EStoreSim - User Guide

M. Amir Moulavi

September 2011

Contents

List of Figures

Listings

Chapter 1

Check out and build

The code for simulation framework is distributed under Apache Software License. Check out the source code from `Github` repository:

```
 # git clone git://github.com/amir343/ElasticStorage.git
```

You should have `Maven` 2 installed in order to be able to build the source code:

```
 # mvn clean install
```

This will download the required libraries from Maven repositories and builds the source code.

Chapter 2

How to construct the cloud?

A cloud environment consists of several elements that each one is described in the following can be passed to `Cloud` class:

- `cloudProviderAddress(address, port)*`: binds the cloud provider to this address and port. The address should refer to `localhost`.
- `data(name, size, size unit)*`: defines the data blocks that will be used during the simulation and their corresponding sizes. The names must be unique.
- `replicationDegree(number)*`: defines the replication degree to meet by Elastic Load Balancer.
- `addressPoll(file name)*`: points to address ranges that are available for this simulation.
- `node(name, address, port)`: defines a storage instance with a name and address, port to bind to. The name should be unique.
  - `cpu(n)`: defines the CPU for the instance with frequency of `n` GHz.
  - `memoryGB(n)`: defines the memory for the instance with size of `n` GB.
  - `bandwidthMB(n)`: defines the bandwidth for the instance with capacity of `n` MB.
- `sla()`: enables the SLA violation calculation according to the parameter that will be defined as the following:
  - `cpuLoad(n)`: SLA requirements for percentage of average CPU load in the system.
  - `responseTime(n)`: SLA requirements for average response time (`ms`) in the system.

  – `bandwidth(n)`: SLA requirements for average bandwidth per download (`B/s`) in the system.

Using these elements a complete cloud environment can be built. Note that elements related to instances are not mandatory. Instances can be launched by cloud provider as well. After this definition, the object that holds theses elements should be started by calling the method `start()`.

Controller also should be defined but separately. The class that is responsible for controller is `ControllerApplication` and has the following element:

- `controllerAddress(address, port)`: defines the address and port that controller will be bind to.

Note that the address should be different from address that are included in `addressPoll` and cloud provider address. Controller is started with a method called `start()`. These definition can be put in a source code with desire name for class inside a `main` method. It is a good practice to put your class in package `scenarios.executor`. To start the simulation it is enough to run this class. An example scenario can be like the following:

Listing 2.1: A sample scenario to create cloud environment

```java
package scenarios.executor;

import cloud.CloudProvider;
import econtroller.ElasticController;
import instance.Instance;
import instance.common.Size;
import org.apache.log4j.PropertyConfigurator;
import scenarios.manager.Cloud;
import scenarios.manager.ControllerApplication;

public class TestScenario {

    public static final void main(String[] args) {
      Cloud cloud = new Cloud(CloudProvider.class, Instance.class) {
        {
          cloudProviderAddress("127.0.0.1", 23444);
          node("node1", "127.0.0.1", 23445).
            cpu(2.2).
            memoryGB(8).
            bandwidthMB(2);
          data("block1", 2, Size.MB);
          data("block2", 4, Size.MB);
          data("block3", 3, Size.MB);
          data("block4", 1, Size.MB);
          data("block5", 4, Size.MB);
          replicationDegree(2);
          addressPoll("addresses.xml");
          sla()
            .cpuLoad(30)
            .responseTime(1000);
        }
      };

      cloud.start();

      ControllerApplication controller =
    new ControllerApplication(ElasticController.class) {
        {
          controllerAddress("127.0.0.1", 23443);
        }
      };

      controller.start();
    }
```

