EDACC
User Guide
version 0.1

Abstract

We present the main capabilities of EDACC and describe how to use EDACC for managing solvers and instances, create experiments with them, launch them on different computer clusters, monitor them and then analyze the results.

disclaimer!

Contents

1 Outline

Here we will have an overview of this user guide specifying where the user can find what!

2 Introduction

Specification of the problems algorithm engineers are often confronted with!

Algorithm engineering: When designing and implementing algorithms one is at the end of the process confronted with the problem of evaluatating the implementation on the targeted problem set. As the authors of *EDACCare familiar with algorithms for the satisfiability problem we will take this sort of algorithms as examples. After designing and implementing a SAT solver we would like to see how this performs on a set of instance problems (let us suppose that our solver is an implementation of a stochastic one i. e.,the result of the solver on the same instance will be a random variable).*

Normally we would start our solver on each instance and record the runtime or some quality measure. This is a sequential process and could be easily performed with the help of a simple shell script. But there are some questions that have to be answered before starting this evaluation process.

1. *How long is the solver allowed to compute on one instance?*
2. *In the case of randomized solvers, how often do we call the solver on each problem set?*
3. *Do we limit the resources used by the solver (i. e.,maximum of memory, maximum stack size)?*

What is the task?
What are the tools?
What would we like to achieve?

Description of two example scenarios:

Example 1: *See how the performance of a deterministic solver differs from other solvers!*

Example 2:

2.1 What is EDACC

Describe the main tasks of EDACC, what is it for, what problems can be solved with EDACC?

2.2 System Requirements

2.3 Getting started

Components of EDACC

DB: *Here we are going to describe the data that EDACC uses.*

UI: *How to work with EDACC.*

Client:

→ Chapter 4 *For more details about the requirements of the client and the command line see 4*

3 User Interface

4 Client

4.1 Introduction

EDACC (Experiment Design and Analysis for Computer Clusters) is a software platform for the experimental analysis of algorithms. It consists of multiple components: a MySQL database, a Java GUI application, a web frontend and the computation client (also simply called client in this document). The GUI application can be used to create experiments that consist of a set of solvers that should run on a set of problem instances multiple times each. This results in computation tasks that are also called jobs in EDACC. All instance and solver binaries as well as the experiments and their results are stored in the database. To compute the jobs the computation client is used. Usually there have to be a lot of jobs computed to evaluate some experiments and since they are independent from each other, this task can be parallelized across many CPU cores. The computation client can be started on arbitrarily many machines and will manage the available CPUs and fetch and compute jobs from the created experiments. It connects to the central database and downloads all required ressources such as instances and solver binaries and writes back the results to the database.

4.2 System requirements

System requirements client:

The client is written in C/C++ and should be able to run on most Linux distributions where a MySQL C connector library is available. Because the central storage location for all required computational ressources, experiment metadata and results is a MySQL database, the client has to be able to establish a connection to the machine that hosts the database.

TCP/IP Connections:

This means that the machines where the client runs on have to be able to establish a TCP/IP connection to the database machine.

no TCP/IP connection:

The client was mainly tested on the bwGRID[1]*, a distributed computer cluster that consists of several hundred nodes at several physical locations at universities of Baden-Wrttemberg, Germany. Even though the machine hardware is homogenous, the network topology of bwGRID is not. In cases where direct network access from the computation nodes back to the database server is not possible it is usually possible to tunnel a connection over the cluster's login node back to the database via SSH.*

Other than that, the client has to be able to write temporary files to some location on the filesystem. This can be configured if it differs from the client binary location.

Shared file systems:

Because the client will download missing solver binaries and instances and upload results it also needs a reasonably fast network connection to the database. Shared file systems can considerably reduce the required bandwidth since every file is only downloaded once. As alternative you can pre-package the required files and put them into the correct folders before starting the client. However, if you modify experiments while the client is running it will still download missing files.

[1] http://www.bw-grid.de/

4.3 Usage

4.3.1 Configuration

Configuration file:

After designing experiments and creating jobs using the GUI application the client has to be configured. This is done by some command line arguments and a simple configuration file (called "config"). In the configuration file you have to specify the database connection details and which hardware the client runs on. This is done by configuring so called "grid queues" in the GUI application. They contain some basic information about the computation hardware such as number of CPUs per machine. The client will then use this information to run as many parallel jobs as the grid queue information allows it on each machine where it is launched. Here is a sample configuration file:

Example 3:

```
host = database.host.foo.com
username = dbusername
password = dbpassword
database = dbname
gridqueue = 3
verifier = ./verifiers/SAT
```

Note that the gridqueue is simply the ID of the grid queue. Another (optional) configuration option is the verifier line. It tells the client if it should run a program on the output that a solver generated. For example, if your experiments consist of attempting to solve boolean propositional logic formulas you can use a SAT verifier that tests if the solution given by a solver is actually correct. The verifier configuration option is simply a path to an executable that the client calls. See section 4.4 for more information on verifiers.

