COCOMA Documentation

Release 1

Carmelo Ragusa, Philip Robinson, Sergej Svorobej

June 06, 2013

Contents

1 Controlled Contentious and Malicious framework

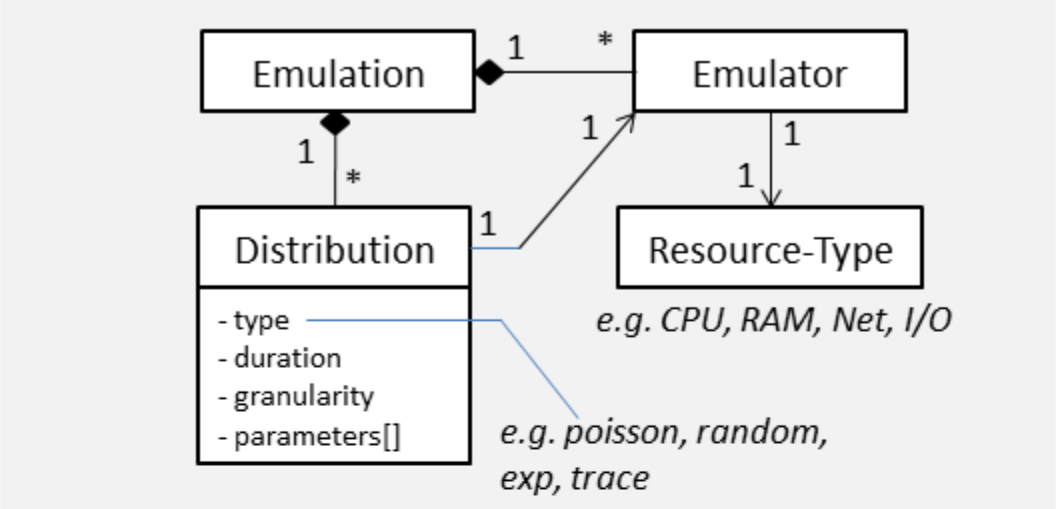
The aim of COCOMA framework is to create, monitor and control contentious and malicious system workload. By using this framework experimenters are able to create operational conditions under which tests and experiments can be carried out. This allows more insight into the testing process so that various scenarios of the cloud infrastructure behaviour can be analysed by collecting and correlate metrics of the emulated environment with the test results.

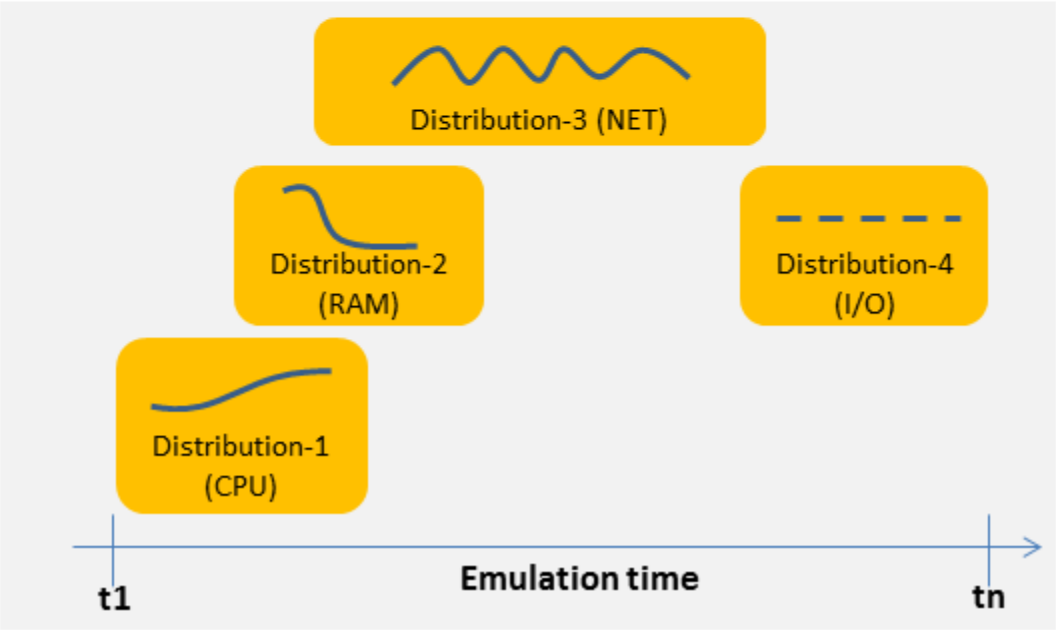
COCOMA is provided by BonFIRE as a service within a VM already configured that can be added to an experiment and used.

2 Contents

2.1 Introduction

In order to use COCOMA, an experimenter defines an *emulation* which embeds all environment operational conditions as shown in the figure below. The actual operational conditions are defined in what are called *distributions*, which create specific workloads over the targeted resource of a specific resource type. For example, *distribution 1* targets the CPU creating an exponential trend over a specific time range within the whole emulation. Each distribution time is divided into multiple time-slots based on the distribution granularity then broken down into multiple runs each one injecting a different load level per time slot, which depends on the discrete function of the distribution.





2.2 Getting Started

COCOMA is installed in *"/usr/share/pyshared/cocoma"*. In this section we provide information about the components that have to be running in order to fully use the framework, and how can a user interact with it.