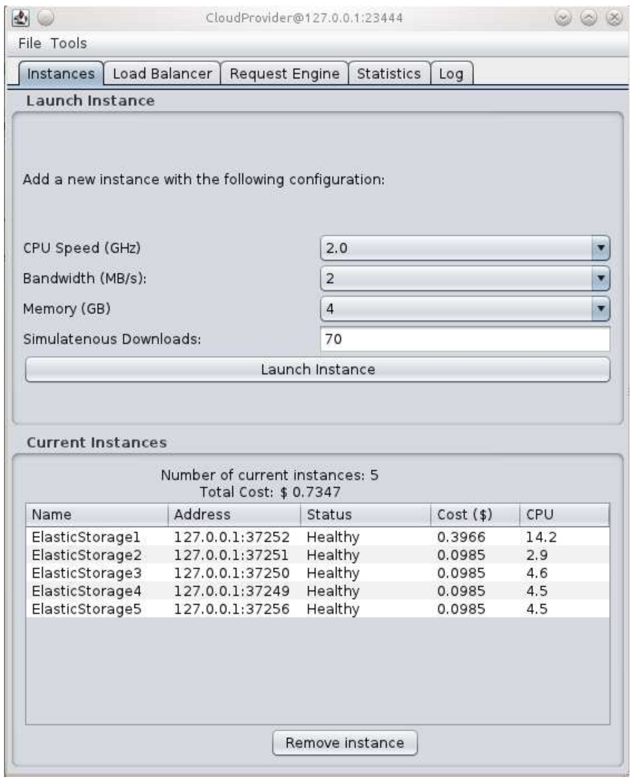
2.1 addressPoll

`addressPoll` is an XML file that defines the elastic IP address ranges that can be used by cloud provider. A simple example is like:

Listing 2.2: A sample Elastic IP addresses definition

```xml
<addressPoll>
  <addresses>
    <addressRange>
      <ip>127.0.0.1</ip>
      <startPort>37000</startPort>
      <endPort>38001</endPort>
    </addressRange>
    <addressRange>
      <ip>127.0.0.1</ip>
      <startPort>47000</startPort>
      <endPort>48005</endPort>
    </addressRange>
    <addressRange>
      <ip>127.0.0.1</ip>
      <startPort>51000</startPort>
      <endPort>51857</endPort>
    </addressRange>
  </addresses>
</addressPoll>
```

This definition includes 2866 unique elastic addresses. Note that `startPort` can not be greater than `endPort`. The cloud provider window is demonstrated in Fig. 2.1



Add a new instance with the following configuration:

CPU Speed (GHz)	2.0
Bandwidth (MB/s):	2
Memory (GB)	4
Simulatenous Downloads:	70

Number of current instances: 5
Total Cost: $ 0.7347

Name	Address	Status	Cost ($)	CPU
ElasticStorage1	127.0.0.1:37252	Healthy	0.3966	14.2
ElasticStorage2	127.0.0.1:37251	Healthy	0.0985	2.9
ElasticStorage3	127.0.0.1:37250	Healthy	0.0985	4.6
ElasticStorage4	127.0.0.1:37249	Healthy	0.0985	4.5
ElasticStorage5	127.0.0.1:37256	Healthy	0.0985	4.5

Figure 2.1: Simulation Framework - Cloud Provider Window

2.2 Taking snapshot

At any point during the running of the simulation, you can take snapshots from any of cloud provider, instance or controller. This can be done by using `ALT + s` keys. You can take as many snapshots as you may need. These snapshots are saved into *Snapshots* panel in controller/instances windows and *Statistics* panel in cloud provider. You can save any of the snapshots to disk later on by selecting and right clicking on your desire one in the corresponding panel. If you need only the log messages to be saved you can right click on the log panel and select `save to file`.

Each snapshot includes the set of all diagrams and also log messages for that specific node in the system.

2.3 Headless Run

In order to consume less resources on the target machine, you can specify `headless()` in the cloud class of your scenario. This will prevent any construction of GUI for storage instances and save a lot of JVM heap memory.

An instance windows is shown in Fig 2.2.

