The Journal of Open Source Software

Mallob: Scalable SAT Solving On Demand With Decentralized Job Scheduling

Peter Sanders [1] **and Dominik Schreiber** [1]

1 Karlsruhe Institute of Technology, Germany

DOI: 10.21105/joss.04591

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @ARMartinelli
- @massimotorquati

Submitted: 29 June 2022
Published: 23 July 2022

Summary

Propositional satisfiability (SAT) is the problem of finding a variable assignment for a given propositional formula (i.e., a composition of Boolean variables using logical operators NOT, AND, OR) such that the formula evaluates to **true**, or reporting that no such assignment exists. The platform **Mallob** (**Mal**leable **Lo**ad **B**alancer, or **Ma**ssively P**a**r**all**el **Lo**gic **B**ackend) enables processing SAT jobs in a (massively) parallel and distributed system on demand. Mallob's flexible, fair, and decentralized approach to online job scheduling results in scheduling latencies in the range of milliseconds, near-optimal system utilization, and high resource efficiency.

Statement of need

Despite SAT being a notoriously difficult problem (S. A. Cook, 1971), practically efficient SAT solving approaches have empowered a wide range of real-world applications for SAT such as software verification (Kleine-Büning et al., 2019), circuit design (Goldberg et al., 2001), cryptography (Massacci & Marraro, 2000), automated planning (Schreiber, 2021a), and theorem proving (Heule et al., 2016). With respect to modern computing paradigms such as cloud computing and high-performance computing (HPC), the limited scalability of established parallel and distributed SAT solvers has become a pressing issue (Hamadi & Wintersteiger, 2012). Moreover, since processing times of SAT jobs are unknown in advance, conventional HPC scheduling approaches applied to such jobs can lead to prohibitively large scheduling latencies and to suboptimal utilization of computational resources. Instead, we suggest making use of so-called *malleable scheduling*. A parallel computing task is called malleable if the amount of computational resources allotted (i.e., the number of cores or machines) can be adjusted during its execution (Feitelson, 1997). Malleability is a powerful paradigm in the field of job scheduling and load balancing as it allows scheduling incoming jobs rapidly and continuously rebalancing the present tasks according to their priority and other properties.

We believe that a cloud service that combines a scalable SAT solving engine with malleable job scheduling can significantly improve many SAT-related academic and industrial workflows in terms of productivity and (resource-)efficiency. Moreover, our decentralized scheduling approach is applicable to further applications beyond SAT where processing times are unknown in advance and a modest amount of data is transferred between the workers.

System overview

Mallob is a C++ application designed for parallel and distributed systems between 16 and 10000 cores; it allows resolving propositional problems in a (massively) parallel manner. Our SAT solving engine scales up to two thousand cores (Schreiber & Sanders, 2021) using a portfolio of established sequential SAT solvers (Audemard & Simon, 2009; Biere, 2017; Biere et al., 2020) and a careful exchange of information among them. Our submissions of Mallob to

the International SAT Competitions (Schreiber, 2020, 2021b) won multiple prizes (Balyo et al., 2021; Froleyks et al., 2021). Following this success, Mallob has been referred to as "by a *wide* margin, the most powerful SAT solver on the planet" (B. Cook, 2022). Each distributed SAT solving task is malleable, allowing users to submit formulae to Mallob at will, with scheduling latencies in the range of milliseconds (Sanders & Schreiber, 2022b). Computational resources are allotted proportionally to each job's priority and also respecting each job's (maximum) demand for resources. Our scheduling approach is fully decentralized and uses a small part of each worker's CPU time ($< 5\%$) to perform scheduling negotiations. As such, Mallob achieves optimal system utilization, i.e., either all processes are utilized or all resource demands are fully met. We achieve this feat by arranging each active job as a binary tree of workers and growing or shrinking each job tree dynamically based on the current system state. For an in-depth discussion of these techniques, we refer to Schreiber & Sanders (2021), Sanders & Schreiber (2022a), and Sanders & Schreiber (2022b).

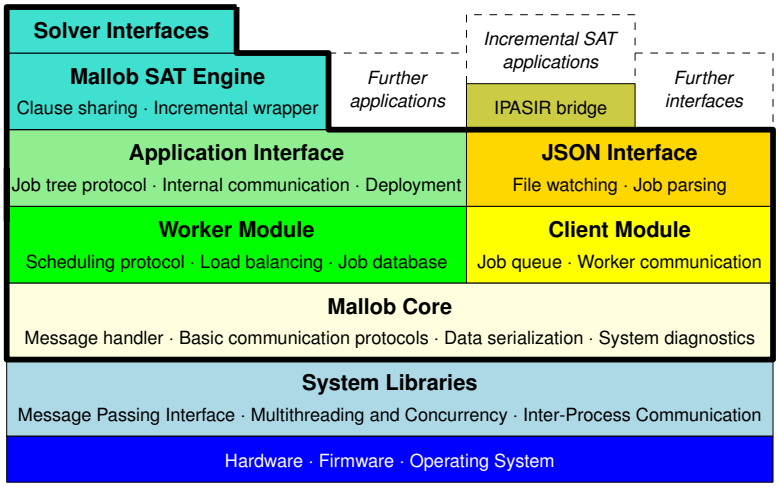

Figure 1: Technology stack of Mallob.

