Reproducibility Report for ACM SIGMOD 2023 Paper: "SafeBound: A Practical System for Generating Cardinality Bounds"

XINGGUANG CHEN, The Chinese University of Hong Kong, China
SAEED FATHOLLAHZADEH, Concordia University, Canada
SIBO WANG, The Chinese University of Hong Kong, China

We have successfully replicated a portion of the key performance results presented in the experimental section of the original paper. The authors provided valuable support for the replication process, including data files, detailed scripts for running experiments and plotting figures, and a helpful guide that outlined the step-by-step replication process. Their assistance greatly contributed to the successful recreation of the results. We conducted experiments with two sets of independent environments (AWS from Concordia University and Ubuntu from The Chinese University of Hong Kong). Thereafter, we will use Setting 1 to indicate tested by Saeed and use Setting 2 to indicate the Ubuntu environment tested by Xingguang and Sibo.

1 INTRODUCTION

A system called SafeBound is proposed to generate cardinality bounds from degree sequences, even when conditioned on predicates. The authors also employ an enhanced compression technique for degree sequences to maintain the cardinality of the base tables. Experimental results using PostgreSQL on four benchmarks demonstrate that SafeBound outperforms alternative methods in terms of lower end-to-end running time, query planning time, and memory overhead.

2 SUBMISSION

The work offers ample resources for replicating its effectiveness and performance experiments. A comprehensive README file is available in a GitHub repository, providing references to the source code repository and offering detailed step-by-step instructions for the replication process setup.

- Code repository on GitHub: https://github.com/kylebd99/SafeBound
- Readme: The README provides a detailed guide for setting up the necessary environment to conduct experiments. It covers steps such as building the SafeBound library, configuring PostgreSQL 13 as the database engine, loading benchmark databases, and running experiments. Users can simply follow the instructions sequentially on a machine with Anaconda installed.
- Dataset: The repository offers convenience by providing shell scripts that automate the download and preprocessing of benchmark datasets. These datasets, including "JOBLight", "JOBLightRanges", "JOBM" and "Stats" are used for the experiments.

3 HARDWARE AND SOFTWARE ENVIRONMENT

Table 1 for a comprehensive analysis contrasting the authors' initial setup with Setting 1 on an AWS machine and Setting 2 on a Linux machine. As indicated in the README file, the rough durations for running BuildExperiments InferenceExperiments, and RuntimeExperiments are 4-8 hours, 10-30 minutes, and 1-2 days, respectively, closely corresponding to our findings throughout the review process on Settings 1 and 2.

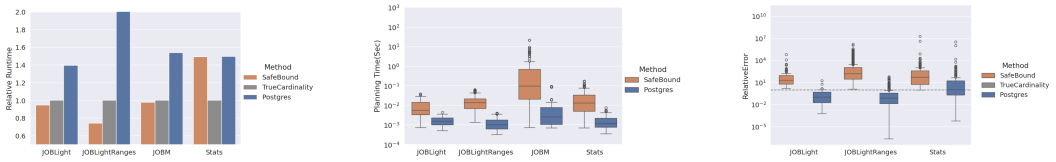
4 REPRODUCIBILITY EVALUATION

4.1 Process

We meticulously adhered to the Reproducibility section, comprising seven steps (1-7) encompassing installation, preparation, and the execution of experimental runs. We configure PostgreSQL 13 with

Table 1. Hardware & Software environment

	Paper	Repro Review Setting 1	Repro Review Setting 2
Node	AWS EC2 (m5.8xlarge)	AWS EC2 (m5.8xlarge)	Ubuntu 22.04.2 LTS
CPU	Intel Xeon Platinum 8000	Intel Xeon Platinum 8000	Intel Xeon Gold 5218R
Cores	32	32	40
RAM	128 GB	128 GB	512 GB
Storage	HDD 6,800 Mbps	HDD 6,800 Mbps	HDD 6,800 Mbps



(a) Runtimes Relative. (b) Planning Time. (c) Relative Errors.

