GridTPT: a distributed platform for Theorem Prover Testing*

Thomas Bouton Diego Caminha B. de Oliveira David Déharbe
Pascal Fontaine
david@dimap.ufrn.br
[Thomas.Bouton,Diego.Caminha,Pascal.Fontaine]@loria.fr
Universidade Federal do Rio Grande do Norte, Brazil
LORIA, INRIA & Nancy University, Nancy, France

Abstract

Programming provers is a complex task; completeness or even soundness may often be broken by apparently harmless bugs. A good testing platform can contribute in detecting problems early and helping development. This paper presents GridTPT, the distributed platform for testing the veriT SMT solver. Its features are fairly standard, but it allows to easily distribute the task in a cluster.

We plan to make this platform available as an open source tool for the community of developers of automated theorem provers. This presentation to PAAR'2010 will provide the opportunity to discuss the need for such a tool and the necessary features in a broader context. We would like to extract a requirement specification from this discussion, that would be useful to get dedicated implementation resources for distribution, maintenance and future development of GridTPT.

1 Introduction

The implementation of efficient automated theorem provers requires intricate data structures and algorithms and is therefore error-prone. As a consequence, establishing the functional correctness of those tools includes applying large test suites, in addition to other measures such as third-party certification of intermediate and final results through e.g. proof generation and proof checking. The faster those verification results are available, the sooner mistakes are discovered and can be corrected by the developers. Also, automated theorem proving is intrinsically of a heuristic nature and requires experimenting with many different combinations of parameters. Again, this experimental study needs frequently applying large test suites.

Testing over a large number of benchmarks can easily be done in parallel (at least from a theoretical viewpoint). However, owning and maintaining a large cluster of machines is both time-consuming and financially expensive, and most prover developers do not have the resources to do this work in addition to research and implementation of the prover. Nevertheless, many research and university environments do have large clusters that are not fully used. More and more often these computing facilities are again clustered via a grid infrastructure that provides access to hundreds and even thousands of cores. It is often possible to obtain a low priority (i.e. when not in use for the financing projects) access to those clusters, and this low priority access will most of the time be suitable for the use of prover testing. Once the cluster is found, one needs to develop the software infrastructure for running the tests. Although a set of *ad hoc* scripts would do the basic job, a dedicated platform developed over a long period could provide many useful services.

Our goal is to share with the theorem proving community the software for a distributed testing platform for automated provers that we have built incrementally to support the development of the SMT solver veriT[1]. This software reduced the testing time from a week-end to a few minutes. For instance,

*This work is partly supported by the ANR project DECERT, and the INRIA-CNPq project SMT-SAVeS.

[1]For the development of veriT, we have been kindly granted access to a large grid infrastructure of INRIA known as Grid5000 [6].

the approximately nine thousand formulas in the categories for which veriT is complete are checked in 12 minutes with 80 cores and a 30 seconds time-out. As another example, the whole TPTP (around 14000 files) is run on E [12] in 20 minutes using 160 cores and a 30 seconds time-out. We plan to release this powerful and customizable platform under the open-source BSD license as well as offering maintenance to meet new requirements of external users. For theorem prover developers this would reduce the problem of having a good testing infrastructure to finding the cluster to run our software on.

There already exists platforms for evaluating solvers, and in particular, the platforms for the various annual competitions, for instance CASC [10, 13] and SMT-COMP [1, 2].[2] The main focus for those frameworks is to precisely and fairly measure the running time for the various solvers on the instances chosen for the competition. The purpose of GridTPT is different: it includes comparing versions of the same solver/prover but being precise in measuring running time is not the main objective. Much more importantly, the tool gathers statistics, and provides to the user ways to understand the tendencies and the relations between various quantities.

The platform is now stable and has reached a point where its use in a larger context, for slightly different goals, and in various environments, requires the feedback of the community, which we would like to get at PAAR. Following the presentation of GridTPT, we expect to get, from potential users, additional requirements to enhance and make the platform more attractive. Additionally, after we show the benefit of using the platform, we expect some of the participants will be interested in being users.

