Artifact Appendix of Unison

A Artifact Appendix

A.1 Abstract

This artifact includes Unison's implementation based on ns-3.36.1, along with experiments for profiling and performance comparison between Unison and existing PDES approaches.

A.2 Description & Requirements

A.2.1 How to access. The artifact is publicly visible at the GitHub repository of our organization via https://github.com/NASA-NJU/Unison-for-ns-3.

A.2.2 Hardware dependencies. In order to run all experiments of Unison exactly, you should have at least 144 CPU cores and 512GB of memory. These computation resources can come from either a single host with many CPU cores (*e.g.*, ARM or AMD Threadripper) or multiple identically configured hosts within a LAN cluster.

However, most of the experiments can be performed on a single commodity computing server with at least 24 CPU cores and 128GB of memory. Moreover, about half of the experiments can be performed just on a single commodity PC with at least 8 CPU cores and 16GB of memory.

If your current hardware doesn't meet the very first requirements (144 cores, 256GB of memory), we provide some alternative small-scale experiments that can be run on a single commodity PC (8 cores, 16GB of memory) or a single commodity computing server (24 cores, 128GB of memory) and reflect similar phenomena at a large scale.

We also provide SSH access to our computing cluster with 6 hosts for the AEC. Each host has 24 CPU cores and 256GB of memory. Together, they satisfy all the hardware requirements above and they are already set up and installed with all software dependencies.

A.2.3 Software dependencies. A Unix-like operating system installed with Python 3.8 and above, Git, CMake, Linux Perf, OpenMPI library and NFS server/client is required to compile and run Unison and all of its related experiments. The minimum compiler versions supported by Unison are g++-7, clang-10 and Xcode 11. We recommend to run our experiments with Ubuntu 22.04. To plot the figures presented in the paper from the experiment data, a TeX installation with the PGFPlots package is required.

A.2.4 Benchmarks. Our artifact requires WAN network topology data from the Internet Topology Zoo and traffic CDF data from previous publications. We already packed these data into the artifact repository. You can find these data in `scratch/topos` and `scratch/cdf` in the `unison-evaluations` branch.

A.3 Set-up

You should first get the source code of Unison by cloning its GitHub repository to your host. Then, switch to the `unison-evaluations` branch and follow the `README.md` file in that branch to set up.

A.4 Evaluation workflow

A.4.1 Major Claims. Here we list all of our major claims below. It is notable that our artifact does not validate Figure 9 and Figure 12, since they both 1) have to modify and evaluate other artifacts of ML-based data-driven simulators and 2) require expensive GPUs to run these ML-based simulators and we cannot provide such hardware to the AEC persistently during the review. The validation of the figures not mentioned in this section is included in our artifact, but they are relatively less important so we do not treat them as major claims to save the space.

Claim 1. *Unison can achieve 10× speedup over existing PDES approaches.* This is proven by Exp 1 and Exp 2 whose results are reported in Figure 1.

Claim 2. *The synchronization time of existing PDES approaches gradually dominates as the traffic inhomogeneity increases.* This is proven by Exp 3 whose results are reported in Figure 5a.

Claim 3. *The synchronization time ratio is high in a transient time window for existing PDES approaches, even if the traffic pattern is balanced in macro.* This is proven by Exp 4 whose results are reported in Figure 5b.

Claim 4. *The synchronization time is long for low-latency and high-bandwidth networks for existing PDES approaches.* This is proven by Exp 5 and Exp 6 whose results are reported in Figure 5c and Figure 5d.

Claim 5. *Unison can significantly reduce the synchronization time to near-zero.* This is proven by Exp 7 and Exp 8 whose results are reported in Figure 10a and Figure 10b.

Claim 6. *Unison exhibit super-linear speedup and its parallelism is flexible to set.* This is proven by Exp 9, Exp 10 and Exp 11 whose results are reported in Figure 9.

Claim 7. *Unison is also fast with other topologies and under different traffic patterns.* This is proven by Exp 12, Exp 13 and Exp 14 whose results are reported in Figure 11b.

Claim 8. *The output of Unison is deterministic under multiple runs while other PDES approaches are not.* This is proven by Exp 15 whose results are reported in Figure 13.

Claim 9. *Fine-grained partition of Unison can reduce cache misses which can further reduce the simulation time.* This is proven by Exp 16 whose results are reported in Figure 14a.

Claim 10. *The default scheduling metric of* Unison *performs better than others and without scheduling.* This is proven by Exp 17 whose results are reported in Figure 14b.

A.4.2 Experiments. It is easy to run experiments of our artifact. We provide a script `exp.py` and all you need to do is pass an argument indicating the experiment name. The results of the experiments are saved in the `results` folder by their name and timestamps when launched. We also provide `process.py` to convert these results into CSV corresponding to the figure data in each claim for ploting. Here we list all the experiments required to validate our major claims. We also list the corresponding name and its expected machine time for each experiment.

