

Experiments Setup Guide

Look Ahead Distributed Planning

Contents

Introduction.....	2
Environment Setup.....	2
Database.....	2
Web Servers.....	2
Load Balancer	4
Feeder.....	5
Proxy Servers	7
Slave.....	7
Master	9
Proxy Server Configuration.....	10
General Configuration	10
Scenario Definitions.....	12
Adaptation Options	13
Access to Outputs.....	14
Adaptations Count.....	14
Arrival Rates.....	14
Average Adaptation Duration.....	14
Bandwidth	14
Bandwidth Weights	15
Response Times	15
Servers Count	15
Utilization	15
Violations Count	15

Introduction

This document is devoted to providing a systematic guide for running experiments and setting up their prerequisites. Following the instructions in the presented order is necessary. The experiments environment is composed of one Database server, three Web Servers, one Load Balancer, one Feeder, and two Proxy Servers (Master and Slave). AMI images for all nodes have been created and are available.

Environment Setup

In this section, we provide the necessary information required for setting up the experiment environment using existing AMI images and from scratch.

Database

First, we need to set up a database server. The database server hosts a single node MySQL server. Create a new instance using the `DP-Database` AMI image (ID: `ami-ace8f2d3`). Please use the following configuration to set up a new instance of the database server:

```
Instance Type: m4.large (vCPUs: 2, Memory: 8GB)
Network: DPNet
Primary IP: 10.1.0.61
Storage: GP2 (16GB)
Security Group: DPSecGroup
```

By running the newly created instance, a MySQL server will be started, which hosts a database named `eshop`. If you need to recreate the database, a backup file named `databse.sql` file is attached to this document.

Web Servers

It is time to setup three web servers. Each web server hosts an Apache Tomcat 8.0. Create a new instance of each webserver using `DP-WebServer1` (ID: `ami-01b7075b646ceebcd`), `DP-WebServer2` (ID: `ami-0e90d96fe3a11f0f0`), and `DP-WebServer3` (ID: `ami-0c0ec5530729cf2e3`) AMI images. Please use the following configuration to setup a new instances of web servers:

```
Instance Type: m4.large (vCPUs: 2, Memory: 8GB)
Network: DPNet
Primary IP: WebServer1: 0.1.0.51, WebServer2: 0.1.0.52, WebServer3: 0.1.0.53
Storage: GP2 (16GB)
Security Group: DPSecGroup
```

By running newly created instances, an Apache Tomcat instance will be started on port 8080. A sample application is already deployed as the root application of Tomcat servers. However, if you want to set up web applications from scratch, after installing Open JDK 8 and Apache Tomcat 8.0, a data source definition should be added to the `context.conf` file of the Tomcat configuration:

```
<Resource
  name="jdbc/EshopPool"
  auth="Container"
  type="javax.sql.DataSource"
  driverClassName="com.mysql.jdbc.Driver"
  url="jdbc:mysql://10.1.0.61/eshop?useUnicode=true&characterEncoding=UTF-
8&connectionCollation=utf8_persian_ci"
  username="root"
  password="#####"
  maxActive="200"
/>
```

You can access the source code via the following GitHub repository. The source code should be cloned into `/home/src` directory:



<https://github.com/FarzinZaker/DP-Application>
[git@github.com:FarzinZaker/DP-Application.git](https://github.com/FarzinZaker/DP-Application.git)

The sample application is developed on the Grails framework version 3.2.6. Download links for different versions of the Grails framework are available on its official website:



<http://grails.org/download.html>

Download and unzip the appropriate version of Grails framework into `/home/grails` directory. Make sure that the `ubuntu` user has full access to the specified path.

To build a war file and deploy it on Tomcat, you need to create a build script named `build.sh` in `/home/Ubuntu` directory:

```
sudo systemctl stop tomcat

git -C /home/src/DP-Application pull

rm -rf /home/src/DP-Application/build/libs/*.war

(cd /home/src/DP-Application && /home/grails/grails-3.2.6/bin/grails clean)

(cd /home/src/DP-Application && /home/grails/grails-3.2.6/bin/grails war)

sudo rm -rf /opt/tomcat/webapps/ROOT
sudo rm -rf /opt/tomcat/webapps/ROOT.war

sudo cp /home/src/DP-Application/build/libs/*.war /opt/tomcat/webapps/ROOT.war

sudo systemctl start tomcat
```

