Using Quicktrace to collect runtime execution traces easily and automatically

Vincent Nelis

Luís Miguel Pinho

CISTER-TR-151208

Using Quicktrace to collect runtime execution traces easily and automatically

Vincent Nelis, Luís Miguel Pinho

CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

http://www.cister.isep.ipp.pt

Abstract

Using Quicktrace to collect runtime execution traces easily and automatically

Vincent Nelis
CISTER/INESC-TEC, ISEP
Porto, Portugal
Email: nelis@isep.ipp.pt

Luís Miguel Pinho
CISTER/INESC-TEC, ISEP
Porto, Portugal
Email: lmp@isep.ipp.pt

In many application domains, it is an elementary step in the design of a software, typically before its deployment, to exercise parts of its functionality by running some of its code on the target platform and collect informations about its runtime behaviour. Those informations may be used for debugging purpose or to assess the responsiveness of the application for example.

Taking measurements and collecting runtime data and is typically a tedious process that comprises many smaller but more specific tasks such as, e.g. (1) upload the source code to be tested on the target device, (2) compile the code remotely, (3) parametrize the device and set up the execution conditions, (4) run the compiled program to be analysed, (5) download the collected information (called *traces* hereafter), (6) process and analyse those traces, and (7) display the results of the analysis. Those are just a few example steps that are typically found as part of a testing process.

Besides the tremendous time that it takes to execute all these subtasks, it is also very difficult to automatize the whole process, mostly because all those steps are generally implemented using different programming languages. For example it is not uncommon that all the remote operations such as compiling, linking, and running the program to be analysed are basic shell scripts run on the target device through SSH, whereas the scripts used to analyse and display the traces are typically written in Matlab, R, or any human-friendly language capable of generating nice-looking plots. As a result, each time new measurements are needed it is required to *manually* re-execute the whole chain of subtasks, which may be very tedious, error-prone, and time-consuming.

This paper introduces and show-cases our new and innovative tool called "Quicktrace". Quicktrace has been developed in the scope of the Eurpoean FP7 STREP project P-SOCRATES[1] to ease the process of collecting runtime execution traces. However, the tool can be used for many other purposes as it offers a generic interface to easily implement, connect, and run together a set of scripts. Those scripts may be written in different programming languages and yet they can share common variables. Quicktrace is based on three main concepts: **commands**, **actions**, and **variables**.

[1]The P-SOCRATES Consortium, P-SOCRATES (Parallel Software Framework for Time-Critical Many-core Systems): http://p-socrates.eu

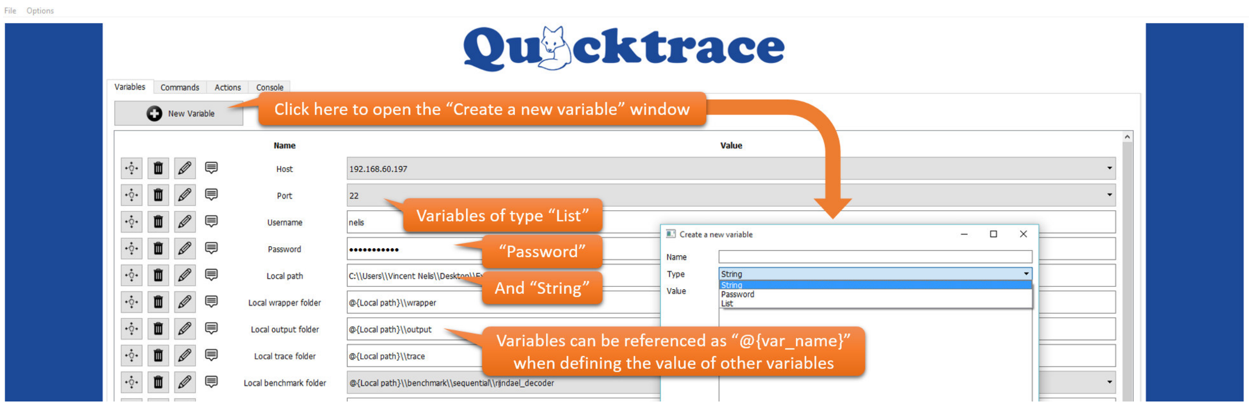
1) The **commands** are the basic blocks of the tool. A command is defined in a specific programming language and has a pre-defined type. A command is typically a small script that is used to perform a very specific task and its type describes how it must be executed. For example, one can define the commands "Upload the source code to the testing device" or "Compile the code remotely". The former is of type "SFTP – Put" and its code lists the files to be uploaded (following a pre-defined syntax), whereas the latter can be a Shell script of type "SSH - Remote command[s]". The current version of the tool supports scripts written in R as well as in any shell script supported by the machine running the script. The next version will provide support for running scripts written in python and matlab. Files can be sent and received through SSH or SFTP and Shell scripts can be executed remotely through SSH. Figure 2 shows the "Commands" panel of the tool.

