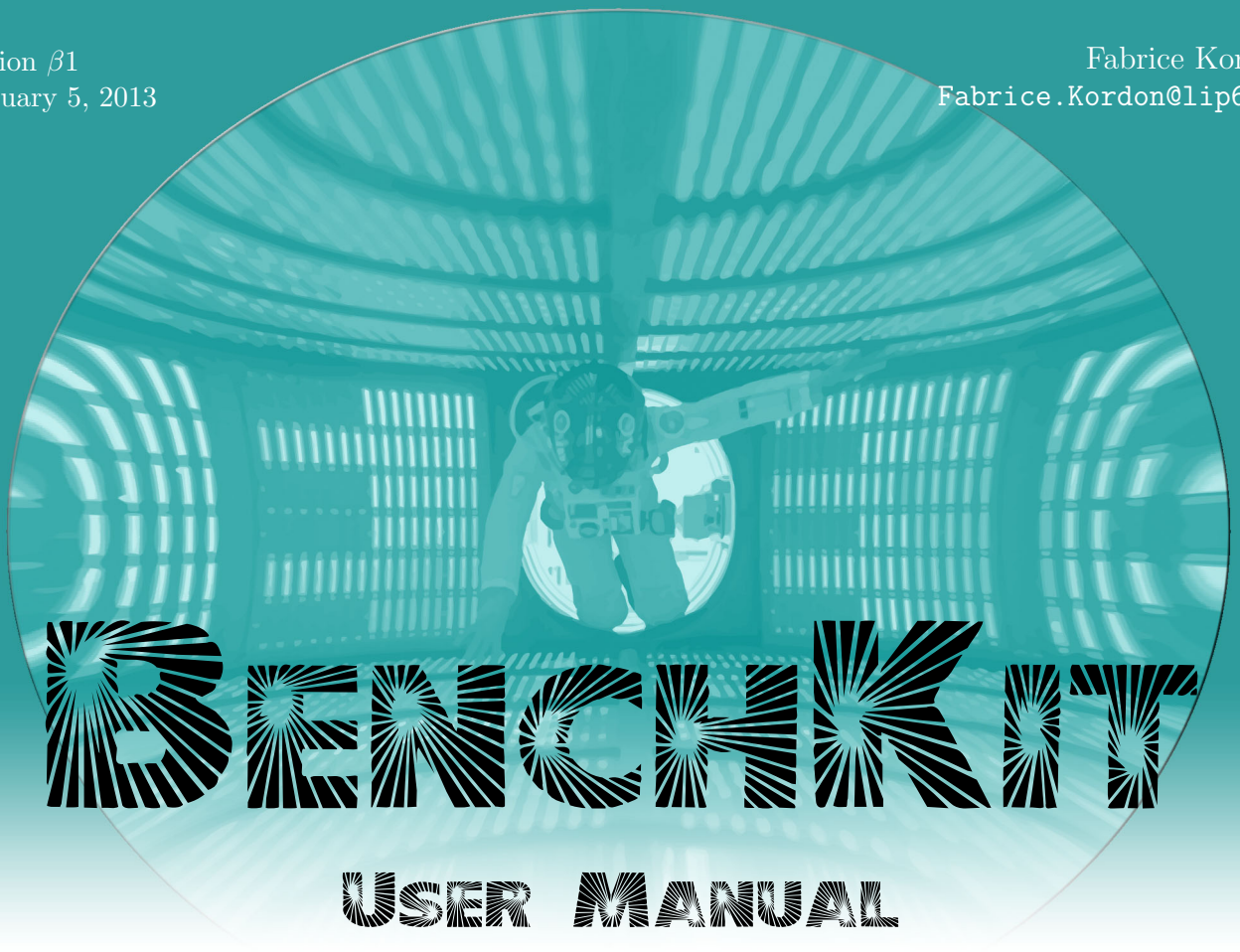
Version $\beta 1$
February 5, 2013

Fabrice Kordon
Fabrice.Kordon@lip6.fr



BENCHKIT

User Manual

UPMC SORBONNE UNIVERSITÉS

http://benchkit.cosyverif.org

LIP6

Contents

Warning. the current version of BenchKit does not yet handle the execution of several virtual machine on the same remote node that is mentioned in the current document. This is to be fixed in a later version of this tool.