The build script is responsible for performing the following actions:

1. Stopping the Tomcat if it is running
2. Pulling latest changes from the repository
3. Removing older WAR files from the output directory

4. Cleaning the project
5. Building a new WAR file
6. Removing old WAR files from `webapps` folder of the Tomcat
7. Copying the WAR file to the `webapps` folder
8. Starting the Tomcat

Do not forget to make the script file executable using the following command:

```
Chmod u+x ~/build.sh
```

Now, after each modification in the source code, the deployment process is as easy as executing the following command:

```
~/build.sh
```

Load Balancer

An Nginx web server is responsible for load balancing incoming requests between 1 to 3 available web servers. Although an up and running instance is available for the load balancer, you can create a new instance using `DP-LoadBalancer` AMI image (ID: `ami-01a54431e23a1c264`). Please use the following configuration to setup a new load balancer:

```
Instance Type: t2.medium (vCPUs: 2, Memory: 4GB)
Network: DPNet
Primary IP: WebServer1: 0.1.0.41
Storage: GP2 (8GB)
Security Group: DPSecGroup
```

The load balancer is configured to distribute incoming requests between three available web servers. Make sure that the second and the third web servers are excluded in the `/etc/nginx/nginx.conf` file at the beginning. The proxy server will take advantage of other web servers if required during serving incoming requests. In addition, forwarding rules for each scenario should be declared the load balancer configuration.

```

upstream site {
    server 10.1.0.51:8080;
    #server 10.1.0.52:8080;
    #server 10.1.0.53:8080;
}

location /static {
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_pass http://site/static;
    proxy_read_timeout 600s;
}

location /product {
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_pass http://site/product;
    proxy_read_timeout 600s;
}

location /browse {
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_pass http://site/browse;
    proxy_read_timeout 600s;
}

location /basket {
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_pass http://site/basket;
    proxy_read_timeout 600s;
}

```

Feeder

The feeder node is responsible for sending requests to the proxy server using an executable file name `feeder.jar`. Unlike other nodes, a new feeder node should be created each time using the DP-Feeder AMI image (ID: `ami-0fee0b71`). Since the network interfaces of the feeder node are being controlled using an overlay network, on each reboot connection to the node will be lost. In order to achieve more concurrency level, initialize the feeder node using the following configuration:

```

Instance Type: t2.xlarge (vCPUs: 4, Memory: 16GB)
Network: DPNat
Primary IP: 0.1.0.21
Storage: GP2 (8GB)
Security Group: DPSecGroup

```

Since we need to test four scenarios, four different instances of the `feeder.jar` should be deployed on the feeder server as shown below:

```
? ~/feeders
? 1
? feeder.jar
? 2
? feeder.jar
? 3
? feeder.jar
? 4
? feeder.jar
```

You can access the source code of the `feeder.jar` via the following GitHub repository, modify it and create new JAR files:



<https://github.com/FarzinZaker/DP-Feeder>
<git@github.com:FarzinZaker/DP-Feeder.git>

In addition, make sure to copy the `AccessLogs` folder to the home folder of the `ubuntu` user. Finally, to finalize the overlay network configuration run the following command:

```
~/ovs.sh
```

The `ovs.sh` file, creates a bridge on `eth0` named `feeder`, modifies routing rules to route packets through the `feeder` bridge and then creates four different ports (`p0`, `p1`, `p2`, and `p3`) on the `feeder` bridge for four different scenarios. These ports will be used by the proxy server to modify bandwidth limits for each scenario.

```

export mac=$(ifconfig -a | grep -Po 'HWaddr \K.*$')
sudo modprobe openvswitch
sudo /etc/init.d/openvswitch-switch start
sudo ovs-vsctl del-br feeder
sudo ovs-vsctl add-br feeder
sudo ovs-vsctl set bridge feeder other-config:hwaddr=$mac
sudo ovs-vsctl add-port feeder eth0
sudo ip addr del 10.1.0.21/24 dev eth0
sudo ip addr add 10.1.0.21/24 dev feeder
sudo ip route del default via 10.1.0.1 dev eth0
sudo ifconfig feeder up
sudo ip route add default via 10.1.0.1 dev feeder
sudo ovs-vsctl show
sudo ovs-vsctl list-br

sudo ip tuntap add mode tap p0
sudo ifconfig p0 up
sudo ip tuntap add mode tap p1
sudo ifconfig p1 up
sudo ip tuntap add mode tap p2
sudo ifconfig p2 up
sudo ip tuntap add mode tap p3
sudo ifconfig p3 up

sudo ovs-vsctl add-port feeder p0
sudo ovs-vsctl add-port feeder p1
sudo ovs-vsctl add-port feeder p2
sudo ovs-vsctl add-port feeder p3

sudo ip addr add 10.1.0.21/24 dev p0
sudo ip addr add 10.1.0.21/24 dev p1
sudo ip addr add 10.1.0.21/24 dev p2
sudo ip addr add 10.1.0.21/24 dev p3

