

Eliminating Double Amplifications of Hashing Indexes for Persistent Memory

Paper ID: 98

1 SUMMARY OF IMPLEMENTATIONS

We implement our proposed ElimDA with C++. We elaborate on the dependent software, representative hash indexes, and testing tools as follows.

Dependent softwares: the implementation of ElimDA relies on a suite of software, including ndctl, ipmctl, Intel PMWatch, and PMDK, which can be reached via the following URLs.

```
$ wget https://github.com/pmem/ndctl
$ wget https://github.com/intel/ipmctl
$ wget https://github.com/intel/intel-pmwatch
$ wget https://github.com/pmem/pmdk
```

Representative hash indexes: we use four representative hash indexes for comparison, namely CCEH [3], Dash [2], level hashing [4] and clevel hashing [1], which can be downloaded via the following URLs. Note that we used the PMDK version of the level hashing implemented by Dash and add uniqueness check to CCEH for index correctness.

```
$ wget https://github.com/DICL/CCEH/tree/master/
CCEH-PMDK
$ wget https://github.com/pmdk/dash
$ wget https://github.com/chenzhangyu/Clevel-Hashing
```

Testing tools: we employ YCSB in our evaluation. It can be reached via the following URL.

```
$ wget https://github.com/brianfrankcooper/YCSB/
releases/download/0.17.0/ycsb-0.17.0.tar.gz
```

2 EVALUATION DETAILS

Hardware configurations: We conduct extensive experiments on a single machine equipped with two 2.10 GHz Intel Golden 5318Y CPUs (with 24 cores in total), 128 GB of DRAM memory and four Optane PMem DIMMs of 200 series (128 GB per DIMM and 512 GB in total). The Optane PMem DIMMs are configured in interleaved AppDirect Mode, which are connected to one processor. Figure 1 shows the major hardware configurations of our testbed and Figure 2 shows detailed information about the Optane PMem used in our evaluation.

We perform extensive testbed experiments, including (i) experiments to understand the property of ElimDA, and (ii) experiments to understand the sensitivity of ElimDA.

Experiments with YCSB: We measure the access performance of ElimDA, CCEH, Dash, level hashing, and clevel hashing using YCSB. We generate the workloads through YCSB and then modify their format so that they can be used for testing. We then use these workloads to test each hashing index.

The following script is an example of how to generate an insert-only workload using YCSB:

```
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
Address sizes: 46 bits physical, 57 bits virtual
CPU(s): 48
On-line CPU(s) list: 0-47
Thread(s) per core: 1
Core(s) per socket: 24
Socket(s): 2
NUMA node(s): 2
Vendor ID: GenuineIntel
CPU family: 6
Model: 106
Model name: Intel(R) Xeon(R) Gold 5318Y CPU @ 2.10GHz
Stepping: 6
CPU MHz: 800.779
CPU max MHz: 3400.0000
CPU min MHz: 800.0000
BogoMIPS: 4200.00
Virtualization: VT-x
L1d cache: 2.3 MiB
L1i cache: 1.5 MiB
L2 cache: 60 MiB
L3 cache: 72 MiB
```

Figure 1: Configurations of our testbed.

DimmID	Capacity	LockState	HealthState	FWVersion
0x0010	126.742 GiB	Disabled, Frozen	Healthy	02.02.00.1516
0x0210	126.742 GiB	Disabled, Frozen	Healthy	02.02.00.1516
0x0110	126.742 GiB	Disabled, Frozen	Healthy	02.02.00.1516
0x0310	126.742 GiB	Disabled, Frozen	Healthy	02.02.00.1516

Figure 2: The configurations of the Intel Optane PMem DIMMs of 200 series used in the evaluation.

```
#!/bin/bash
$ ycsb load basic -P uniformworkload > uniform/load.text
$ ycsb run basic -P insertworkload > uniform/insert.text
We run the following script to modify the format of workloads.
#!/bin/bash
$ dir=./uniform/load.txt
$ sed -i '1,17d' $dir
$ sed -i 's/READ usertable user/r /' $dir
$ sed -i 's/INSERT usertable user/i /' $dir
$ sed -i 's/UPDATE usertable user/u /' $dir
$ sed -i 's/\[ field0=/ /' $dir
$ sed -i 's/\] / /' $dir
```

