# Readme for reproducibility submission of paper ID 338

## A Source code info

**Repository** We provide one repository containing scripts for reproducing the experiments and two repositories containing source code. You only need to clone the first repository.

- The *reproducibility repository* is available under `https://github.com/jmuehlig/mxtasking-reproducibility`.
- The code repositories for *(a)* `MxTasking` and *(b)* comparative tree benchmarks using threads, Intel TBB, etc. are available under

  *(a)* `https://github.com/jmuehlig/mxtasking`

  *(b)* `https://github.com/jmuehlig/btree-benchmarks`.

**Programming Language** *C++* (17) for benchmarks, *Python* (3) for scripts, and *Java* for YCSB workload generation.

**Compiler Info** Clang in version *10.0.0-4ubuntu1*.

**Packages/Libraries Needed** You can verify all dependencies installed using the `check.sh` script.

If you are using *Ubuntu*, you can install all dependencies by executing `install_dependencies.sh`.

- `CMake` in version $\geq$ 3.10 for generating the Makefile
- `Clang` in version $\geq$ 10 for compilation
- Package `libnuma-dev`
- `Python` in version $\geq$ 3 for generating plots
- `curl` for downloading the YCSB workload
- `Java` for generating the YCSB workload
- Package `libtcmalloc-minimal4`
- Packages `libjemalloc-dev` and `libjemalloc2`
- `Git` for downloading the repositories
- `Gnuplot` for plotting
- `Perf`
- `Intel VTune` for cycle-based analysis

## B Datasets info

**YCSB** We used the YCSB workload (in version 0.16.0) for the tree benchmarks. The script will download and build the workload automatically. The used workload specifications are `workloada` and `workloadc`, located in `workloads/ycsb`. The official repository is available at `https://github.com/brianfrankcooper/YCSB`. *This is just for your information, you do not have to change anything.*

**TPC-H** For the hash join experiment, we used the TPC-H benchmark with **scale factor** 100. The relevant files are `customer.tbl` and `orders.tbl`. **Notice**:

- Please download[1] and compile the TPC-H `dbgen` tool and set the directory containing the executable in `setup_environment.sh` (variable `DIR_TPCH_DBGEN`)
- **or** copy both `customer.tbl` and `orders.tbl` files into `workloads/tpch`.

If you set the `DIR_TPCH_DBGEN` variable, the script will automatically generate the needed files with scale factor 100 and move them into `workloads/tpch`.

## C Hardware Info

**Processor** Two sockets with CPU *Intel(R) Xeon(R) Gold 6226 CPU @ 2.70GHz* (12 physical cores per socket, 24 logical cores per socket).

**Caches** The cache sizes for the entire system are specified.

- *L1d*: 768 KiB (32 KiB per physical core)
- *L1i*: 768 KiB (32 KiB per physical core)
- *L2*: 24 MiB
- *L3*: 38.5 MiB (19.25 MiB per socket)

**Memory**

- 6 DIMMs per socket (12 DIMMs total)
- Size: 15.26 GiB per DIMM (183 GiB total)
- Max Capable Speed: 2933 MHz

**Secondary Storage** Storage is not used.

**Network** Network is not used.

## D Experimentation Info

We will describe how to use the scripts to reproduce the experiments, followed by a description of the files from the reproducibility project. In our experiments, we used Ubuntu 20.04 as the operating system.

[1] `http://tpc.org/tpc_documents_current_versions/download_programs/tools-download-request5.asp?bm_type=TPC-H&bm_vers=3.0.0&mode=CURRENT-ONLY`

### Instructions

**Before running the experiments** Please note before running the experiments.

- Please install all dependencies listed in A.
  - You might execute `check.sh` to verify all dependencies installed.
  - If you are running *Ubuntu* as the OS, you can execute `install_dependencies.sh` to automatically install missing dependencies, including Intel VTune.
- Please disable *NUMA balancing* by setting `/proc/sys/kernel/numa_balancing` to `0`. *This will be done by the scripts, if you execute them as a root user.*
- Please set `/proc/sys/kernel/perf_event_paranoid` to `-1` to allow the usage of (almost) all performance events. The performance counters are read by the executables. *This will be done by the scripts, if you execute them as a root user.*
- Please set `/proc/sys/kernel/kptr_restrict` to `0` when you are not executing the scripts as a root user.
- When executing the scripts as a root user, the scripts will enable CPU performance mode before running the benchmarks and disable it afterwards. *It is not required to run the scripts as root, however, for the experiments in our paper, we enabled the CPU performance mode.*

**Running all at once** You can run all experiments at once by executing `run_experiments.sh`. This will execute all the `run_*` files described next. However, please note the description for every single script below.

**Running one experiment after the other** You do not need to execute the following scripts, if you execute `run_experiments.sh`.

- The script `run_tree_experiments.sh` will produce Figure 10, Figure 11, and Figure 12 of the paper. Please note that the experiments might run for a long time ($\sim$ 36 hours in our setting).
- The script `run_hashjoin_experiments.sh` will produce Figure 9 of the paper. This should take about one hour.
  - **Please note:** You need to set `DIR_TPCH_DBGEN` (in `setup_environment`.sh) **or** generate the TPC-H files *customer.tbl* and *orders.tbl* on your own and copy them into `workloads/tpch`.
- The script `run_cycle_experiments.sh` will produce Figure 7 and Figure 13 of the paper.
  - **Please note:** You need to install *Intel VTune*[2] and set the variable `VTUNE_VARS_SCRIPT` in `setup_environment.sh` pointing to the *VTune* source script.

[2] `https://www.intel.com/content/www/us/en/develop/documentation/installation-guide-for-intel-oneapi-toolkits-linux/top/installation.html`

**Results** The plots will be stored in `plots/` and named like the Figures within the paper.

### Configuration

There are some configurations that might be necessary depending on the underlying system. The default configuration should match the system described in C.

**Library path** Some executables need specific allocation libraries `tcmalloc` and `jemalloc`. You can specify their location by changing the value of `DIR_LIB` in `setup_environment.sh`.

**Performance Counter** We use different performance counter to better interpret the throughput-based results. One of them is the *stalled memory cycles* counter (`CYCLE_ACTIVITY.STALLS_MEM_ANY`). In our experience, this counter can be addressed differently on different architectures. The counter can be configured

- in `projects/mxtasking/src/benchmark/perf.cpp` (line 39) for `MxTasking`
- and in `project/tree-bench/src/util/perf.h` (line 104) for the tree benchmarks.

### Description

The project includes scripts for downloading all sources and building executables needed for the experiments.

**Shipped files** Besides the mentioned scripts, the following files and folders are also included into the project.

- `workloads/` contains all files needed to build/run the workloads.
- `scripts/` contains some helper scripts for plotting and processing the raw output of the executables.
- `clear.sh` will remove all results, plots and downloaded sources.
- `check.sh` will check all dependencies and list missing libraries and packages.

**Generated files** The following files and folders are generated by running the scripts.

- `benchmark-results/` will contain the output generated by running the experiments and processed data (from the raw data) for plotting.
- `projects/` will contain the source code of *(a)* `MxTasking` and *(b)* a project for comparable tree experiments (thread-based B$^{\text{link}}$-tree, Intel TBB-based B$^{\text{link}}$-tree, OLCTree, open BwTree, and Masstree).
- `plots/` will contain all generated plots. The plots are named **Fig-*i*.pdf** where *i* is the number of the Figure within the paper.