Distributed Solving of Mixed-Integer Programs with Coin-OR CBC and Thrift

Ramon Janssen, Jakob Görner
ramon.janssen@stud.hn.de,jakob.goerner@stud.hn.de

ABSTRACT

We present a distributed branch and bound solver for mixed-integer programming (MIP) problems. Our solver utilizes Coin-OR CBC for solving subproblems. Interprocess communication is achieved by using the remote procedure call framework Thrift. Our aim is to provide an easy to use and free of charge alternative to commercial solvers like CPLEX and Gurobi. Our work extends the approach presented in [**?**]

1 INTRODUCTION

2 ARCHITECTURE

3 DETERMINISM

3.1 Motivation to use Determinism

Non-deterministic behaviour in the context of parallel branch and bound is the possibility that different (but valid) search paths are explored at different times due to exogenous effects like high occupancy of a machine where a worker is running. Providing a deterministic implementation is very important in practice, cause the expectation of most customers is the same result in approximately the same time for multiple times solving the same problem under the same configuration. Especially returning the same result is important to prevent customers losing confidence into the software.

3.2 Implementation

We compared three implementations to reach determinism. In the beginning we used a simple approach using a Barrier which synchronised the worker after each job. Problem with this a approach is that the perfomance drops to 40 percent of the Non-deterministic solution. We use a timeout between 10-25 seconds until a job has to be finished which causes a lot of idle time for worker solving jobs much faster than the timeout. This explains the huge perfomance drop.

```
void worker(){
   ...
   processJob();
   if(barrier()) //returns true for the last thread leaving
        the barrier
      processResultsOfFinishedJobs();
   barrier();
   ...
}
```

To avoid the idl time we implemented a second solution. In the second solution we let the worker solve all subproblems we created so far but the resulst where not processed until the job list is empty. After the job list is empty we processed the results and added new jobs to the job list. This worked well for smaller problems but for bigger problems the programm timed out without solving the problem. Turns out that the jobs where processed in manner that is close to an breadth-first search. Only differences between a breadth-first search is that we priorize the execution order of a level of the search tree. This can be fast for some problems but in general the execution time is much higher. [**?**]

Finaly we combined both approaches and defined a small number $n = 4 \cdot workerCount$ and allowed the worker to solve n Jobs before being synchronised by the barrier. Also we don't process the results of the n Jobs until all of them are processed by a worker. With this solution we was able to avoid most of the idle time and the priorization of jobs is not manipulated any more.

4 EVALUATION

Setup We evaluated our solution on a cluster of 16 Machines of the same type. The Machines are equipped with a Intel(R) Core(TM) i7-8700 CPU with a clock frequenzy of 3.20 GHz and 16 GB of RAM. The RAM is running with a clock frequency of 2666 MHz. The Machines are connected with Ethernet to a Switch. All solvers ran in opportunistic mode.

Debian 10 is the installed operation system on all machines. We use Thrift Version 0.13.0, Coin-OR CBC Version 2.10.5 and GLPK Version 4.65.

As test data we used several problems from the miplib 2003 and 2010 test set which contain various types of problems and are made to benchmark MIP Solvers [**?**]. Also we used several problems from the OR-Library which is a set of multidimensional knapsack problems [**?**].

A major problem when assessing MIP solvers is that there are many sources of variability that can have an impact on the measuring results. The two major factors causing variability are Non-deterministic behaviour and perfomance variability [**?**]. Perfomance variability arises through different but mathematical equivalent inputs causing differnet execution paths. For example we could permute the rows and columns of the constraint matrix which could lead to changes in the branching order without changing the problem itself. Non-deterministic behaviour can be caused when jobs finishing in different order from run to run caused by external effects like other programms running on the same machine as a worker. This can have critical effect on the branching order which results in varying execution times and also can have an impact on the results itself. To address this problems we repeated execution of a problem several times and calculated the geometric mean which is more stable against statistic outliers. We calculate the geoemtric mean as follows:

$$G(N) = (\prod_{k=1}^{n}(x_k + s))^{\frac{1}{n}} - s$$

$$N := \{x_1, x_2, ..., x_n\}$$

Table 1: Solve duration on various instances. We report the geometric mean of the solve duration over multiple trials in seconds.

Instance	GLPK 4x4	CBC 4x4	CBC 8x4	Gurobi
OR30x100-0.25_9	1954	376	169	237
OR10x250-0.75_1	2633	273	136	209
OR10x250-0.75_2	> 3600	626	349	641
OR10x250-0.75_5	3259	697	253	230
OR10x250-0.75_9	1321	112	77	77
mas74	1915	106	61	73
danoint	530	747	403	309

Comparison to distributed GLPK We compared our CBC based solution with the original GLPK based solution. In most cases we observed a significant perfomance increase with the CBC based solution compared to the GLPK based one. There was some cases where the GLPK based solution outperfomed the CBC based solver.

Distributed CBC Speedup We analysed the speedup of our solution. To do so we compared the runtime for several problems from our test set on a setup with 4 machines - each running 4 worker - to an setup with 8 machines - also with 4 worker per machine. In other words we duplicated the amount of worker. The measurements did not give a clear result cause the speedup was dependend to the problem instance. Some problems scaled very good but some instances don't. By this fact it is hard to specify a overall speedup.

Comparison to Gurobi We compared our solution to Gurobi which is a state of the art solver. With Gurobi we solved problems from our test set on a machine similar to the one we used for our CBC based solver. Only difference is that the machine used by Gurobi has a i7-8700K CPU instead of the i7-8700 CPU. The i7-8700K CPU has a clock frequency of 3.7 GHz instead of 3.2 GHz. Gurobi used 12 Threads.

The results shows that in most cases we are faster than Gurobi when running the CBC based solution on 8 machines with 4 worker per machine. Due to the fact that Gurobi uses heuristics and pseudocost branching in a very efficent way, in some cases Gurobi outperformces our solution even if we use more than twice as much worker as Gurobi uses Threads.

CBC Multithreaded vs Distributed CBC We compared the execution time from CBC Console Application using muliple threads on the same machine with our solution using same amount of workers running in single threaded mode to evaluate if we should start multiple workers or one worker with multiple threads on one machine.

Suprisingly, the worker-based solution is often on par with the multi-threaded CBC solver. In Table 2 it can be seen that it is problem dependend if the multi threaded version is faster than the one with multiple worker running one thread. Therfore we decided to stick to the multiple single threaded worker per machine setup.

Impact of Determinism Finally we evaluated the impact of determinism. As allready mentioned in chapter Determinism the pure barrier solution without load balancing was very slow caused by lots of idle time which can be seen in Table 3. A perfomance drop up to 60 percent can be measured compared to the oppertunistic results. Scalibility also suffers from huge idle times.

Table 2: CBC Console Appilcation with multithreading and distributed CBC solver results

Instance	CBC 4 Thr.	CBC 8 Thr.	CBC 1x4
OR10x250-0.75_9	384	312	343
OR10x250-0.75_3	614	429	759
OR10x250-0.25_6	1765	1287	1307
mas74	311	241	324

Table 3: Solver running in deterministic mode on various instances using barrier after each job.

Instance	CBC 4x4 det	CBC 4x4	CBC 8x4 det	CBC 8x4
OR10x250-0.75_9	248	112	215	77
mas74	259	106	198	61

The barrier solution using load balancing is working much faster. Only x percent of the execution time is lost compared to non-deterministic results. We also evaluated which number of sub-problems should be solved bevor the results get processed and the worker are synchronised. Through experiments we found out that y jobs should be processed to improve performance as good as possible.

5 CONCLUSION