Poster: gbdt-rs: Fast and Trustworthy Gradient Boosting Decision Tree

Tianyi Li[1], Tongxin Li[2], Yu Ding[2], Yulong Zhang[2], Tao Wei[2], Xinhui Han[1]
[1]Peking University [2] Baidu X-Lab
{litianyi, hanxinhui}@pku.edu.cn
{litongxin, dingyu02, ylzhang, lenx}@baidu.com

***Abstract*—For data analysis services, the capability of preserving user privacy has become an eagerly demanded trait. However, most existing approaches are either unsound to close some major information leakage chanels or prohibitively expensive for practical deployment. We propose `gbdt-rs` , a secure and fast implementation of the gradient boosting decision tree algorithm, which is widely used in data mining and machine learning tasks. `gbdt-rs` is designed to fit traditional computing platforms as well as hardware-enforced trusted environments, i.e., Intel SGX secure enclaves. `gbdt-rs` is coded in the Rust programming language to exterminate memory errors, while its performance is barely sacrificed due to sophisticatedly optimized memory access patterns. Our experiments show that `gbdt-rs` can provide up to 10x speedup on inference tasks compared to XGBoost, a reputed C++ implementation of the same algorithm. In addition, `gbdt-rs` is highly auditable due to its relatively small code base.**

I. Introduction

Privacy and security has risen as one of the major concerns about big data analysis. Both the academia and industry seek high-assurance privacy-preserving data analysis solutions that scale to tremendous amounts of sensitive data. Among various proposed solutions, hardware-assisted trusted computation is drawing more attention for being able to deliver privacy-preserving general-purpose computation with modest cost. Competitor approaches, such as the Fully Homomorphic Encryption (FHE) and multi-party secure computation, are more than 1000 times slower than hardware-assisted solutions and incur bloated communication complexity. To the best our knowledge, hardware-enabled trusted computing is the only practical solution to privacy-preserving data analysis at this point.

In this paper, we propose `gbdt-rs`, a fast and trustworthy implementation of the gradient boosting decision tree (GBDT) algorithm. `gbdt-rs` is able to run both inside and outside Intel SGX secure enclaves. Written in Rust, `gbdt-rs` benefits from an advanced type system to minimize the occurence of memory bugs. The code base of `gbdt-rs` is as small as about 2000 lines of code, making it extremely auditable. Moreover, by utilizing highly-optimized memory access patterns, the performance of `gbdt-rs` is barely compromised as the cost of additional security. Compared to XGBoost [1], one of the most reputed and widely deployed GBDT implementation, `gbdt-rs` can be up to 10 times faster on inference tasks. Even when exeucted in Intel SGX enclaves, the slow down is confined within 10%. Overall, we make the following contributions in this work:

- We show some deep insight on secure and trustworthy data analysis software stack and propose a novel approach by leveraging Rust and hybrid memory-safety rules-of-thumb.
- We use gradient boosting decision tree as a concrete example to show how the methodology can help achieve strong security guarantees, auditable trustworthiness, as well as good performance.
- We share the details about a safe and secure implementation of the gradient boosting decision tree and demonstrate its uncompromized performance with experiments.

The code of `gbdt-rs` is hosted on Github[1] which will be made public in the near future. The project is based on rust-sgx-sdk [2] and follows the hybrid memory-safety rules-of-thumb proposed in that paper.

II. Challenges and Design

One of the design principles for `gbdt-rs` is "tiny and clean." Its implementation seems to be pretty straightforward. The major challenge for this 2000-line project is to provide memory-safety guarantees and trustworthiness for the entire software stack. One of its competitors is XGBoost. XGBoost contains about 15K lines of C and C++ code with immediate dependencies on additional 66.5K lines of third-party code[2]. The third party code further depends on other libraries. The software stack of XGBoost is too complicated to be audited and it is extremely difficult, if not impossible, to formally prove any memory-safety property about it.

To simulteneously achieve safety, security, auditability, and satisfying performance, we chose Rust as the coding language of `gbdt-rs`. We leverage the rich semantics and strict syntax of Rust to properly design and implement each level of APIs. With the powerful type/lifetime/borrowship checker of Rust, and its official lint tool rust-clippy, we guarantee that `gbdt-rs` is appropriately designed and faithfully implemented. An expert can easily examine the implementation of each function and establish trust in the library.

Rust provides strong memory-safety guarantees and it has been partially verified [4]. The verification results indicate

[1]https://github.com/mesalock-linux/gbdt-rs
[2]20K lines of dmlc-core, 6.5K lines of rabit and 40K lines of cub

TABLE I
INFERENCING PERFORMANCE IN REGULAR ENVIRONMENT

Test	i7-8086K/Linux	i7-8850H/macOS	Intel J5005/Linux
XGBoost 10K	5.6494s	13.0135s	34.5524s
`gbdt-rs` 10K	1.4221s	1.8590s	3.4988s
XGBoost 100K	57.1083s	129.1581s	345.6155s
`gbdt-rs` 100K	15.9581s	24.1202s	45.1806s

TABLE II
SGX ENCLAVE PERFORMANCE TEST

Test	10K	100K
XGBoost 1-thread	34.5224s	345.6155s
XGBoost 4-thread	8.4867s	86.4985s
XGBoost 8-thread	8.2702s	83.8158s
`gbdt-rs`	3.4988s	45.1806s
`gbdt-rs` SGX	3.9662s	174.9028s
`gbdt-rs` SGX	3.9662s	36.0156s(batchsize=10K)

that Rust guarantees memory safety in `safe` code blocks. Although Rust indeed provides `unsafe` code blocks where checkers are temporarily disabled, we *do not* have any `unsafe` code in `gbdt-rs`. Every line of `gbdt-rs` is checked by the Rust compiler and memory-safety is guaranteed.