Developers. BenchKit was primarily developed by Fabrice Kordon with the precious help of Nicolas Gibelin, Francis Hulin-Hubard and Franck Pommereau (mostly for the virtual machine stuff, as well as the monitoring aspects).

BenchKit is developed within the context of the **CosyVerif**[1] project founded by three laboratories in the Paris area: LIP6 (université Pierre & Marie Curie), LIPN (Université Paris Nord) and LSV (École Nationale Supérieure de Cachan).

[1] `http://cosyverif.org`

Fabrice Kordon
Fabrice.Kordon@lip6.fr

1 Introduction: What is **BenchKit**

Do you want to evaluate time and memory consumption of any software? *BenchKit* is made for you. This is a tool that allows you to evaluate these data. It measures:

- the time to run your software (the "user experience" point of view),
- the CPU average consumption of your software,
- the maximum memory used by your software,
- the evolution of time and CPU over the execution of your software (as a percentage of what is allocated).

BenchKit is of particular interest when you want to evaluate these on numerous inputs for a given program. You also confine the software execution up to a maximal execution time or memory usage.

Compared to other solutions such as `memtime`[2], an efficient solution developed by the UPPAAL community, *BenchKit* is able to evaluate memory and CPU consumption of multi-processes (or multi-threaded) applications.

A Typical use. *BenchKit* was elaborated aside the `Model Checking Contest @ Petri Net` in 2011 and 2012 where several model checking tools are evaluated against a set of models [2, 1]. Preliminary versions were also used to benchmark prototype tools for various papers.

Where BenchKit could be operated. *BenchKit* has been designed to be operated on any type of machine. It is mostly of interest when you have access to a cluster or a machine with numerous cores and a large amount of memory.

BenchKit runs under Unix but is based on virtual machines technology. Thus, the virtual machine can embed other types of operating systems. So far, it has been tested for Linux windows virtual machines.

2 The **BenchKit** Architecture

This section presents the overall architecture of *BenchKit*. It follows a classical "master/slave" structure. By similarity to the terminology of cluster computing, we call "head" the master part and "remote" the slave nodes.

Hardware Architecture. The *BenchKit* architecture is depicted in Figure 1. We distinguish two types of machine: the *head machine* and the *remote nodes*. The distinction is conceptual since all these machines can be located in a unique physical one. However, these two types of machine do have different roles.

The *head machine* is the one where an analysis is started. It contains the scripts and the three configuration files:

- a list of inputs to be processed,
- a list of programs (associated to VMs) to process these tests,
- a list of remote machines where these programs will operate the tests.

The *remote nodes* are the computers processing the tests. This notion is logical since there can be several remote nodes per physical computer. Typically, a multi-core computer can host several nodes if it contains a sufficient amount of memory.

[2] See `http://www.uppaal.org/`.

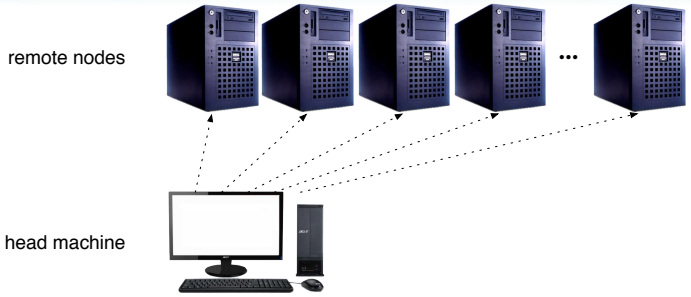


Figure 1: the *BenchKit* hardware architecture

To operate *BenchKit*, we assume that all physical machines operate Unix (*BenchKit* was developed and experimented under Linux). Execution of the programs will be performed under virtual machines ; we heavily experimented virtual machines under Linux but Windows is supported too (provided that CygWin[3] is installed).

