Efficient Algorithms for Persistent Transactional Memory

There are three different benchmarks in this artifact: one for evaluating multiple concurrent persistent data structures, and two benchmarks for KV stores (db_bench and YCSB). Building these benchmarks requires a compiler with C++14 support.

To validate the performance results of our implementations, the data structure benchmarks require at least 2 GB of Optane DC Persistent Memory, while the KV benchmarks need 32 GB.
However, some of the other KV stores used in our comparisons (such as RedoDB) have a higher memory usage and therefore, require 122 GB of Optane.

1. Required Libaries

1.1. PMEMKV

pmemkv requires the PMDK library. To build and install PMDK follow the instructions in https://github.com/pmem/pmdk

```
        cd ~
        git clone https://github.com/pmem/pmdk.git
        sudo apt install g++ gcc-9 g++-9 make python pandoc pkg-config libndctl-
dev libdaxctl-dev graphviz asciidoc asciidoctor doxygen m4 clang cmake libgtest-
dev libgflags-dev
        cd pmdk
        make
        sudo make install
```

Next install https://github.com/pmem/libpmemobj-cpp

```
        cd ~
        git clone https://github.com/pmem/libpmemobj-cpp.git
        cd libpmemobj-cpp
        mkdir build
        cd build
        cmake ..
        make
        sudo make install
```

Install https://github.com/memkind/memkind

```
        sudo apt install libnuma-dev autoconf libtool
        cd ~
        git clone https://github.com/memkind/memkind.git
        cd memkind
        ./autogen.sh
        ./configure
        make
        sudo make install
```

Followed by installing pmemkv https://github.com/pmem/pmemkv

```
sudo apt install libtbb-dev rapidjson-dev valgrind
cd ~
git clone https://github.com/pmem/pmemkv.git
cd pmemkv
mkdir ./build
cd ./build
cmake ..
make
sudo make install
```

After it is also necessary to install pmemkv-tools https://github.com/pmem/pmemkv-tools

```
cd ~
git clone https://github.com/pmem/pmemkv-tools.git
cd pmemkv-tools
make bench
sudo ldconfig
```

You can confirm the pmem pool can be created using the following command:

```
pmempool create -l "pmemkv" -s 2G obj /mnt/pmem0/pmemkv_pool
```

The binary for PMEMKV db_bench is expected to be placed in `~/pmemkv-tools/pmemkv_bench`

1.2. ROCKSDB

To install RocksDB 6.5 follow the instructions on https://github.com/facebook/rocksdb/blob/master/INSTALL.md

```
cd ~
sudo apt install libbz2-dev liblz4-dev libz-dev libsnappy-dev libzstd-dev
git clone https://github.com/facebook/rocksdb.git
cd rocksdb/
git checkout remotes/origin/6.5.fb
make release
```

The binary for RocksDB db_bench is expected to be placed in `~/rocksdb/db_bench`

1.3. PRONTO

Pronto repository can be found here, https://zenodo.org/record/3605351/files/pronto-v1.1.tar.gz

Unfortunately, Pronto requires several modifications in order to be able to run. We do not have a step by step instruction guide.

2. Data Structure Benchmarks

2.1. Building the data structure benchmarks

After unzipping `durabletx.zip`, to build the data structures benchmarks go into the `graphs` folder and type `make`

```
cd graphs/
make
```

This will generate the following benchmarks executables in the `graphs/bin/` folder.

```
bin/pq-ll-romlogfc
bin/pq-ll-oflf
bin/pq-ll-pmdk
bin/pq-ll-trinityfc
bin/pq-ll-trinityvrfc
bin/pq-ll-trinityvrtl2
bin/pq-ll-quadrafc
bin/pq-ll-quadravrfc

bin/pset-tree-1m-romlogfc
bin/pset-tree-1m-oflf
bin/pset-tree-1m-pmdk
bin/pset-tree-1m-trinityfc
bin/pset-tree-1m-trinityvrfc
bin/pset-tree-1m-trinityvrtl2
bin/pset-tree-1m-quadrafc
bin/pset-tree-1m-quadravrfc

bin/pset-btree-1m-romlogfc
bin/pset-btree-1m-oflf
bin/pset-btree-1m-pmdk
bin/pset-btree-1m-trinityfc
bin/pset-btree-1m-trinityvrfc
bin/pset-btree-1m-trinityvrtl2
bin/pset-btree-1m-quadrafc
bin/pset-btree-1m-quadravrfc

bin/pset-ziptree-1m-romlogfc
bin/pset-ziptree-1m-oflf
bin/pset-ziptree-1m-pmdk
bin/pset-ziptree-1m-trinityfc
bin/pset-ziptree-1m-trinityvrfc
bin/pset-ziptree-1m-trinityvrtl2
bin/pset-ziptree-1m-quadrafc
bin/pset-ziptree-1m-quadravrfc
```

2.2. Running the data structure benchmarks

Still in the `graphs/` folder, type `'./run-conc.py'`. This will run the relevant benchmarks and save the results of each benchmark in `data/<filename>.txt`

```
./run-conc.py
```

3. KV store benchmarks

3.1. Building the KV store benchmarks (DB_BENCH and YCSB)

DB_BENCH

To build the KV store db_bench benchmarks for TrinityDB(TL2) and TrinityDB(FC) go into the ptmdb folder and type make

```
cd ptmdb/
make
```

This will generate the following benchmarks executables in the `ptmdb/bin/` folder.

```
bin/db_bench_trinvrfc

bin/db_bench_trinvrtl2
```

To build the KV Store db_bench benchmarks for RedoDB go into the `ptmdb/otherdb/redodb` folder and type `make`

```
cd ptmdb/otherdb/redodb
make
```

This will generate the following executable in the `ptmdb/otherdb/redodb/bin/` folder.

```
ptdmdb/otherdb/redodb/bin/db_bench_redoopt
```

3.2. Run KV store benchmarks for db_bench

To run all KV store database for the db_bench benchmarks

```
./run-db.py
```

All the outputs of db_bench are available in `ptmdb/results_db_bench_<kv_name>.txt`

3.3. Run KV store benchmarks for YCSB

Generate the files with workloads A and B for YCSB:

```
cd ptmdb/
./generate-ycsb.py
```

This will create the necessary files in `ptmdb/workloads/1m/`

To run all KV store database for the db_bench benchmarks

```
./run-ycsb.py
```

All the outputs of YCSB are available in `ptmdb/results_ycsb_<kv_name>.txt`