Reproducibility Report for ACM SIGMOD 2023 Paper: "MRVs: Enforcing Numeric Invariants in Parallel Updates to Hotspots with Randomized Splitting"

XIAOJUN DONG, University of California, Riverside, United States
LETONG WANG, University of California, Riverside, United States
BINYANG DAI, Hong Kong University of Science and Technology, Hong Kong, China
NUNO FARIA, INESCTEC and University of Minho, Portugal
JOSÉ PEREIRA, INESCTEC and University of Minho, Portugal

We have successfully evaluated and reproduced the tables and figures in the paper using the artifact provided by the authors on our own machine. The artifact is well-documented with a single-run script. The trends of the reproduced figures have been verified and are aligned with the arguments from the paper.

1 INTRODUCTION

The original paper [1] of the artifact is by Faria and Pereira from the University of Minho. They studied the performance of transactional systems, which is bottlenecked by updating hotspots in previous work since conflicts can lead to heavy contention. The authors addressed the challenge by proposing the Multi-Record Values (MRVs) technique, which uses randomization to split and access the numeric values in multiple records to reduce the probability of conflicts.

All experimental results from the paper are reproducible using the provided artifact. The figures from the reproduced experiments are shown in section 4.

2 SUBMISSION

The artifact is publicly available on GitHub, which consists of a single-run script to deploy and reproduce the experiment and a full step-by-step documentation to set up and test each database individually. The README file on GitHub is well-documented with sufficient details to reuse the code later.

- GitHub repository with code: https://github.com/nuno-faria/mrv/tree/main
- Detailed readme file at: https://github.com/nuno-faria/mrv/blob/main/README.md
- Scripts for reproduction at: https://github.com/nuno-faria/mrv/tree/main/reproducibility

3 HARDWARE AND SOFTWARE ENVIRONMENT

The original results from the paper were run with N1 vCPUs and persistent SSDs from the Google Cloud Platform. In the reproduced experiments, we used a physical machine with 24 CPUs and 252GB HDD. Note that the given scripts to install the dependencies are based on the Ubuntu system and may not work with other operating systems.

Table 1. Hardware & Software environment

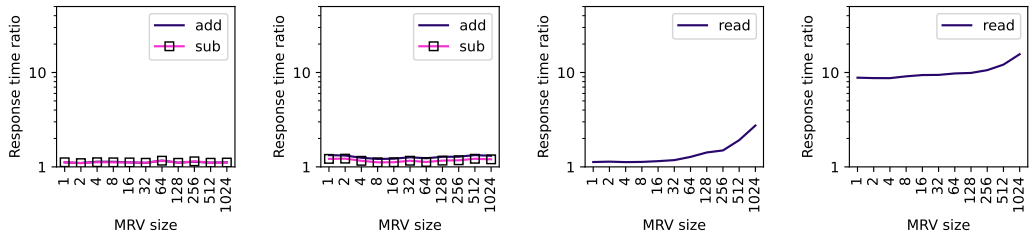
	Paper	Repro Review	Repro Review #3
CPU	N1 vCPUs from GCP	Intel(R) Xeon(R) CPU E5-2670 v2	Intel(R) Xeon(R) Gold 6354
cores	24	24	36
RAM	24GB	24GB	1TB
Storage	500GB SSD	252 GB HDD	1.6 TB SSD
System	Ubuntu 18.04	Ubuntu 22.04	Ubuntu 20.04

4 REPRODUCIBILITY EVALUATION

4.1 Process

We conducted all the experiments using the scripts from the repository provided by the authors. The scripts worked mostly without issues. However, we encountered a few dependencies that were not listed in the README. We had to install Jinja2 manually. The entire process took approximately 30 hours. We then generated the figures using the scripts provided by the authors.