Prerequisite. *BenchKit* requires `bash`[4] and `python`[5] to be installed on the head machine and the remote nodes. It also requires `qemu` to be installed on the remote nodes. It is highly recommended that the physical machines where remote nodes are executed operate hardware acceleration of virtual machines (otherwise, execution is very slow).

Software Architecture. The software architecture of *BenchKit* is dispatched over threes entities: the head machine, the remote nodes and the virtual machines that contain the software to be executed.

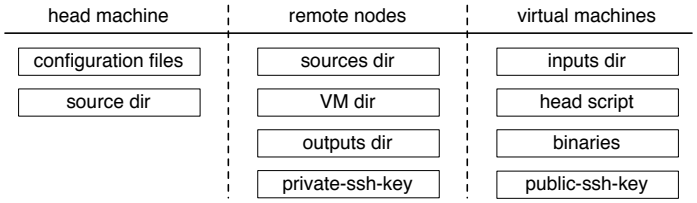


Figure 2: the *BenchKit* software architecture

The Head Machine hosts the source the configuration files.

The main script is `launch_benchmark.sh`. It is operated when you want to launch and distribute the execution of your tools over the remote nodes. It uses information gathered from five configuration files:

- `configuration.sh` allows you to refer to the other configuration files and to set up the time and memory you allocate to your software to be executed,
- `tool_list.txt`[6] defines the list of tools and functions to be operated. Each line defines a tool to be operated and must respect the following format.

  `examination_name:software_name:vm_image_name`

[3] see `http://www.cygwin.com`.
[4] Checks were done with bash version 4.2.8.
[5] Checks were done with `python` version 2.7.1+.
[6] This the default name of this file, you can modify it by changing the default value of the corresponding environment variable in `configuration.sh`.

  - `examination_name` is an identifier that will be passed to the virtual machine; it denotes the function to activate in your software,
  - `software_name` is the identifier of the program to be activated on the virtual machine,
  - `vm_image_name` is the image disk to be operated by the virtual machine.

  You can set-up as many programs as you want in a virtual machine as long as the head *BenchKit* script (see later) is able to recognize them and launch the appropriate one.

- `test_list.txt`[6] defines a list of test identifier to be processed for each tool. There is one test identifier per line. Each of this identifier corresponds to a directory where all the date to be processed for an examination by a given program is located.

- `rmachine_list.txt`[6] defines the list of nodes that will execute the programs. It must define the number of remote nodes in the following way.

  `NB_REMOTE_NODES:<a value>`

  where `<a value>` is the number of nodes to be activated. Nodes are numbered from 1 to `<a value>`. Each one must be described in a line having the following format.

  `node_number:host_name:login`

  - `node_number` it is a value between 1 and the number of remote nodes specified as shown above; it denotes a logical note where virtual machines will be executed to process programs on the examinations for each test,
  - `host_name` this is the name or IP address of the physical host to be contacted,
  - `login` is the login to access this host.

  Several logical node can be executed on the same physical node with one or more logins. It is better that a public key is installed on the physical machines that will operate the tests.

For the three `.txt` configuration files, empty lines, as well as lines beginning with a `#`, are ignored.

Based on these configuration files, the main script (`launch_benchmark.sh`) generates one shell script per remote node, upload the system to each physical machine and then execute the scripts for each remote node.

It is possible to set-up notification emails (in the `configuration.sh` file) to be warned when a run is finished or when the execution on a remote node is terminated.

Remote Nodes execute a subset of the benchmark tests. For each involved physical machine, the following elements must be set up:

- a directory to put *BenchKit* sources and configuration files,
- a directory to put the image disks for the virtual machines,
- a directory where *BenchKit* drops its outputs.

These elements must be specified in the `configuration.sh` file. All the physical nodes must respect the same location rules. Be sure that the login used for remote nodes have write permission access to the directory where outputs ill be dropped.

