Reproducibility Report for ACM SIGMOD 2023 Paper: "Polaris: Enabling Transaction Priority in Optimistic Concurrency Control"

SONGLIN JIANG, Aalto University, Finland and Technical University of Denmark, Denmark
BO ZHAO, Aalto University, Finland
XIAODONG LI, The University of Hong Kong, China
YIXIANG FANG, The Chinese University of Hong Kong, Shenzhen, China
CHENHAO YE, University of Wisconsin-Madison, USA
WUH-CHWEN HWANG, University of Wisconsin-Madison, USA
KEREN CHEN, University of Wisconsin-Madison, USA
XIANGYAO YU, University of Wisconsin-Madison, USA

We have successfully reproduced the evaluation results in the paper. We appreciate the authors have prepared detailed documents and automation scripts for reproducibility.

1 INTRODUCTION

This report describes the reproducibility process and results of the paper *Polaris: Enabling Transaction Priority in Optimistic Concurrency Control* [1] authored by Chenhao Ye, Wuh-Chwen Hwang, Keren Chen, and Xiangyao Yu from the University of Wisconsin-Madison. The authors present Polaris, an optimistic concurrency control protocol that supports multiple priority levels. The authors evaluate Polaris using the YCSB benchmark and TPC-C benchmark. In addition, they conduct a comprehensive comparison against other concurrency control protocols including Silo, No-Wait, Wait-Die, and Wound-Wait.

Our reproduced results demonstrate that Polaris outperforms Silo and other concurrency control protocols with higher throughput and lower tail latency, which supports the scientific contributions/claims of the original paper [1].

2 SUBMISSION

The submission comprises detailed instructions for installing the code base and dependencies, along with Bash and Python scripts to execute the experiments with a single-line command. Additionally, the provided Python scripts can automatically generate all the figures and plots in PDF format.

- GitHub repository with code and scripts at https://github.com/chenhao-ye/polaris
- Detailed instructions for reproduction at https://github.com/chenhao-ye/polaris/blob/main/ARTIFACT.md, which describes how to generate all the figures presented in paper [1].

3 HARDWARE AND SOFTWARE ENVIRONMENT

Table 1 demonstrates the hardware specification of the paper-recommended setup [1] and our own machines used in the reproduction review. Note that the setup in our reproduction review only reflects the resources we have reserved via slurm. More details can be found in the specifications of our Triton HPC cluster https://scicomp.aalto.fi/triton/overview/.

4 REPRODUCIBILITY EVALUATION

4.1 Process

We managed to generate all plots and figures are reproduced out of the box when following the instructions and executing the codebase. It's important to note that when we run on Triton HPC

Table 1. Hardware & Software environment

	Paper [1] (CloudLab c6420)	Reproduction Review (Aalto Triton, fn3, shared)
OS	Ubuntu 20.04	CentOS Linux 7
CPU	Intel Xeon Gold 6142	Intel Xeon Gold 6148
cores (threads)	2 x 16 (x2)	39 (x2)
GHz	2.6	2.4
RAM	384GB DDR4-2666	624GB DDR4-2666

cluster, we initially allocate 64 CPUs, each with 6GB of memory, mirroring the specifications of CloudLab c6420. However, when the thread count exceeds 48, we find a significant amount of anomalous data in some of the reproduced graphs (0 throughput and almost infinite latency). Consequently, we opted to reserve additional resources and the situation improved a little bit, but we still have some anomalous results in the reproduced graph. It can be caused by the disk lags since we share resources in HPC systems. We also manually change the y-axis scale a little bit in `plot.py` so that it can fit our reproduced data.