```

Proxy Servers

There are two different types of proxy servers. One of the proxy servers plays the role of the master node, and the other one is a slave. The master node receives incoming requests, routes them to the load balancer and sends the response back to the requester. If the response time SLA for a scenario violates, the proxy server automatically runs the look ahead distributed planning routines to find and apply the best adaptation option. In case of high pressure, the master proxy server takes advantage of the slave and deploys some actors on the slave proxy server.

Slave

Create a new instance of slave proxy server using `DP-ProxyServer-Slave` AMI image (ID: `ami-0e365654c087a2ebd`). Please use the following configuration to set up a new slave proxy server:

```

Instance Type: t2.medium (vCPUs: 2, Memory: 4GB)
Network: DPNet
Primary IP: WebServer1: 0.1.0.32
Storage: GP2 (8GB)
Security Group: DPSecGroup

```

The slave proxy server is deployed on an Apache Tomcat instance and runs at the startup. However, if you want to set up proxy servers from scratch, you need to install Open JDK 8 and Apache Tomcat 8.0. You can access the source code via the following GitHub repository. The source code should be cloned into `/home/src` directory:



<https://github.com/FarzinZaker/DP-ProxyServer>
`git@github.com:FarzinZaker/DP-ProxyServer.git`

The slave proxy server is developed on the Grails framework version 3.2.6. Download and unzip the appropriate version of Grails framework into `/home/grails` directory. Make sure that the `ubuntu` user has full access to the specified path.

Before deploying the slave proxy server, make sure that the `hostname` variable in `application.conf` is set to the IP address of the slave proxy server:

```
hostname = "10.1.0.32"
```

To build a war file and deploy it on Tomcat, you need to create a build script named `build.sh` in `/home/Ubuntu` directory:

```
sudo systemctl stop tomcat

git -C /home/src/DP-ProxyServer pull

rm -rf /home/src/DP-ProxyServer/Web/build/libs/*.war

(cd /home/src/DP-ProxyServer/Web && /home/grails/grails-3.2.6/bin/grails clean)

(cd /home/src/DP-ProxyServer/Web && /home/grails/grails-3.2.6/bin/grails war)

sudo rm -rf /opt/tomcat/webapps/ROOT
sudo rm -rf /opt/tomcat/webapps/ROOT.war

sudo cp /home/src/DP-ProxyServer/Web/build/libs/*.war /opt/tomcat/webapps/ROOT.war

sudo systemctl start tomcat
```

The build script is responsible for performing the following actions:

1. Stopping the Tomcat if it is running
2. Pulling latest changes from the repository
3. Removing older WAR files from the output directory
4. Cleaning the project
5. Building a new WAR file
6. Removing old WAR files from `webapps` folder of the Tomcat
7. Copying the WAR file to the `webapps` folder
8. Starting the Tomcat

Do not forget to make the script file executable using the following command:


```
Chmod u+x ~/build.sh
```

Now, after each modification in the source code, the deployment process is as easy as executing the following command:

```
~/build.sh
```

After starting the slave proxy server, check the Tomcat log file using the following command:

```
sudo tailf /opt/tomcat/logs/catalina.out
```

Make sure that the following outputs have been generated after startup:

```
[akka.remote.Remoting] Starting remoting
[akka.remote.Remoting] Remoting started; listening on addresses
: [akka.tcp://ForwardServer@10.1.0.32:2552]
[akka.remote.Remoting] Remoting now listens on addresses:
[akka.tcp://ForwardServer@10.1.0.32:2552]
```

Master

Create a new instance of the Master proxy server using `DP-ProxyServer-Slave` AMI image (ID: `ami-00c89f9305366b808`) yourself. Please use the following configuration to setup a new master proxy server:

```
Instance Type: t2.medium (vCPUs: 2, Memory: 4GB)
Network: DPNat
Primary IP: WebServer1: 0.1.0.41
Storage: GP2 (8GB)
Security Group: DPSecGroup
```

Although the master proxy server is deployed on an Apache Tomcat instance after each reboot should be started manually. If you want to set up proxy servers from scratch, you need to install Open JDK 8 and Apache Tomcat 8.0 similar to the slave proxy server. The source code is the same for both master and slave proxy servers and should be cloned into `/home/src` directory.

