Journal on Satisfiability, Boolean Modeling and Computation 7 (2010) 77–82

Experiment design and administration for computer clusters for SAT-solvers (EDACC)

SYSTEM DESCRIPTION

Adrian Balint adrian.balint@uni-ulm.de
Daniel Gall daniel.gall@uni-ulm.de
Gregor Kapler gregor.kapler@uni-ulm.de
Robert Retz robert.retz@uni-ulm.de
Institute of Theoretical Computer Science
Faculty of Engineering and Computer Science
Ulm University, Germany

Abstract

The design of a SAT-solver or the modification of an existing one is always followed by a phase of intensive testing of the solver on a benchmark of instances. This task can be very time consuming even when using multi-core computers or computer clusters. To speed up this process we designed EDACC. A system capable of managing solvers with their parameters, instances, creating experiment jobs and running them on arbitrary multi-core systems. After loading solvers and instances into the system the user is able to configure the parameters of the solvers and choose the instances for the experiment and the settings of the runs. Then EDACC generates the jobs for the experiment and stores them in a database. The user only has to start the EDACC-clients on the target computing system. Each client will load jobs from the database and try to use the full computing capacity of the system to accomplish its tasks. The client will also monitor the resources of the solvers such as time and memory. When a job is finished the client will write the results back to the database and choose another job until all jobs are finished. The clients can be distributed on a grid to increase the throughput. System crashes will not affect the results because the results are saved to a remote database. The user only has to restart the client. With the help of this system, testing and comparing solvers can be done much faster and much more efficiently.

KEYWORDS: *SAT-solver testing, design of tests, computer clusters, jobs scheduling*

Submitted March 2010; revised May 2010; published August 2010

1. Introduction

The progress made by SAT-solvers in the last years is quite remarkable as we can see from the results of the SAT Competitions [4]. The size of the problems that are in reach of modern SAT-solvers increases every year. This progress can be seen as the result of the development of better heuristics. The new heuristics are rather complex and have more parameters when compared to the ones a couple of years ago. The advantage of being better tunable is also their disadvantage because the testing and tuning phase gets longer. If the running time of a solver is a random variable then the test-set increases further.

For example testing the solver Sparrow [1] on a set of 104 instances, each instance being solved 100 times resulted in 10400 jobs with run-times between a couple of seconds and 1200 seconds (time-out). Running this test on a quad-core computer would have taken half a year. On a computer cluster it took about a week to finish all jobs. The problem that now arises is to equally distribute the jobs on the computer cluster to get maximal throughput. A popular method is to write shell scripts which start only a part of the jobs and then to submit these scripts as jobs on the computer cluster. We cannot submit each of our jobs to a computer cluster because no queuing system is able to handle so many jobs for one user. Without knowing the runtime of the solver on a given instance a priori (which is always the case for probabilistic solvers) we can not provide an optimal splitting of the jobs in shell scripts. Another problem that arises when dealing with such a high number of jobs is the management of the results that have to be analyzed.

To overcome these problems we have designed and implemented a system called EDACC (**E**xperiment **D**esign and **A**dministration for **C**omputer **C**lusters). The system can be split into three main parts: the database (DB), the graphical user interface (GUI) and the client. The DB is able to store solvers with their parameters, instances and experiments. Experiments represent a selection of solvers with a given parameter setting, a selection of instances and a configuration such as maximum time, maximum memory and number of runs. The DB also contains information about the computer cluster queues and resources. Section 3 describes the DB in detail.

All information from the DB can be managed with a GUI, which also facilitates the design of experiments and the analysis of the results. The user can select solvers, configure their parameters, then choose the instances and generate the jobs. Details about the GUI can be found in Section 4.

The last part of the system is the client. The client, when started on a computer or computer cluster, first analyzes the hardware configuration of the system, then loads jobs from the DB, starts them and monitors them on the target system. The client is designed to maximize the use of all resources. It will try to assign jobs to all free CPU's. When jobs are finished the results are written back to the DB. Crashes of the computing system will not affect the experiment because the client can be restarted and broken jobs will be recomputed. Details about the client are presented in section 5.