We use the following script to submit as a slurm batch job on HPC Triton:

```
#!/bin/bash -l
#SBATCH --time=20:00:00
#SBATCH --cpus-per-task=78
#SBATCH --mem-per-cpu=8G

cd $WRKDIR
git clone https://github.com/chenhao-ye/polaris
cd polaris

# Install dependencies
module load anaconda
pip3 install -r requirements.txt

# Run all experiments
bash experiments/run_all.sh

# Draw graphs
python3 parse.py ycsb_latency ycsb_prio_sen ycsb_thread ycsb_readonly ycsb_zipf
    tpcc_thread ycsb_aria_batch
python3 plot.py
```

As indicated in the ARTIFACT.md, the estimated running time for the entire experiment is more than 9 hours, which is consistent with our observations during the review (15 hours). Here are some utilization metrics to share during the job running period:

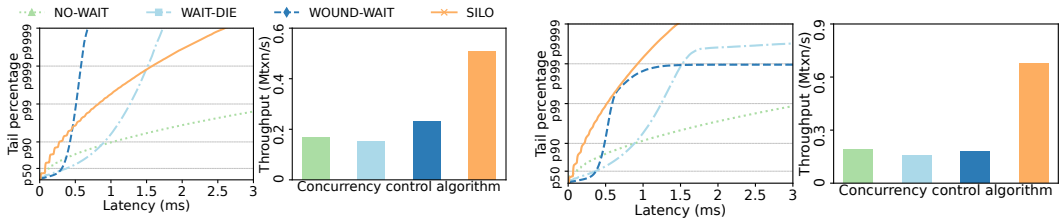
- *CPU Utilized*: 17-23:58:22
- *CPU Efficiency*: 36.92% of 48-17:56:06 core-walltime
- *Job Wall-clock time*: 14:59:57
- *Memory Utilized*: 109.11 GB
- *Memory Efficiency*: 17.49% of 624.00 GB

4.2 Results

Our replicated figures generally match those presented in the paper [1] and show that Polaris can perform better. The authors acknowledge and clarify the divergence noted in Figure 10, attributing it to the Aria p999 tail latency exhibiting relatively high variation across runs. This observation is also reported and explained in the paper as follows:

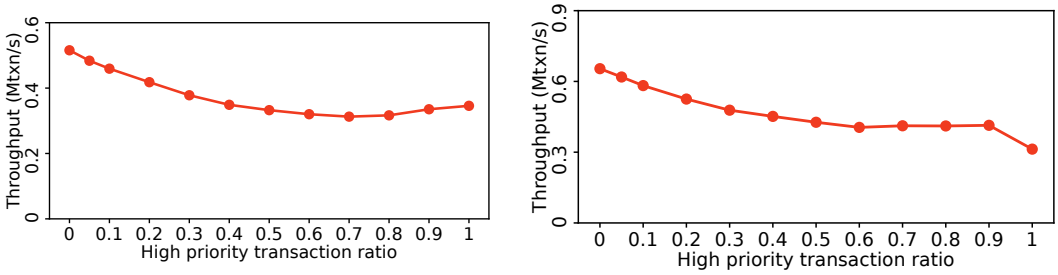
"We observe Aria has fluctuating p999 tail latency when there are more than 32 threads at $\theta = 0.5$. Under this workload, only < 0.08% of transactions have experienced aborts, so the transaction at p999 tail commits at its first execution. We think it is the large batch sizes that amplify noise (e.g., one slow transaction causes all transactions in the batch to wait) and lead to fluctuation." [1]

Figure 1–10 give both the figures from the paper (a) and our reproduced version (b).