Another key challenge is *deterministics*. A system is said to be "trusted" [3] when it *satisfies an individual's expectations in providing a particular service*. Although the requirement is straightforward to state and understand, it can be extremely difficult to implement such a deterministic system, mainly because current systems use privileged resource management. Typically, the OS kernel and VM hypervisor have higher privileges to intercept and control the user applications. To minimize interactions with those previleged entities and therefore their impacts, `gbdt-rs` is designed to be single-threaded and the number of I/O requests is reduced to the lowest possible level. This design decision closely aligns with the programming model of recent IoT/Intel SGX devices where limited/no threading/IO exists. We expect that `gbdt-rs` can be seamlessly deployed on IoT devices as a trustworthy and lightweight gradient boosting decision tree inference engine.

Performance optimization is the most attractive of `gbdt-rs`. We carefully profiled XGBoost and the results reveals that the major factor is the speed of memory access. To this end, we use contiguous memory in `gbdt-rs` to store the trees and this solution increases the locality of reference significantly. One step further, we find that CPU cache works more efficiently on caching model instead of caching data. So `gbdt-rs` conducts inferencing on each individual tree with batched data. In a classic setting of {`nfeature=32,depth=6,ntree=10000`}, the `gbdt-rs` is 10x faster than single-threaded XGBoost. This optimization is SGX-friendly and benefits SGX enclaves in a higher acceleration ratio, since Intel SGX v1 only provides 128MB EPC. The current page-fault driven software-based memory swapping mechanism of Intel SGX causes 1000x slow down on random access, but only 6x on sequential memory access. So the optimization is almost perfect for Intel SGX.

Multi-threading `gbdt-rs` supports multi-threading because it provides thread-safety in almost all of its APIs. One could use multi-threading mode to boost inference tasks. However, the multi-threading mode depends on the scheduler of OS kernel which brings in more uncertainty into the solution. We recommend considering about the trade-off between safety/trustworthiness and performance before enabling the multi-threading mode.

Easy to use `gbdt-rs` can directly use the model generated by XGBoost. One can easily load the model and benefit from the performance boost by using `gbdt-rs`.

Optional Training support `gbdt-rs` provides APIs for training and saving model to disk. At this time, we do not provide multi-thread support for training in `gbdt-rs`.

III. PRELIMINARY RESULTS

a) Trustworthiness: `gbdt-rs` consists of 2000 lines of Rust codes with clear API definition and is organized well. So the code can prove itself on trustworthiness. This philosophy follows "proof without words". Challengers who doubt on its honesty could spend some time on code auditing and then establish the trust. And the Rust compiler guarantees its safety.

b) Performance: `gbdt-rs` provides high accelerate rate on inferencing. We the following testbeds: i7-8086K/Linux stands for the most powerful desktop platform. i7-8850H/macOS is the latest Macbook Pro. Intel J5005/Linux is classic IoT platforms. We use the aforementioned model settings of {`nfeature=32,depth=6,ntree=10000`} and two input dataset: 10K and 100K. Results of this regular performance test could be found in table I. We can see that `gbdt-rs` provides 4-10x faster on 10K dataset and 4-7x faster on 100K dataset. Table II shows the result of SGX performance test. we can see that `gbdt-rs` SGX has 13% slowdown on 10K datasets but still 8.7x faster than single-threaded XGBoost and 2.34x faster than 8-threaded XGBoost. On the SGX test, we test `gbdt-rs` with 100K dataset in two modes: one single 100K, and 10 group of 10K (batched). Theoretically, the test with one single 100K data would be 6x slower then that of 10x10k due to the 6x slowdown caused by memory swapping and the results confirmed that. We can see the time cost in batched 100k is almost 10x of 10K, which means `gbdt-rs` is a linear time implementation.

REFERENCES

[1] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[2] Y. Ding, R. Duan, L. Li, Y. Cheng, Y. Zhang, T. Chen, T. Wei, and H. Wang. Poster: Rust sgx sdk: Towards memory safety in intel sgx enclave. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 2491–2493, New York, NY, USA, 2017. ACM.

[3] D. A. Fisher, J. M. McCune, and A. D. Andrews. Trust and trusted computing platforms. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2011.

[4] F. Wang, F. Song, M. Zhang, X. Zhu, and J. Zhang. Krust: A formal executable semantics of rust. In *2018 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 44–51, Aug 2018.