The further development of Mallob is an ongoing effort. As such, we are in the process of integrating engines for NP-hard applications beyond SAT, such as k-Means clustering or hierarchical planning, into Mallob.

Acknowledgements

We wish to thank the numerous people whose code we make thankful use of, including Balyo et al. (2015), Biere et al. (2020), Audemard & Simon (2009), Eén & Sörensson (2003), Lohmann (2022), Goetghebuer-Planchon (2022), Ankerl (2022), and Dusíková (2022). The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC-NG at Leibniz Supercomputing Centre (www.lrz.de). Moreover, some of this work was performed on the HoreKa supercomputer funded by the Ministry of Science, Research and the Arts Baden-Württemberg and by the Federal Ministry of Education and Research. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 882500).

References

Ankerl, M. (2022). *robin_hood unordered map & set*. https://github.com/martinus/robin-hood-hashing

Audemard, G., & Simon, L. (2009). Predicting learnt clauses quality in modern SAT solvers. *International Joint Conference on Artificial Intelligence*, 399–404.

Balyo, T., Froleyks, N., Heule, M. J., Iser, M., Järvisalo, M., & Suda, M. (2021). *The results of SAT competition 2021*. https://satcompetition.github.io/2021/slides/ISC2021-fixed.pdf

Balyo, T., Sanders, P., & Sinz, C. (2015). Hordesat: A massively parallel portfolio SAT solver. *International Conference on Theory and Applications of Satisfiability Testing*, 156–172. https://doi.org/10.1007/978-3-319-24318-4_12

Biere, A. (2017). Cadical, lingeling, plingeling, treengeling and yalsat entering the SAT competition 2018. *SAT Competition*, 14–15.

Biere, A., Fazekas, K., Fleury, M., & Heisinger, M. (2020). CaDiCaL, kissat, paracooba, plingeling and treengeling entering the SAT competition 2020. *SAT Competition 2020*, 50.

Cook, B. (2022). *Automated reasoning's scientific frontiers*. Amazon Science. https://www.amazon.science/blog/automated-reasonings-scientific-frontiers

Cook, S. A. (1971). The complexity of theorem-proving procedures. *ACM Symposium on Theory of Computing*, 151–158. https://doi.org/10.7551/mitpress/12274.003.0036

Dusíková, H. (2022). *Compile time regular expressions*. https://github.com/hanickadot/compile-time-regular-expressions

Eén, N., & Sörensson, N. (2003). An extensible SAT-solver. *International Conference on Theory and Applications of Satisfiability Testing*, 502–518. https://doi.org/10.1007/978-3-540-24605-3_37

Feitelson, D. G. (1997). *Job scheduling in multiprogrammed parallel systems*.

Froleyks, N., Heule, M., Iser, M., Järvisalo, M., & Suda, M. (2021). SAT competition 2020. *Artificial Intelligence*, *301*, 103572. https://doi.org/10.1016/j.artint.2021.103572

Goetghebuer-Planchon, T. (2022). *A C++ implementation of a fast hash map and hash set using robin hood hashing*. https://github.com/Tessil/robin-map

Goldberg, E. I., Prasad, M. R., & Brayton, R. K. (2001). Using SAT for combinational equivalence checking. *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*, 114–121. https://doi.org/10.1109/date.2001.915010

Hamadi, Y., & Wintersteiger, C. (2012). Seven challenges in parallel SAT solving. *Proceedings of the AAAI Conference on Artificial Intelligence*, *26*, 2120–2125. https://doi.org/10.1609/aimag.v34i2.2450

Heule, M. J., Kullmann, O., & Marek, V. W. (2016). Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. *International Conference on Theory and Applications of Satisfiability Testing*, 228–245. https://doi.org/10.1007/978-3-319-40970-2_15

Kleine-Büning, M., Balyo, T., & Sinz, C. (2019). Using DimSpec for bounded and unbounded software model checking. *International Conference on Formal Engineering Methods*, 19–35. https://doi.org/10.1007/978-3-030-32409-4_2

Lohmann, N. (2022). *JSON for modern C++*. https://github.com/nlohmann/json

Massacci, F., & Marraro, L. (2000). Logical cryptanalysis as a SAT problem. *Journal of Automated Reasoning*, *24*(1), 165–203.

Sanders, P., & Schreiber, D. (2022a). *Artifact and instructions to generate experimental results for the Euro-Par 2022 paper: "Decentralized Online Scheduling of Malleable NP-hard Jobs"*. https://doi.org/10.6084/m9.figshare.20000642

Sanders, P., & Schreiber, D. (2022b). Decentralized online scheduling of malleable NP-hard jobs. *European International Conference on Parallel Processing*.

Schreiber, D. (2020). Engineering HordeSat towards malleability: Mallob-mono in the SAT 2020 cloud track. *SAT Competition 2020*, 45.

Schreiber, D. (2021a). Lilotane: A lifted SAT-based approach to hierarchical planning. *Journal of Artificial Intelligence Research*, *70*, 1117–1181. https://doi.org/10.1613/jair.1.12520

Schreiber, D. (2021b). Mallob in the SAT competition 2021. *SAT Competition 2021*, 38.

Schreiber, D., & Sanders, P. (2021). Scalable SAT solving in the cloud. *International Conference on Theory and Applications of Satisfiability Testing*, 518–534. https://doi.org/10.1007/978-3-030-80223-3_35