4.2 Results

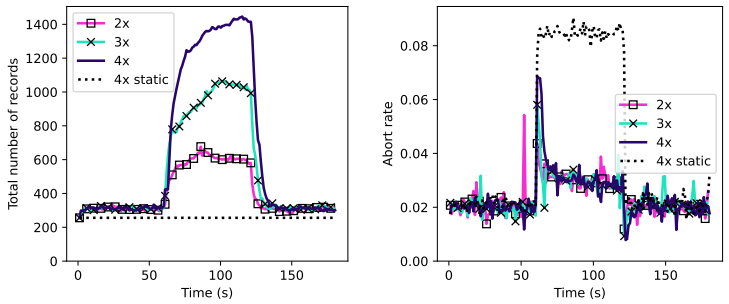


(a) Write PG **(b)** Write MD **(c)** Read PG **(d)** Read MD

Fig. 1. Comparison of the response time ratio between MRVs and baseline (single record) in the microbenchmark, for PostgreSQL (PG) and MongoDB (MD).

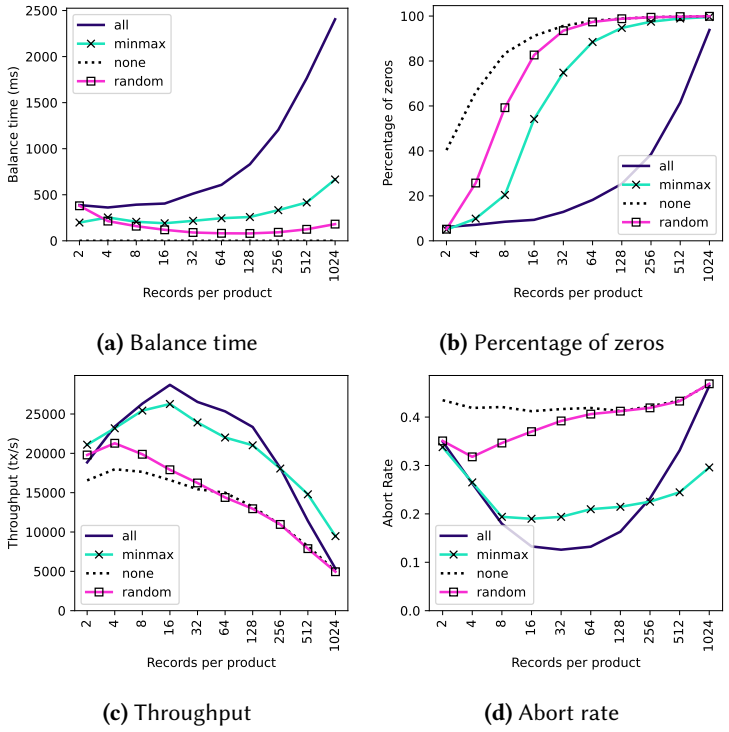
Table 2. MRVs storage overhead relatively to the baseline.

# of clients	1	2	4	8	16	32	64	128	256	512
1 hotspot	*1*	*40*	*109*	*239*	*409*	*699*	*1024*	*1024*	*1024*	*1024*
1 column	2.04	2.03	2.02	2.01	2.18	2.44	2.58	2.01	1.98	1.96
TPC-C	1.00	0.99	0.99	1.03	1.09	1.06	1.06	1.02	1.00	0.94
1 col. static	195.74									



(a) Number of records **(b)** Abort rate

Fig. 2. Evolution of the number of records and abort rate based on different load changes in MRVs. $\times n$ means an n times increase in the number of clients between 60 and 120 seconds.



(a) Balance time **(b)** Percentage of zeros

(c) Throughput **(d)** Abort rate

Fig. 3. Comparison between different balance algorithms.

