TUNING PARAMETERS OF A MIXED INTEGER PROGRAMMING SOLVER IN THE CLOUD

Smirnov S. PhD.[1]
Institute for Information Transmission Problems of the Russian Academy of Sciences (Kharkevich Institute), Russia [1]
sasmir@gmail.com

Abstract: *We present a software system that tunes configuration parameters of an algorithm. Parameters are tuned to minimize the solving time for a set of problems. SCIP is a mixed integer programming solver developed at Zuse Institute Berlin. The solver has more than 1500 configuration parameters. Most of the parameters are related to the solution process, others apply to solver's input/output. There are both discrete and continuous parameters. Our system modifies parameters one by one to find ones having the most impact on the solving time. Then combinations of the best parameter values are evaluated. This approach implies that a great amount of solver runs is needed: 1-2 values of every parameter multiplied by the number of parameters multiplied by the number of test problems. Thus we employ a public cloud to create a temporary computational cluster for faster processing. The paper presents an overview of the system, a method to measure algorithm's performance in the cloud and numerical results of system's use on several problem sets.*

Keywords: ALGORITHMIC PARAMETER OPTIMIZATION, PARAMETER TUNING, CLOUD COMPUTING

1. Introduction

Growing number of Infrastructure as a service (IaaS) providers we observe today is a direct effect of computation costs getting cheaper and of infrastructure automatization levels getting higher. Cloud services make it possible to automate more programmer's work making him more productive. It may be considered as another step in continuous process of adding more abstraction levels to a computer system: high-level programming languages, interactive debugging, automatic build systems, etc. For developers it allows for rapidly creating development and test sandboxes, quickly provisioning virtual machines with needed software, testing load and scalability.

There are lots of problems that can be automated by the use of clouds. One of such problems is fine tuning an algorithm to make it work better in some sense, for example, faster. It can be done in many ways: modifying hard coded parameters inside a program, smart analysis of the program's source code, adjusting parameters inside configuration files of program's modules. In our study we have chosen the last described way: fine tuning configuration parameters for the SCIP (Solving Constraint Integer Programs) solver. SCIP [1] is currently one of the fastest non-commercial solvers for mixed integer programming (MIP) and mixed integer nonlinear programming (MINLP). It is also a framework for constraint integer programming and branch-cut-and-price. It allows for total control of the solution process and the access of detailed information down to the guts of the solver. Although SCIP is a very fast solver even with default parameters, it should be possible to fine tune the parameters for one's work. It is quite simple if there are a couple of parameters and not many test problems. However SCIP is very configurable having more than a thousand parameters. Apparently having such a large set of configuration parameters makes fine tuning it quite time consuming. That is why we tried to automate this process making a system choosing best configuration setting for a set of problem instances. Due to vast number of SCIP runs needed we had to use a cloud to make the process quick.

In this paper we use the following terminology. A program has configuration parameters controlling how it works. Every parameter has a value assigned. A set of parameter values is called the settings. When we run a solver it is given settings in form of a configuration file and a problem instance. The goal of our system may be thought as finding the settings yielding the shortest running time on a set of problem instances.

Other works on general optimization of algorithmic parameters include Selection Tool for Optimization Parameters (STOP) [2] based on intelligent sampling of settings throughout the space and OPAL framework [3] based on mesh adaptive direct search. The former tool works with a small set of parameters having discrete values. Our approach allows working with large numbers of parameters and their values.

2. Implementation

Let us begin with a brief overview of the system. One begins using it by specifying the number of computing hosts in the Vagrant's [4] configuration file and then starting the system by running init-virtualbox.sh or init-digitalocean.sh script. After a while one has a cluster of a master host and the specified number of slave hosts where Simple Linux Utility for Resource Management (SLURM) [5] and other essential software are installed and running. Then one connects to the master host by issuing vagrant ssh command where one can manage the system with optctl.py command.

Currently the settings optimization process consists of three phases: time check, big step and inter step. During the first phase every problem's instance is evaluated once on each computing node with default settings. The main goal of this step is to get an estimate of maximum time allowed for a problem instance to run until it's killed by SLURM. The big step phase is the most computationally intensive one. On this step huge number of settings with only one parameter different from defaults is evaluated. As a result the big step allows us to sort parameter values based on their impact on solving time. Next, on the inter step phase, four best parameter values from the head of the big step's sorted list are chosen and all their possible combinations are evaluated. After this step we have the best settings in terms of running time. This step is not very time consuming and can be repeated multiple times.

