COCOMA Documentation

Release 1

Carmelo Ragusa, Philip Robinson, Sergej Svorobej

April 11, 2013

Contents

COCOMA framework was designed by SAP as part of EU funded BonFIRE project. Task of COCOMA framework is to create, monitor and control contentious and malicious system workload. By using this framework experimenters are able to make testing process more accurate and anticipate various scenarios of cloud infrastructure behaviour, collect and correlate metrics of the emulated environment with the test results.

1 Contents

1.1 Introduction

In order to use COCOMA framework experimenter creates an emulation using XML language(see below Examples section). Emulation should contain all the neccessary information about duration, magnitude and required resource usage. Once XML document is received by COCOMA, the framework will automatically schedule and execute required workload on the chosen resource(-s) such as CPU, IO, Memory and/or Network.

Installation

The framework is designed to run on GNU/Linux and released in *.deb* package only. Once you have downloaded latest COCOMA version install it by running:

```
$ dpkg -i cocoma_X.X-X_all.deb
```

The application will be installed to folder *"/usr/share/pyshared/cocoma"*. All the additional required programs and libraries will be downloaded and installed on the fly if missing. To check check if it was installed correctly run:

```
$ ccmsh -v
```

Starting Components

To avail full functionality of COCOMA two daemons need to be started:

- Scheduler daemon (mandatory)
- API Daemon (optional if REST API functionality is required)

Scheduler daemon - runs in the background and executes workload with differential parameters at the time defined in the emulation properties. to start scheduler use command:

```
$ ccmsh --start scheduler
```

Default network interface is *eth0*, port *51889* you can change that by adding required interface name and port number at the end:

```
$ ccmsh --start scheduler wlan0 5180
```

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

```
$ ccmsh --start scheduler wlan0 5180 debug
```

Note: Scheduler needs to be running otherwise nothing will work. Always start it first!!

API daemon - represents RESTfull web API which exposes COCOMA resources for use over the network. It follows the same startup pattern as the Scheduler:

```
$ ccmsh --start api
```

By default web API will try to start using *eth0* network interface on port *5050*, but it can be changed by supplying own parameters:

```
$ ccmsh --start api wlan0 3030
```

The log level will be always same as the *Scheduler*.

Command Line Arguments

The COCOMA **ccmsh** command line interface has several 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 will it will list information for that specific emulation

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

-j, -list-jobs
: Querries scheduler for the list of jobs which is to be executed. Gives jobs names and planned execution time

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

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

-x, -xml `<file name>`
: If you have a local XML file with emulation parameters, you can use it to create emulation.

-n, -now `(used with -x option only)`
: If your local XML file emulation has set start date in past or in future, but you want to override it and start the test right now, without modifying the file, then you can add this option after the file name i.e. `ccmsh -x <file name> -n`

-d, -delete `<emulation name>`
: Deletes specific emulation from the database, logs will remain and will be available until manualy deleted from *"/usr/share/pyshared/cocoma/logs"* folder

-p, -purge
: Wipe all DB entries, removes all scheduled jobs, logs will remain and will be available until manualy deleted from *"/usr/share/pyshared/cocoma/logs"* folder

-start `<api interface port>, <scheduler interface port>`
: Launch 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 interface is not specified then Scheduler daemon will run on *eth0* port *51889* and API daemon runs on *eth0* with port *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

If the web API daemon has been started successfully, then COCOMA toolkit can be accessed remotely using its RESTfull API.

The API URIs summary list:

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

REST API Description

GET **/**

Title root

Responses

- **200** – OK
- **404** – Not Found

The **root** method returns *collection* of all the available resources. Example 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>
```

GET **/emulations**

Title emulations

Responses

- **200** – OK
- **404** – Not Found

The **emulations** method returns *collection* of all the available emulation resources. Example 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" state="inacti
     <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>
```

GET **/emulations/*{name}***

Path arguments **name** – Name of emulation that you want to get more info

Responses

- **200** – OK
- **404** – Not Found

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>
```

POST **/emulations**

Query params XML (*string*) – Emulation parameters defined via XML as shown in the examples section.

Responses

- **201** – Emulation was created successfully
- **400** – Bad Request

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>
```

GET **/emulators**

Responses

- **200** – OK
- **404** – Not Found

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>
```

GET **/emulators/*{name}***

Path arguments **name** – Name of emulator that you want to get more info

Responses

- **200** – OK
- **404** – Not Found

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

```
<?xml version="1.0" ?>
<emulator href="/emulator/lookbusy" xmlns="http://127.0.0.1/cocoma">
  <info>
    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:
    &lt;emulator-params&gt;
        &lt;resourceType&gt;CPU&lt;/resourceType&gt;
        &lt;ncpus&gt;0&lt;/ncpus&gt;
    &lt;/emulator-params&gt;