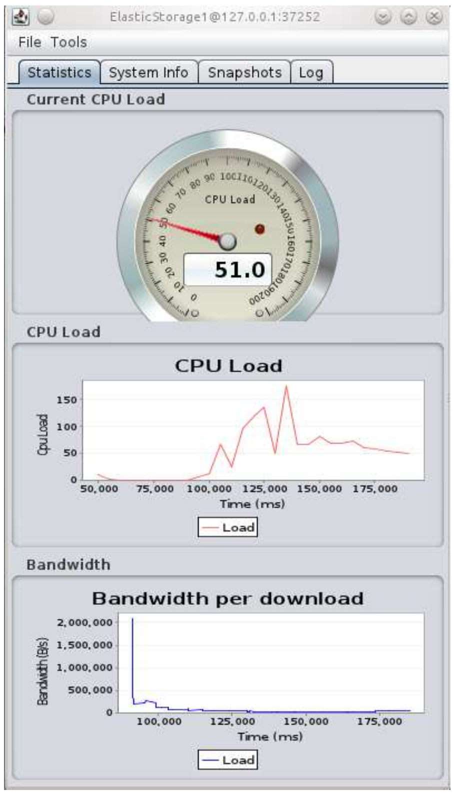


Figure 2.2: Simulation Framework - Instance Window

2.4 How to add your own controller

If you require to write your own controller, you can do so by implementing the `ControllerDesign` interface. When you run the simulation, your controller will be listed with the name you have chosen for the class that implements the interface. It is a good practice to put the implemented class in package `econtroller.design.impl`. The name should not be used before and should be unique.

2.5 How to define custom distributions

You can define your own custom distribution to be used by Request Generator when sending request to cloud provider. Simply create a file and put your timeouts one in each line. So if you have something like

```
3
10
```

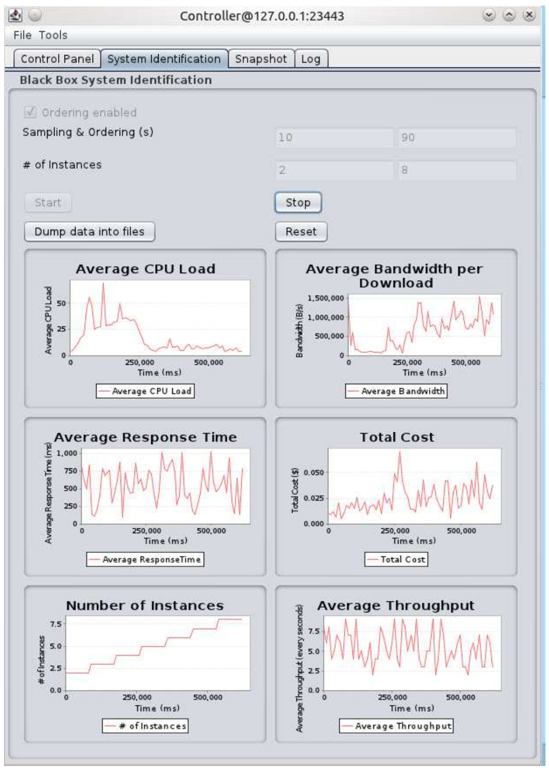
it means that the first request will be sent after 2 seconds, second request after 5 and the third one after 15 seconds. When the request generator reaches the end of the file, it repeats the distribution from the beginning. After you create the file, choose *Custom* from drop down list in Request Generator panel in cloud provider window.

2.6 How to do System Identification?

First define your scenario with Cloud and controller classes. Run your scenario. Try to launch at least two instances by cloud provider. Then connect the controller to cloud provider. In the meantime the instances are launching, select the distribution you want the Request Generator to send request and start it. When the instances are launched and ready to use, browse to System Identification panel in controller window. Make sure that settings are the ones you need start the modeler.

At any time during the system identification, you can stop the modeler or dump the collected training data into files. After one period that instances are scaled up and down, the modeler dumps the collected data into files and takes a snapshot as well which is listed in snapshot panel. It is recommended to do the system identification in headless mode that is described earlier. A sample system identification is depicted in Fig. 2.3.

By default `Ordering enabled` check box is checked and it means that modeler acts as a System Identifier and orders new to launch/shut down instances. However, if you need to just monitor the system without affecting the number of instances, you can disable this option.


Controller@127.0.0.1:23443

Black Box System Identification

☑ Ordering enabled

Sampling & Ordering (s)	10	90
# of Instances	2	8

Figure 2.3: Controller Window - System Identification