Reproducible Cluster(less) Speedup Analysis

Maximilian Heisinger[1] and Martina Seidl[1]

Johannes Kepler University Linz, Austria,
`{maximilian.heisinger, martina.seidl}@jku.at`

1 Introduction and Context

When developing new libraries or improving on old implementations, the aim generally lies on outperforming the solutions realized in the past. This improvement of performance may be captured in multiple ways, be it increasing number of solved instances, decreasing wall-clock run-time, or similar measures. In this work we share our improved benchmarking setup to cover as many scenarios as possible with one combined scripting framework, called the Simsala Script Collection, which could also be described as self-hosted and primarily CLI-focused StarExec. We use this setup in our distributed Cube-and-Conquer SAT and QBF solver research project Paracooba and in our upcoming generic solver interfacing library QuAPI. We hope that the developed methods and tools can be helpful to other groups as-well, both during development of new solvers and when organizing competitions. In the workshop, we will interactively go through some usage scenarios. A public release of our scripts is available at:

http://simsala.pages.sai.jku.at/ (website with tutorials)
https://gitlab.sai.jku.at/simsala/simsala (repository)

2 Executing and Benchmarking Eval-Tools

In the following, we define an eval-tool to be a general benchmarking program that can be run on sets of input files, printing one row of tabular data for each one. Inputs may be generated or taken from a benchmark library. The tool may directly execute different configurations and generate a combined output, or be exclusive to one configuration.

Scripted Eval-Tool Execution In order to realize such an eval-tool, an execution environment is needed. Multiple possibilities exist, e.g. slurm [8] or local batch jobs using e.g. GNU parallel [6]. To create an extendable benchmarking pipeline, our `submit.pl` script generates tasks, which then execute the eval-tool with different input-files on either a slurm cluster or on regular (unmanaged) workstations using GNU parallel. As a regular eval-tool enforces no resource limits, an additional wrapper for their enforcement is recommended, e.g. BenchExec [1] or RunLim [2]. For memory and wall-clock, we use a statically-linked RunLim. `submit.pl` itself may be compared to BenchExec, although supporting multiple execution backends.

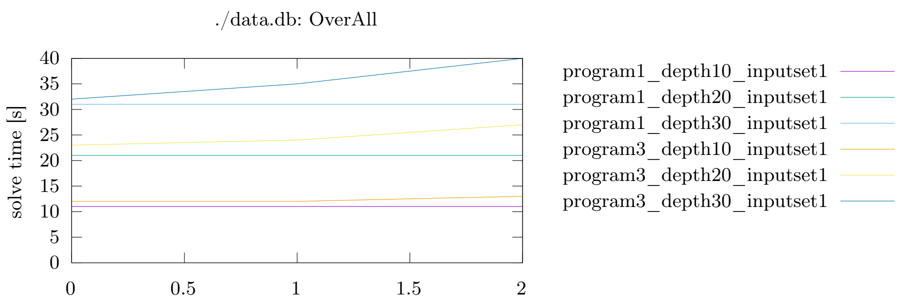

Fig. 1. Quick cactus plot of example data.

Extracting Structured Results As the generated logs share the same format, they can be analyzed using the same tools. This makes them more stable and incentivizes long-term investments. We provide a script to quickly extract wall-clock run-times and other statistics from the logs, inserting them into an SQLite3 [4] database. Such a database then contains all the different configurations of the used eval-tool, which can all be compared with per-file granularity. This database is the crucial improvement compared to the old setup based purely on ad-hoc log parsing, as the structured data can now be analyzed and compared directly.

3 Analysing Results

To get an overview over the data we use SQLitebrowser [5], which also provides convenient in-built plotting. More specialized comparisons, e.g. cactus plots as in Figure 1, are already provided as a ready-made script. Custom-built plots are then based on SQL queries written by the user, e.g. to calculate and visualize speedups between different configurations. To store a growing library of such queries, we use Org Mode [3] for its support of inline (and parameterized) SQL statements. The resulting files may then also be exported and shared as PDF or HTML documents, both providing results their source queries.

Combining Gnuplot with SQLite and Publishing In order to publish the results, we use Gnuplot [7] for its support of standalone LaTeX documents for plotting. By prepending `<` to the data file argument of a `plot` command in Gnuplot, it is interpreted as an executable to be run instead of a file to be read. When embedding SQLite3 queries into these commands, Gnuplot gathers all required data using these embedded queries. For this to work correctly, `set datafile separator '|'` has to be included the plotting commands.

As the number of different configurations and plotting scripts grows, the calls to Gnuplot themselves should be automated. For this we use a `Makefile` with pattern targets to both generate the TeX and PDF files, with the matched pattern (represented by the `$(*)` variable) being transformed into a Gnuplot argument. These plots can then be analyzed by themselves or directly included into a publication. All the produced data is then checked into a repository, which can then also be shared. Because every step of the pipeline is exchangeable and all steps taken are documented in the checked-in scripts, replication studies can work in the same setup and compare findings more easily.

Case Study We used our tools to track Paracooba & QuAPI step-by-step. Due to its distributed and parallel nature, especially the first required extensive tests and evaluation to understand the impact of changes and to monitor progress.

References

1. Beyer, D., Löwe, S., Wendler, P.: Reliable benchmarking: requirements and solutions. International Journal on Software Tools for Technology Transfer **21**, 1–29 (2017)
2. Biere, A., Jussila, T.: RunLim (2022), http://fmv.jku.at/runlim/
3. Dominik, C., et al.: Org Mode (2022), https://orgmode.org
4. Hipp, R.D., et al.: SQLite (2022), https://www.sqlite.org/index.html
5. Kleusberg, M., et al.: DB Browser for SQLite (2022), https://sqlitebrowser.org
6. Tange, O.: Gnu parallel - the command-line power tool. ;login: The USENIX Magazine **36**(1), 42–47 (Feb 2011), http://www.gnu.org/s/parallel
7. Williams, T., Kelley, C., many others: Gnuplot 5.4: an interactive plotting program (2022), http://gnuplot.sourceforge.net/
8. Yoo, A.B., Jette, M.A., Grondona, M.: Slurm: Simple linux utility for resource management. In: JSSPP (2003)