After modifying the format, we measure the performance of all hashing indexes. Due to space limits, we present the following script to evaluate ElimDA. The experiments with other hashing indexes are similar.

```
#!/bin/bash
dir=/mnt/pmem1-node0/pool1
loadsize=50000000
runsize=50000000
FLAG='-mavx -mavx2 -mavx512f -mavx512bw -mavx512vl'

g++ -O3 -std=c++17 -I./ -lrt -c -o src/ElimDA.o -g \
src/ElimDA.cpp -DINPLACE -lpmemobj -lpmem \
```

```

$FLAG -lprofiler -lunwind

g++ -O3 -std=c++17 -I./ -lrt -o bin/run_all_ycsb \
-g src/run_all_ycsb.cpp src/ElimDA.o include/dlock.o \
-lpmemobj -lpmem -lpthread -DMULTITHREAD -lpqos

outdir=ycsb_all
for test in 1 2 3 4
do
    for oper in "w50r50.txt"
    do
        for num in 1 2 4 8 16 24
        do
            echo " " >> $outdir
            echo "test zipfian "$num" "$oper >> $outdir
            ycsb_dir="./ycsb/zipfian/"$oper
            load_dir="./ycsb/zipfian/load.txt"
            rm -rf $dir

            numactl -N 0 -m 0 \
            ./bin/run_all_ycsb $dir $loadsize $runsize
            $num $ycsb_dir $load_dir
            |grep -a -E "num|Ops|Start" >> $outdir
        done
    done
done

```

Experiments on Sensitivity: We also use the impact of the P-bucket size (Exp#12) as an example to clarify how we assess the performance of ElimDA. Other experiments are similar.

```

#!/bin/bash
dir=/mnt/pmem1-node0/pool1
num=1
loadsize=500000000
FLAG='-mavx -mavx2 -mavx512f -mavx512bw -mavx512vl'
outdir=./sendata
srcdir=./sensitive.cpp
for testnum in 1 2 3 4 5
do
    for sbucketsize in 64 128 256 512 1024
    do
        sed -i '27d' ./src/ElimDA.h
        sed -i '27i #define S_Bucket_Size '\
        $sbucketsize' ./src/ElimDA.h
        rm -rf $dir

        g++ -O3 -std=c++17 -I./ -lrt -c \
        -o src/ElimDA.o -g src/ElimDA.cpp \
        -DINPLACE -lpmemobj -lpmem \
        $FLAG -lprofiler -lunwind

        g++ -O3 -std=c++17 -I./ -lrt \
        -o bin/multi_threaded_ElimDA -g \
        src/test.cpp src/ElimDA.o include/dlock.o \
        -lpmemobj -lpmem -lpthread \
        -DMULTITHREAD -lpqos
    done
done

```

```

numactl -N 0 -m 0 \
./bin/multi_threaded_ElimDA $dir $loadsize $num \
|grep -a "Ops/sec" >> $outdir
done
sed -i '27d' ./src/ElimDA.h
sed -i '27i #define S_Bucket_Size 256' ./src/ElimDA.h
done

```

REFERENCES

- [1] Zhangyu Chen, Yu Hua, Bo Ding, and Pengfei Zuo. Lock-free concurrent level hashing for persistent memory. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 799–812, 2020.
- [2] Baotong Lu, Xiangpeng Hao, Tianzheng Wang, and Eric Lo. Dash: Scalable hashing on persistent memory. *Proceedings of the VLDB Endowment*, pages 1147–1161, 2020.
- [3] Moohyeon Nam, Hokeun Cha, Young-ri Choi, Sam H Noh, and Beomseok Nam. Write-optimized dynamic hashing for persistent memory. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 31–44, 2019.
- [4] Pengfei Zuo, Yu Hua, and Jie Wu. Write-optimized and high-performance hashing index scheme for persistent memory. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 461–476, 2018.