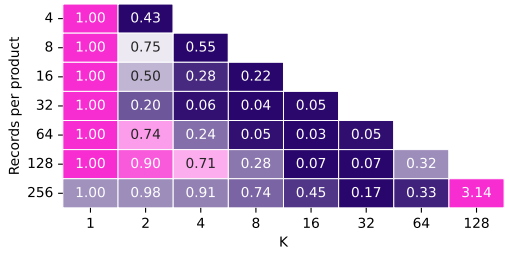
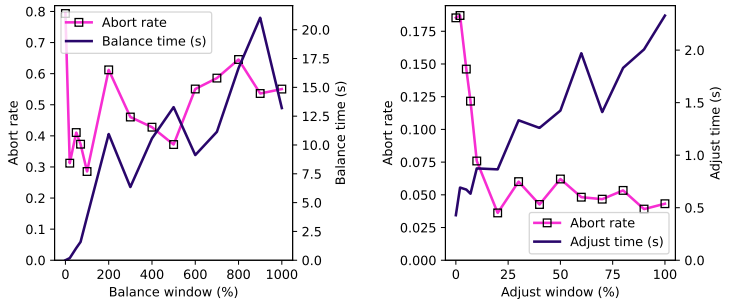
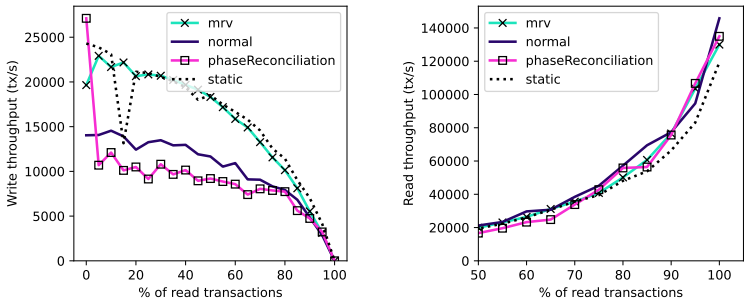


Fig. 4. Comparison between different balance sizes (K) in MRVs performance ($c_i = \frac{\text{Zeros}_{K=i}}{\text{Zeros}_{K=1}} \cdot \frac{\text{Time}_{K=i}}{\text{Time}_{K=1}}$). Lower is better.



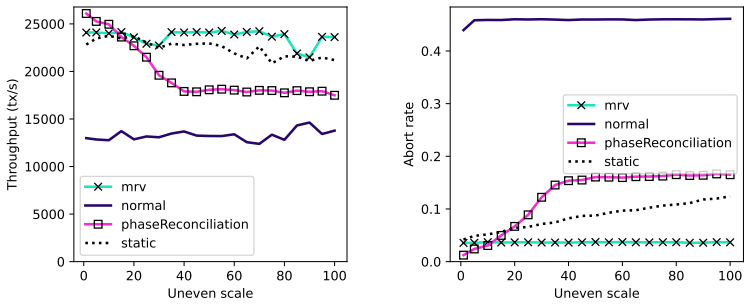
(a) Balance worker **(b)** Adjust worker

Fig. 5. Effects of different windows in the MRVs workers.



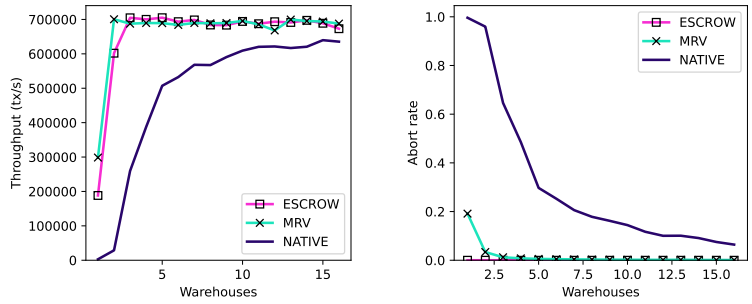
(a) Write throughput **(b)** Read throughput

Fig. 6. Comparison between baseline (native), MRV, and *phase reconciliation* using the microbenchmark with a variable read percentage (PostgreSQL Repeatable Read).



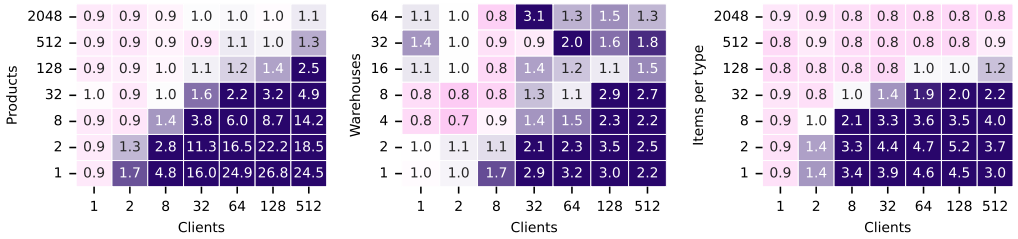
(a) Throughput **(b)** Abort rate

Fig. 7. Comparison between baseline (native), MRV, and *phase reconciliation* using the microbenchmark with various *uneven* scales (writes only; PostgreSQL R. Read). $x = 1$: 1-unit *add* per 1-unit *sub*; $x = 5$: 5-unit *add* per five 1-unit *subs*, and so on.