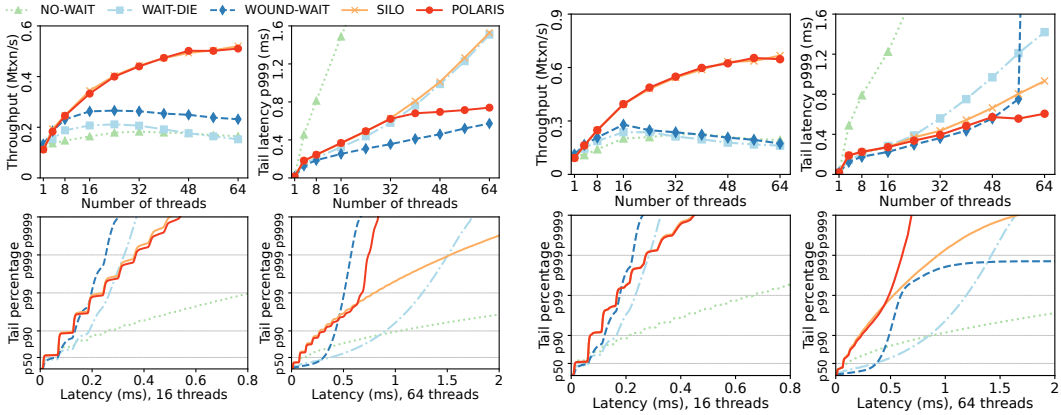
(a) Paper (b) Reproduced

Fig. 1. Latency distribution and overall throughput of four concurrency control algorithms (YCSB-A, $r = 50\%$, $w = 50\%$, $\theta = 0.99$, 64 threads).



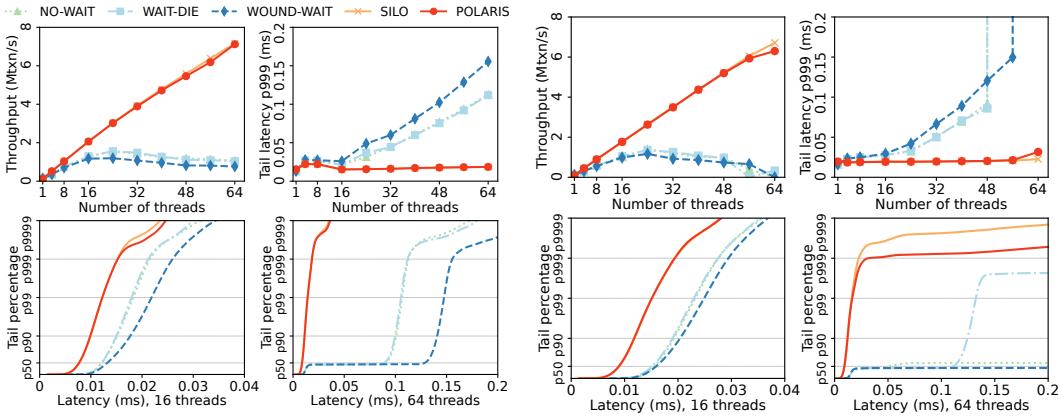
(a) Paper (b) Reproduced

Fig. 2. Polaris throughput with varying ratio of high priority transaction (YCSB-A, $r = 50\%$, $w = 50\%$, $\theta = 0.99$, 64 threads).



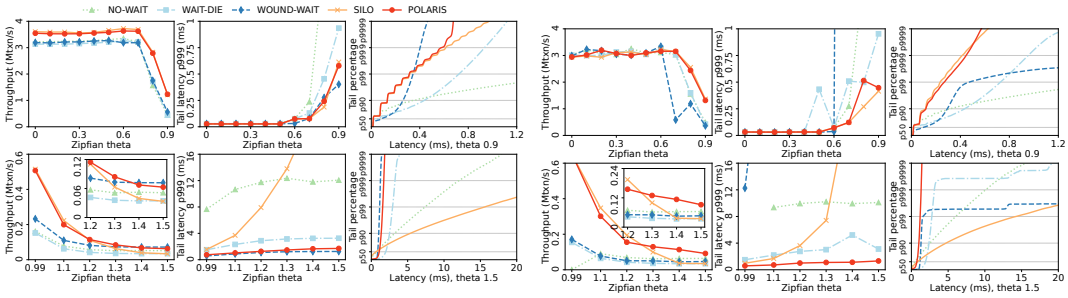
(a) Paper (b) Reproduced

Fig. 3. Throughput and p999 tail latency over a spectrum of thread numbers; latency distribution in the cases of 16 and 64 threads (YCSB-A, r = 50%, w = 50%, θ = 0.99).