It is important that, in the directory containing *BenchKit* sources, there is a file `bk-private-key` with the private key associated to the login used to connect the virtual machines. This file must have `rwx------`

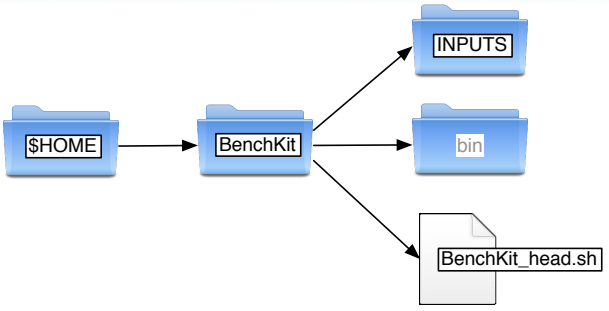


Figure 3: Structure of the VM for the login specified for the execution (labels in squares are the mandatory ones)

permission access[7]. If this file is not present with the appropriate permission, then a password will be required each time *BenchKit* runs a test.

You may generate any pair, store the private key in `bk-private-key` and install the corresponding public key in the `.ssh` directory of both `root`[8] and the login you use for your tests. A default Key is provided[9] ; it is installed in the default VM we propose.

The Virtual Machine executes one software on one examination and for one test. Execution is performed using a login specified in the `configuration.sh` file (environment variable `VM_LOGIN`).

The corresponding user account must be configured as presented in Figure 3. The `$HOME` must contain a directory named `BenchKit` that must contain the *BenchKit* "head script" (*i.e.* the one communicating with the invocation system). It also contains the `INPUTS` directory (for inputs), itself containing one directory per test to be performed.

We recommend to locate in `BenchKit`, a **bin** directory that contains the binaries of your software. However, this last directory can be located elsewhere.

Important: be sure to provide binaries compatible with the architecture of the VM. Moreover, these binaries must be standalone (*i.e.* they are compatible with the installation of the VM you use).

Structure of the `INPUTS` directory The philosophy is to establish one directory per test. This directory must contain all the files needed to process your tool on a given test for all the examinations listed in the `tool_list.txt` file.

It is important to note that *BenchKit* goes in this directory prior to invoke `BenchKit_head.sh`. Thus, your software can find all the required data in its current directory.

The role of `BenchKit_head.sh` this shell script is invoked by *BenchKit* with the following environment variables set up:

- `BK_TOOL`: this variable contains the name of the invoked program. Then, if you have several programs located in the same virtual machine, `BenchKit_head.sh` can select the appropriate one.
- `BK_EXAMINATION`: this variable contains the name of the action your software must perform. This is a way to select an action when the evaluated software may perform several type of computations.

[7] This is required by ssh, otherwise, it does not accept the Key.
[8] Of, course, only the one of the virtual machine.
[9] `http://benchkit.cosyverif.org/BENHMARKS/bk-private_key.zip`

```
$ ls -lh
total 6704
4768 -rw-r--r--  1 fko  fko   2,3M 21 oct 22:36 BenchKit.pdf
   8 -rw-------  1 fko  fko   1,6K 21 oct 22:36 bk-private_key
   8 -rw-r--r--@ 1 fko  fko   1,2K 21 oct 22:36 configuration.sh
   8 -rw-r--r--@ 1 fko  fko    52B 21 oct 22:36 halt_a_vm.sh
   8 -rw-r--r--@ 1 fko  fko   983B 21 oct 22:36 invocation_template.txt
   8 -rw-r--r--@ 1 fko  fko   2,7K 21 oct 22:36 launch_a_command.sh
   8 -rw-r--r--@ 1 fko  fko   1,2K 21 oct 22:36 launch_a_run.sh
   8 -rw-r--r--@ 1 fko  fko   219B 21 oct 22:36 launch_a_vm.sh
  24 -rw-r--r--@ 1 fko  fko   8,7K 21 oct 22:36 launch_benchmark.sh
1824 -rw-r--r--  1 fko  fko   911K 21 oct 22:36 monitor-supp.tgz
   8 -rw-r--r--@ 1 fko  fko   561B 21 oct 22:36 rmachine_list.txt
   8 -rw-r--r--@ 1 fko  fko   260B 21 oct 22:36 test_list.txt
   8 -rw-r--r--@ 1 fko  fko   655B 21 oct 22:36 tool_list.txt
   8 -rw-r--r--@ 1 fko  fko   943B 21 oct 22:36 vm.sh
```

Figure 4: Content of the distribution

- `BK_INPUT`: this variable contains the name of the current input being processed. This can be useful when invoking your tool but it allows *BenchKit* to go in the appropriate test directory prior to the invocation of your software.