Starting Components

The two main components of COCOMA are:

- Scheduler daemon (mandatory, needs to be the first started)
- API Daemon (optional, if the REST API functionality is required)

The **Scheduler daemon** - runs in the background and executes workload with differential parameters at the time defined in the emulation properties. To check if the scheduler is running use the following:

```
$ ccmsh --show scheduler
```

To start the scheduler on a specific network interface and port, use the command:

```
$ ccmsh --start scheduler eth1 55555
```

If the network interface and port are omitted, the default values are respectively *eth0* adn *51889*

If more detailed output information is needed, the *Scheduler* can also be started in *DEBUG* mode:

```
$ ccmsh --start scheduler eth1 55555 debug
```

The **API daemon** - provides the RESTfull web API, which exposes COCOMA resources to be used over the network. It follows the same command structure as the Scheduler. To check if the scheduler is running use the following:

```
$ ccmsh --show api
```

To start the api on a specific network interface and port, use the command:

```
$ ccmsh --start api eth2 77777
```

If the network interface and port are omitted, the default values are respectively *eth0* and *5050*

The log level is the same specified for the *Scheduler*.

CLI

The COCOMA CLI is called *ccmsh*, and provides the following options:

-h, -help
Display help information of the available options

-v, -version
Display installed version information of COCOMA

-l, -list `<emulation name>`
Display list of all emulations that are scheduled or already finished. If emulation name is provided then it lists information for that specific emulation

-r, -results `<emulation name>`
Display list of results of all emulations that are scheduled or already finished. If emulation name is provided, then it lists information for that specific emulation

-j, -list-jobs
Queries the scheduler for the list of jobs that have to be executed. For each one, it gives the job name and the planned execution time

-i, -dist `<distribution name>`
Scans the *"/usr/share/pyshared/cocoma/distributions"* folder and displays all available distribution modules. If a distribution name is provided, then it shows the help information for that specific distribution

-e, -emu `<emulator name>`
Scans the *"/usr/share/pyshared/cocoma/emulators"* folder and displays all available emulator wrapper modules. If an emulator name is provided, then it shows the help information for that specific emulator wrapper

-x, -xml `<file name>`
It create and emulation based on the local XML

-n, -now `(used with -x option only)`
Override any start date in the local XML emulation file without modifying the file, i.e. `ccmsh -x <file name> -n`

-d, -delete `<emulation name>`
Deletes a specific emulation from the database

-p, -purge
Remove all DB entries, all scheduled jobs

-start `<api interface port>, <scheduler interface port>`
Start Scheduler or API daemon by specifying network interface and port number i.e. `ccmsh --start api eth0 2020` or `ccmsh --start scheduler eth0 3030`. By default if the network interface is not specified, the Scheduler daemon will run on *eth0* and *51889*, and the API daemon will run on *eth0* and *5050*.

-stop `<api>, <scheduler>`
Stop Scheduler or API daemon

-show `<api>, <scheduler>`
Show OS information on Scheduler or API daemon, displays PID numbers

REST API

Index

The API URIs summary list is as follow:

```
* /
* /emulations
* /emulations/{name}
* /distributions
* /distributions/{name}
* /emulators
* /emulators/{name}
* /results
* /results/{name}
* /tests
* /tests/{name}
* /logs
* /logs/system
* /logs/emulations
* /logs/emulations/{name}
```

Description

`http:method::  GET /`

The **root** returns a *collection* of all the available resources. Example of a XML response:

```
<?xml version="1.0" ?>
<root href="/">
        <version>0.1.1</version>
        <timestamp>1365518303.44</timestamp>
        <link href="/emulations" rel="emulations" type="application/vnd.bonfire+xml"/>
        <link href="/emulators" rel="emulators" type="application/vnd.bonfire+xml"/>
        <link href="/distributions" rel="distributions" type="application/vnd.bonfire+xml"/>
        <link href="/tests" rel="tests" type="application/vnd.bonfire+xml"/>
        <link href="/results" rel="results" type="application/vnd.bonfire+xml"/>
        <link href="/logs" rel="logs" type="application/vnd.bonfire+xml"/>
</root>
```

`http:method::  GET /emulations`

The **emulations** returns a *collection* of all the available emulation resources. Example of a XML response:

```
<?xml version="1.0" ?>
<collection href="/emulations" xmlns="http://127.0.0.1/cocoma">
        <items offset="0" total="3">
                <emulation href="/emulations/1-Emu-CPU-RAM-IO" id="1" name="1-Emu-CPU-RAM-IO" st
                <emulation href="/emulations/2-CPU_EMU" id="2" name="2-CPU_EMU" state="inactive"
                <emulation href="/emulations/3-CPU_EMU" id="3" name="3-CPU_EMU" state="inactive"
        </items>
        <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

`http:method::  GET /emulations/{name}`

Displays information about emulation by name. The returned *200-OK* XML is:

```
<?xml version="1.0" ?>
<emulation href="/emulations/1-Emu-CPU-RAM-IO" xmlns="http://127.0.0.1/cocoma">
        <id>1</id>
        <emulationName>1-Emu-CPU-RAM-IO</emulationName>
        <emulationType>mix</emulationType>
        <resourceType>mix</resourceType>
        <emuStartTime>2013-04-09T13:00:01</emuStartTime>
        <emuStopTime>180</emuStopTime>
        <scheduledJobs>
                <jobsempty>No jobs are scheduled</jobsempty>
        </scheduledJobs>
        <distributions ID="1" name="Distro1">
                <startTime>5</startTime>
                <granularity>3</granularity>
                <duration>30</duration>
                <startload>10</startload>
                <stopload>90</stopload>
        </distributions>
        <distributions ID="2" name="Distro2">
                <startTime>5</startTime>
                <granularity>3</granularity>
                <duration>30</duration>
                <startload>10</startload>
                <stopload>90</stopload>
```