2. EDACC

From an abstract point of view EDACC can be seen as a management system for solvers, instances and results and as a queuing system that can be run on top of other arbitrary queuing systems, being almost independent of them. The user only has to provide some generic scripts for the queuing system that runs on the target computer system. EDACC is released under the MIT license and is available at the project site [5]. Before describing the components of EDACC we will give a typical work-flow, which starts with the GUI and contains the following steps:

1. Load solvers into the DB and configure their parameters.
2. Load instances into the DB and optionally categorize them into classes.
3. Create a configuration for the computing system.

4. Create an experiment by:
   (a) Choosing the solvers.
   (b) Configuring their parameters.
   (c) Selecting instances.
   (d) Configuring properties of the runs.
5. Choose target computing systems.
6. Generate jobs for the experiment.
7. Generate a package with all needed files (solver binaries, instances, etc.).
8. Copy the package on the computing system and start/submit the clients.
9. Monitor or analyze the results of the jobs with the help of the GUI.
10. Export results for further analysis.

Through the rest of the paper, by a computing system we mean a computer, a computer cluster or a grid (with homogeneous architecture). An experiment is a set of jobs consisting of solvers, parameters, instances and run properties.

3. EDACC - Database

The DB can be hosted on an arbitrary MySQL server where the user has read and write access. To avoid file-system inconsistencies all files like solver binaries and the instances are saved within the DB. Files loaded or downloaded to the DB are always checked with the help of the MD5-sums. Next we will give a brief overview of the information that is saved in the DB.

The main entities saved in the DB are the solvers with their parameters, the instances that can be categorized into source or user-defined classes, the experiments and their results. An experiment is the cross product of a selection of solvers, parameters, with certain values and instances together with some further run-time information. An element of this cross product is called a job. The database also includes information about queues of different computer clusters. The complete DB-model can be downloaded from the project page [5].

4. EDACC - GUI

The GUI client is a Java Swing application and is therefore independent of the operating system. It requires the Java Runtime Environment in version 6. The GUI can be divided into two modes, the manage DB mode and the experiment mode.

The manage DB mode provides the functionality for loading, updating and deleting solvers and instances from the DB. It also manages the configuration of the solvers' parameters and the classification of instances in source and user-defined classes. Furthermore all solvers and instances can be exported to the file system with the help of the GUI .

For creating experiments the user has to switch to the experiment mode. This mode enables the user to choose the solvers and to configure their parameters. It is also possible to

choose a solver multiple times but with different parameter configurations. This facilitates the test for different parameter settings. Afterwards the instances for the experiment are selected. The selection of the instances is alleviated by the classification of the instances into classes and by an instance filter.

To complete the creation of an experiment, the user has to configure the experiment details like the number of runs per solver and instance or the automated generation of random seeds. From the given data, the GUI generates a list of all jobs for this experiment and a package which contains all chosen solvers, instances, the cluster client and some scripts. This package has to be copied on the computer cluster and has to be unpacked to start the experiment. On a grid the client has to be submitted to the queuing-system of the grid.

It is possible to edit running experiments by adding new instances or new solvers and then generating the additional jobs. In this case it is not necessary to generate a new package for the computer cluster, the client will download all missing files from the DB. An experiment browser enables the monitoring of the jobs and their results.

With the help of the GUI complex experiments are created within minutes enabling the user to start experiments much faster. Another great benefit of the GUI together with the DB is the distributed administration of solvers and instances. The management can be performed by many users at the same time using the same DB. Thereby every user can access a large collection of different solvers and instances from his colleagues. With the user privilege management of MySQL, complex requirements on user rights are still satisfied.

5. EDACC - Client

The client is written in C and makes heavy use of Linux built-in commands, therefore the operating system of the computer system has to be related to Linux. A client is able to handle one computer, hence when running an experiment on a computer cluster the client should be started on each node (this is done automatically by the queuing system of the cluster in most cases). The client will first read an experiment ID and the connection information for the MySQL-server from a configuration file. After connecting to the DB it will check if all needed binaries and instances are available on the computing system. If not it will download them from the DB. After that it will check how many cores the computer has and will try to start a free job on each core. The status of the job is changed to 'computing'. If a job is finished the client writes the output and the exit code of the solver in the DB together with the run time of the job and changes the status to finished. During the run of a job the client monitors its time- and memory-usage (at the moment this is done by the Linux build-in command *ulimit*, but will be replaced in the future by the runsolver program from the SAT Competition). If some specified limits for time and memory are exceeded, the job will be stopped and the status will be set to 'failed'. As long as there are 'free' jobs in the DB and not all cores are busy the client will start a new free job. At the moment the client does not use any scheduling policies. This is because the client is limited to compute all jobs of an experiment and in most cases the analysis of the results is only possible if all jobs are finished.