2 State of the platform

The testing platform has been used and improved to support the development of the SMT solver veriT [7] for more than a year. The test data used by the platform are the different categories of SMT-LIB [11, 3] benchmarks that are supported by the solver. The best way to run the tests and to access data is through the web interface, but the reports are in plain text, and all the scripts may be run from the command line.

Three types of tests can be performed over a selected set of benchmarks:

- functional test: the satisfiability status (satisfiable, unsatisfiable, or unknown), execution time (or failure, or time out) and other (user defined) statistics are gathered.
- consistency test: for each benchmark, the solver generates verification conditions corresponding to intermediate results. External solvers[3] are applied to recheck these conditions to ensure that not only the final result, but also the reasoning leading to this result is correct.
- memory test: memory leaks are detected using Valgrind (see `http://valgrind.org/`).

The latter two tests only generate a brief report to notify the developers if further debugging is required on particular benchmarks.

A prototype version of the script was sequential. An extensive test over the SMT-lib used to take several days to complete. The present version distributes the work over several multi-core computers, drastically reducing the total execution time to a few minutes. It uses a master/slave architecture, where one node assumes the role of the master, distributes the benchmarks and gathers the results, while the other nodes are slaves and execute the solver. It is fault tolerant: in case the connection to a slave is lost (due to a network failure, node hanging, ...), the full test is not affected. A test can be suspended at any time, and resumed later without significant duplicated work. Finally and most importantly, the

[2] Some organizations even give access to the clusters outside the competitions.

[3] In the case of veriT, we use CVC3 [4] to check intermediate conflict clauses and PicoSAT [5] to check the overall Boolean abstraction (MiniSAT [9] is used internally).

framework has been written to be easily portable: its implementation language is Python with a few OS-specific scripts written in bash.

Competitions usually distribute one process per computer, to eliminate interferences between processes. This is required in order to accurately and fairly measure the running time of the various solvers on the competition benchmarks. Since we prefer to get short testing time, we allow to send one job for every core available on the cluster, even if these are on the same processor. This introduces some slight variations in running times, but this is an acceptable price to pay to divide the overall testing time by a factor of 8 (in our case). Our experiments show that, even so, times are measured quite accurately.

The statistics collected by the tool and their value are not hard coded, but rather gathered from the output of the prover. They should be prefixed with a configurable character string – so that these statistics can be recognized from irrelevant information, such as an execution trace – followed by the name of the statistic and its value. Similarly, error messages need to be prefixed by a definable string. Notice that the statistics should at least provide the result of the prover on the formula. The execution time can be computed by the command *time* (on *nix systems). Figure 1 presents a typical output from veriT. Obviously, it is easy to put the information in the required form without modifying the internals of the prover by simply using a shell script wrapper.

```
verit 200907 - the veriT solver (UFRN/LORIA).
[...]
STAT_DESC: clauses: Number of clauses generated
STAT_DESC: res: 0 (UNSAT), 1 (SAT), -1 (UNKNOWN)
STAT_DESC: nodes: Number of nodes in the input formula as a DAG representation
STAT_DESC: nodes_tree: Number of nodes in the input formula as a tree representation
STAT_DESC: atoms: Number of atoms in the input formula as a tree representation
STAT_DESC: total_time: Total time
STAT: clauses=1486
STAT: res=0
STAT: nodes=799
STAT: nodes_tree=4114
STAT: atoms=1825
STAT: total_time=1.204
[...]
```

Figure 1: A typical output from veriT.

New tests are triggered automatically by cron jobs (if no test exists for the current version in the subversion repository), or manually through the developer-only section on the website of the solver. In the latter case, the tester has the opportunity to choose the solver revision, the solver options, the list of benchmarks on which the solver is to be run, and an optional comment. Reports are automatically generated and can be consulted on line, via the project website. The access to the reports is restricted to developers only. Other available features include the capacity to compare either graphically or textually two functional reports and to extract CSV (comma-separated values format) files for reports or for comparison of reports in order to do more sophisticated treatments using other tools (such as spreadsheets). Some of these features are demonstrated in the next section.