```
        </distributions>
        <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
        <link href="/emulations" rel="parent" type="application/vnd.bonfire+xml"/>
</emulation>
```

The returned *404 – Not Found* XML is:

```
<error>Emulation Name: 1-Emu-CPU-RAM-IO1 not found. Error:too many values to unpack</error>
```

```
http:method::  POST /emulations
```

```
:param string XML: Emulation parameters defined via XML as shown in the examples section.
```

The returned *201-Created* XML:

```
<?xml version="1.0" ?>
<emulation href="/emulations/4-CPU_EMU" xmlns="http://127.0.0.1/cocoma">
        <ID>4-CPU_EMU</ID>
        <EmuNotes>OK</EmuNotes>
        <DistroNotes>OK</DistroNotes>
        <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
        <link href="/emulations" rel="parent" type="application/vnd.bonfire+xml"/>
</emulation>
```

The returned *400 – Bad Request* XML:

```
<?xml version="1.0" ?>
<error>XML is not well formed Error: syntax error: line 1, column 0</error>
```

```
http:method::  GET /emulators
```

Displays emulators list. The returned *200- OK* XML:

```
<?xml version="1.0" ?>
<collection href="/emulators" xmlns="http://127.0.0.1/cocoma">
        <items offset="0" total="3">
                <emulator href="/emulators/lookbusy" name="lookbusy"/>
                <emulator href="/emulators/stressapptest" name="stressapptest"/>
                <emulator href="/emulators/iperf" name="iperf"/>
        </items>
        <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

```
http:method::  GET /emulators/{name}
```

```
:arg name: Name of emulator that you want to get more info
```

Displays information about emulator by name. The returned *200- OK* XML:

<?xml version=”1.0” ?> <emulator href=”/emulators/lookbusy” xmlns=”http://127.0.0.1/cocoma“>

<info> <help> Emulator lookbusy can be used for following resources: 1)Loads CPU with parameters:

ncpus - Number of CPUs to keep busy (default: autodetected)

2)Loads Memory(MEM) with parameters: memSleep - Time to sleep between iterations, in usec (default 1000)

3)Changing size of files to use during IO with parameters: ioBlockSize - Size of blocks to use for I/O in MB ioSleep - Time to sleep between iterations, in msec (default 100)

XML block example: <emulator-params>

<resourceType>CPU</resourceType>
<ncpus>0</ncpus>

</emulator-params>

</help> <resources>

<cpu>

<ncpus> <upperBound>100</upperBound> <lower-Bound>100</lowerBound>

</ncpus>

</cpu> <io>

<iosleep> <upperBound>999999999</upperBound> <lower-Bound>999999999</lowerBound>

</iosleep> <ioblocksize>

<upperBound>9999999</upperBound> <lower-Bound>9999999</lowerBound>

</ioblocksize>

</io> <mem>

<memsleep> <upperBound>999999999</upperBound> <lower-Bound>999999999</lowerBound>

</memsleep>

</mem>

</resources>

</info> <link href=”/” rel=”parent” type=”application/vnd.bonfire+xml”/> <link href=”/emulators” rel=”parent” type=”application/vnd.bonfire+xml”/>

</emulator>

`http:method:: GET /distributions`

Displays distributions list. The returned *200- OK* XML:

```
<?xml version="1.0" ?>
<collection href="/distributions" xmlns="http://127.0.0.1/cocoma">
        <items offset="0" total="3">
                <distribution href="/distributions/linear" name="linear"/>
                <distribution href="/distributions/linear_incr" name="linear_incr"/>
                <distribution href="/distributions/trapezoidal" name="trapezoidal"/>
        </items>
        <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

`http:method:: GET /distributions/{name}`

`:arg name: Name of distributions that you want to get more info`

Displays information about distributions by name. The returned *200- OK* XML:

```
<?xml version="1.0" ?>
<distribution href="/distributions/linear" xmlns="http://127.0.0.1/cocoma">
        <info>Linear distribution takes in start and stop load parameters and gradually increasi
        <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
        <link href="/distributions" rel="parent" type="application/vnd.bonfire+xml"/>
</distribution>
```

http:method:: GET /tests

Displays tests list. The returned *200- OK* XML:

```
<?xml version="1.0" ?>
<collection href="/tests" xmlns="http://127.0.0.1/cocoma">
        <items offset="0" total="20">
                <test href="/tests/01-CPU-Linear-Lookbusy_10-95.xml" name="01-CPU-Linear-Lookbus
                <test href="/tests/03-NET-Linear_incr-Iperf-100-1000.xml" name="03-NET-Linear_in
                <test href="/tests/02-IO-Linear-Stressapptest_1-10.xml" name="02-IO-Linear-Stres
                <test href="/tests/02-IO-Linear_incr-Stressapptest_1-10.xml" name="02-IO-Linear_
                <test href="/tests/02-MEM-Linear_incr-Stressapptest_100-1000.xml" name="02-MEM-L
                <test href="/tests/01-CPU-Trapezoidal-Lookbusy_10-95.xml" name="01-CPU-Trapezoid
                <test href="/tests/01-IO-Trapezoidal-Lookbusy_1-10.xml" name="01-IO-Trapezoidal-
                <test href="/tests/01-NET_TEST.xml" name="01-NET_TEST.xml"/>
                <test href="/tests/03-MEM-500-1000MB-overlap.xml" name="03-MEM-500-1000MB-overla
                <test href="/tests/01-CPU-Linear_incr-Lookbusy_10-95.xml" name="01-CPU-Linear_in
                <test href="/tests/01-IO-Linear_incr-Lookbusy_1-10.xml" name="01-IO-Linear_incr-
                <test href="/tests/02-IO-Trapezoidal-Stressapptest_1-10.xml" name="02-IO-Trapezo
                <test href="/tests/03-CPU-opposite.xml" name="03-CPU-opposite.xml"/>
                <test href="/tests/01-MEM-Linear_incr-Lookbusy_100-1000.xml" name="01-MEM-Linear
                <test href="/tests/03-MEM-500-1000MB.xml" name="03-MEM-500-1000MB.xml"/>
                <test href="/tests/03-MEM-Linear-Stressapptest_500-1000MB.xml" name="03-MEM-Line
                <test href="/tests/01-MEM-Trapezoidal-Lookbusy_100-1000.xml" name="01-MEM-Trapez
                <test href="/tests/02-MEM-Trapezoidal-Stressapptest_100-1000.xml" name="02-MEM-T
                <test href="/tests/03-NET-Trapezoidal-Iperf-100-1000.xml" name="03-NET-Trapezoid
                <test href="/tests/01-IO-Linear-Lookbusy_1-10.xml" name="01-IO-Linear-Lookbusy_1
        </items>
        <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

