

Figure 3: After installing Markov4JMeter, the Logic Controller menu shows two new entries: Markov State and Markov Session Controller.

Adding a Markov Session Controller

After installing Markov4JMeter, the new Logic Controllers Markov State and Markov Session Controller appear in the respective menu (see Figure 3).

A Markov Session Controller needs to be added to the *Thread Group*. This is the root element of any probabilistic session model consisting of a number of Markov States and transitions between them. Also, it contains the configuration of the behavior mix and the Session Arrival Controller. A *Gaussian Random Timer* added to the Markov Session Controller emulates client-side think times between subsequent requests.

It is highly recommended to use at most one Markov Session Controller within a Test Plan. The Markov Session Controller should be placed at the root level of the Thread Group. Especially, Markov Session Controllers must not be nested.

Name	**Path**	**Method**	**Parameters**	
			Name	Value
Markov State: Index				
index.shtml	[JPSROOT]/index.shtml	GET		
Markov State: Sign On				
signonForm.shtml	[JPSROOT]/signonForm.shtml	GET		
signon.shtml	[JPSROOT]/signon.shtml	POST	username password submit	j2ee j2ee Login
Markov State: View Category				
viewCategory.shtml	[JPSROOT]/viewCategory.shtml	GET	categoryId	REPTILES
Markov State: Sign Off				
signoff.shtml	[JPSROOT]/signoff.shtml	GET		

Table 1: Data to fill in to the *HTTP Request* configuration dialogs. "[JPSROOT]" needs to be replaced with "/servlets/jpetstore5/shop". Also, the check boxes "Redirect Automatically" and "Follow Redirects" must be selected.

Adding Markov States

After adding four Markov States named "Index", "Sign On", "View Category", and "Sign Off" to the Markov Session Controller, the Test Plan has the tree structure shown in Figure 4. Markov States must be placed directly underneath a Markov Session Controller and must especially not be nested – neither directly nor indirectly. Each Markov State must have a unique name. *HTTP Request Samplers* should be added to the Markov States according to Table 1.

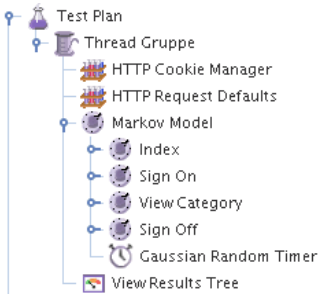

Figure 4: Markov4JMeter Test Plan.

Defining Transition Guards and Actions

When selecting a Markov State within the Test Plan, the configuration dialog including the table to define guards and actions for transitions to all states of the same Markov Session Controller appears. The table is automatically updated each time Markov States are added, removed or renamed. Transitions can be assigned *guards* in order to allow a transition to be taken only if the specified expression evaluates to *true*. By selecting the respective check box, transitions can be deactivated completely, which is equivalent to entering a guard evaluating to *false*. An *action* is a list of statements, such as function calls or variable assignments, separated by a semicolon which is evaluated when a transition is taken.

In our example a variable *signedOn* is used to remember whether a user has logged in or not. A *User Parameters Pre-Processor* to the Markov Session Controller with a new variable named *signedOn* with the value *false* to initialize the variable. The check box "Update Once Per Iteration" needs to be activated. The guards and actions of the transitions should be configured as listed in Table 2.

Source State	Destination State	Disabled	Guard	Action
any	Sign On		!${signedOn}	signedOn=true
any	Sign Off		${signedOn}	signedOn=false

Table 2: Guards and actions used to restrict transitions to the states "Sign On" and "Sign Off". The variable *signedOn* is used to remember whether a user has logged in or not.

Creating User Behavior Models and Defining the BehaviorMix

A behavior file template can be exported by clicking the button "Generate Template" within the Markov Session Controller. This file can then be edited in a spread sheet application or a text editor. The sum of probabilities in each row must be 1.0. This step needs to be performed for each behavior model to be used.

The behavior mix, i.e. the assignment of behavior models to their relative frequency of occurrence during execution, is defined in the configuration dialog of the Markov Session Controller. Entries can be added and removed using the buttons "Add" and "Delete". Again, the sum of relative frequencies must be 1.0. Absolute and relative filenames can be used. Relative filenames are always relative to the directory JMeter has been started from. Figure 5 shows an example behavior mix.

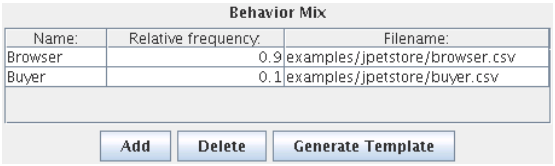

Figure 5: Example Behavior Mix.

If an error occurs while loading the behavior models, the entire test is aborted immediately. Details concerning this error are written to the file *jmeter.log*.

Using the Session Arrival Controller

The Session Arrival Controller controls the number of active sessions, given an expression evaluating to an integer value. It is configured within the configuration dialog of the Markov Session Controller. JMeter allows to use BeanShell [1] scripts within text expressions. Particularly, this allows for varying the number of active sessions depending on the elapsed experiment time. Markov4JMeter adds

[1] Notice that the BeanShell Jar file is not included in the JMeter release. The file can be downloaded from `http://www.beanshell.org/download.html` and must be copied to the *lib/opt/* directory of the JMeter installation. See `http://jakarta.apache.org/jmeter/usermanual/functions.html#__BeanShell` for a description on how to use BeanShell scripts within JMeter expressions.

```
import org.apache.jmeter.util.JMeterUtils;

long startMs = ↩
    Long.parseLong(JMeterUtils.getPropDefault("TEST.START.MS",""));
long curMs = System.currentTimeMillis();
double expMin = (curMs-startMs)/(1000*60);
return (int)Math.ceil(allowedNum(expMin));
```

Figure 6: Example BeanShell script which returns the elapsed experiment minute as an integer using the Markov4JMeter variable *TEST.START.MS*.

the global variable *TEST.START.MS* to the JMeter context which is accessible in BeanShell scripts. The variable is set when the Test Plan is started.

For example, when using the BeanShell script listed in Figure 6, the Session Arrival Controller limits the number of active sessions based on the elapsed experiment minute: in the i-th minute i active sessions are allowed in parallel. Figure 7 shows how to use this BeanShell script within the Session Arrival Controller configuration.

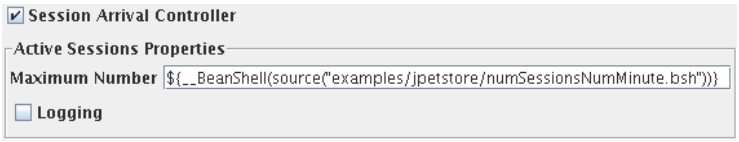

Figure 7: BeanShell scripts can be used with the function *BeanShell*. The BeanShell function *source* includes a script file.

The Session Arrival Controller doesn't create any new threads. It simply blocks threads in a queue in case the number of active sessions would exceed the given maximum number. Hence, the maximum number of active sessions is limited to the number of threads configured in the *Thread Group*.

If an error occurs while evaluating the active sessions formula, the entire test is aborted immediately. Details concerning this error are written to the file *jmeter.log*.

3 Running the Test

Start the test run just as you would start any JMeter test. You should always check *jmeter.log* in order to get information about possible errors. You can use JMeter to render the resulting HTML response data for each request in order to make sure you set up your Test Plan properly.