3 Illustration

This section contains illustrative information on the following capacities of the platform: functional report, a textual comparison report, and a graphical comparison report.

3.1 Report example

An extract of a sample test report is given below:

```
-veriT report----------------
Date            : 20090904133605
-informations----------------
Host name       : Grid5000
Number of cores : 80
CPU type        : xeon-harpertown at 2.5GHz
Executable      : ./verit
Build time      : 20090903181241
Options         : --enable-simp --enable-unit-simp --cnf-p-definitional -v
Number of files : 8965
CPU limit       : 30s
-grid statistics-------------
Cumulative time  : 876m (14h)
Total time       : 12m
Theoretical time : 11m
-legend-----------------------
total_time: Total time
res: 0 (UNSAT), 1 (SAT), -1 (UNKNOWN)
atoms: Number of atoms in the input formula as a tree representation
nodes_tree: Number of nodes in the input formula as a tree representation
clauses: Number of clauses generated
nodes: Number of nodes in the input formula as a DAG representation
-summary----------------------
Total number of benchmarks : 8965
Number of success : 7638
     between 0 and 5 sec : 6835
     between 5 and 10 sec : 465
     between 10 and 15 sec : 199
     between 15 and 20 sec : 89
     between 20 and 25 sec : 36
     between 25 and 30 sec : 14
Number of "CPU time limit exceeded" : 1327
-data-------------------------
Name                                                    total_time res       atoms  nodes_tree clauses  nodes
QF_IDL/Averest/binary_search/BinarySearch_live_bgmc000.smt        0.000   1      207793      208901       0    339
QF_IDL/Averest/binary_search/BinarySearch_live_bgmc002.smt        0.000   1      415523      417695       0    607
QF_IDL/Averest/binary_search/BinarySearch_live_bgmc003.smt        0.000   0      623253      626489       0    875
QF_IDL/Averest/binary_search/BinarySearch_live_blmc000.smt        0.000   1      623314      626594       1    942
QF_IDL/Averest/binary_search/BinarySearch_live_blmc002.smt        0.004   0     1246566     1253082       0   1814
QF_IDL/Averest/binary_search/BinarySearch_safe_bgmc000.smt        0.000   0         406         546       0    203
QF_IDL/Averest/binary_search/BinarySearch_safe_bgmc001.smt        0.000   1          99         175       0    109
QF_IDL/Averest/binary_search/BinarySearch_safe_bgmc002.smt        0.000   0      208393      209661       0    550
QF_IDL/Averest/binary_search/BinarySearch_safe_bgmc003.smt        0.000   1      416380      418776       2    897
QF_IDL/Averest/binary_search/BinarySearch_safe_blmc000.smt        0.000   0      416266      418750       0   1008
QF_IDL/Averest/binary_search/BinarySearch_safe_blmc001.smt        0.004   1      831718      836406       8   1329
QF_IDL/Averest/binary_search/BinarySearch_safe_blmc002.smt        0.004   1     1247479     1254435       3   1950
[...]
```

The presentation of the report may need to be adapted for other solvers. However, as mentioned above, the list of statistics is not hardcoded, and is built during the parsing of the output of the solver, assuming that it follows some formatting instructions.

3.2 Textual comparison

The comparison tool has the following parameters:

- the two functional reports to be compared;
- the categories of benchmarks to compare the reports on;
- the minimum spread, in percent of execution time, for the benchmark to be shown;
- the minimum spread, in absolute execution time, for the benchmark to be shown.

If the two reports are on different sets of benchmarks, only the common subset is shown. The statistics from both reports are shown. Optionally, the comparison tool may hide benchmarks for which running time are not sufficiently different. The spread between the execution times must then be higher than a specified percentage and a minimum value. Indeed, for benchmarks solved very quickly, 0.01 second is twice as fast as 0.02, but the running time difference may still be considered negligible.