2.1. Measuring running time in the cloud

Measuring running time of a program in the cloud reliably is not very simple. Naive approach like wall-clock time or processor cycles are not reliable due to computer resources overcommit by a cloud provider. Depending on the load other virtual machines express on the hypervisor host, program's running time can change dramatically. There is a better approach: one can measure the number of instructions executed by the CPU while running the program. Of course, different instructions may need different numbers of cycles to complete so it may be hard to correlate running time to the number of instructions executed. Instruction count becomes handy when comparing performance the same program expresses with the same input but with different settings.

In x86 CPUs instructions can be counted in hardware by the Performance Monitoring Unit (PMU). One can use PAPI or perf tool to set up and access the hardware counters. Not every hypervisor supports PMU virtualization, e.g. VirtualBox does not. However modern KVM releases has such support.

In our system we used perf tool to measure user space instruction count which gives very stable results independent of the hypervisor host's load.

Here is a sample run of SCIP under perf-stat. Six runs of SCIP were made, average counter values and their standard deviations can be observed:

```
$ perf stat -r 6 -e cpu-clock,task-clock,\
cycles,instructions,instructions:u,\
instructions:k scipampl TSP_Uniform_50_10.nl

 Performance counter stats for 'scipampl
TSP_Uniform_50_10.nl' (6 runs):

        81154.175629 cpu-clock
        81154.063870 task-clock
     175,626,392,898 cycles
     267,235,503,611 instructions
     265,101,243,265 instructions:u
       2,134,260,346 instructions:k
        81.224668399 seconds time elapsed
```

Instaructions:u counter gives much more stable results than software counters or cycles counted in hardware.

Same single CPU virtual machine with two SCIP instances running simultaneously:

```
 Performance counter stats for 'scipampl
TSP_Uniform_50_10.nl':

        82580.457064 cpu-clock
        82579.334255 task-clock
     181,566,274,355 cycles
     267,300,821,128 instructions
     265,099,385,783 instructions:u
       2,201,435,345 instructions:k
       167.578326122 seconds time elapsed


 Performance counter stats for 'scipampl
TSP_Uniform_50_10.nl':

        82581.195083 cpu-clock
        82580.104484 task-clock
     181,589,302,031 cycles
     267,299,923,846 instructions
     265,099,381,995 instructions:u
       2,200,541,851 instructions:k
       167.589704033 seconds time elapsed
```

Again, instructions:u are much more accurate.

As we can see from this examples, instructions:u is the most stable event counter at least with SCIP. It even allows for running multiple solver instances simultaneously with acceptable timing accuracy.

3. Results and discussion

We have performed testing with two different sets of problem instances. One of the sets was tested with two version of SCIP: 3.0.2 and 3.1.0. Throughout the tests, 48 computing nodes with identical virtual machines were used.

First problem set consisted of ten randomly generated traveling salesman problem instances of the same size. SCIP 3.1.0 was used. Big step for this set consisted of 28810 jobs and took six hours and a half to complete while total CPU time consumed was 296 hours, as if 46 machines were used. After one inter step optimal settings were obtained. Second inter step showed no improvement. If we compare the sums of running times for default settings and for optimized ones, we observe 3x speedup with the latter (see Table 1). Optimized settings consisted of only one parameter value:

```
lp/scaling = FALSE
```

Table 1: *Traveling salesman problem, learning data set.*

Problem instance	Defaults, [sec]	Optimized, [sec]
TSP_Uniform_50_1	3,0	2,5
TSP_Uniform_50_2	10,4	6,4
TSP_Uniform_50_3	29,4	16,1
TSP_Uniform_50_4	6,8	8,7
TSP_Uniform_50_5	39,9	36,6

For testing purposes more TSP instances were generated and run with the same optimized parameters (see Table 2), the speedup is just 1,41x here.