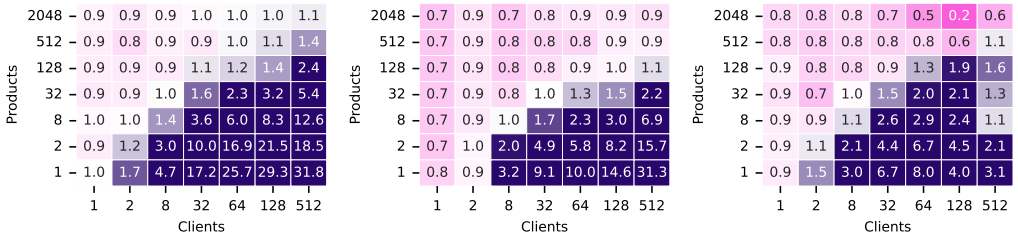
(a) Average throughput **(b)** Average abort rate

Fig. 8. Comparison between baseline (native), MRV, and *escrow locking* using TPC-C's *payment* in DBx1000.



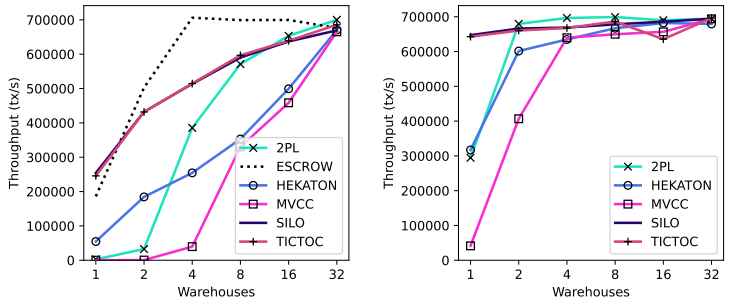
(a) *Microbenchmark* (scale=Products) **(b)** *TPC-C* (scale=Warehouses) **(c)** *STAMP Vacation (opt.)* (scale=Items per type)

Fig. 9. Throughput comparison between MRVs and baseline (native) using different workloads with PostgreSQL's Repeatable Read. A value of 1.0 means MRVs and native have the same throughput, 2.0 means double the throughput for MRV, and so on.



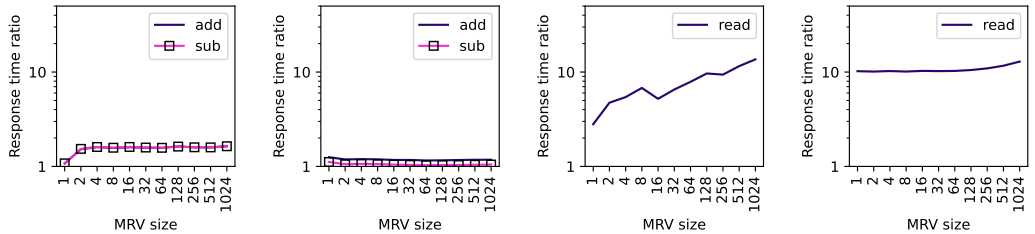
(a) *Single-writer SQL* (PostgreSQL) **(b)** *Single-writer NoSQL* (MongoDB) **(c)** *Multi-writer SQL* (MySQL Group Replication)

Fig. 10. Throughput comparison between MRVs and baseline (native) using the microbenchmark with different database management systems. A value of 1.0 means MRVs and native had the same throughput, 2.0 means double the throughput for MRV, and so on.



(a) Without MRVs **(b)** With MRVs

Fig. 11. Throughput comparison between different concurrency control techniques with and without MRVs. 2PL a) is equivalent to *native* of Figure 11; 2PL b) is equivalent to *mrv* in Figure 11.