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

GET **/distributions**

Responses

- **200** – OK
- **404** – Not Found

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>
```

GET **/distributions/*{name}***

Path arguments name – Name of distributions that you want to get more info

Responses

- **200** – OK
- **404** – Not Found

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 increasing res
  <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
  <link href="/distributions" rel="parent" type="application/vnd.bonfire+xml"/>
</distribution>
```

GET **/tests**

Responses

- **200** – OK
- **404** – Not Found

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-Lookbusy_10-95.xml"
    <test href="/tests/03-NET-Linear_incr-Iperf-100-1000.xml" name="03-NET-Linear_incr-Iperf-100
    <test href="/tests/02-IO-Linear-Stressapptest_1-10.xml" name="02-IO-Linear-Stressapptest_1-1
    <test href="/tests/02-IO-Linear_incr-Stressapptest_1-10.xml" name="02-IO-Linear_incr-Stressa
    <test href="/tests/02-MEM-Linear_incr-Stressapptest_100-1000.xml" name="02-MEM-Linear_incr-S
    <test href="/tests/01-CPU-Trapezoidal-Lookbusy_10-95.xml" name="01-CPU-Trapezoidal-Lookbusy_
    <test href="/tests/01-IO-Trapezoidal-Lookbusy_1-10.xml" name="01-IO-Trapezoidal-Lookbusy_1-1
    <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-overlap.xml"/>
    <test href="/tests/01-CPU-Linear_incr-Lookbusy_10-95.xml" name="01-CPU-Linear_incr-Lookbusy_
    <test href="/tests/01-IO-Linear_incr-Lookbusy_1-10.xml" name="01-IO-Linear_incr-Lookbusy_1-1
    <test href="/tests/02-IO-Trapezoidal-Stressapptest_1-10.xml" name="02-IO-Trapezoidal-Stressa
    <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_incr-Lookbu
    <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-Linear-Stressapp
    <test href="/tests/01-MEM-Trapezoidal-Lookbusy_100-1000.xml" name="01-MEM-Trapezoidal-Lookbu
    <test href="/tests/02-MEM-Trapezoidal-Stressapptest_100-1000.xml" name="02-MEM-Trapezoidal-S
    <test href="/tests/03-NET-Trapezoidal-Iperf-100-1000.xml" name="03-NET-Trapezoidal-Iperf-100
    <test href="/tests/01-IO-Linear-Lookbusy_1-10.xml" name="01-IO-Linear-Lookbusy_1-10.xml"/>
  </items>
  <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

GET **/tests/{name}**

Path arguments name – Name of tests that you want to get more info

Responses

- **200** – OK
- **404** – Not Found

Displays Content of XML file.

POST **/tests**

Query params string – name of the test that is located on COCOMA server

Responses

- **201** – Emulation was created successfully
- **400** – Bad Request

Create emulation from already 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>
```

GET **/results**

Responses

- **200** – OK
- **404** – Not Found

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" state="inac
    <results failedRuns="0" href="/results/2-CPU_EMU" name="2-CPU_EMU" state="inactive"/>
    <results failedRuns="0" href="/results/3-CPU_EMU" name="3-CPU_EMU" state="inactive"/>
    <results failedRuns="0" href="/results/4-CPU_EMU" name="4-CPU_EMU" state="inactive"/>
    <results failedRuns="0" href="/results/5-CPU_EMU" name="5-CPU_EMU" state="inactive"/>
  </items>
  <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

GET **/results/*{name}***

Path arguments **name** – Name of tests that you want to get more info

Responses

- **200** – OK
- **404** – Not Found

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>
```

GET **/logs/system**

Responses

- **200** – OK
- **404** – Not Found

Return Zip file with system logs.

GET **/logs/emulations**

Responses

- **200** – OK
- **404** – Not Found

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>
```

GET **/logs/*{name}***

Path arguments **name** – Name of emulation logs that you want to get more info

Responses

- **200** – OK
- **404** – Not Found

Return Zip file with emulation logs.

XML payload structure

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>
```

XML document defines emulation experiment details and consists of three blocks:

- **Emulatuion**

```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>
  ...
</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 emulation named *CPU_EMU* running for *60* sec. and stating right *now*. Emulation will include one distribution called *CPU_Distro* which starts at the same time as emulation, will run for *60* sec. using *linear* algorithm. In its duration it will increase workload of *CPU* from *10%* to *95%* in *20* steps by using *lookbusy* emulator. Workload produced by the application will be logged every second with very detailed information.

1.2 Creating Emulation via CLI

Once *Scheduler* was started and running we can now create stress emulations for the resources. We are using local XML `emulation.xml` file:

```
 <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/stressapptest" 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/stressapptest" 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>
```

start with simple CLI command:

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

If everything went right, you will see on the screen list of scheduled jobs:

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

```
11  6-Emu-CPU-RAM-IO
```

Each line from *3-8* shows information of a single scheduled emulation job. If we break it down, the line *3* from above as an example we have:

- **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 type of the resource used by this run
- **90** - stress value applied to this run
- **Duration 10.0sec.** - how long will job run
- **Start Time: 2013-04-10 09:43:01 End Time: 09:43:11** - time gap when run will be executed

We can write run name notation in this way:

`(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 will appear only if *log* section is stated in XML and is optional. Logger job runs for the duration of the whole emulation and collects system resource usage information. Logger job name notation can be described in this way:

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

1.3 Creating Emulation via API Client (Restfully)

Here you can find examples for building ruby script XML payloads for restfully client.

First you need to create configuration file like `cocoma.yml`. It contains the IP address URI of COCOMA web API and used mediatype name:

```
uri: http://10.55.164.223:5050/
require: [ApplicationVndBonfireXml]
```

Next we creating `MEM-emulation.rb` file which contains XML payload and the path to config file:

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

session = Restfully::Session.new(
 :configuration_file => "~/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",
            :name => "linear"},
           :startLoad => "1000",
            :stopLoad => "18000",
            :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",
                                :name => "linear"},
                :startLoad => "18000",
                :stopLoad => "1000",
                :emulator =>{
                                :href => "/emulators/stressapptest",
                                :name => "stressapptest"},
                :'emulator-params' =>{
                                :resourceType =>"MEM",
                                :memThreads => "1"}
                }]
 )

end
```

Finally we launch the script using `resfully` client

```
$ restfully emulation.rb
```

If then you would like to check if the emulation was created you can list emulations again using `restfully` :

```
$ restfully -c cocoma.yml

>> #<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 web site.

1.4 XML Examples

This section shows XML payload examples for creating emulation experiments

CPU

Emulation XML for the CPU contention:

```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>120</emustopTime>

  <distributions>
   <name>CPU_Distro</name>
     <startTime>0</startTime>
     <!--duration in seconds -->
     <duration>120</duration>
     <granularity>24</granularity>
     <distribution href="/distributions/linear_incr" name="linear_incr" />
     <!--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>
```

I/O

Emulation XML for the I/O contention:

```xml
<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/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>
   <logLevel>debug</logLevel>
  </log>

</emulation>
```

2nd example using *trapezoidal* distribution:

```xml
<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/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>
   <logLevel>debug</logLevel>
  </log>

</emulation>
```

Memory

Emulation XML for the 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>
      <emulator href="/emulators/lookbusy" name="lookbusy" />
      <emulator-params>
        <resourceType>MEM</resourceType>
```

```
       <!--time between iterations in usec (default 1000)-->
       <malloclimit>4004</malloclimit>
      <memSleep>0</memSleep>
      </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>
```

2nd example 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>4000</malloclimit>
      <emulator href="/emulators/lookbusy" name="lookbusy" />
      <emulator-params>
        <resourceType>MEM</resourceType>
       <!--time between iterations in usec (default 1000)-->
      <memSleep>0</memSleep>
      </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

For this emulation to work you need to deploy two COCOMA VM's. One will act as a client (the one where XML is sent) and the other will act as a server. Emulation XML for the network contention:

```xml
<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" name="linear" />
   <!--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>10.55.164.223</serverip>
   <!--Leave "0" for default 5001 port -->
   <serverport>5001</serverport>
        <clientip>127.0.0.1</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

You can create multiple distributions within one 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 in the same time frame. If distributions for the same resource will overlap, the runs might crash if not enough resources available.

- CPU and Memory example

```xml
<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>
        <emulator href="/emulators/lookbusy" name="lookbusy" />
        <emulator-params>
          <resourceType>MEM</resourceType>
         <!--time between iterations in usec (default 1000)-->
         <malloclimit>4004</malloclimit>
        <memSleep>0</memSleep>
        </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>
                 <emulator href="/emulators/lookbusy" name="lookbusy" />
                 <emulator-params>
                   <resourceType>MEM</resourceType>
                  <!--time between iterations in usec (default 1000)-->
                  <malloclimit>4004</malloclimit>
                 <memSleep>0</memSleep>
                 </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>
```

2 Indices and tables

- *genindex*
- *search*

Index