Here is a small excerpt of a comparison report:

Name	total time		result	
	20090729191611	20080729142114	20090729191611	20080729142114
QF_UFIDL/pete3/bug_file3.smt	0.800	Failed	1	Failed
QF_UFIDL/pete3/bug_file4.smt	198.404	Failed	1	Failed
QF_UFIDL/pete3/bug_file5.smt	25.286	4.75	1	1
QF_UFIDL/uclid/22s.smt	0.332	0.58	0	0
QF_UFIDL/uclid/43s.smt	6.604	2.28	0	0
QF_UFIDL/uclid/cache.inv10.smt	3.536	5.46	0	0
QF_UFIDL/uclid/cache.inv14.smt	105.999	Failed	0	Failed
QF_UFIDL/uclid/cache.inv8.smt	0.652	1.08	0	0
QF_UFIDL/uclid/elf.rf10.smt	18.149	25.23	0	0
QF_UFIDL/uclid/elf.rf8.smt	0.320	0.62	0	0
QF_UFIDL/uclid/elf.rf9.smt	2.536	3.38	0	0
QF_UFIDL/uclid/ooo.rf10.smt	25.942	Failed	0	Failed
QF_UFIDL/uclid/ooo.rf8.smt	1.208	1.56	0	0
QF_UFIDL/uclid/ooo.tag10.smt	3.736	5.02	0	0
QF_UFIDL/uclid/ooo.tag12.smt	39.114	Failed	0	Failed
QF_UFIDL/uclid/q2.12.smt	15.201	19.58	0	0
QF_UFIDL/uclid/q2.14.smt	77.169	Failed	0	Failed
QF_UFIDL/uclid2/bug1.smt	9.721	13.57	1	1
QF_UFIDL/uclid2/bug2.smt	0.780	1.45	1	1
QF_UFIDL/uclid2/ooo.rf11.smt	158.894	Failed	0	Failed

On the web page, for each benchmark, the color code explicitly highlights improvement or regression.

3.3 Graphical comparison

Usually, on large sets of benchmarks, a graphical comparison helps to highlight the difference in execution time between two revisions of the solver. The web interface also displays an XY logarithmic graph (see figure 2). Again, more in-depth analysis may be done very quickly using the CSV data extraction facility and using a spreadsheet.

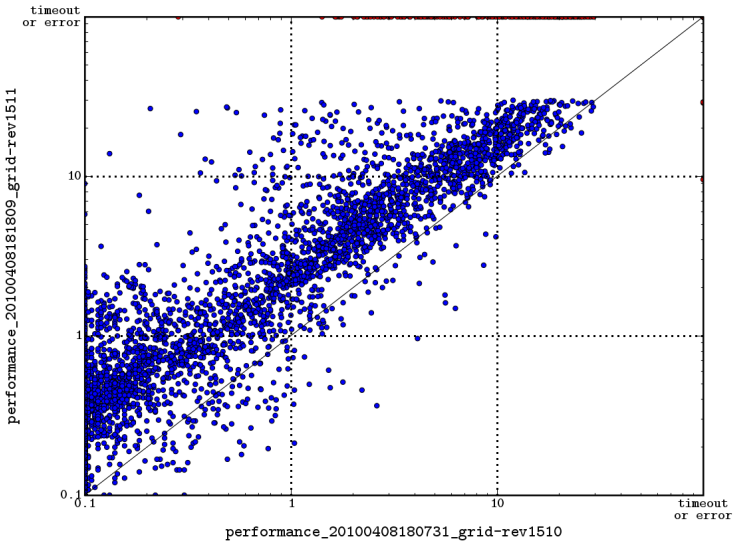

Figure 2: Example of a graphical comparison. A blue dot corresponds to one benchmark.

4 Conclusion and future work

We presented GridTPT, the testing platform used daily in the development of the veriT SMT solver. Since most prover developers have the same kind of needs for such a platform, we feel that this work may benefit other groups in the ATP community. The platform was used internally on several third-party SMT solvers. First positive experiments were also carried on with a first-order theorem prover (namely, the E prover [12]).

