Adaptive Optimization of Very Large Join Queries

Reproducibility Submission Readme, SIGMOD 2018: mod066

Thomas Neumann (neumann@in.tum.de)
Bernhard Radke (radke@in.tum.de)

This readme describes how to reproduce the experimental results of the SIGMOD 2018 paper "Adaptive Optimization of Very Large Join Queries". The original paper is available at: https://dl.acm.org/citation.cfm?id=3183733.

TL;DR

1. download the ReproZip package from db.in.tum.de/~radke/mod066/repro.rpz
2. extract the ReproZip package:
   `reprounzip docker setup repro.rpz <setupdir>`
3. optionally, to run the "Other Systems" experiment for Figure 11 of the paper (see Section 3 for details):
   - download and extract db.in.tum.de/~radke/mod066/other.tar.xz
   - execute `make` in the extracted directory
   - upload the resulting `runtimes.csv` into the container: `reprounzip docker upload <setupdir> runtimes.csv:runtimes.csv`
4. run the experiments and generate the paper using the results of the experiments (20 runs in total): `reprounzip docker run <setupdir>`
5. fetch the generated paper and review the new data:
   `reprounzip docker download <setupdir> main.pdf`

1. Environment

We provide a ReproZip package containing the environment required to run the experiments (db.in.tum.de/~radke/mod066/repro.rpz). This package includes executables for the various experiments as well as the workloads used for evaluation. The experiments

for Figure 11 of the paper (Other Systems) use themselves docker containers and are therefore not included into the ReproZip package (see Section 3 on how to reproduce this experiment).

To create a docker container from the ReproZip package, issue `reprounzip docker setup <setupdir>`.

2. Experiments

The following experiments are carried out by the ReproZip package:

- measurement of optimization time and normalized costs for standard benchmarks
  binary: `bin/dp-master`; input files: `queries/<benchmark>/*`
- measurement of optimization time and normalized costs for randomly generated tree queries
  binary: `bin/dp-master`; input files: `queries/generated/*`
- measurement of plan quality under different cost models
  binaries: `bin/dp-{seeking-truth,index}`; input files: `queries/generated/*`
- measurement of normalized true costs for randomly generated tree queries with noise on cardinality estimates
  binary: `bin/dp-noise`; input files: `queries/generated/*`
- measurement of speedups achieved for the adaptive optimization framework using the various bitset implementations described in the paper
  binary: `bin/dp-{128,var,sparse}`; input files: `queries/generated/*`
- measurement of speedup of GOO and IKKBZ using the join lookup table
  binary: `bin/dp-joinlookup`; input files: `queries/generated/*`

To run the experiments, issue `reprounzip docker run <setupdir>`. This will run the experiments and generate a version of the paper (`main.pdf`) using the results of the performed experiments. There are 20 different runs in total, that each generate a plot or table. The last run (run 19) generates the complete paper. Issue `reprounzip docker download <setupdir> main.pdf` to download the newly generated paper as `mod066.pdf` into the current directory.

2.1. Details

There are 19 runs, each generating a different plot or table:

- Run 0 generates Figure 9 (times100.pdf)
- Run 1 generates Figure 10 (times1000.pdf)
- Run 2 generates Figure 11 (other.pdf, see Section 3)

- Run 3 generates Figure 12 (times5000.pdf)
- Run 4 generates Figure 13 (joinlookup.pdf)
- Run 5 generates Table 1 (benchmarks.tex)
- Run 6 generates Table 2 (costs-master.tex)
- Run 7 generates the inline table on the seeking the truth cost model in Section 6.4 (seek-summary.tex)
- Run 8 generates the inline table on the Cmm cost model in Section 6.4 (index-summary.tex)
- Run 9 generates Table 3 (card-err-costs.tex)
- Run 10 generates the inline table on the quality of linearized DP in Section 6.7 (linearized-summary.tex)
- Run 11 generates Table 4 (bitset-summary.tex)
- Run 12 generates Table 5 (times-details-10.tex)
- Run 13 generates Table 6 (times-details-40.tex)
- Run 14 generates Table 7 (times-details-100.tex)
- Run 15 generates Table 8 (times-details-1000.tex)
- Run 16 generates Table 9 (times-details-5000.tex)
- Run 17 generates Table 10 (costs-seek.tex)
- Run 18 generates Table 11 (costs-index.tex)
- Run 19 generates the full paper (main.pdf)

Note that the individual runs are independent from each other. Each run will either reuse the raw data from previous runs or run the required experiments, if they have not been run yet. If you would like to run experiments again, the raw data of the experiments has to be removed by clearing the container (`reprounzip docker reset <setupdir>`). Further note that, due to gurobi licensing issues, we cannot provide an experiment that reproduces the results of the MILP optimization. Therefore, there will be no numbers of the MILP approach in Tables 1 and 2.

2.1.1. Runtime of the Experiments

Running all 20 reprounzip runs on a 12 core Intel(R) Core(TM) i7-6850K CPU @ 3.60GHz with 64 GB of RAM took around 17 hours.