Table 2: *Traveling salesman problem, control data set.*

Problem instance	Defaults, [sec]	Optimized, [sec]
TSP_Uniform_50_11	52,6	27,3
TSP_Uniform_50_12	11,5	9,2
TSP_Uniform_50_13	1,4	2,2
TSP_Uniform_50_14	7,0	5,2
TSP_Uniform_50_15	270,8	230,6
TSP_Uniform_50_16	64,0	10,5
TSP_Uniform_50_17	4,0	3,7
TSP_Uniform_50_18	27,1	14,4
TSP_Uniform_50_19	5,0	12,0

Second problem set consisted of five instances which solved quickly with SCIP 3.0.2 and very slowly with SCIP 3.1.0.

An attempt was made to find parameters making SCIP 3.1.0 working on the problem as good as 3.0.2. We took all parameters that changed their default values, were renamed or added in 3.1.0, which resulted in 186 parameters (against 1547 total parameters). One of the instances (w6_t19_test_8) was dropped after the time check phase due to hitting memory limit (512 MB RAM in VM). Big step consisted of 1260 jobs for the first four instances and took five hours and a half to complete. Total CPU time spent in SCIP was 237 hours which equals to 43 hosts working. After one inter step optimized settings were obtained. Second inter step showed no improvement. If we compare the sums of running times for default settings and for optimized ones, we observe 1,65x speedup when the latter is used. It should be noted that optimized settings also improved time for the problem w6_t19_test_8 that was not involved in the tests due to memory limitation. As we can see, our system was not able to find settings making SCIP 3.1.0 perform as good as SCIP 3.0.2 for this problem, however a noticeable speedup was obtained. Optimized settings:

```
heuristics/rins/minnodes = 25
lp/checkdualfeas = FALSE
lp/disablecutoff = 1
```

Table 3: *Load balancing problem solving times.*

Problem	SCIP 3.1.0, defaults, [sec]	SCIP 3.1.0, optimized, [sec]	SCIP 3.0.2, defaults,[sec]
w6_t15_test_4	5,95	3,11	1,97
w6_t18_test_4	586,18	186,8	70,3
w6_t19_test_4	1420,4	823,9	223,9
w6_t19_test_5	941,7	737,6	138,7
w6_t19_test_8	11596,3	7077,7	382,6

We also tried optimizing settings for this problem set in SCIP 3.0.2 on all its parameters. During big step 13200 jobs were run in 13 hours and a half, 594 hours were spent in the solver as if 44 hosts were working. After two inter steps optimized settings were

obtained, third interstep yielded no improvement. Here 1,26x speedup was obtained. Optimized settings after first inter step:

```
constraints/linear/upgrade/setppc = FALSE
lp/solvefreq = 0
```

After the second inter step:

```
constraints/linear/upgrade/setppc = FALSE
lp/solvefreq = 0
conflict/preferbinary = TRUE
heuristics/fracdiving/freqofs = 1
heuristics/veclendiving/freq = -1
```

4. Conclusion

As a result of the study the system described was made. It works in the cloud, uses Vagrant for virtual machine management, SLURM for batch job processing, Python [6] for automation and Virtualbox [7] for debugging. It was tested on a number of problem classes and noticeable speedup was shown.

It is possible to extend the system on other solvers e.g. CBC or Ipopt. Another possible improvement may be made by making the system accessible on the Web. It is also planned to publish the source code on GitHub after some cleanup.

In conclusion we expect that the service may become popular among SCIP users. Another conclusion is that cloud computing is very convenient and cheap nowadays which is definitely a good driver for developing new and nonconventional approaches.

5. References

[1] Tobias Achterberg, SCIP: solving constraint integer programs, Mathematical Programming Computation, volume 1, number 1, 2009, pp. 1–41.

[2] Baz, M., Hunsaker, B., Brooks, P., & Gosavi, A. Automated tuning of optimization software parameters. University of Pittsburgh Department of Industrial Engineering Technical Report, 7, 2007.

[3] Audet, C., Dang, K. C., & Orban, D. Optimization of algorithms with OPAL. Mathematical Programming Computation, 2012. pp. 1-22.

[4] Vagrant, http://www.vagrantup.com/

[5] Yoo, Andy B., Morris A. Jette, and Mark Grondona. "SLURM: Simple linux utility for resource management." Job Scheduling Strategies for Parallel Processing. Springer Berlin Heidelberg, 2003.

[6] Sanner, Michel F. "Python: a programming language for software integration and development." J Mol Graph Model 17.1, 1999, pp. 57-61.

[7] Oracle VM VirtualBox, https://www.virtualbox.org/