http:method:: GET /tests/{name}

:arg name: Name of tests that you want to get more info

Displays Content of XML file.

http:method:: POST /tests

:param string: name of the test that is located on COCOMA machine

Create emulation from available tests. The returned *201- Created* XML:

```
<?xml version="1.0" ?>
<test href="/tests/5-CPU_EMU" xmlns="http://127.0.0.1/cocoma">
        <emulationName>5-CPU_EMU</emulationName>
        <startTime>2013-04-09T18:57:32</startTime>
        <durationSec>60</durationSec>
</test>
```

The returned *400- Not Found* reply XML:

```
<?xml version="1.0" ?>
<error>error message</error>
```

http:method:: GET /results

Displays results list. The returned *200- OK* XML:

```
<?xml version="1.0" ?>
<collection href="/results" xmlns="http://127.0.0.1/cocoma">
        <items offset="0" total="5">
                <results failedRuns="0" href="/results/1-Emu-CPU-RAM-IO" name="1-Emu-CPU-RAM-IO"
                <results failedRuns="0" href="/results/2-CPU_EMU" name="2-CPU_EMU" state="inacti
                <results failedRuns="0" href="/results/3-CPU_EMU" name="3-CPU_EMU" state="inacti
                <results failedRuns="0" href="/results/4-CPU_EMU" name="4-CPU_EMU" state="inacti
                <results failedRuns="0" href="/results/5-CPU_EMU" name="5-CPU_EMU" state="inacti
        </items>
        <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

http:method:: GET /results/{name}

:arg name: Name of tests that you want to get more info

Displays information about results by name. The returned *200- OK* XML:

```
<?xml version="1.0" ?>
<results href="/results/1-Emu-CPU-RAM-IO" xmlns="http://127.0.0.1/cocoma">
        <emulationName>1-Emu-CPU-RAM-IO</emulationName>
        <totalRuns>6</totalRuns>
        <executedRuns>6</executedRuns>
        <failedRuns>0</failedRuns>
        <emuState>inactive</emuState>
</results>
```

http:method:: GET /logs

Displays logs list. The returned *200- OK* XML:

```
<?xml version="1.0" ?>
<logs href="/logs">
        <link href="/logs/emulations" rel="emulations" type="application/vnd.bonfire+xml"/>
        <link href="/logs/system" rel="system" type="application/vnd.bonfire+xml"/>
</logs>
```

http:method:: GET /logs/system

Return Zip file with system logs.

http:method:: GET /logs/emulations

Displays emulations logs list. The returned *200- OK* XML:

```
<?xml version="1.0" ?>
<collection href="/logs/emulations" xmlns="http://127.0.0.1/cocoma">
        <items offset="0" total="3">
                <emulationLog href="/logs/emulations/3-CPU_EMU" name="3-CPU_EMU"/>
                <emulationLog href="/logs/emulations/5-CPU_EMU" name="5-CPU_EMU"/>
                <emulationLog href="/logs/emulations/4-CPU_EMU" name="4-CPU_EMU"/>
        </items>
        <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
        <link href="/logs" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

```
http:method::  GET /logs/{name}

    :arg name: Name of emulation logs that you want to get
```

Return Zip file with emulation logs.

XML payload structure

A COCOMA emulation is specified in XML. The user can directly create the XML and send it to COCOMA through the CLI client or any REST client. An Emulation must contain all the neccessary information about starting time, duration, target resource and required resource usage. Once the XML document is received by COCOMA, the framework automatically schedules and executes the required workload on the chosen resource(s), CPU, IO, Memory or Network.

Consider this sample XML document code:

```xml
<emulation>
  <emuname>CPU_EMU</emuname>
  <emuType>Mix</emuType>
  <emuresourceType>CPU</emuresourceType>
  <!--date format: 2014-10-10T10:10:10 -->
  <emustartTime>now</emustartTime>
  <!--duration in seconds -->
  <emustopTime>60</emustopTime>

  <distributions>

   <name>CPU_Distro</name>
     <startTime>0</startTime>
     <!--duration in seconds -->
     <duration>60</duration>
     <granularity>20</granularity>
     <distribution href="/distributions/linear" name="linear" />
     <!--cpu utilization distribution range-->
      <startLoad>10</startLoad>
      <stopLoad>95</stopLoad>

      <emulator href="/emulators/lookbusy" name="lookbusy" />
      <emulator-params>
        <!--more parameters will be added -->
        <resourceType>CPU</resourceType>
       <!--Number of CPUs to keep busy (default: autodetected)-->
       <ncpus>0</ncpus>
      </emulator-params>

  </distributions>

  <log>
   <!-- Use value "1" to enable logging(by default logging is off)  -->
   <enable>1</enable>
   <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec)  -->
   <frequency>1</frequency>
   <logLevel>debug</logLevel>
  </log>

</emulation>
```