Fig. 1. Reproduction (excerpt) of Figure 5 in [1] on Setting 1: Workload runtimes, planning time, and estimation error

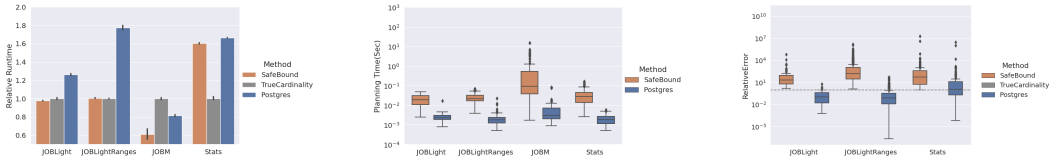
shared memory set to 4GB, worker memory to 2GB, and max parallel workers to 6, as recommended. The necessary software requirements were installed on the AWS machine with Setting 1 and the Linux machine with Setting 2 respectively, and the experiments were conducted. To ensure the correct execution of experiments, guarding against issues like software mismatches, unavailable data, and system crashes, we executed all scripts with the "nohup" command prefixed to them and conducted regular checks.

Additionally, as previously mentioned, we closely monitored the hardware infrastructure provided by the author, regularly scrutinizing system access and log entries. While the paper included a variety of experiments, the repository lacked some of the baseline implementations. Despite reaching out to the authors, they were unable to supply reproducibility code for methods other than Postgres. Their methods entailed specific handholding and installation steps that couldn't be easily replicated within the repository.

Upon completing a meticulous step-by-step execution of the experiments, the results were stored in the "Data/Results" path. Subsequently, we utilized the Jupyter notebooks, namely "BuildGraph", "InferenceGraphs", and "RuntimeGraphs", provided by the author to generate visual representations of the results.

4.2 Results

The reproducibility framework employed in our study faithfully replicates the key experiments outlined in Section 5 of the original paper. Our review analysis of the experimental results involves a comprehensive comparison, including the paper's proposed solution, against two other baselines, Postgres and TrueCardinality. While we encountered challenges, such as missing certain measures and baselines due to the intricacies of specific installation steps, our locally obtained results align with the critical Figures 5 (excluding baseline implementations: Postgres2D, PostgresPK, PessEst, Simplicity, NeuroCard, and BayesCard), 6, 7 and 8 (excluding baseline implementations: Postgres2D, Simplicity, NeuroCard, BayesCard) from the original paper. As we discussed, we cannot reproduce Figures 9, 10, and 11.



(a) Runtimes Relative. (b) Planning Time. (c) Relative Errors.

Fig. 2. Reproduction (excerpt) of Figure 5 in [1] on Setting 2: Workload runtimes, planning time, and estimation error

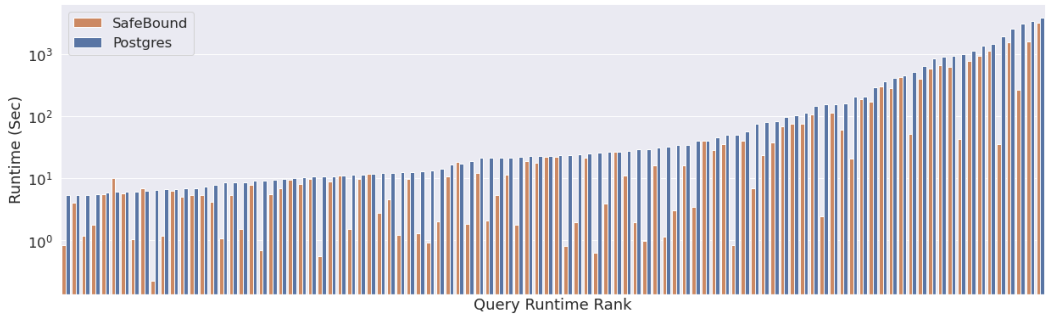


Fig. 3. Reproduction of Figure 6 in [1] on Setting 1: The runtime of the 80 longest-running queries across all benchmarks. Estimates from SafeBound result in faster query execution on the most expensive queries with a p05/p25/p50/p75/p95 quantile speedup of 1.01x/1.3x/1.7x/10.1x/30.3x.