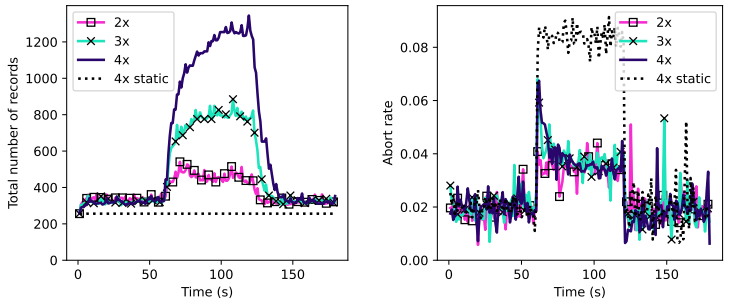


(a) Write PG **(b)** Write MD **(c)** Read PG **(d)** Read MD

Fig. 12. Comparison of the response time ratio between MRVs and baseline (single record) in the microbenchmark, for PostgreSQL (PG) and MongoDB (MD).

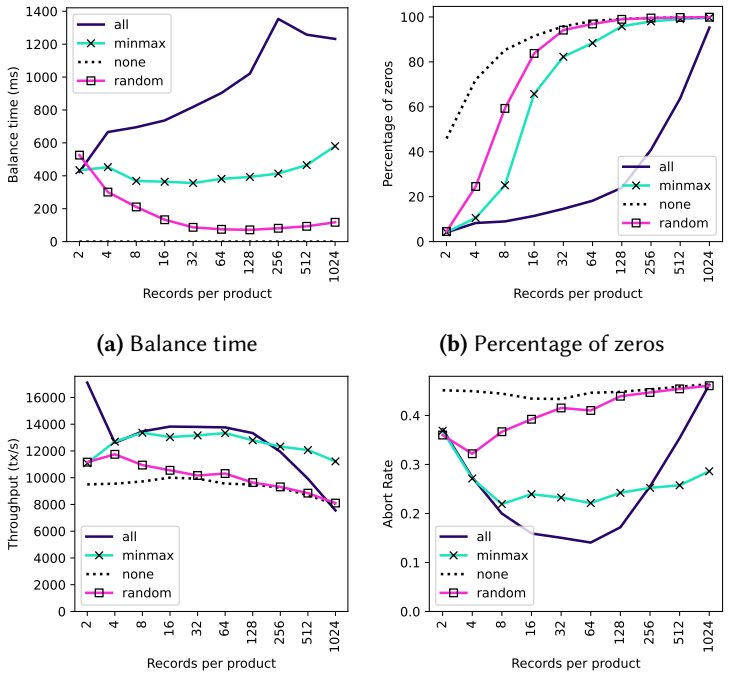
# of clients	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**
1 hotspot	*1*	*60*	*134*	*212*	*406*	*672*	*1024*	*1024*	*1024*	*1024*
1 column	2.05	2.04	2.04	2.01	1.97	1.98	2.38	1.91	1.88	1.48
TPC-C	0.99	1.00	1.00	1.01	1.05	1.10	1.19	1.65	1.65	1.53
1 col. static	200.11									

Table 3. MRVs storage overhead relatively to the baseline.



(a) Number of records **(b)** Abort rate

Fig. 13. Evolution of the number of records and abort rate based on different load changes in MRVs. $\times n$ means an n times increase in the number of clients between 60 and 120 seconds.



(a) Balance time **(b)** Percentage of zeros

(c) Throughput **(d)** Abort rate

Fig. 14. Comparison between different balance algorithms.