The XML document defines the emulation experiment details, which consists of three blocks:

- **Emulation**

```
<emulation>
  <emuname>CPU_EMU</emuname>
  <emuType>Mix</emuType>
  <emuresourceType>CPU</emuresourceType>
  <!--date format: 2014-10-10T10:10:10 -->
  <emustartTime>now</emustartTime>
  <!--duration in seconds -->
  <emustopTime>60</emustopTime>
  ...
</emulation>
```

- **Distribution**

```
  <distributions>

   <name>CPU_Distro</name>
     <startTime>0</startTime>
     <!--duration in seconds -->
     <duration>60</duration>
     <granularity>20</granularity>
     <distribution href="/distributions/linear" name="linear" />
     <!--cpu utilization distribution range-->
      <startLoad>10</startLoad>
      <stopLoad>95</stopLoad>

      <emulator href="/emulators/lookbusy" name="lookbusy" />
      <emulator-params>
        <!--more parameters will be added -->
        <resourceType>CPU</resourceType>
       <!--Number of CPUs to keep busy (default: autodetected)-->
       <ncpus>0</ncpus>
      </emulator-params>

  </distributions>
```

- **Log** (optional)

```
  <log>
   <!-- Use value "1" to enable logging(by default logging is off)  -->
   <enable>1</enable>
   <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec)  -->
   <frequency>1</frequency>
   <logLevel>debug</logLevel>
  </log>
```

In plain english it means - create an emulation named *CPU_EMU* starting *now* and running for *60* sec. The Emulation includes one distribution called *CPU_Distro*, which starts at the same time as emulation, runs for *60* sec, using *linear* pattern. The pattern increases the workload of the *CPU* from *10%* to *95%* in *20* steps by using the *lookbusy* emulator. The workload produced by the application is logged every second with debug level information.

2.3 Creating Emulation via CLI

To create an emulation via CLI, a local XML `emulation.xml` file as the following can be used:

```
   <emulation>
     <emuname>CPU_Emulation</emuname>
```

```
    <emuType>Mix</emuType>
    <emuresourceType>Mix</emuresourceType>
    <emustartTime>now</emustartTime>
    <!--duration in seconds -->
    <emustopTime>180</emustopTime>

    <distributions>
       <name>Distro1</name>
       <startTime>5</startTime>
       <!--duration in seconds -->
       <duration>30</duration>
       <granularity>3</granularity>
       <distribution href="/distributions/linear" name="linear" />
     <!--cpu utilization distribution range-->
        <startLoad>90</startLoad>
        <stopLoad>10</stopLoad>
        <emulator href="/emulators/lookbusy" name="lookbusy" />
        <emulator-params>
          <!--more parameters will be added -->
          <resourceType>CPU</resourceType>
     <!--Number of CPUs to keep busy (default: autodetected)-->
     <ncpus>0</ncpus>

        </emulator-params>
    </distributions>

     <distributions>
       <name>Distro2</name>
       <startTime>5</startTime>
       <!--duration in seconds -->
       <duration>30</duration>
       <granularity>3</granularity>
       <distribution href="/distributions/linear" name="linear" />
     <!--cpu utilization distribution range-->
        <startLoad>10</startLoad>
        <stopLoad>90</stopLoad>
        <emulator href="/emulators/lookbusy" name="lookbusy" />
        <emulator-params>
          <!--more parameters will be added -->
          <resourceType>CPU</resourceType>
     <!--Number of CPUs to keep busy (default: autodetected)-->
     <ncpus>0</ncpus>

        </emulator-params>
    </distributions>

    <log>
     <!-- Use value "1" to enable logging(by default logging is off)  -->
     <enable>1</enable>
     <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec)  -->
     <frequency>3</frequency>
    </log>

  </emulation>
```

The comman to start the emulation via CLI is:

```
$ ccmsh -x emulation.xml
```

Once sent, the list of scheduled jobs is shown on screen as follow:

```
1  $ ccmsh -x emulation.xml
2  INFO:XML Parser:Finished running
3  INFO:Distriburion Manager:Scheduler reply: 6-CPU_Emulation-7-0-Distro1-lookbusy-cpu: 90 Duration: 10.
4  INFO:Distriburion Manager:Scheduler reply: 6-CPU_Emulation-7-1-Distro1-lookbusy-cpu: 50 Duration: 10.
5  INFO:Distriburion Manager:Scheduler reply: 6-CPU_Emulation-7-2-Distro1-lookbusy-cpu: 10 Duration: 10.
6  INFO:Distriburion Manager:Scheduler reply: 6-CPU_Emulation-8-0-Distro2-lookbusy-cpu: 10 Duration: 10.
7  INFO:Distriburion Manager:Scheduler reply: 6-CPU_Emulation-8-1-Distro2-lookbusy-cpu: 50 Duration: 10.
8  INFO:Distriburion Manager:Scheduler reply: 6-CPU_Emulation-8-2-Distro2-lookbusy-cpu: 90 Duration: 10.
9  INFO:Emulation Manager:##Emulation 6-Emu-CPU-RAM-IO created
10 INFO:Emulation Manager:Started logger:6-CPU_Emulation-logger interval-3sec.StartTime:2013-04-10 09:42
11 6-Emu-CPU-RAM-IO
```

Each line from *3-8* shows information of a single scheduled emulation job. Each line provides job's information, for example line *3*:

- **INFO:Distriburion Manager:Scheduler reply:** -just a generic logger part
- **6-CPU_Emulation** - emulation name, which is a combined string of emulation ID from the DB and `emuname` value in the XML file
- **7** - database ID number for distribution
- **0** - run number of this distribution
- **Distro1** - name of the distribution taken from XML file
- **lookbusy** - distribution module used to calculate each run parameters
- **cpu** - the target resource used by this run
- **90** - stress value applied to this run
- **Duration 10.0sec.** - how long the job run
- **Start Time: 2013-04-10 09:43:01 End Time: 09:43:11** - time interval when the run is/was executed

More generally, the run/job notation is as follow:

```
(logger reply) - (emulationID-name) - (distribution ID) - (run number} -
(distribution name) - (distribution module) - (resource) - (stress value) -
(run duration) - (execution time)
```

Line *10* shows another job which was created for the logger. This job appears only if the optional *log* section is stated in the XML. The logger job executes for the duration of the whole emulation and collects system resource usage information. The logger job name notation can be described in this way:

```
(logger reply) - (emulationID-name) - (logger mark) - {poll interval} - (start
time)
```

2.4 Creating Emulation via API Client (Restfully)

This sections provides examples on how to use the REST API via the restfully client.

First you need to create a configuration file for restfully `api.cocoma.yml`, containing the public IP address of COCOMA:

```
uri: http://131.254.204.223/
require: [ApplicationVndBonfireXml]
```

The example below creates an emulation with two distributions over the MEM resource. The file can be saved as a .rb and used by restfully. It contains the XML payload for COCOMA and a reference to the config file to connect to the COCOMA VM:

```
require 'rubygems'
require 'restfully'
require 'logger'

session = Restfully::Session.new(
 :configuration_file => "~/api.cocoma.yml"

)

session.logger.level = Logger::INFO

emulation = nil

begin
 emulation = session.root.emulations.submit(
   :emuname => "MEM-emulation",
   :emutype => "Contention",
   :emuresourceType => "RAM",
   :emustartTime => "now",
   :emustopTime => "240",
   :distributions =>[{
         :name => "MEM-increase",
            :startTime =>"0",
            :duration =>"120",
            :granularity =>"10",
            :distribution => {
                              :href => "/distributions/linear_incr",
                              :name => "linear_incr"},
            :startLoad => "10%",
            :stopLoad => "80%",
            :emulator =>{
            :href => "/emulators/stressapptest",
            :name => "stressapptest"},
            :'emulator-params' =>{
                              :resourceType =>"MEM",
                              :memThreads => "1"}
      },
             {
                        :name => "MEM-decrease",
                        :startTime =>"121",
                        :duration =>"119",
                        :granularity =>"10",
                        :distribution => {
                                :href => "/distributions/linear_incr",
                                :name => "linear_incr"},
                        :startLoad => "80%",
                        :stopLoad => "10%",
                        :emulator =>{
                                :href => "/emulators/stressapptest",
                                :name => "stressapptest"},
                        :'emulator-params' =>{
                                :resourceType =>"MEM",
                                :memThreads => "1"}
                        }]
 )
```

```
end
```

The script can be executed as:

```
$ restfully emulation.rb
```

You can access the COCOMA VM interactively thorugh the `restfully` client, and check if the emulation was created successfully:

```
$ restfully -c cocoma.yml

>> pp root.emulations
>> #<Collection:0x45f9f3e uri="/emulations"
>>  RELATIONSHIPS
>>    parent, self
>>  ITEMS (0..2)/2
>>    #<Resource:0x45b5d3e name="7-CPU_Stress" uri="/emulations/7-CPUStress">
>>    #<Resource:0x4489eb0 name="8-MEM-emulation" uri="/emulations/8-MEM-emulation">>
>> => nil
```

To get more client tutorials check the restfully page.

2.5 XML Examples

This section provides XML payload examples for creating different emulations over various resources.

CPU

Emulation XML for CPU contention:

```
<emulation>
  <emuname>CPU_EMU</emuname>
  <emuType>Mix</emuType>
  <emuresourceType>CPU</emuresourceType>
  <!--date format: 2014-10-10T10:10:10 -->
  <emustartTime>now</emustartTime>
  <!--duration in seconds -->
  <emustopTime>120</emustopTime>

  <distributions>
   <name>CPU_Distro</name>
     <startTime>0</startTime>
     <!--duration in seconds -->
     <duration>120</duration>
     <granularity>24</granularity>
     <distribution href="/distributions/linear" name="linear" />
     <!--cpu utilization distribution range-->
      <startLoad>10</startLoad>
      <stopLoad>95</stopLoad>
      <emulator href="/emulators/lookbusy" name="lookbusy" />

      <emulator-params>
        <!--more parameters will be added -->
        <resourceType>CPU</resourceType>
       <!--Number of CPUs to keep busy (default: autodetected)-->
       <ncpus>1</ncpus>
```

```
      </emulator-params>
  </distributions>

  <log>
   <!-- Use value "1" to enable logging(by default logging is off)  -->
   <enable>1</enable>
   <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec)  -->
   <frequency>1</frequency>
   <logLevel>debug</logLevel>
  </log>

</emulation>
```