Exp 1 (`fat-tree-distributed`, 7d)**.** This experiment will simulate 48-cluster to 144-cluster fat-trees under incast traffic on multiple hosts, assuming the number of cores used in total is equal to the number of clusters and each host has 24 cores. If your hosts have a different number of cores (*e.g.*, 16), you can change the cluster parameter to 32, 48, 64, *etc.* 24 clusters (cores) should be enough to produce the 10× relative speedup result. Therefore if you do not have so many cores, you can run a small-scale experiment with 8 to 24 clusters, but the relative speedup is only about 6× at 8 clusters.

Exp 2 (`fat-tree-default`, 4d)**.** This experiment will use the default sequential simulation kernel to run Exp 1. If you have changed any parameters of Exp 1, please adjust these parameters in this experiment accordingly.

Exp 3 (`mpi-sync-incast`, 18h)**.** This experiment runs a $k = 8$ fat-tree with existing PDES algorithms using 8 cores under different incast traffic ratios, and records the average P, S and M of every LP. It is expected that S will increase to over 70% of the total time as the incast traffic ratio increases.

Exp 4 (`mpi-sync`, 1h)**.** This experiment runs a $k = 8$ fat-tree with the barrier synchronization algorithm using 8 cores under balanced traffic, and records the S ratio of each round. It is expected that the S ratio will fluctuate and will be above 20% for most of the time.

Exp 5 (`mpi-sync-delay`, 20min)**.** This experiment runs a $k = 8$ fat-tree with existing PDES algorithms using 8 cores under different link delay, and records the average S ratio of every LP. It is expected that the S ratio will decrease as the link delay increases.

Exp 6 (`mpi-sync-bandwidth`, 10min)**.** This experiment runs a $k = 8$ fat-tree with existing PDES algorithms using 8 cores under different link bandwidth, while keeping the same traffic load, and records the average S ratio. It is expected that the S ratio will increase as the link bandwidth increases.

Exp 7 (`mtp-sync-incast`, 3h)**.** This experiment runs a $k = 8$ fat-tree with Unison using 8 threads under different incast traffic ratios, and records the average P, S and M of every thread. It is expected that S is less than 5% for every case.

Exp 8 (`mtp-sync`, 40min)**.** This experiment runs a $k = 8$ fat-tree with Unison using 8 threads under balanced traffic, and records the S ratio of each round. It is expected that the S ratio will be near zero in almost every round.

Exp 9 (`flexible`, 1d)**.** This experiment runs a $k = 8$ fat-tree with Unison using 2-24 threads. It is expected that 24 threads can achieve about 2.5× speedup relative to 8 threads. If you do not have so many cores, you can run a small-scale experiment (a $k = 4$ fat-tree) with 2-8 threads.

Exp 10 (`flexible-barrier`, 3d)**.** This experiment runs a $k = 8$ fat-tree with the barrier synchronization algorithm using 2-8 cores. It is expected that the barrier synchronization algorithm is slower than Unison in Exp 9 under the same number of cores. If you do not have so many cores, you can run a small-scale experiment (a $k = 4$ fat-tree) with 2-4 cores.

Exp 11 (`flexible-default`, 1d)**.** This experiment will use the default sequential simulation kernel to run Exp 9 and Exp 10. If you change any parameters of Exp 9 and Exp 10, please adjust these parameters in this experiment accordingly.

Exp 12 (`bcube`, 40min)**.** This experiment runs a 3-level $n = 8$ BCube with Unison using 8 and 16 threads. It is expected that 16 threads can achieve about 1.3-1.6× speedup relative to 8 threads.

Exp 13 (`bcube-old`, 2h)**.** This experiment runs a 3-level $n = 8$ BCube with Unison existing PDES algorithms 8 cores. It is expected that the existing PDES algorithms are slower than Unison in Exp 12.

Exp 14 (`bcube-default`, 1d)**.** This experiment will use the default sequential simulation kernel to run Exp 12 and Exp 13. If you change any parameters of Exp 12 and Exp 13, please adjust these parameters in this experiment accordingly.

Exp 15 (`deterministic`, 4h)**.** This experiment runs a $k = 8$ fat-tree with Unison and existing PDES algorithms 20 times. It records the number of processed events and flow statistics. It is expected that Unison's event count and flow statistics are the same across multiple runs, while other PDES algorithms are not.

Exp 16 (`partition-cache`, 1d)**.** This experiment runs a 12×12 torus with Unison using only 1 thread. However, the automatic partition of Unison is disabled and we change the granularity of the partition (*i.e.*, number of LPs) manually. It is expected that Unison's cache miss and simulation time are reduced while the partition granularity increases.

Exp 17 (`scheduling-metrics`, 4h)**.** This experiment runs a $k = 8$ fat-tree with Unison with different scheduling metrics. It is expected that the slowdown factor of Unison's default scheduling metric `ByExecutionTime` is the smallest among others.