The `BenchKit_head.sh` file is your way to interface your programs with *BenchKit*. You most probably want to normalize outputs in order to perform post-processing of the collected data (see section 4).

Distribution. The content of the *BenchKit* distribution is shown in Figure 4. The main files to be considered are:

- `BenchKit.pdf`: this documentation,
- `bk-private-key`[10]: the default private key associated to the default public kay proposed in the default disk image (allows access to root and the default user login proposed in the virtual machine that executes your program) ; it is to be replaced by another couple public/private key if you want so (do not mismatch with the public/private keys used to connect to remote nodes),
- `configuration.sh`: the script where the main configuration variables are set. You need to check these in order to adapt *BenchKit* to your own configuration.
- `launch_benchmark.sh`: the main script that launches the execution of your program on all tests once the virtual machine and all data have been configured.
- a default image disk to be used in a VM[11]. This dish image is compatible with `qemu` (used by *BenchKit*) or VirtualBox. On a Linux machine with `qemu-kvm` installed, it can be operated with the `launch_a_vm.sh` and `halt_a_vm.sh` service scripts that allow to manually start/stop a virtual machine to configure it.

A default virtual machine (Linux 32bits Unbuntu) is also provided in a separate file: `default-vm.vmdk`. You may use it to set-up your programs (default user login is `mcc`) or set-up another one with your dedicated settings.

Deployment of *BenchKit* on the remote nodes (as well as some checks) is performed using `launch_benchmark.sh`. This commands provides online help as shown below.

[10] `http://benchkit.cosyverif.org/BENHMARKS/bk-private_key.zip`
[11] Downloadable at `benchkit.cosyverif.org/BENCHMARKS/BK-empty.vmdk.zip`. It runs a Linux Debian 7.0 in 32 bit mode.

```
$ sh launch_benchmark.sh -help
usage: launch_benchmark.sh [-deploy] [-effective]
       -deploy:    enforces the deployment of all scripts on the target machines
                   and checks for the existence of image  files for the  VMs.
       -effective: requested to execute the files on the remote nodes, otherwise
                   only the files to be executed are generated (for checks)
```

BenchKit is distributed under the GPL licence.

3 How to use BenchKit

This section shows how to use *BenchKit* to perform measures on a software. There are two steps:

- preparation of your software,
- launching a benchmark once your software is prepared.

Preparing Your Software. Preparing your software to be executed under *BenchKit* is relatively simple. You just have to elaborate the image disk to be executed in the virtual machine. The main actions are:

1. Compile your programs (taking in account the constraints of the VM OS),
2. set up the image disk of the virtual machine with your compiles programs and the tests to be processed. In particular, you must install the public key that allows to connect to this virtual machine (if you d not use the provided default one). You must also implement `BenchKit_head.sh` to invoke appropriately your tool as explained in section 2,
3. deploy the disk images on the remote nodes,
4. Configure *BenchKit* as specified in section 2.

Launching a benchmark. Once the virtual machine is ready and deployed on the nodes, you must deploy *BenchKit* unless this has been already done. Type the following command:

```
$ sh launch_benchmark.sh -deploy
```

BenchKit deploy the sources on the executing physical machines (you must do that each time you change the remote node configuration) and checks if the disk images are located where expected.

The scripts to be executed on the remote nodes are also generated (you may want to check them, they are named `BenchKit_node_`⟨*node*⟩`.sh`). If you want to regenerate these scripts, you may type:

```
$ sh launch_benchmark.sh
```

To generate these scripts and launch the tests on the remote nodes, then type

```
$ sh launch_benchmark.sh -effective
```

This can take a while, it can be useful to have emails sent regularly to follow the progression of the execution. This is configured in `configure.sh`. Emails title are formatted to let you set-up rules, then enabling a redirection into a dedicated mailbox.