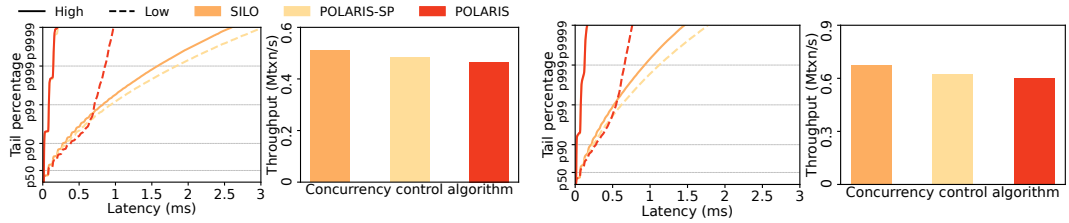
(a) Paper (b) Reproduced

Fig. 4. Throughput and p999 tail latency over a spectrum of thread numbers (YCSB-C, r = 100%, θ = 0.99); latency distribution in the cases of 16 and 64 threads.



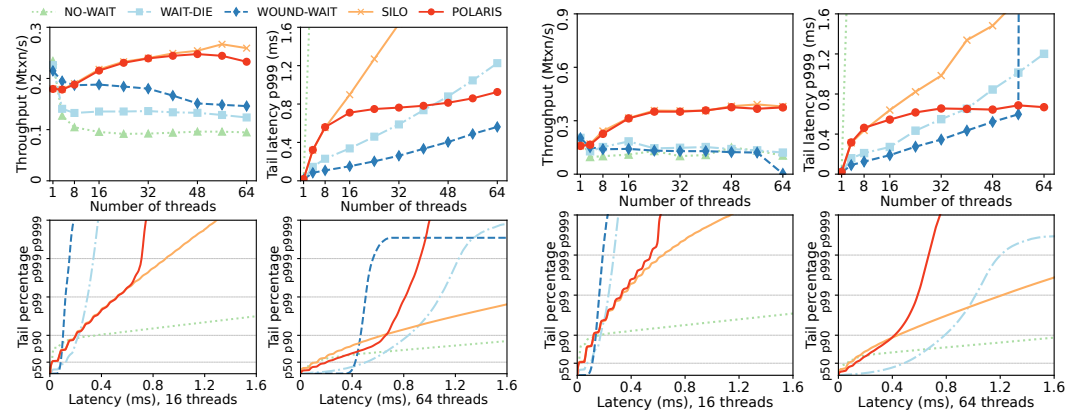
(a) Paper (b) Reproduced

Fig. 5. Throughput and p999 tail latency with varying contention levels; latency distribution when Zipfian θ equals to 0.9 and 1.5 (YCSB-A, r = 50%, w = 50%, 64 threads).



