Engineering HordeSat Towards Malleability: mallob-mono in the SAT 2020 Cloud Track

Dominik Schreiber
Institute of Theoretical Informatics
Karlsruhe Institute of Technology
Karlsruhe, Germany
dominik.schreiber@kit.edu

***Abstract*—We briefly present the massively distributed SAT solver which we submit to the Cloud Track of the SAT Competition 2020, being the solver engine of a novel framework for massively parallel and distributed malleable job scheduling applied to SAT solving. Our solver is based on HordeSat; notable differences include completely asynchronous communication, a much more careful clause exchange, and some internal performance improvements.**

***Index Terms*—Parallel SAT solving, distributed SAT solving**

I. Introduction

In order to improve massively parallel problem solving "on demand" in a cloud context, we introduce malleability to parallel SAT solving as a part of a novel framework named "mallob" for massively parallel and distributed malleable job scheduling [1]. Malleability is the property of a computation to dynamically handle a varying amount of computational resources (i.e. cores or nodes) during its execution, opening up vast possibilities for performing highly dynamic load balancing on many jobs of varying demand and priority that run in parallel on some large-scale infrastructure.

However, the SAT competitions do not involve malleable computations nor solving multiple instances at the same time. As a consequence, we added a special configuration to our system for the sole purpose of solving a single instance with full computational power from the beginning and named it "mallob-mono" (mallob mono instance mode). In the following, we will describe the most relevant aspects of this solver engine and the surrounding architecture.

II. Overview

On each node we start one MPI process for each set of four available (virtual) CPUs such that each process can employ four solver threads. HordeSat [2] serves as a foundation for the solver engine residing on each process. Internally, we use Lingeling as a solver backend just like HordeSat's default configuration. However, we updated the used Lingeling version from *ayv* (2014) [3] to *bcj* (2018) [4]. We also updated the native diversification routines of Lingeling according to the diversification of the 2018 version of Plingeling. We let one out of 14 solvers in our portfolio perform local search (using YalSAT [4] as a backend) while the others are CDCL solvers with different set options.

We adjusted and replaced significant portions of the codebase of HordeSat in order to match the requirements of our malleable framework. As such, we enabled the suspension and resumption of particular solver instances, made all communication among the nodes completely asynchronous, and enabled descriptions of SAT formulae to be serialized and transferred directly over message passing instead of assuming that the formula resides on each node. Many of these changes are unimportant for the SAT competition. Some general performance improvements were integrated; for example, we reduce lots of unnecessary `getrusage` system calls by supplying a cheap and approximative time measuring callback over the Lingeling interface instead.

In the following we describe our clause exchange mechanism and the related clause filtering, which are the most prominent differences between HordeSat and our solver engine.

III. Clause Exchange

HordeSat initiates an All-to-all exchange of learnt clauses every second by a synchronous collective operation (`MPI_Allreduce`). The clause buffer size of each node is of fixed length 1500 and the entire buffer is sent around regardless of the degree to which it is filled. Duplicate clauses are detected by HordeSat's clause filters only after the full operation succeeded. If an exported local clause buffer is filled to less than 80%, one of the local solver threads is asked to increase its clause production. Unit clauses are are always shared and are exempt from being filtered. As a result, the first few clause exchanges are often flooded with large numbers of highly redundant unit clauses after first simplifications and preprocessing steps.

We have made the clause exchange entirely asynchronous while ensuring that one broadcast of a globally aggregated clause buffer takes place every second. We aggregate buffers of learnt clauses along a binary tree of all computing nodes. Clause buffers sent over this tree are always in compact shape, i.e., without any unused portions of memory. During the reduction, instead of just *concatenating* the buffers, inner nodes do a three-way *merge* of their local clauses and the clauses of their children, preferring short clauses and filtering out duplicates with an additional Bloom filter, a datastructure

Evaluation of this work was partially performed on the supercomputer ForHLR funded by the Ministry of Science, Research and the Arts Baden-Württemberg and by the Federal Ministry of Education and Research.

that we took from original HordeSat [2]. Thereby, we limit the maximum length $b(u)$ of a merged clause aggregation containing clauses from u nodes:

$$b(u) = \lceil u \cdot \alpha^{\log_2(u)} \cdot 1500 \rceil$$

Note that $\alpha = 0.5$ makes the length of a clause aggregation converge to 1500 the more nodes are involved, and $\alpha = 1.0$ makes the limit grow linearly in the number of nodes just like in HordeSat. We set $\alpha = 0.75$ to find a middle ground between these extremes.

Additionally, no clauses of length greater than five are shared. With this strict limitation we expect to avoid a lot of communication volume and internal work in the SAT solvers while still sharing lots of potentially interesting information among the solvers.

After the reduction reaches the binary tree's root node, the clause aggregation is broadcast through the tree to all other nodes and locally digested when appropriate.

IV. Clause Filtering

We also made some adjustments to HordeSat's clause filtering mechanic used when clauses are exported or imported. We added duplicate checking for unit clauses both to each clause filter and to our duplicate checking during the reduction. This check does not rely on Bloom filters but functions with exact hash sets, using one of the commutative hash functions that are employed in the Bloom filters. This way we do not get any false positives for unit clauses and make sure that each such clause is being shared at least once.

Last but not least, we implemented a mechanic similar to restarts into the clause filters. The authors of original HordeSat already intended to periodically clear clause filters in order to be able to share clauses after some time, but it was not implemented. We introduce a quite careful "forgetting" of shared clauses: Every five minutes, in one iteration over all set bits in the filter each bit is unset with probability $\sqrt[4]{0.5} \approx 15.91\%$. As every clause inserted into the filter sets four bits from four hash functions, the probability that a clause is forgotten is close to $P(\textit{forgotten}) = P(\geq \textit{1 bit unset}) = 1 - P(\textit{0 bits unset}) = (\sqrt[4]{0.5})^4 = 0.5$. For the unit clauses, every element in the explicit set is forgotten with probability 0.5. Overall, approximately half of all clauses are effectively forgotten and can be shared again.

V. License

Our system mallob and, by extension, our submitted solver is licensed under the GNU Lesser General Public License (LGPLv3). As the licensing of Lingeling was changed to MIT with the 2018 version, our system consists of fully Free Software.

VI. Conclusion

We described the central aspects of our massively parallel SAT solver and are excited to see how it performs in the AWS environment of the competition.

While our competitor does include some computational overhead due to its malleable job scheduling aspects, we still expect that our solver will overall outperform original HordeSat due to various improvements of the internal workings of the portfolio solver and notably the improved clause exchange.

Acknowledgment

Many thanks to Tomáš Balyo and Markus Iser for fruitful discussions regarding HordeSat and its clause exchange.

References

[1] P. Sanders and D. Schreiber, "Massively parallel malleable job scheduling for SAT solving." to be published.

[2] T. Balyo, P. Sanders, and C. Sinz, "Hordesat: A massively parallel portfolio SAT solver," in *International Conference on Theory and Applications of Satisfiability Testing*, pp. 156–172, Springer, 2015.

[3] A. Biere, "Yet another local search solver and Lingeling and friends entering the SAT competition 2014," *Sat competition*, vol. 2014, no. 2, p. 65, 2014.

[4] A. Biere, "CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT entering the SAT competition 2018," *Proc. of SAT Competition*, pp. 13–14, 2018.