VM-MAD Documentation

Release development (SVN** $Revision$**)

Simon Barkow, Peter Kunszt, Sergio Maffioletti, Riccardo Murri, C

November 26, 2012

CONTENTS

Contents:

CHAPTER

ONE

INSTALLATION OF VM-MAD

Author Riccardo Murri <riccardo.murri@gmail.com>

Date 2010-10-06

Revision $Revision$

1.1 Installation

These instructions show how to install VM-MAD from the source repository into a separate python environment (called virtualenv). Installation into a virtualenv has two distinct advantages:

- All code is confined in a single directory, and can thus be easily replaced/removed.
- Better dependency handling: additional Python packages that VM-MAD depends upon can be installed even if they conflict with system-level packages.

0. Install software prerequisites:
   - On Debian/Ubuntu, install packages: `subversion`, `python-dev`, `python-profiler` and the C/C++ compiler:

     ```
     apt-get install subversion python-dev python-profiler gcc g++
     ```

   - On CentOS5, install packages `subversion` and `python-devel` and the C/C++ compiler:

     ```
     yum install subversion python-devel gcc gcc-c++
     ```

   - On other Linux distributions, you will need to install:
     - the `svn` command (from the SubVersion VCS)
     - Python development headers and libraries (for installing extension libraries written in C/C++)
     - the Python package `pstats` (it's part of the Python standard library, but sometimes it needs separate installation)
     - a C/C++ compiler (this is usually installed by default).
1. Choose a directory where the VM-MAD software will be installed; any directory that's writable by your Linux account will be ok.

   If you are installing system-wide as `root`, we suggest you install VM-MAD into `/opt/vm-mad`.

   If you are installing as a normal user, we suggest you install VM-MAD into `$HOME/vm-mad`.
2. If it's not already installed, get the virtualenv Python package and install it:

```
  wget http://pypi.python.org/packages/source/v/virtualenv/virtualenv-1.5.1.tar.gz
  tar -xzf virtualenv-1.5.1.tar.gz && rm virtualenv-1.5.1.tar.gz
  cd virtualenv-1.5.1/

If you are installing as `root`, the following command is all you
need:

  python setup.py install

If instead you are installing as a normal, unprivileged user,
things get more complicated::

  export PYTHONPATH=$HOME/lib64/python:$HOME/lib/python:$PYTHONPATH
  export PATH=$PATH:$HOME/bin
  mkdir -p $HOME/lib/python
  python setup.py install --home $HOME

(You will also need to add the two `export` lines above to the
`$HOME/.bashrc` file -if using the `bash` shell- or to the
`$HOME/.cshrc` file -if using the `tcsh` shell.)

In any case, once `virtualenv`_ has been installed, you can exit
its directory and remove it::

  cd ..
  rm -rf virtualenv-1.5.1
```

3. Create a virtualenv to host the `vm-mad` installation at the directory you chose in Step 1.:

   ```
   virtualenv $HOME/vm-mad # use '/opt/vm-mad' if installing as root
   cd $HOME/vm-mad/
   source bin/activate
   ```

4. Check-out the `vm-mad` files in a `src/` directory:

   ```
   svn co http://vm-mad.googlecode.com/svn/trunk src
   ```

5. Install the `vm-mad` in "develop" mode, so any modification pulled from subversion is immediately reflected in the running environment:

   ```
   cd src/
   env CC=gcc ./setup.py develop
   cd .. # back into the `vm-mad` directory
   ```

   This will place all the VM-MAD command into the `vm-mad/bin/` directory.

1.2 Upgrade

These instructions show how to upgrade the VM-MAD scripts to the latest version found in the GC3 svn repository.

1. *cd* to the directory containing the VM-MAD virtualenv; assuming it's named `vm-mad` as in the above installation instructions, you can issue the commands:

   ```
   cd $HOME/vm-mad # use '/opt/vm-mad' if root
   ```

2. Activate the virtualenv

```
source bin/activate
```

3. Upgrade the *vm-mad* source and run the *setup.py* script again:

```
cd src
svn up
env CC=gcc ./setup.py develop
```

1.3 HTML Documentation

HTML documentation for the VMlib programming interface can be read online at:

http://vm-mad.googlecode.com/svn/trunk/doc/html/index.html

You can also generate a local copy from the sources:

```
cd $HOME/vm-mad # use '/opt/vm-mad' if root
cd src/docs
make html
```

Note that you need the Python package *Sphinx <http://sphinx.pocoo.org>* (at least versdion 1.0) in order to build the documentation locally.