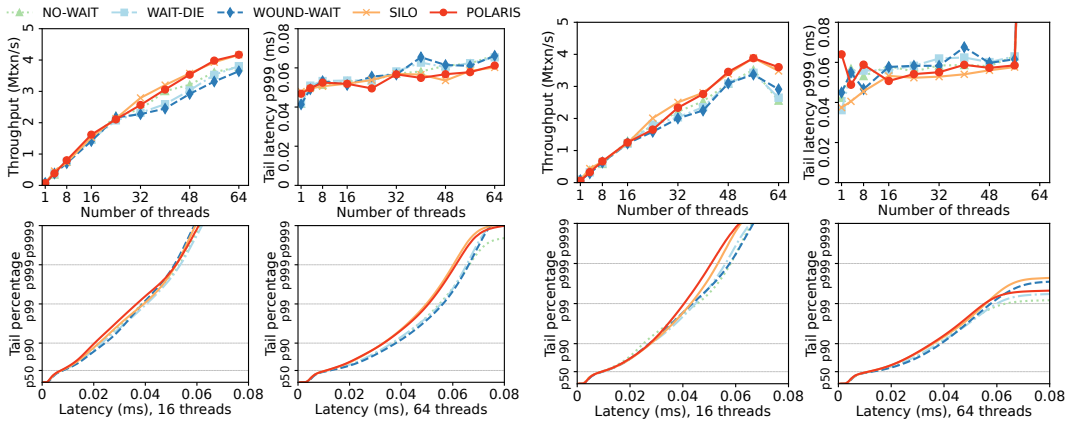
(a) Paper (b) Reproduced

Fig. 6. Latency distribution of high/low-priority transactions and overall throughput (YCSB-A, r = 50%, w = 50%, θ = 0.99, 64 threads).



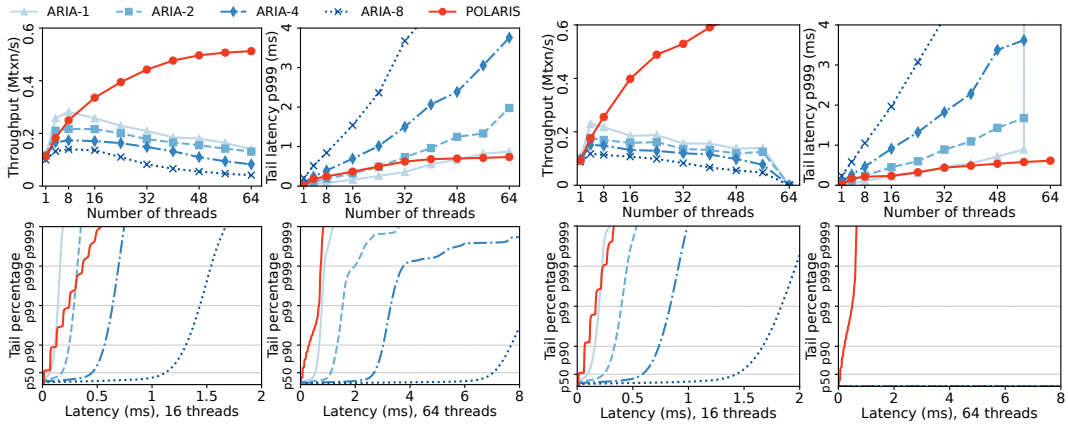
(a) Paper (b) Reproduced

Fig. 7. Throughput and p999 tail latency over a spectrum of thread numbers; latency distribution in the cases of 16 and 64 threads (TPC-C, 1 warehouse).



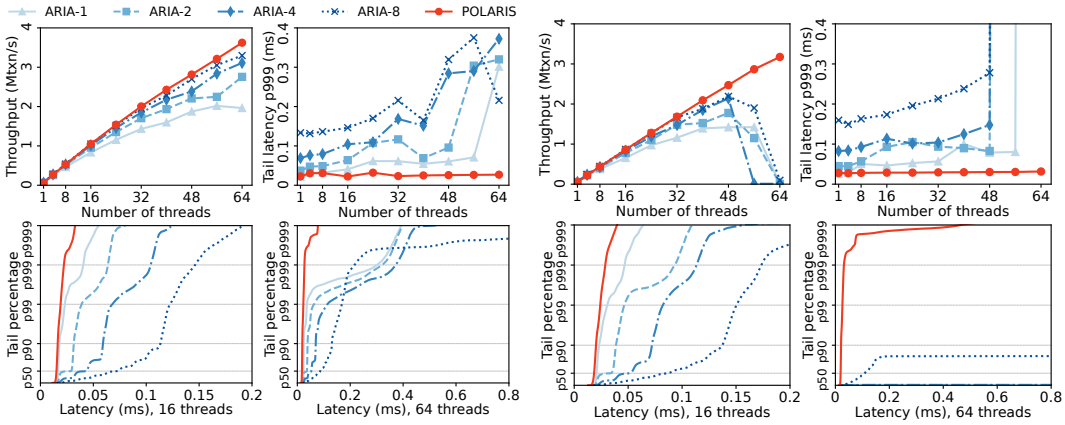
(a) Paper (b) Reproduced

Fig. 8. Throughput and p999 tail latency with over a spectrum of thread numbers; latency distribution in the cases of 16 and 64 threads (TPC-C, 64 warehouses).