I/O

Emulation XML for I/O contention:

```
<emulation>
  <emuname>IO_EMU</emuname>
  <emuType>Mix</emuType>
  <emuresourceType>IO</emuresourceType>
  <!--date format: 2014-10-10T10:10:10 -->
  <emustartTime>now</emustartTime>
  <!--duration in seconds -->
  <emustopTime>60</emustopTime>

  <distributions>

   <name>IO_Distro</name>
     <startTime>0</startTime>
     <!--duration in seconds -->
     <duration>60</duration>
     <granularity>5</granularity>
     <distribution href="/distributions/linear_incr" name="linear_incr" />
      <startLoad>1</startLoad>
      <stopLoad>10</stopLoad>
      <emulator href="/emulators/stressapptest" name="stressapptest" />

      <emulator-params>
        <!--more parameters will be added -->
        <resourceType>IO</resourceType>
       <!--Size of mem in MB used-->
       <memsize>1000</memsize>
       <!--Number of threads-->
       <memThreads>10</memThreads>
      </emulator-params>

  </distributions>

  <log>
   <!-- Use value "1" to enable logging(by default logging is off)  -->
   <enable>1</enable>
   <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec)  -->
   <frequency>3</frequency>
   <logLevel>debug</logLevel>
  </log>

```

```
</emulation>
```

In this example we use a different distribution called *trapezoidal*:

```
<emulation>
  <emuname>IO_EMU</emuname>
  <emuType>Mix</emuType>
  <emuresourceType>IO</emuresourceType>
  <!--date format: 2014-10-10T10:10:10 -->
  <emustartTime>now</emustartTime>
  <!--duration in seconds -->
  <emustopTime>60</emustopTime>

  <distributions>

   <name>IO_Distro</name>
     <startTime>0</startTime>
     <!--duration in seconds -->
     <duration>60</duration>
     <granularity>5</granularity>
     <distribution href="/distributions/trapezoidal" name="trapezoidal" />
      <startLoad>1</startLoad>
      <stopLoad>10</stopLoad>
      <emulator href="/emulators/stressapptest" name="stressapptest" />

      <emulator-params>
        <!--more parameters will be added -->
        <resourceType>IO</resourceType>
       <!--Size of mem in MB used-->
       <memsize>1000</memsize>
       <!--Number of threads-->
       <memThreads>10</memThreads>
      </emulator-params>

  </distributions>

  <log>
   <!-- Use value "1" to enable logging(by default logging is off)  -->
   <enable>1</enable>
   <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec)  -->
   <frequency>3</frequency>
   <logLevel>debug</logLevel>
  </log>

</emulation>
```

Memory

Emulation XML for memory contention:

```
<emulation>
  <emuname>MEM_EMU</emuname>
  <emuType>Mix</emuType>
  <emuresourceType>MEM</emuresourceType>
  <!--date format: 2014-10-10T10:10:10 -->
  <emustartTime>now</emustartTime>
  <!--duration in seconds -->
```

```
   <emustopTime>60</emustopTime>

   <distributions >
      <name>MEM_Distro</name>
      <startTime>0</startTime>
      <!--duration in seconds -->
      <duration>60</duration>
      <granularity>5</granularity>
      <distribution href="/distributions/linear_incr" name="linear_incr" />
      <!--Megabytes for memory -->
       <startLoad>100</startLoad>
       <stopLoad>1000</stopLoad>
               <malloclimit>4095</malloclimit>
       <emulator href="/emulators/stressapptest" name="stressapptest" />
       <emulator-params>
                       <resourceType>MEM</resourceType>
                       <memThreads>0</memThreads>
       </emulator-params>
   </distributions>

   <log>
    <!-- Use value "1" to enable logging(by default logging is off)  -->
    <enable>1</enable>
    <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec)  -->
    <frequency>3</frequency>
    <logLevel>debug</logLevel>
   </log>

</emulation>
```

Example for memory emulation using *trapezoidal* distribution:

```
<emulation>
   <emuname>MEM_EMU</emuname>
   <emuType>Mix</emuType>
   <emuresourceType>MEM</emuresourceType>
   <!--date format: 2014-10-10T10:10:10 -->
   <emustartTime>now</emustartTime>
   <!--duration in seconds -->
   <emustopTime>60</emustopTime>

   <distributions >
      <name>MEM_Distro</name>
      <startTime>0</startTime>
      <!--duration in seconds -->
      <duration>60</duration>
      <granularity>5</granularity>
      <distribution href="/distributions/trapezoidal" name="trapezoidal" />
      <!--Megabytes for memory -->
       <startLoad>100</startLoad>
       <stopLoad>1000</stopLoad>
       <malloclimit>4095</malloclimit>
       <emulator href="/emulators/stressapptest" name="stressapptest" />
       <emulator-params>
                       <resourceType>MEM</resourceType>
                       <memThreads>0</memThreads>
       </emulator-params>
   </distributions>
```

```
  <log>
   <!-- Use value "1" to enable logging(by default logging is off)  -->
   <enable>0</enable>
   <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec)  -->
   <frequency>3</frequency>
   <logLevel>debug</logLevel>
  </log>

</emulation>
```

Network

The newtork emulation needs two COCOMA VM's, one that acts as a client and the other as a server. Normally those two VMs are placed in different nodes. The SuT should be composed of at least two VMs placed on the same two nodes of COCOMA. The emulation XML for network contention looks like:

```
<emulation>
  <emuname>NET_emu</emuname>
  <emuType>Mix</emuType>
  <emuresourceType>NET</emuresourceType>
  <!--2014-02-02T10:10:10-->
  <emustartTime>now</emustartTime>
  <!--duration in seconds -->
  <emustopTime>155</emustopTime>

  <distributions>
   <name>NET_distro</name>
     <startTime>0</startTime>
     <!--duration in seconds -->
     <duration>150</duration>
     <granularity>10</granularity>
     <distribution href="/distributions/linear_incr" name="linear_incr" />
   <!--cpu utilization distribution range-->
      <startLoad>100</startLoad>
      <!-- set target bandwidth to bits per sec -->
      <stopLoad>1000</stopLoad>
      <emulator href="/emulators/iperf" name="iperf" />
    <emulator-params>
        <resourceType>NET</resourceType>
        <serverip>172.18.254.234</serverip>
                <!--Leave "0" for default 5001 port -->
                <serverport>5001</serverport>
                <clientip>172.18.254.236</clientip>
                <clientport>5001</clientport>
                <packettype>UDP</packettype>
                </emulator-params>
  </distributions>

  <log>
   <!-- Use value "1" to enable logging(by default logging is off)  -->
   <enable>0</enable>
   <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec)  -->
   <frequency>3</frequency>
  </log>

</emulation>
```