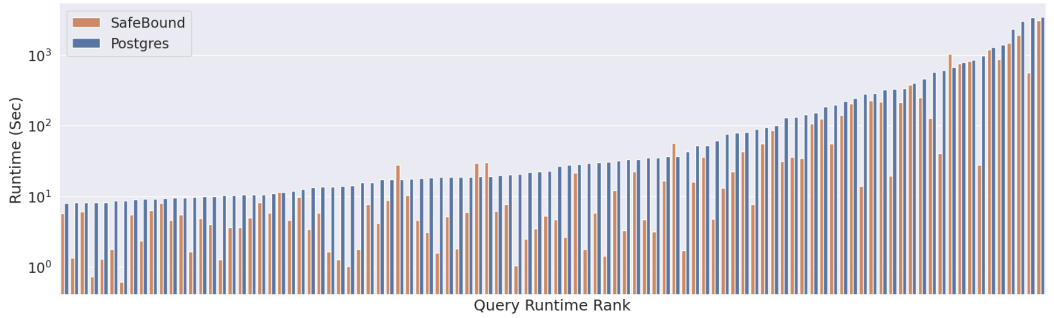


Fig. 4. Reproduction of Figure 6 in [1] on Setting 2: The runtime of the 80 longest-running queries across all benchmarks. Estimates from SafeBound result in faster query execution on the most expensive queries with a p05/p25/p50/p75/p95 quantile speedup of 1.01x/1.3x/1.7x/10.1x/30.3x.

5 SUMMARY

The materials provided by the authors greatly facilitate the reproducibility of the proposed system. In summary, the outcomes we two groups achieved partially substantiate the assertions made in the original paper.

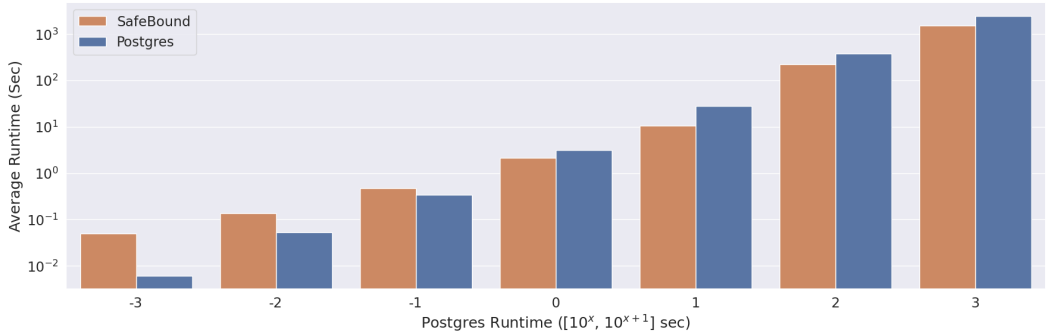
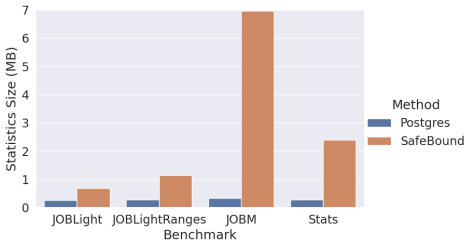


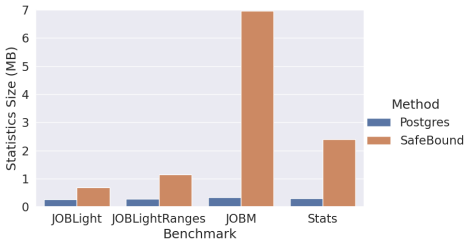
Fig. 5. Reproduction of Figure 7 in [1] on Setting 2: The average runtime of queries binned by their runtime using Postgres' estimates. SafeBound outperforms Postgres' estimates for queries with runtime over one second.



(a) Across all benchmarks, SafeBound uses 3x-6.8x less space than ML methods

(b) SafeBound achieves up to 17x lower offline statistics construction time than ML methods.

Fig. 6. Reproduction (excerpt) of Figure 8 in [1] on Setting 1: Statistic Size and Construction Time



(a) Across all benchmarks, SafeBound uses 3x-6.8x less space than ML methods

(b) SafeBound achieves up to 17x lower offline statistics construction time than ML methods.

Fig. 7. Reproduction (excerpt) of Figure 8 in [1] on Setting 2: Statistic Size and Construction Time

REFERENCES

[1] Kyle B. Deeds, Dan Suciu, and Magdalena Balazinska. 2023. SafeBound: A Practical System for Generating Cardinality Bounds. *Proc. ACM Manag. Data* 1, 1 (2023), 53:1–53:26. https://doi.org/10.1145/3588907