The master proxy server is developed on the Grails framework version 3.2.6. Download and unzip the appropriate version of Grails framework into `/home/grails` directory. Make sure that the `ubuntu` user has full access to the specified path.

Before deploying the master proxy server, make sure that the `hostname` variable in `application.conf` is set to the IP address of the master proxy server:

```
hostname = "10.1.0.31"
```

To build a war file and deploy it on Tomcat, you need to create a build script named `build.sh`, which is the same as the `build.sh` file for the slave proxy server.

Before starting the master proxy server, make sure that the IP address of the feeder node has been added to the list of known hosts in the master proxy server:

```
sudo ssh -i /home/DPKeyPair.pem ubuntu@10.1.0.21
```

Any modification in the source code can be published on the server using the following command:

```
~/build.sh
```

After starting the master proxy server, Make sure that the following outputs have been generated after startup:

```
[akka.remote.Remoting] Starting remoting  
[akka.remote.Remoting] Remoting started; listening on addresses  
:[akka.tcp://ForwardServer@10.1.0.31:2552]  
[akka.remote.Remoting] Remoting now listens on addresses:  
[akka.tcp://ForwardServer@10.1.0.31:2552]
```

Proxy Server Configuration

After launching the proxy servers, it is time to configure the master proxy server.



After any configuration change, a restart is mandatory for the master proxy server.


General Configuration


In order to modify general configurations of the proxy server use the following URL:



[http:\[Master Proxy Server IP\]/configuration/show/1](http://[Master Proxy Server IP]/configuration/show/1)


The following figure shows the general configuration settings available via the mentioned URL:

 Proxy Server

 Configuration


Show Configuration


Name	Default
Feeder Address	10.1.0.21
Proxy Server Master Address	10.1.0.31
Proxy Server Slave Address	10.1.0.32
Load Balancer Address	10.1.0.41
Web Server1 Address	10.1.0.51
Web Server2 Address	10.1.0.52
Web Server3 Address	10.1.0.53
Database Address	10.1.0.61
Simulation Steps	100
Response Time SLA s	<ul style="list-style-type: none"> browse [1200] - p0 product [800] - p1 static [1000] - p3 basket [500] - p2
Adaptation Options	<ul style="list-style-type: none"> 0.5 1.0 0.25 0.75

 Edit

CERAS Lab @ York University

Use the edit button at the bottom of the configuration page to modify any of the available fields. As shown in the following figure, the general configuration page is composed of *Name* field, eight text fields for storing IP addresses of different servers, and a numeric field for specifying the number of *Simulation Steps*. In our paper, we have used 1, 25, 50, 75 and 100 for *Simulation Steps* in separate experiments to show the effectiveness of our proposed approach.

 Proxy Server

 Configuration

Edit Configuration

Name

Default

Feeder Address

10.1.0.21

Proxy Server Master Address

10.1.0.31

Proxy Server Slave Address

10.1.0.32

Load Balancer Address

10.1.0.41

Web Server1 Address

10.1.0.51

Web Server2 Address

10.1.0.52

Web Server3 Address

10.1.0.53

Database Address

10.1.0.61

Simulation Steps *

100

Response Time SLA s


- [product \[800\] - p1](#)
- [static \[1000\] - p3](#)
- [basket \[500\] - p2](#)
- [browse \[1200\] - p0](#)

Adaptation Options

- [1.0](#)
- [0.75](#)
- [0.25](#)
- [0.5](#)

[Add ScenarioConfig](#)

