Artifact for Non-Blocking Interpolation Search Trees with Doubly-Logarithmic Running Time

Getting Started Guide

We prepared a Docker container that can be used to run the experiments from the paper. The requirements are:

- A relatively recent Linux distribution (we used Ubuntu to prepare the artifact).
- An installation of a recent Docker (see instructions further below).
- A multicore CPU. We used a 4-socket system, with four Intel 8160 CPUs (192 threads total). A less parallel CPU will also suffice, although you will replicate less of our experiments.
- If you want to reproduce all the results, we suggest to have 384 GB of RAM memory, which is what our machine had available. If you have less RAM, it is still possible to reduce the dataset sizes in most experiments (for details, see instructions in the Step-by-Step guide).

The steps to download and run the artifact are as follows (note -- you might have to use `sudo` for Docker):

1. Install a newer version of Docker to your system. For example, we are using `Docker version 19.03.5, build 633a0ea838` . You can run:

   ```
   $ docker --version
   ```

   to check the version. If you are using the Ubuntu distribution, see the instructions here: https://docs.docker.com/install/linux/docker-ce/ubuntu/

2. Download the `ppopp20-artifact.tar.gz` file that is the docker image from: https://www.dropbox.com/s/xs34t8mmi53e6oh/paper-241.tar.gz?dl=0

3. Load the docker image from the downloaded file:

   ```
   $ docker load -i paper-241.tar.gz
   ```

4. Run the following to check that the image was loaded:

   ```
   $ docker images
   ```

5. Start a Docker container from the artifact image. Note that you *have to run in the privileged mode* so that the artifact can use thread-to-CPU pinning and some other facilities. Run the following:

   ```
   $ docker run -i -t --privileged ppopp20-artifact /bin/bash
   ```

6. Go to the `root/artifact` folder:

   ```
   $ cd /root/artifact
   ```

7. Run `ls` . You should see several folders, in particular `microbench` and `macrobench` . In this image, the source files have already been compiled for both the microbenchmarks and the macrobenchmarks, but if you want to compile them again, you should delete `microbench/bin` and `macrobench/bin` , and then run `microbench/compile.sh` and `macrobench/compile.sh` .

8. To test that you can run the benchmarks, please first go to `microbench/experiments` . Then run `ls` . You will see several folders, one for each experiment. Run:

   ```
   $ cd istree_exp1_scaling_threads
   ```

   to enter the first experiment. If necessary, make the `run.sh` script executable like this: `chmod a+x run.sh` . Then run the script:

   ```
   $ ./run.sh
   ```

   You should see output like this.

   ```
   Estimated 5 hours to run
   filename,DS_TYPENAME,size_node,RECLAIM,ALLOC,POOL,MILLIS_TO_RUN,INS,DEL,RQ,RQSIZE,MAXKEY,PREFILL_THREADS,TOTAL_THRE
   step 10001/10128: istree_exp1_scaling_threads/step10001.txt,brown_ext_ist_lf,72,reclaimer_debra,allocator_new,pool_


   step 10002/10128: istree_exp1_scaling_threads/step10002.txt,brown_ext_ist_lf,72,reclaimer_debra,allocator_new,pool_
   ...
   ```

   Press CTRL-Z, run `top` and kill the `run.sh` process.

9. Now go to the `macrobench/experiments` folder, and the macro benchmark.

   ```
   $ cd /root/artifact/macrobench/experiments/istree_exp1
   $ ./run.sh "< thread-counts >"
   ```

where the `< thread-counts >` is the list of thread counts you want to run it with. For example, if your CPU has 8 cores, you can run it with `"2 4 8"`. Please make sure to include the double quotes. Also, do not use thread counts larger than the number of CPUs, because the `taskset` command, which is used in the benchmark, will fail. You should see output like this:

```
Estimated running time of the experiment is 2 hours.

                      alg             nthreads                theta              runtime           throughput
...
```

Press CTRL-Z, run `top` and kill the `run.sh` process.

If you managed to run these steps, then you should be able to run the experiments.

Step-by-Step Instructions

The artifact supports the following claims from the paper:

- `microbench/experiments/istree_exp1_scaling_threads` -- supports Figure 9., which shows the throughput of the basic operations for different data structures. See also Figure 15. in the appendix.
- `microbench/experiments/istree_exp2_memory_static` -- supports Figure 13., which shows the memory footprints of the different data structures.
- `microbench/experiments/istree_exp3_disable_multicounter` -- supports Figure 19. from the appendix, which evaluates the effect of multicounters.
- `microbench/experiments/istree_exp4_disable_rebuild_helping` -- supports Figure 12., which shows the effect of collaborative rebuilding.
- `microbench/experiments/istree_exp6_rebuilding_time` -- supports Figure 11., which shows the amount of time spent in C-IST rebuilding vs in other operations. See also Figures 17. and 18. in the appendix.
- `microbench/experiments/istree_exp9_zipf` -- supports Figure 21. from the appendix, which shows the performance of basic ops on Zipfian key distributions.
- `macrobench/experiments/istree_exp1` -- supports Figure 20. form the appendix, which shows the performance of different data structures on the YCSB database benchmark.

Of these experiments, the most important is `microbench/experiments/istree_exp1_scaling_threads`.

To run each of these experiments:

1. Enter the respective folder.
2. Run the `./run.sh` script (setting permissions if necessary).
3. After the experiment completes, inspect the CSV files in that folder.

Note that, for each experiment, you can modify the `run.sh` scripts to change various parameters of the experiment. For example, in `istree_exp1_scaling_threads`, you can change the `thread_counts` variable to manually set the thread counts that you want to run with. Similarly, in `istree_exp2_memory_static`, you can change `key_range_sizes` to change the key counts.

When an experiment completes, it will produce a CSV file, which contains all the data. You can manually inspect the numbers in each CSV file, or you can open the Excel spreadsheet in each folder, paste the CSV file contents into it, and the graphs will automatically be generated for you if you click on the "refresh all" button (you need Excel macros to run for this, you can check them if you press ALT-F11).

The artifact technically does not reproduce the hardware counter performance, since we did not manage to run them within Docker. However, if you copy the folder with the artifact to your host system (`docker cp`), recompile everything there, and re-run the benchmarks, then you should be able to reproduce the hardware counter numbers (this data is included the CSV file of microbenchmark `istree_exp1_scaling_threads`).