Command line arguments:

Beside the configuration file there are several command line options the client accepts, please also see "./client –help":

```
-v <verbosity>: integer value between 0 and 4 (from lowest to highest verb
-l: if flag is set, the log output is written to a
file instead of stdout.
-w <wait for jobs time (s)>: how long the client should
wait for jobs after it didn't get any new jobs before exiting.
-i <handle workers interval ms>: how long the client should
wait after handling workers and before looking for a new job.
-k: whether to keep the solver and watcher output files or
to delete them after uploading to the DB.
```

Verbosity controls the amount of log output the client generates. A value of 4 is only useful for debugging purposes, a value of 0 will make the client log important messages and all errors.

If the "l" flag is set, log output goes to a file whose name includes the hostname and IP address the client runs on to avoid name clashes in shared filesystems typically found in computer clusters. Otherwise log output goes to standard output.

With the "-w" option you can tell the client how long to wait before exiting after it didn't start any jobs. This can be useful to keep the client running and ready to process new jobs while you evaluate preliminary results and add new jobs or whole experiments. The wait option is also used to determine how long attempts should be made to reconnect to the database after connection losses. The default value is 10 seconds.

The "-i" option controls how long the client should wait between its main processing loop iterations. If this value is low, it will more often look for new jobs when there are unused CPUs. For maximum job throughput this value should be lower than the average job processing time but lower values will also put more strain on the database and increase the client's CPU usage. The default value is 100ms which should work fine in most cases. The client will also adapt to situations where there are free CPUs but no more jobs and increase the interval internally and fall back to the configured value once it got another job.

The "-k" flag tells the client if it should keep temporary job output files after a job is finished of if it should delete them. The default is to delete them.

4.3.2 Launching

After configuration you can simply run the client on your computation machines. On computer clusters there are often queuing systems that you have to use to gain access to the nodes. On bwGRID for example, we could use the following short PBS (portable batch script) and submit our client to 10 nodes with 8 cores per node:

```
#!/bin/sh
#PBS -l walltime=10:00:00
#PBS -l nodes=10:ppn=8:bwgrid
cd /path/to/shared/fs/with/client/executable
./client -v0 -l -i200 -w120
```

You should always run the client from within its directory (i.e. cd to the directory) to avoid problems with relative paths such as the verifier path from the example configuration above.

As soon as clients start you should be able to see jobs changing their status from "not started" to "running" in the GUI's or Web frontend's job browsers.

4.3.3 Troubleshooting

If errors or failures occur the client will always attempt to shut down cleanly, that is stop all running jobs and set their status to "client crashed" and write the last lines of its log output as "launcher output" to each job. This can fail when network connections fail or the client receives a SIGKILL signal causing it to exit immediately. In case of network failures you should still be able to find useful information in the client's logfile on the local filesystem.

4.4 Verifiers

Verifiers are programs that the client runs after a job finishes. Verifiers are getting passed the instance of the job and the solver output as arguments and are supposed to exit with a status code that indicates if the solution the solver gave for the instance is correct or other result descriptions. This code will be written to the database as "result code" and "verifier exit code". Any output the verifier writes to standard out will be written as "verifier output". The call specification for a verifier binary looks like this:

```
./verifier_binary <path_to_instance> <path_to_solver_output>
```

We provide a verifier for the SAT problem that works on CNF instances in DIMACS format and solvers that adhere to a certain output format (see the source code). Note that you have to make sure that your possible result codes are specified in the ResultCodes table in the database before running clients or there will be errors when the client tries to write results. By convention, the web frontend and GUI application consider status codes that begin with a decimal "1" as correct answers.

4.5 Experiment priorization

EDACC experiments can be marked as inactive and individual jobs can be prioritized. Only jobs of active experiments with priority equal to or greater than 0 are considered for processing by the client. Futhermore, experiments can be assigned a priority. The clients will then try to match the relative number of CPUs working on an experiment with its relative priority to all other experiments that are assigned to the same grid queue. For example, if you have three experiments with priorities 100, 200 and 300 respectively and a total number of 100 CPUs managed by some running clients, the clients will try to have 16% of CPUs working on the first, 33% of CPUs working on the second and 50% of CPUs working on the third experiment.

! → *The client is running solely on Unix and is not distributed yet for Windows systems.*

5 Web Frontend

6 Monitor

7 Troubleshooting

8 Glossary

contains definitions of the most used terms within this document. solver instance parameter algorithm deterministic vs. stochastic las vegas vs. monte carlo

Index