4 BenchKit Outputs

This section describes the output provided by *BenchKit*. These can be sent automatically by email when the execution on a remote node is terminated. After the execution, all the information remains in the output directory you specified. It is erased when you launch a new benchmark on the same machines with the same configurations.

The outputs from BenchKit. The archive (as well as the remaining information on the physical machine) is structured as shown in Figure 5. It contains three directories:

- `node_`⟨*i*⟩`_CONFIGURATIONS`: the information was initially intended for debug purpose but it remains since it can be useful to understand how your programs behave on the virtual machine. It basically provides the list of commands executed to operate your tool for a given configuration (one file per test × examination processed on this node),
- `node_`⟨*i*⟩`_CSV`: it contains two types of CSV files summarizing the executions performed on the remote node (see Figure 6).

  files `summary-`⟨*tool*⟩`.csv` provide a summary of the executions performed for *tool* on the node. Each line reports a run in the following format (column per column):

  - the name of the tool (in order to merge these files later),
  - the name of the test,
  - the name of the examination performed on the test,
  - the maximum amount of memory used during the execution (percentage),
  - the average percentage of CPU used during the execution,
  - the time required to execute your program,
  - the status of the termination (normal or timeout when time confinement is reached),
  - a unique run identifier[12] that allows you to access additional data.

  files `run-`⟨*tool*⟩`-`⟨*test*⟩`-`⟨*runId*⟩`_n_`⟨*i*⟩`.csv` provide, for a given run identifier, regular samples of CPU and memory consumption (as a percentage). Each line contains three columns:

  - time (seconds since 1970) of sampling,
  - CPU sampled at this time (percentage),
  - memory sampled at this time (percentage).

  Figure 6 shows examples of the data *BenchKit* produces.

[12] This identifier is unique, even considering an execution on several remote nodes.

```
$ ls -lh archive_node_1
total 0
0 drwxr-xr-x   4 xxx  xxx   136B 18 oct 13:27 node_1_CONFIGURATIONS/
0 drwxr-xr-x  42 xxx  xxx   1,4K 18 oct 13:50 node_1_CSV/
0 drwxr-xr-x  58 xxx  xxx   1,9K 18 oct 13:49 node_1_OUTPUTS/
```

Figure 5: Structure of the archive provided for a node (1 in our example)

```
tool,test_1,ltl,2.68151048738,17.5,7.23868918419,normal,135055571400033_n_1
tool,test_2,ltl,2.55138092947,16.1,6.51366186142,timeout,135055571400035_n_1
```

(a) Summary file

```
1350560133.4,2.18068239019,13.6
1350560133.98,2.18020601597,13.5
1350560134.57,2.23562421696,13.6
1350560135.17,2.17822112339,13.9
1350560135.76,2.18362003122,14.5
1350560136.35,2.66309068418,15.7
1350560136.91,2.72247867034,16.8
1350560137.5,2.81846807578,17.9
```

(b) Sampled CPU and time

Figure 6: Example of *BenchKit* CSV outputs

- `node_⟨i⟩_OUTPUTS`: there are two types of outputs. Your program execution traces (`stdout`) and the error traces (`stderr`). They respectively display what was displayed during the execution on the virtual machine on `stdout` and `stderr`.

  Figure 7 shows an example of execution capture for a tool. The trace is encapsulated into two tags: `START` and `STOP` that show the time (in seconds) measured at the beginning of the execution and the end of the execution. This allows to investigate more deeply in the run files to suppress the data sampled before your tool start.

  These files can be associated to an execution summary thanks to the unique run identifier. We highly recommend that the output of your tool is strictly formatted if you want to post-process your files in order to extract dedicated information.

Post-analysis. The CSV structure, as well as the identification of each run via its identifier, allows an easy post-processing of the collected data. The Figure 8 shows an example of chart generated with `gnuplot`[13] from the sampling files during the `model checking contest@Petri Nets 2012`. It is a quasi immediate use of the data collected by *BenchKit*.