2) The **Variables** are defined by the user. They are simply characterised by a name, a type, and a value. Before a command is executed, Quicktrace performs a simple "search and replace" on the code of the command to replace every reference to a variable with its value. Therefore, variables can be used and accessed by every command, irrespective of its programming language, simply by referring to it as "@{variable_name}" in the command's code. Figure 1 shows the "Variables" panel of the tool.

3) The **Actions** are the mean to connect the commands together. They are defined by a set of commands and executing an action simply runs all its commands in the order defined by the user. Note that Quicktrace also provides special control-flow commands that allow to implement loops and if-statements. Those control-flow commands are kept relatively simple as the ambition of the tool is (for now) not to design a new programming language. Figure 3 shows the "Actions" panel of the tool.

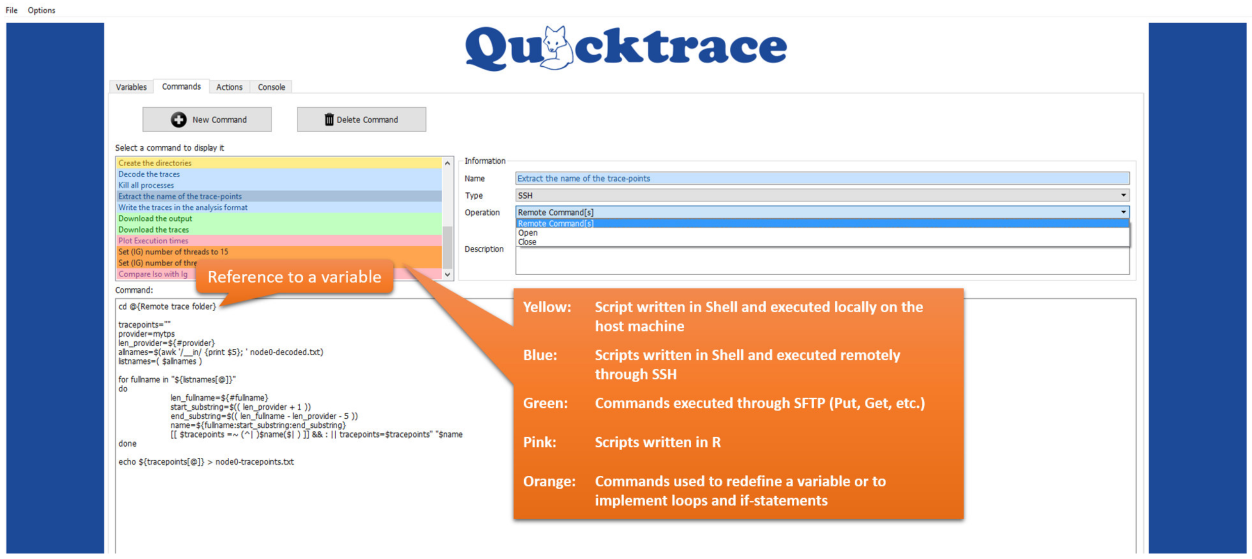
Acknowledgement

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER Research Centre); also by the European Union under the Seventh Framework Programme (FP7/2007-2013), grant agreement n° 611016 (P-SOCRATES).


File Options
Quicktrace
Variables Commands Actions Console
New Variable
Click here to open the "Create a new variable" window
Variables of type "List"
"Password"
And "String"
Variables can be referenced as "@{var_name}" when defining the value of other variables
Create a new variable
Name
Type
Value
String
Password
List

Name	Value
Host	192.168.60.197
Port	22
Username	nelis
Password	•••••••••••
Local path	C:\\Users\\Vincent Nelis\\Desktop\\E
Local wrapper folder	@{Local path}\\wrapper
Local output folder	@{Local path}\\output
Local trace folder	@{Local path}\\trace
Local benchmark folder	@{Local path}\\benchmark\\sequential\\rijndael_decoder

Fig. 1. In the "Variables" panel, the user can define all the variables that will then be shared by all the **commands**, irrespective of their programming language. Note that variables can be used within variables to allow for more flexibility.