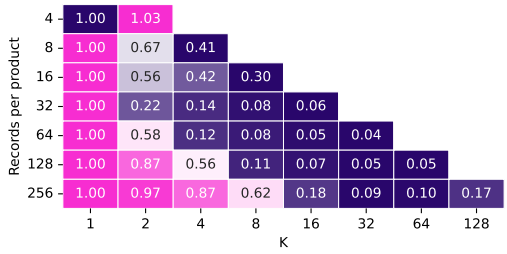
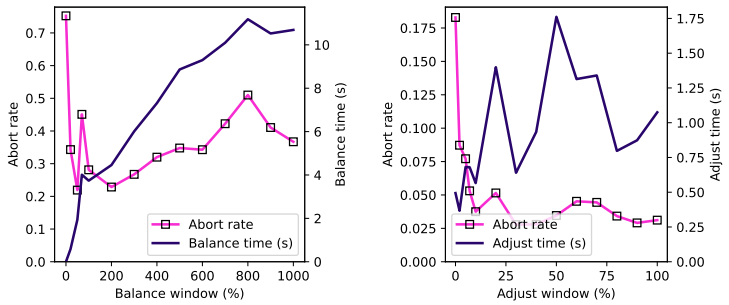
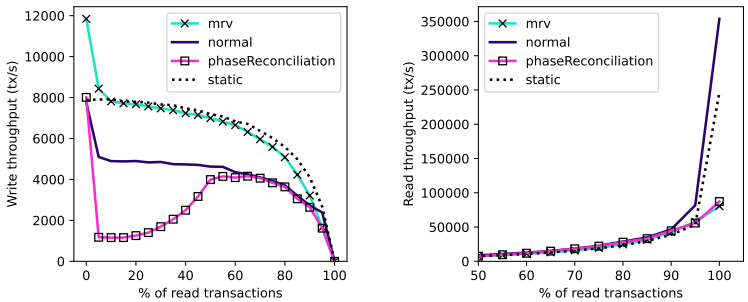


Fig. 15. Comparison between different balance sizes (K) in MRVs performance ($c_i = \frac{\text{Zeros}_{K=i}}{\text{Zeros}_{K=1}} \cdot \frac{\text{Time}_{K=i}}{\text{Time}_{K=1}}$). Lower is better.



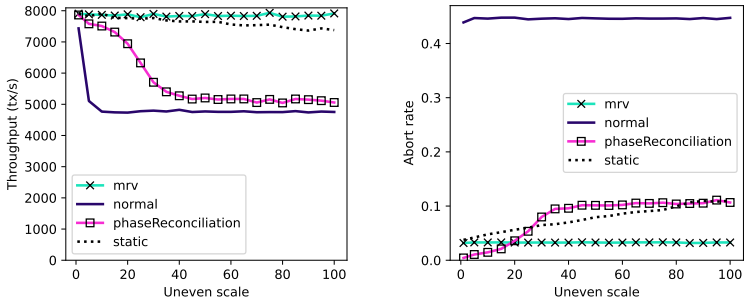
(a) Balance worker **(b)** Adjust worker

Fig. 16. Effects of different windows in the MRVs workers.



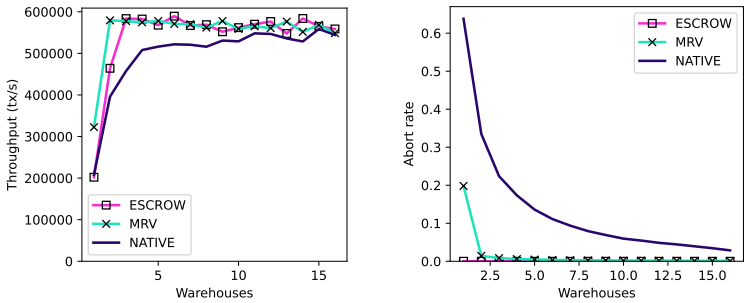
(a) Write throughput **(b)** Read throughput

Fig. 17. Comparison between baseline (native), MRV, and *phase reconciliation* using the microbenchmark with a variable read percentage (PostgreSQL Repeatable Read).



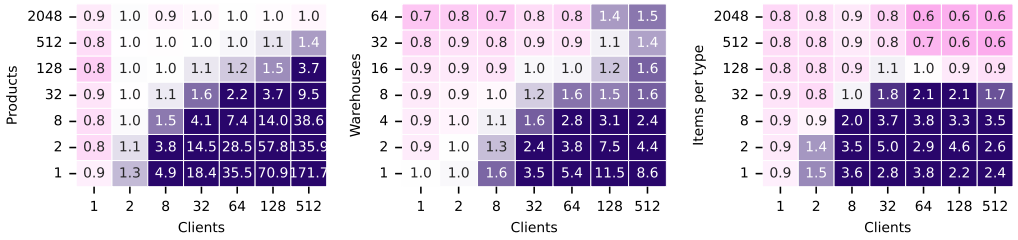
(a) Throughput **(b)** Abort rate

Fig. 18. Comparison between baseline (native), MRV, and *phase reconciliation* using the microbenchmark with various *uneven* scales (writes only; PostgreSQL R. Read). $x = 1$: 1-unit *add* per 1-unit *sub*; $x = 5$: 5-unit *add* per five 1-unit *subs*, and so on.