[13] See `http://www.gnuplot.info`.

```
execution on node 1: cluster1u26.lip6.fr (runId=135055571500055\_n\_1)
========================================================================
runnning ExplDecMStrongSE05+ on anderson\_4\_F003 (ltl)
We got on stdout:
Probing ssh
Waiting ssh to respond
Ssh up and responding
sleep 20
========================================================================
 Generated by BenchKit 1.0

 Test is anderson\_4\_F003, examination is ltl
========================================================================

--------------------
content from stdout:

START 1350557182
FORMULA properties VIOLATED Explicite SE05\_OPT,VIOLATED,BA,0,21,21,22,"1U("P\_1.p3"|F("P\_1.p2"<->FG"P\_0.p1"))"
STOP 1350557183

--------------------
content from stderr:
```

Figure 7: Example of the execution capture from a tool

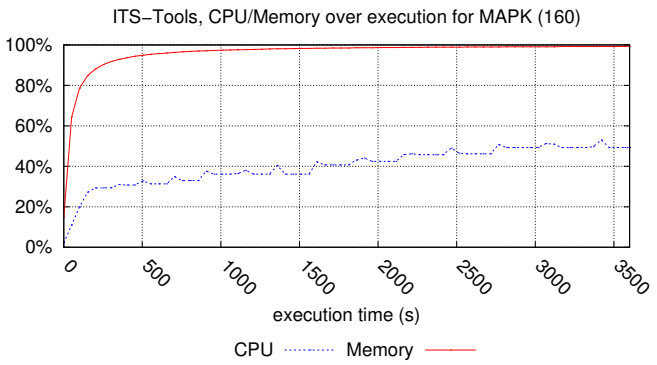


Figure 8: Example of chart generated for the `Model Checking Contest@Petri Net` in 2012

References

[1] F. Kordon, A. Linard, D. Buchs, M. Colange, S. Evangelista, L. Fronc, L-M. Hillah, N. Lohmann, E. Paviot-Adet, F. Pommereau, C. Rohr, Y. Thierry-Mieg, and K. Wolf. Raw Report on the Model Checking Contest at Petri Nets 2012. Technical report, CoRR, September 2012.

[2] F. Kordon, A. Linard, D. Buchs, M. Colange, S. Evangelista, K. Lampka, N. Lohmann, E. Paviot-Adet, Y. Thierry-Mieg, and H. Wimmel. Report on the Model Checking Contest at Petri Nets 2011. *Transactions on Petri Nets and Other Models of Concurrency (ToPNoC)*, VI:169–196, march 2012.

A An Example

We provide here the default configuration for a simple example. It is based on the configuration to be used for the Model Checking Contest @ Petri Nets 2013.

The `configuration.sh` file. It specifies the main elements where data are taken by *BenchKit* and results sent by *BenchKit*. Here, confinement is 5 minutes and 2GByte.

```
################################################################################
# This is the configuration file of the benchmark kit
################################################################################

#################################################################
# Benchkit (not touch please)
BENCHKIT_VERSION="1.0"

#################################################################
# General configuration
EMAIL_GLOBAL_REPORT="Fabrice.Kordon@lip6.fr" # empty if no global report by email (they are in the output_directory)
EMAIL_RUN_REPORT="Fabrice.Kordon@lip6.fr" # empty if no execution report by email (they are in the output_directory)
TIME_CONFINMENT="300" # duration in seconds
MEM_CONFINMENT="2048" # memory in megaByte

#################################################################
# On the origin machine (where you start everything)
export TOOL_LIST_FILE="tool_list.txt" # the file containing the list of tools/examination to be processed
export TEST_LIST_FILE="inputs_list.txt" # the file containing the list of inputs to be processed
export REMOTE_MACHINES_LIST_FILE="rmachine_list.txt" # the list of remote nodes

#################################################################
# On the execution machines (where you operate runs)
export BENCHKIT_DIR="BenchKitSources" # source directory in the physical machines executing the remote nodes
export OUTPUT_DIR="/data1" # directory to put execution outputs
export VM_DIRECTORY="/data1" # directory where the disk images for virtual machines are located
export VM_LOGIN="mcc" # the login used on the virtual machine to execute tyour software.
```