2.1.2. Note on normalized costs

As we calculate the normalized costs for each plan relative to the best known plan for this query, there may be slight variations in the normalized costs that result from a reprozip run. Depending on which algorithm runs into timeout during the experiment, an algorithm that did not finish in the original experiment may finish in time and give a better solution than the successful algorithms in the original experiment. Thus, normalized costs of other algorithms for this query will become worse. The same could of course happen the other way around: an algorithm that gave the best known solution in the original experiment does not finish in the reproducibility run. In this case, the normalized costs of this query will decrease for the other algorithms.

3. Other Systems (optional)

The experiment that lead to Figure 11 of the paper requires to run these other database systems. We have setup shell scripts, Makefiles and sql code (schema as well as queries) to create docker containers for PostgreSQL, MS SQL-Server and IBM DB2 and let these systems optimize the queries. As the systems themselves are run in docker containers, this experiment is not part of the main ReproZip package but can be downloaded from db.in.tum.de/~radke/mod066/other.tar.xz.

3.1. Environment

The experiment has to be run on a linux machine and requires bash, docker and make to be installed.

3.2. Running the Experiment

To run the experiments, extract the tarball and run `make` in the extracted directory. Optimization times are collected in `runtimes.csv`.

To upload the new measurements into the main ReproZip, issue `reprounzip docker upload <setupdir> <newly generated runtimes.csv>:runtimes.csv`. To generate Figure 11 with this new data, issue `reprounzip docker run <setupdir> 2`. To generate the complete paper again, incorporating the new data, issue `reprounzip docker run <setupdir> 19`.

Running this experiment on a 12 core Intel(R) Core(TM) i7-6850K CPU @ 3.60GHz with 64 GB of RAM took around 10 hours.

4. Plots and Tables

The following plots are generated by the ReproZip package:

- Figure 9: `plots/times100.pdf`
- Figure 10: `plots/times1000.pdf`

- Figure 11: `plots/other.pdf`
- Figure 12: `plots/times5000.pdf`
- Figure 13: `plots/joinlookup.pdf`

The following tables are generated by the ReproZip package:

- Table 1: `plots/benchmarks.tex`
- Table 2: `plots/costs-master.tex`
- the two inline tables in Section 6.4: `plots/{seek,index}-summary.tex`
- Table 3: `plots/card-err-costs.tex`
- the inline table in Section 6.7: `plots/linearized-summary.tex`
- Table 4: `plots/bitset-summary.tex`
- Tables 5-9: `plots/times-details-{10,40,100,1000,5000}.tex`
- Tables 10 and 11: `plots/costs-{seek,index}.tex`

5. Hardware

The numbers in the original paper were optained by running the experiments on a 4 socket Intel(R) Xeon(R) CPU E7-4870 v2 @ 2.30GHz with 15 cores per socket and 1 TB of main memory.

A. Detail on the executables

The binaries in `bin` are variants of the same program, each with slightly different implementation details:

- `dp-128`: always uses a 128 bit bitset for queries on up to 128 relations
- `dp-index`: uses the C_{mm} cost model
- `dp-joinlookup`: does not use the join lookup table
- `dp-master`: main implementation
- `dp-noise`: adds random noise to the cardinality estimates and investigates the true costs (based on the true cardinalities) of the resulting plans
- `dp-seeking-truth`: uses the "seeking the truth" cost model
- `dp-sparse`: always uses the sparse bitset

- `dp-var`: always uses the variable sized bitset

All these binaries can be run using the following commandline variants (examples can be seen in `queries/Makefile` and `Makefile`:

- `bin/dp-master <algorithm> <output-file> <query>`
  optimize the given `query` using `algorithm`. Write the result into `output-file`.
  `algorithm`: the optimization algorithm to run. One of: dphyp, dpsize, dpsizelinear, ikkbz, ikkbzbushy (linearizedDP), goo, goodp (IDP with dphyp as inner DP), goodp2 (IDP with linearizedDP as inner DP), quickpick, genetic, simplification, adaptive, minsel
  `output-file`: the file to write optimization time, plan costs and the final plan to
  `problem`: a json representation of the query graph to optimize (see our workload at `queries/<workload>/*` for examples)

- `bin/dp-master collecttimes <files>`
  extract the optimization time from `files`. Writes one line per query and algorithm to stdout
  `files`: result files from running `bin/dp-master <algorithm> ...` (filenames have to be of the form `<query>-<algorithm>`)

- `bin/dp-master collectcosts <files>`
  extract the normalized costs from `files`. Writes one line per query and algorithm to stdout. Costs are normalized to the costs of the best plan for the same query in files
  `files`: result files from running `bin/dp-master <algorithm> ...` (filenames have to be of the form `<query>-<algorithm>`)

- `bin/dp-master collectboth <files>`
  extract optimization times and normalized costs from `files`. Writes one line per query and algorithm to stdout.
  `files`: result files from running `bin/dp-master <algorithm> ...` (filenames have to be of the form `<query>-<algorithm>`)