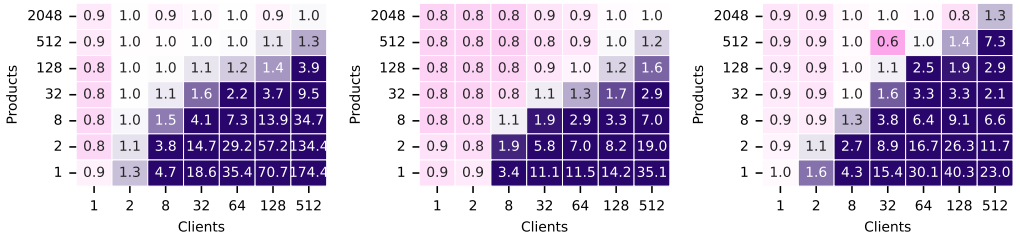
(a) Average throughput **(b)** Average abort rate

Fig. 19. Comparison between baseline (native), MRV, and *escrow locking* using TPC-C's *payment* in DBx1000.



(a) *Microbenchmark* (scale=Products) **(b)** *TPC-C* (scale=Warehouses) **(c)** *STAMP Vacation (opt.)* (scale=Items per type)

Fig. 20. Throughput comparison between MRVs and baseline (native) using different workloads with PostgreSQL's Repeatable Read. A value of 1.0 means MRVs and native have the same throughput, 2.0 means double the throughput for MRV, and so on.

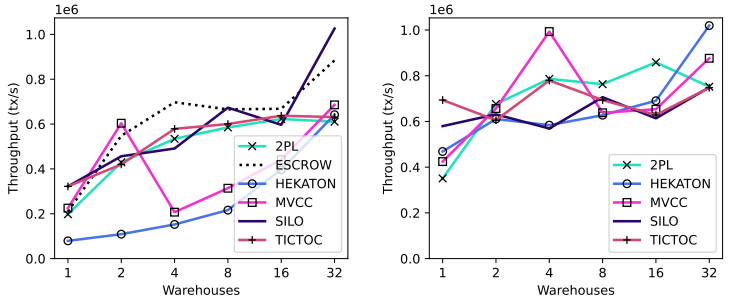


(a) *Single-writer SQL* (PostgreSQL)

(b) *Single-writer NoSQL* (MongoDB)

(c) *Multi-writer SQL* (MySQL Group Replication)

Fig. 21. Throughput comparison between MRVs and baseline (native) using the microbenchmark with different database management systems. A value of 1.0 means MRVs and native had the same throughput, 2.0 means double the throughput for MRV, and so on.



(a) Without MRVs

(b) With MRVs

Fig. 22. Throughput comparison between different concurrency control techniques with and without MRVs. 2PL a) is equivalent to *native* of Figure 11; 2PL b) is equivalent to *mrv* in Figure 11.

5 CHANGES MADE TO THE ARTIFACTS

To answer the issues faced during the reproducibility review, some minor modifications were made to the reproducibility artifacts:

- Upgraded the Jinja2 library used, as the Pandas library to plot the results requires a newer version to work;
- Added a clarification to the README about the usage of SSDs to handle databases' storage;
- Increased the runtime for the DBx1000 tests (Fig. 11), from 100k transactions to 1M, in order to notice significant differences between Native, Escrow, and MRVs with high-end CPUs. Otherwise, the results would finish too quickly;
- Added the VACUUM command to the storage comparison results (Tab. 2), to exclude dead tuples from the measurements;
- Removed the *History* table from the storage measurements in TPC-C (Tab. 2), as this is essentially a log table of payments that increases in size proportionally to the throughput. This table does not use MRVs.

REFERENCES

[1] Nuno Faria and José Pereira. 2023. MRVs: Enforcing Numeric Invariants in Parallel Updates to Hotspots with Randomized Splitting. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–27.