Create the directories
Decode the traces
Kill all processes
Extract the name of the trace-points
Write the traces in the analysis format
Download the output
Download the traces
Plot Execution times
Set (IG) number of threads to 15
Set (IG) number of thre
Compare Iso with Ig

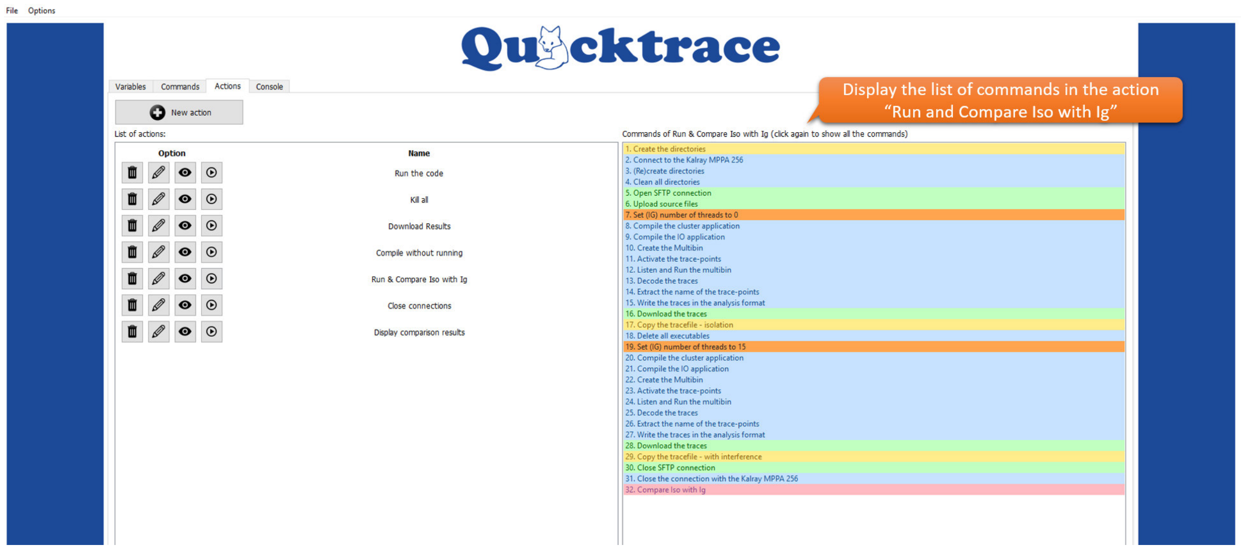
Name: Extract the name of the trace-points
Type: SSH
Operation: Remote Command[s]
Description:

Command:

```
cd @{Remote trace folder}

tracepoints=""
provider=mytps
len_provider=${#provider}
allnames=$(awk '/__in/ {print $5}; ' node0-decoded.txt)
listnames=( $allnames )

for fullname in "${listnames[@]}"
do
		len_fullname=${#fullname}
		start_substring=$(( len_provider + 1 ))
		end_substring=$(( len_fullname - len_provider - 5 ))
		name=${fullname:start_substring:end_substring}
		[[ $tracepoints =~ (^| )$name($| ) ]] && : || tracepoints=$tracepoints" "$name
done

echo ${tracepoints[@]} > node0-tracepoints.txt
```

Yellow:	**Script written in Shell and executed locally on the host machine**
Blue:	**Scripts written in Shell and executed remotely through SSH**
Green:	**Commands executed through SFTP (Put, Get, etc.)**
Pink:	**Scripts written in R**
Orange:	**Commands used to redefine a variable or to implement loops and if-statements**

Fig. 2. In the "Commands" panel, the user defines all the basic blocks/subtasks that will later be assembled to compose an **action**.


File Options
Quicktrace
Variables Commands Actions Console
New action
Display the list of commands in the action "Run and Compare Iso with Ig"

List of actions:

Option	Name
	Run the code
	Kill all
	Download Results
	Compile without running
	Run & Compare Iso with Ig
	Close connections
	Display comparison results

Commands of Run & Compare Iso with Ig (click again to show all the commands)

1. Create the directories
2. Connect to the Kalray MPPA 256
3. (Re)create directories
4. Clean all directories
5. Open SFTP connection
6. Upload source files
7. Set (IG) number of threads to 0
8. Compile the cluster application
9. Compile the IO application
10. Create the Multibin
11. Activate the trace-points
12. Listen and Run the multibin
13. Decode the traces
14. Extract the name of the trace-points
15. Write the traces in the analysis format
16. Download the traces
17. Copy the tracefile - isolation
18. Delete all executables
19. Set (IG) number of threads to 15
20. Compile the cluster application
21. Compile the IO application
22. Create the Multibin
23. Activate the trace-points
24. Listen and Run the multibin
25. Decode the traces
26. Extract the name of the trace-points
27. Write the traces in the analysis format
28. Download the traces
29. Copy the tracefile - with interference
30. Close SFTP connection
31. Close the connection with the Kalray MPPA 256
32. Compare Iso with Ig

Fig. 3. In the "Actions" panel, the user defines new **actions**, the set of commands that compose them, and how they are sequenced within each action.