The most important difficulty we encountered in integrating the platform into our web server (for easy access by developers outside our institution) is the access policy to the cluster and the electronic security policy of our institution. The cluster is strongly firewalled, whereas the web server is outside the protected area; sending jobs and getting back information from the cluster to the web server requires hacks and ssh bounces. We believe that users elsewhere may encounter similar problems. Unfortunately, this prevents to have a clean package and an easy installation procedure that would work out-of-the-box for all cases. It will be necessary to collect and provide off-the-shelf solutions that will allow to circumvent those problems semi-automatically, when an automatic installation is not suitable. Another issue that we will certainly have to face is the variety of tools that clusters use for reserving resources.

The variation in running times with respect to the computer architecture is not linear, since it depends on the instruction set of the processors, the frequencies, the cache sizes and management policy,... that affect differently the running time depending on the program and even on the input data. Nevertheless, we think that it may be useful to investigate some kind of architecture calibration, i.e. recompute an approximation of the running time on a reference architecture. The motivations for such a calibration are twofold. First it would then be possible to use heterogeneous clusters if precise time measurement is not required. Second, it would also allow developers to compare old results (on out-of-use architectures) with newer ones.

Among the ongoing works, we are currently integrating the fuzzing tools for SMT-lib [8] on the distributed architecture. We also have a prototype of an interface to better visualize the differences in running time, with respect to the kind of benchmarks (categorized by their directory, subdirectory, and name prefix). The new version of the SMT-LIB [3] brings a novelty in allowing scripts in the prover input language; finding the right way to nicely integrate testing for scripts will also be necessary.

Acknowledgments: We would like to thank Stephan Merz for his guidance. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see `https://www.grid5000.fr`). We also thank the anonymous reviewers for their comments.

References

[1] C. Barrett, L. de Moura, and A. Stump. SMT-COMP: Satisfiability Modulo Theories Competition. In K. Etessami and S. K. Rajamani, editors, *Computer Aided Verification (CAV)*, volume 3576 of *Lecture Notes in Computer Science*, pages 20–23. Springer, 2005.

[2] C. Barrett, M. Deters, A. Oliveras, and A. Stump. Design and results of the 3rd annual satisfiability modulo theories competition (SMT-COMP 2007). *International Journal on Artificial Intelligence Tools*, 17(4):569–606, 2008.

[3] C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB standard : Version 2.0. First official release of Version 2.0 of the SMT-LIB standard., 2010. See also `http://www.smtlib.org/`.

[4] C. Barrett and C. Tinelli. CVC3. In W. Damm and H. Hermanns, editors, *Computer Aided Verification (CAV)*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer, 2007.

[5] A. Biere. PicoSAT essentials. *JSAT*, 4(2-4):75–97, 2008.

[6] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lantéri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, Nov. 2006. See also `https://www.grid5000.fr`.

[7] T. Bouton, D. C. B. de Oliveira, D. Déharbe, and P. Fontaine. veriT: an open, trustable and efficient SMT-solver. In R. Schmidt, editor, *Proc. Conference on Automated Deduction (CADE)*, volume 5663 of *Lecture Notes in Computer Science*, pages 151–156, Montreal, Canada, 2009. Springer.

[8] R. Brummayer and A. Biere. Fuzzing and delta-debugging SMT solvers. In *SMT '09: Proceedings of the 7th International Workshop on Satisfiability Modulo Theories*, pages 1–5, New York, NY, USA, 2009. ACM.

[9] N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[10] F. J. Pelletier, G. Sutcliffe, and C. B. Suttner. The Development of CASC. *AI Communications*, 15(2-3):79–90, 2002.

[11] S. Ranise and C. Tinelli. The SMT-LIB standard : Version 1.2, Aug. 2006. See also `http://www.smtlib.org/`.

[12] S. Schulz. System Description: E 0.81. In D. Basin and M. Rusinowitch, editors, *Proc. of the 2nd IJCAR, Cork, Ireland*, volume 3097 of *LNAI*, pages 223–228. Springer, 2004.

[13] G. Sutcliffe and C. Suttner. The State of CASC. *AI Communications*, 19(1):35–48, 2006.