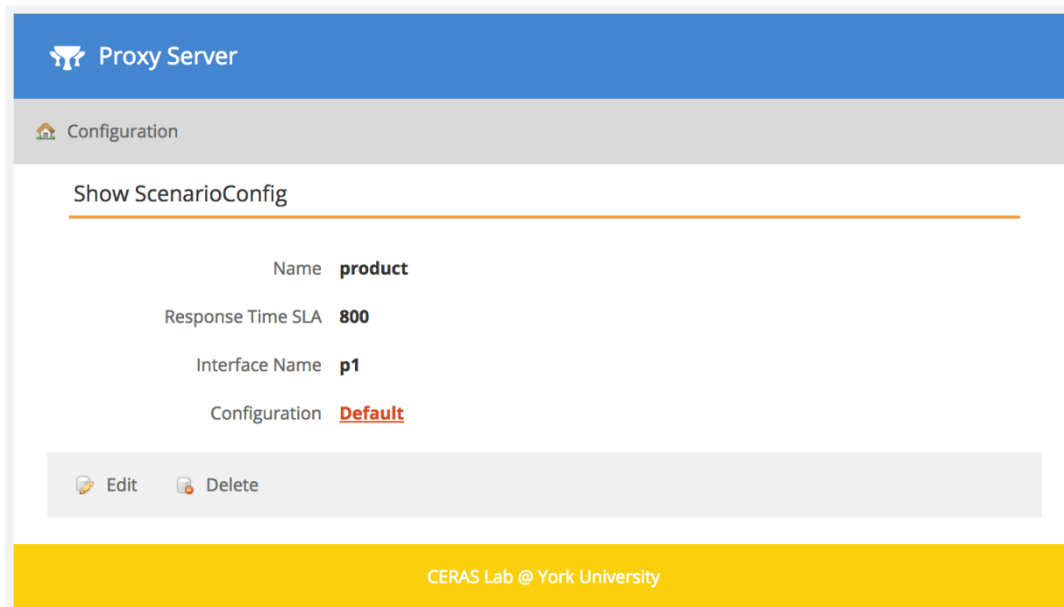
[Add AdaptationOption](#)

 Update

CERAS Lab @ York University

Scenario Definitions

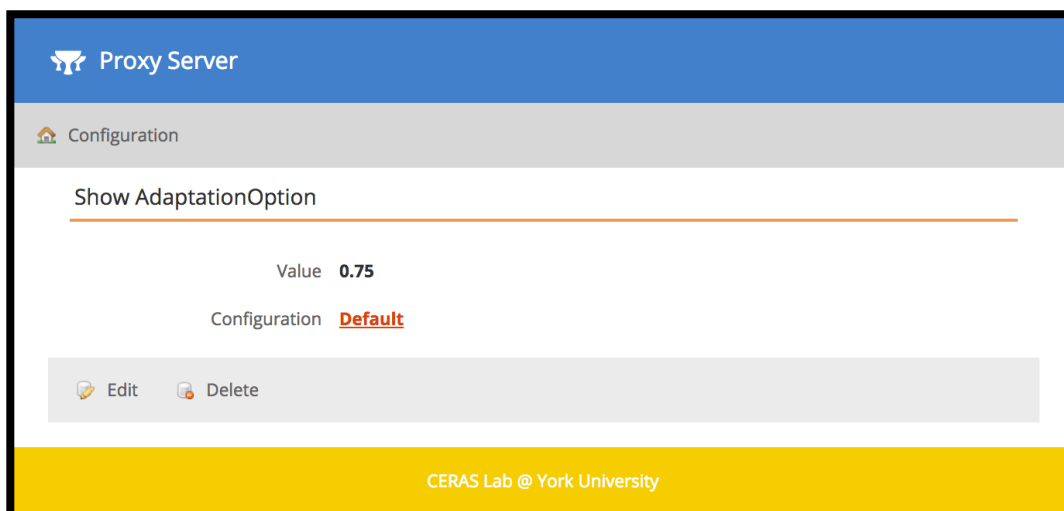
In the *Response Time SLA s* field different scenarios, their assigned response time SLAs and their network ports are listed. You can click each of the items in this list to view an existing scenario definition or use the *Add* button to create a new one.



To modify an existing scenario definition, use the *Edit* button. The *Delete* button removes the current scenario from configuration settings of the proxy server.

Adaptation Options

In the *Adaptation Options* field, different available adaptation options are listed. You can click each of the items in this list to view an existing adaptation option or use the *Add* button to add a new one.



To modify an existing adaptation option, use the *Edit* button. The *Delete* button removes the current option from configuration settings of the proxy server.