Multiple distributions emulation

A good feature of COCOMA is the ability to combine multiple distributions within the same emulation. This allows to specify contention properties for multiple resources or create different patterns for the same resource. Distributions can overlap, meaning two distributions can run at the same time frame. If distributions for the same resource overlap and they exceed the available resources, the runs might crash.

- CPU and Memory example

```
<emulation>
    <emuname>CPU_and_Mem</emuname>
    <emutype>Mix</emutype>
    <emuresourceType>CPU</emuresourceType>
    <emustartTime>now</emustartTime>
    <!--duration in seconds -->
    <emustopTime>80</emustopTime>

    <distributions>
     <name>CPU_distro</name>
     <startTime>0</startTime>
     <!--duration in seconds -->
     <duration>60</duration>
     <granularity>1</granularity>
     <distribution href="/distributions/linear" name="linear" />
     <!--cpu utilization distribution range-->
     <startLoad>10</startLoad>
     <stopLoad>95</stopLoad>
     <emulator href="/emulators/lookbusy" name="lookbusy" />
     <emulator-params>
          <!--more parameters will be added -->
          <resourceType>CPU</resourceType>
          <!--Number of CPUs to keep busy (default: autodetected)-->
          <ncpus>0</ncpus>
     </emulator-params>
    </distributions>

     <distributions >
        <name>MEM_Distro</name>
        <startTime>20</startTime>
        <!--duration in seconds -->
        <duration>60</duration>
        <granularity>5</granularity>
        <distribution href="/distributions/linear_incr" name="linear_incr" />
        <!--Megabytes for memory -->
         <startLoad>100</startLoad>
         <stopLoad>1000</stopLoad>
         <malloclimit>4095</malloclimit>
                        <emulator href="/emulators/stressapptest" name="stressapptest" />
                        <emulator-params>
                              <resourceType>MEM</resourceType>
                              <memThreads>0</memThreads>
                        </emulator-params>
     </distributions>

    <log>
      <!-- Use value "1" to enable logging(by default logging is off)  -->
      <enable>1</enable>
      <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec)  -->
      <frequency>3</frequency>
```

```
        </log>
    </emulation>
```

- CPU, MEM and IO example

```
    <emulation>
        <emuname>CPU_and_Mem</emuname>
        <emutype>Mix</emutype>
        <emuresourceType>CPU</emuresourceType>
        <emustartTime>now</emustartTime>
        <!--duration in seconds -->
        <emustopTime>80</emustopTime>

        <distributions>
         <name>CPU_distro</name>
         <startTime>0</startTime>
         <!--duration in seconds -->
         <duration>60</duration>
         <granularity>1</granularity>
         <distribution href="/distributions/linear" name="linear" />
         <!--cpu utilization distribution range-->
         <startLoad>10</startLoad>
         <stopLoad>95</stopLoad>
         <emulator href="/emulators/lookbusy" name="lookbusy" />
         <emulator-params>
              <!--more parameters will be added -->
              <resourceType>CPU</resourceType>
              <!--Number of CPUs to keep busy (default: autodetected)-->
              <ncpus>0</ncpus>
         </emulator-params>
       </distributions>

        <distributions >
           <name>MEM_Distro</name>
           <startTime>20</startTime>
           <!--duration in seconds -->
           <duration>60</duration>
           <granularity>5</granularity>
           <distribution href="/distributions/linear_incr" name="linear_incr" />
           <!--Megabytes for memory -->
            <startLoad>100</startLoad>
            <stopLoad>1000</stopLoad>
            <malloclimit>4095</malloclimit>
                          <emulator href="/emulators/stressapptest" name="stressapptest" />
                          <emulator-params>
                                <resourceType>MEM</resourceType>
                                <memThreads>0</memThreads>
                          </emulator-params>
        </distributions>

        <distributions>
         <name>IO_Distro</name>
           <startTime>0</startTime>
           <!--duration in seconds -->
           <duration>60</duration>
           <granularity>5</granularity>
           <distribution href="/distributions/linear_incr" name="linear_incr" />
            <startLoad>1</startLoad>
            <stopLoad>10</stopLoad>
```

```
                <emulator href="/emulators/lookbusy" name="lookbusy" />

                <emulator-params>
                  <!--more parameters will be added -->
                  <resourceType>IO</resourceType>
                 <!--Size of blocks to use for I/O, in MB-->
                 <ioBlockSize>10</ioBlockSize>
                 <!--Time to sleep between iterations, in msec-->
                 <ioSleep>100</ioSleep>
                </emulator-params>
              </distributions>

            <log>
              <!-- Use value "1" to enable logging(by default logging is off)  -->
              <enable>1</enable>
              <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec)  -->
              <frequency>3</frequency>
            </log>
          </emulation>
```

3 Indices and tables

- *genindex*
- Glossary

Index

Symbols

C