CHAPTER

TWO

VM-MAD MODULES

2.1 *orchestrator*

2.2 *simul*

2.3 *ge_info*

CHAPTER

THREE

COMMANDS IN VM-MAD

Author Tyanko Aleksiev <tyanko.alexiev@gmail.com>

Date 2012-04-29

Revision $Revision$

3.1 Commands

This article explains the commands available inside VM-MAD. More precisely, it gives an initial overview of the command, followed by a description of the possible interactions with other commands. References to input/output files' format is also being provided.

3.2 Simulation

VM-MAD has an integrated simulation suite which enables processing SGE accounting data. The main idea of this implemetation can be associated with the answer of the question: "What would be the evolution of my cluster's queue during the time if I had on my disposal X always running servers and the possibility to spawn Y Virtual Machines on demand?". Where X and Y are variables that can be chosen by the final user. The simulation process involves three different parts:

- provided accounting data has to be first elaborated from the distil.py tool. For more information see the *Distill* section,
- once the accounting data is available a simulation can be started using the simul.py tool,
- finally the plot_workload.R R script is used for graphically represent the results.

The output produced by the distil.py tool is needed before starting a new simulation. The *Distill Output* section describes in more detail what kind of information the distill tool is providing to the simulator suite.

A new simulation can be set-up by using the provided options, to see all of them:

```
(vm-mad)vm-user@test:~ ./simul.py --help
usage: simul.py [-h] [--max-vms N] [--max-delta N] [--max-idle NUM_SECS]
            [--startup-delay NUM_SECS] [--csv-file String]
            [--output-file String] [--cluster-size NUM_CPUS]
            [--start-time String] [--time-interval NUM_SECS] [--version]

Simulates a cloud orchestrator

optional arguments:
```

```
-h, --help            show this help message and exit
--max-vms N, -mv N    Maximum number of VMs to be started, default is 10
--max-delta N, -md N  Cap the number of VMs that can be started or stopped
                      in a single orchestration cycle. Default is 1.
--max-idle NUM_SECS, -mi NUM_SECS
                      Maximum idle time (in seconds) before swithing off a
                      VM, default is 7200
--startup-delay NUM_SECS, -s NUM_SECS
                      Time (in seconds) delay before a started VM is READY.
                      Default is 60
--csv-file String, -csvf String
                      File containing the CSV information, accounting.csv
--output-file String, -o String
                      File name where the output of the simulation will be
                      stored, main_sim.txt
--cluster-size NUM_CPUS, -cs NUM_CPUS
                      Number of VMs, used for the simulation of real
                      available cluster: 20
--start-time String, -stime String
                      Start time for the simulation, default: -1
--time-interval NUM_SECS, -timei NUM_SECS
                      UNIX interval in seconds used as parsing interval for
                      the jobs in the CSV file, default: 3600
--version, -V         show program's version number and exit
```

The `--max-vms` and `--cluster-size` options are probably the most important as they permit you to simulate different configuration scenarios. The `--max-vms` allows you to set how expandable, in terms of VMs, your cluster could be. The `--cluster-size` options permits you to fix the simulated dimension of your locally availbale cluster.

Once the simulation is completed you can compute the results using the plot_workload.R script:

```
(vm-mad)vm-user@test:~ ./plot_workload.R simulation_output_file output_file
```

Two files are produced at the end: output_file.pdf and output_file.eps. They represent what would be the graphical evolution of your queue with the specified options.

3.2.1 Distill

The purpose of the `distil.py` tool is to elaborate different kind of scheduling information and produce an output in CSV format legible from the simulator suite. The following data input formats are currently recognized by the tool:

- accounting data provided by SGE,
- the output given by querying the SGE scheduler with the `qstat -xml` command. (working in progress)

You can see all the provided options by simply doing `./distil.py -h`

Distill Output

The output produced by the distil.py is in the CSV format tool has the following aspect:

```
JOBID, SUBMITTED_AT, RUNNING_AT, FINISHED_AT, WAIT_DURATION, RUN_DURATION
1,     1282733694,   1282733707, 1282733785,  13,            78
4,     1282736899,   1282736911, 1282737239,  12,            328
6,     1282738136,   1282738141, 1282738141,  5,             0
7,     1282738434,   1282738441, 1282738568,  7,             127
8,     1282739338,   1282739342, 1282740438,  4,             1096
```

The first row of the file is quite self-explaining about what kind of information, each of the columns, is containing.

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*