Access to Outputs

By running the master proxy server, it sends automatically feeding requests to the feeder node and generates outputs for each step. During execution of the master proxy server, you can run the following command:

```
sudo tailf /opt/tomcat/logs/catalina.out
```

In the log file, you can see the current and total number of feeding steps. When the master proxy server finishes feeding and generating reports, a new zip file including generated reports in CSV format will be created in the `/var/log/bwc/` directory. After Unzipping the report file, you can find report files, which will be introduced in this section.

Adaptations Count

The count of adaptations will be logged in the `adaptationsCount.csv` file. As shown below, the first column shows the feeding step and other columns show the total number of adaptations made up to that feeding step for each scenario.

Feeding Step Number	Basket Scenario Adaptations Count	Product Scenario Adaptations Count	Browse Scenario Adaptations Count	Static Scenario Adaptations Count
------------------------	--------------------------------------	---------------------------------------	--------------------------------------	--------------------------------------

Arrival Rates

Arrival Rates will be logged in the `arrivalRates.csv` file. As shown below, the first column shows the feeding step and other columns show the average arrival rate in that feeding step for each scenario.

Feeding Step Number	Basket Scenario Arrival Rate	Product Scenario Arrival Rate	Browse Scenario Arrival Rate	Static Scenario Arrival Rate
------------------------	---------------------------------	----------------------------------	---------------------------------	---------------------------------

Average Adaptation Duration

Average adaptation durations will be logged in the `averageAdaptationDuration.csv` file. As shown below, the first column shows the feeding step and the second column shows the average adaptation duration up to that feeding step in milliseconds.

Feeding Step Number	Average Adaptation Duration
---------------------	-----------------------------

Bandwidth

The actual bandwidths will be logged in the `bandwidth.csv` file. As shown below, the first column shows the feeding step and other columns show the actual bandwidth assigned to incoming requests of each scenario.

Feeding Step Number	Basket Scenario Bandwidth	Product Scenario Bandwidth	Browse Scenario Bandwidth	Static Scenario Bandwidth
------------------------	------------------------------	-------------------------------	------------------------------	------------------------------

Bandwidth Weights

The bandwidth weights will be logged in the `bandwidthWeights.csv` file. Bandwidth weights are being selected between available adaptation options and could be any of 0.25, 0.5, 0.75 and 1.0. As shown below, the first column shows the feeding step and other columns show the bandwidth weight assigned to incoming requests of each scenario.

Feeding Step Number	Basket Scenario Bandwidth Weight	Product Scenario Bandwidth Weight	Browse Scenario Bandwidth Weight	Static Scenario Bandwidth Weight
---------------------	----------------------------------	-----------------------------------	----------------------------------	----------------------------------

Response Times

Average response times will be logged in the `responseTimes.csv` file. As shown below, the first column shows the feeding step and other columns show the average response time in that feeding step for each scenario.

Feeding Step Number	Basket Scenario Response Time	Product Scenario Response Time	Browse Scenario Response Time	Static Scenario Response Time
---------------------	-------------------------------	--------------------------------	-------------------------------	-------------------------------

Servers Count

The number of involved web servers will be logged in the `serversCount.csv` file. As shown below, the first column shows the feeding step and the second column shows the number of web servers being used at that feeding step to serve incoming requests.

Feeding Step Number	Web Servers Count
---------------------	-------------------

Utilization

Processor utilization will be logged in the `utilization.csv` file. As shown below, the first column shows the feeding step and other columns show the average processor utilization of each node in that feeding step.

Feeding Step Number	Database Server Utilization	Web Server 1 Utilization	Web Server 2 Utilization	Web Server 3 Utilization	Load Balancer Utilization	Master Proxy Server Utilization	Slave Proxy Server Utilization
---------------------	-----------------------------	--------------------------	--------------------------	--------------------------	---------------------------	---------------------------------	--------------------------------

Violations Count

The count of SLA violations will be logged in the `violationsCount.csv` file. As shown below, the first column shows the feeding step and other columns show the total number of SLA violations happened up to that feeding step for each scenario.

Feeding Step Number	Basket Scenario Violations Count	Product Scenario Violations Count	Browse Scenario Violations Count	Static Scenario Violations Count
------------------------	-------------------------------------	--------------------------------------	-------------------------------------	-------------------------------------