(a) Paper (b) Reproduced

Fig. 9. Throughput and p999 tail latency over a spectrum of thread numbers; latency distribution in the cases of 16 and 64 threads (YCSB-A, r = 50%, w = 50%, θ = 0.99).



(a) Paper (b) Reproduced

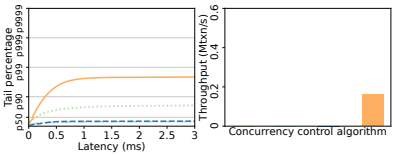
Fig. 10. Throughput and p999 tail latency over a spectrum of thread numbers; latency distribution in the cases of 16 and 64 threads (YCSB-A, r = 50%, w = 50%, θ = 0.5).

5 PORTABILITY EVALUATION ON SINGLE MACHINE

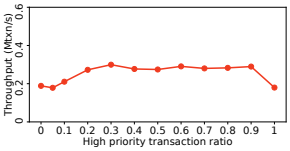
To overcome the issues caused by the shared resources in the Triton HPC cluster (i.e., the 39x2 threads reserved via slurm), e.g., the disk lags in Section 4, the reviewers also evaluated the reproducibility on a single workstation (i.e., SYS-7039A-i, Supermicro SuperWorkstation Mid-Tower), with 32 Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz cores, 250GB DDR4-2666 memory, and Ubuntu 18.04.6 LTS running on it.

It is interesting to observe that even though the performance drops when the number of threads requested by the script exceeds 32, most evaluation tasks can still be finished, and the turning points of the performance curves vary a lot in different evaluation tasks; for the same evaluation task, different baselines appear to own different performance turning points as well.

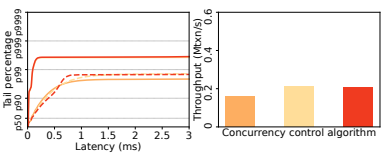
In general, most results can be successfully reproduced, even with 32 or fewer threads from this single machine. The reproduced curves are in good trends when compared with those reproduced in Section 4, based on which the reviewers conclude that they are enough to cover the core thesis of the paper.