The client will assure a maximal throughput of jobs on the computing system independent of the queue policies using all available resources to a maximum. If this is not desired

(avoiding memory bottlenecks), the user has the possibility to reconfigure the client. The architecture of the client also facilitates distributed computing of an experiment. The user only has to start the client on the different components of the grid (Performance comparison on the basis of time is only possible if the grid is homogeneous). The clients will then connect to the same DB and process together jobs from the same experiment. To avoid connectivity problems between the DB-server and the computing nodes on grid-systems a MySQL-proxy can be started on the login node of the grid.

The system is in some sense fail-proof. Suppose that one of the computing nodes crashes, then the status of the jobs controlled by that client will remain with the status 'computing'. This problem is solved with the help of DB-scheduler. A scheduled trigger scans the DB searching for jobs that meet the following condition: $startTime + maxRunningTime < actualTime$. The status of this kind of jobs is then set to 'free' so that other clients can compute them.

6. Related Work

The GUI together with the DB can be seen as a management tool for SAT-solvers and instances, that can be used by different users. The SAT Competition Homepage [4] was one of the first efforts in this direction, enabling the download of a wide range of solvers and instances. The organizers of the SAT Competition have a system that has the same, or maybe even more capabilities than EDACC, also based on a DB. The disadvantage of their system is that it is not freely available, a deficit that we would like to overcome with EDACC.

The client together with the DB can be seen as a scheduling system that can be run on top of another scheduling system. A similar scheduling system is Condor [3]. Condor is able to manage jobs spread around different computers or grids. For each job the user has to write a script which is then submitted to the Condor system. This task of writing job-scripts is simplified in EDACC by the experiment mode, where the user can generate all of its jobs (independent of their complexity) for an experiment within a couple of minutes. EDACC also provides a GUI for the administration of the jobs - a feature that is not present in Condor.

7. Outlook

To facilitate a better analysis of the results of an experiment a graphical result analyzer is developed, which is able to represent the results in graphical form. Comparing the results of two solvers or the results of one solver with different parameters would then be very easy.

Another feature we plan to integrate is an on-line result browser like the one used in the SAT Competition which can be extended by the former mentioned graphical representation of results. This would enable the user to monitor ongoing experiments.

Comparing solvers on heterogeneous grid system based on the run-time is not possible. The ability to configure the parameters of a solver and the fact that jobs can be added during the run of an experiment enables the design of an automated parameter tuning feature. The user could specify the parameters of interest, the range and the granularity

and then the system would create the jobs needed for testing. Depending on the results of the jobs, new jobs will be created to explore the search space of the parameters.

To benefit from the capabilities of the Condor system, particularly *Condor-G*, it would be of great interest to design a plugin for the experiment mode that will be able to generate *Condor-G* jobs that can then be managed by the *Condor-G* system.

8. Acknowledgments

We would like to thank the bwGrid [2] project for providing the test environment. We would also like to thank the students Borislav Junk and Raffael Bild for implementing the code of the client and Daniel Diepold and Simon Gerber for implementing the experiment mode.

References

[1] A. Balint and A. Fröhlich. Improving stochastic local search for SAT with a new probability distribution. In *SAT '10: Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing*. Lecture Notes in Computer Science **6175**, pages 10–15, 2010.

[2] bwGRiD (http://www.bw-grid.de), member of the German D-Grid initiative, funded by the Ministry for Education and Research (Bundesministerium für Bildung und Forschung) and the Ministry for Science, Research and Arts Baden-Württemberg (Ministerium für Wissenschaft, Forschung und Kunst Baden-Württemberg)

[3] D. Thain, T. Tannenbaum, and M. Livny. Distributed Computing in Practice: The Condor Experience. *Concurrency and Computation: Practice and Experience* **17**(2-4):323–356, 2005.

[4] The SAT Competition Homepage: http://www.satcompetition.org

[5] Homepage of the project : http://sourceforge.net/projects/edacc/