The `input_list.txt` file. this file (name can be parameterized in `configuration.sh`) specifies the inputs for beanchmarking. Each input will correspond to a directory in the `$HOME/BenchKit/INPUTS` directory where the data required for the benchmark will be located. We only provide here the first 25 lines of this file (for the MCC'2013, it is about 250 lines length)

```
## this is a comment (starting with #)
##############################
# each line represent one input
CSRepetitions-COL-025
CSRepetitions-COL-049
CSRepetitions-COL-100
CSRepetitions-COL-225
CSRepetitions-COL-400
CSRepetitions-COL-625
CSRepetitions-COL-900
CSRepetitions-PT-025
CSRepetitions-PT-049
CSRepetitions-PT-100
CSRepetitions-PT-225
CSRepetitions-PT-400
CSRepetitions-PT-625
CSRepetitions-PT-900
Dekker-PT-02
Dekker-PT-05
Dekker-PT-10
Dekker-PT-15
Dekker-PT-20
Dekker-PT-50
DrinkVendingMachine-PT-001
Echo-PT-d02r09
...
```

The `tool_list.txt` file. this file (name can be parameterized in `configuration.sh`) associates examination, tools and images disk for virtual machines.

Here, we consider 3 tools: `MyTool1` and `MyTool2` that are installed on the same VM and `MyTool3` that is installed on another image disk. Examinations identifiers are *StateSpace* and *ltl*. Transmitted value of the environment variable `BK_EXAMINATION`.

```
# this is a comment (starting wtith #)
##############################
# each field is separated by a :
# field 1: Examination name
# field 2: name of the tool
# field 3:associated VM disk image
StateSpace:MyTool1:mcc-2013.vmdk
ltl:MyTool1:mcc-2013.vmdk
StateSpace:MyTool2:mcc-2013.vmdk
ltl:MyTool2:mcc-2013.vmdk
StateSpace:MyTool3:mcc2-2013.vmdk
ltl:MyTool3:mcc2-2013.vmdk
```

The `machine_list.txt` file. this file (name can be parameterized in `configuration.sh`) lists the machines associated to a virtual node[14].

Here, we consider two hosts sharing a unique login. It is advised that connection to these machines are handled b a pair of public/private keys.

```
# this is a comment (starting wtith #)
##############################
# each field is separated by a :
# specify number of machines

NB_REMOTE_NODES:2

#specify machines one by one with the following format:
# field 1: number of the machine description (between 1 and the value specified with NB_MACHINE)
# field 2: hostname
# field 3: login for that host (assume ssh key allows to log without requesting a password)

1:cluster1u26.lip6.fr:fkordon
2:cluster1u27.lip6.fr:fkordon
```

The `BenchKit_head.sh` file. this file is the one located on the image Disk to make a link between *BenchKit* and the software to be evaluated.

```
#!/bin/bash
################################################################################
# This is an example for the MCC'2013
################################################################################
# In this script, you will affect values to your tool in order to let
# BenchKit launch it in an apropriate way.
################################################################################

# BK_EXAMINATION: it is a string that identifies your "examination"
# BK_INPUT: it is a string that identifies your test (used to build the name of the directory where you execute)

export PATH="$PATH:/home/mcc/BenchKit/bin/"

case "$BK_EXAMINATION" in
   "StateSpace") /home/mcc/BenchKit/bin/my_tool.sh -gen
      ;;
   "ltl") /home/mcc/BenchKit/bin/my_tool.sh -ltl
      ;;
   *) echo "$0: Wrong invocation:" >> $BK_LOG_FILE
      exit 1
      ;;
esac
```

[14] Remind that this version of *BenchKit* does not handle several virtual nodes to be run on the same physical host yet.

Fabrice Kordon
Fabrice.Kordon@lip6.fr

B Version History

version	date	comment
$\beta 1$	Feb. 5, 2013	First version to be published (only one virtual machine per remote host) in the context of the Model Checking Contest @ Petri Nets 2013