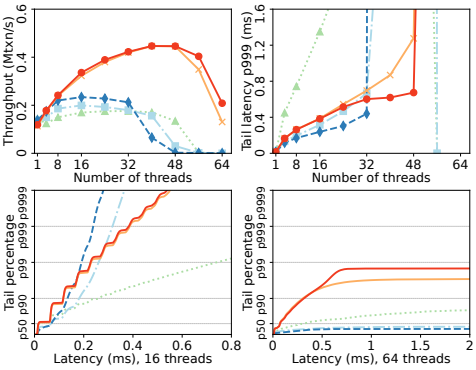
(a) Reproduced Fig. 1



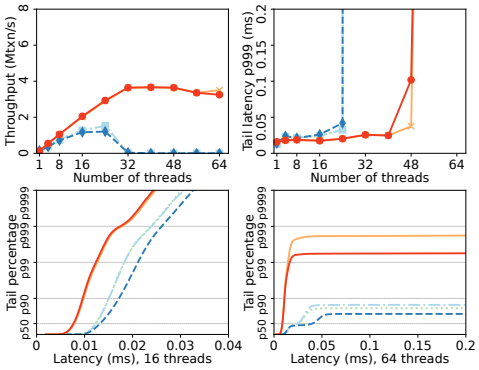
(b) Reproduced Fig. 2



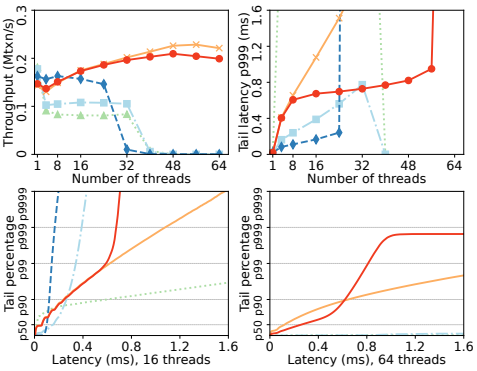
(c) Reproduced Fig. 6



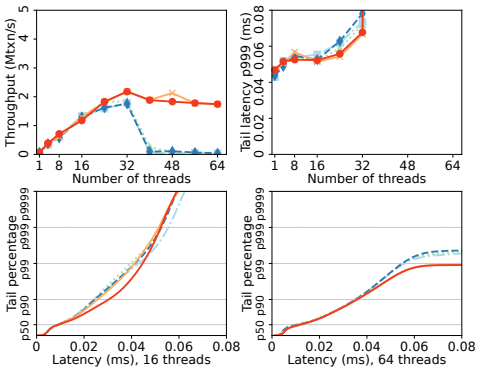
(d) Reproduced Fig. 3



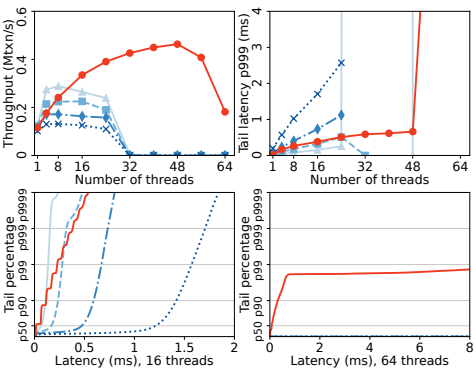
(e) Reproduced Fig. 4



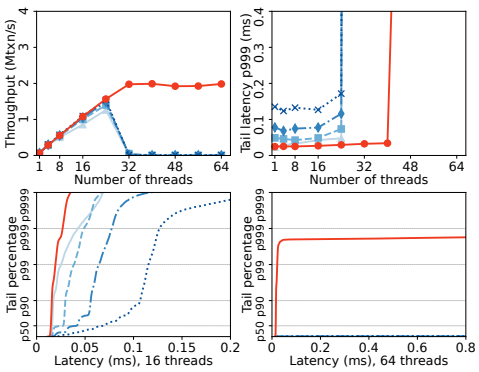
(f) Reproduced Fig. 7



(g) Reproduced Fig. 8



(h) Reproduced Fig. 9



(i) Reproduced Fig. 10

Fig. 11. Evaluation results of a single machine, labeled with the figure numbers in the original paper [1].

6 SUMMARY AND ACKNOWLEDGEMENT

We have reproduced all the figures and plots in the paper [1] on our platforms, which supports the research ideas, claims, and contributions presented by the paper.

The reviewers would like to acknowledge the authors for the detailed documents and automation scripts for reproducibility. For the four reviewers: Songlin and Bo evaluated the reproducibility on the HPC cluster and finished the first four sections of this report, Xiaodong and Yixiang evaluated the portability on a single workstation and finished Section 5 of this report. The reviewers worked independently and all agree to mark the paper as reproduced and award it the availability and reproducibility badges.

REFERENCES

[1] Chenhao Ye, Wuh-Chwen Hwang, Keren Chen, and Xiangyao Yu. 2023. Polaris: Enabling Transaction Priority in Optimistic Concurrency Control. *Proc. ACM Manag. Data* 1, 1, Article 44 (may 2023), 24 pages. https://doi.org/10.1145/3588724