Fray Artifact Evaluation

DOI 10.5281/zenodo.15724289

Introduction

This repository contains artifacts to reproduce the paper "Fray: An Efficient General-Purpose Concurrency Testing Platform for the JVM". This README only includes the instructions to run the evaluation. The documentation of the Fray project is available in the Fray repository and docs folder. We claim the following badge for the paper:

- **Artifact Available**: Both Fray and Fray Benchmark are available on GitHub. Fray uses Nix to provide reproducible builds.
- **Reusable**: Fray is designed to be reusable in real-world applications. It can be used as a JUnit test to find concurrency bugs in Java applications. The Fray debugger plugin is also available for IntelliJ IDEA. The Fray documentation provides detailed instructions on how to use Fray in real-world applications and how to extend Fray to support new concurrency testing algorithms.
- **Results Reproduced**: The results of the paper are reproduced in this repository.

Hardware Dependencies

Hardware

- x86 CPU with at least 6 cores
- 32 GB of RAM

JPF and Fray

- You can run evaluation on any Linux and macOS system

RR (Record and Replay)

- Linux bare-metal machine with Linux kernel version 4.7 or higher
  - If you do not have a Linux system or are running the evaluation in a VM, you cannot evaluate the results with RR.
  - You need to set `kernel.perf_event_paranoid` to `1` or lower to use RR.
    - You can do this by running the following command: `bash echo 'kernel.perf_event_paranoid=1' | sudo tee '/etc/sysctl.d/51-enable-perf-events.conf'`

        - If you are using NixOS, you can set the `kernel.perf_event_paranoid` option in your `configuration.nix` file: `nix boot.kernel.sysctl."kernel.perf_event_paranoid" = 1;`
- Intel CPU is recommended for better performance
    - For AMD CPU, you need extra configuration

Getting Started Guide

Get the Repo

- You may clone this repository: `git clone --recurse-submodules -j8 https://github.com/cmu-pasta/fray-benchmark`.
    - The project uses Nix to manage its dependencies. You may download Nix from here.
- You may also use the pre-configured container image that includes all the dependencies and tools to run the evaluation.
    - `podman run -p 8888:8888 -it --privileged leeleo3x/fray-benchmark bash`
    - We need the `--privileged` flag to run nix and RR.

Build the Project

- Enter the project directory: `cd fray-benchmark`.
    - In the pre-configured container image, you are already in the project directory (`/fray-benchmark`).
- Next, you need to enter the devshell: `nix develop`.
- If you are **not** using the pre-configured container image, run the following command to build all projects: `./scripts/build.sh`.

Kick the Tire (Reproduce Mini RQ 1)

- You can run the following command to run the sctbench benchmark for each technique for 1 minute (mini RQ 1): `bash ./scripts/run_kickthetire.sh`.
- Next you can run the jupyter notebook to visualize the results: `uv run --with jupyter jupyter lab --allow-root --ip 0.0.0.0 --no-browser`.
- Go to the `Kick The Tire` section in the notebook to see the results of the mini RQ 1.

Step-by-Step Instructions

Run Full Evaluation

- The source code of the benchmark applications is available in the `bms/{benchmark_name}` directory.

- Test cases are defined in `fray_benchmark/assets/{benchmark_name}.txt` file.
- (~2 hours) Reproduce the benchmark results (RQ1 and RQ2):
  - `bash ./scripts/run_benchmark.sh`
  - This script only runs the benchmark with 1 repetition for each technique.
  - If you have a powerful machine, you can run the evaluation with more concurrent jobs by using the `--cpu NUM_OF_JOBS` option. For example, `bash ./scripts/run_benchmark.sh --cpu 24` will run the evaluation with 24 concurrent jobs.
- (~30 minutes) Reproduce real-world bugs found by Fray (RQ3 and RQ4):
  - `bash ./scripts/run_realworld.sh`
  - By default, this script only reproduces failures using the POS algorithm. You may add the `--full-evaluation` option to run all algorithms (PCT3, PCT15, and Random) and all techniques (JPF and RR). The full evaluation will take ~4 hours to complete.
  - Note that depending on the hardware, Fray may find more or fewer bugs than reported in the original paper.
  - If you want to run all collected concurrency tests, replacing the `fray_benchmark/assets/{benchmark_name}.txt` with `fray_benchmark/assets/{benchmark_name}.full.txt` will run all tests. Note that this will take days to complete.

Analyze Results

- All results will be saved in the `output` directory.
- You can find the RQ1 and RQ2 results in the `output/benchmark` directory and the RQ3 and RQ4 results in the `output/realworld` directory.
  - `{benchmark_name}/{technique}/iter-0/{run_id}/` contains the output of each technique for each test case.
  - For Fray, the `report` folder contains the output. `report/fray.log` contains the log of Fray and error information if Fray finds a bug.
- We provide a Jupyter notebook to analyze the results. You can run the notebook by using the following command: `uv run --with jupyter jupyter lab --allow-root --ip 0.0.0.0 --no-browser`. The notebook is located in `fray_benchmark/visualizer/visuralize_result.ipynb`.

Real-world Bugs and Corresponding Run ID

- Kafka
  - Run 11: #17112 StreamThread shutdown calls completeShutdown only in CREATED state

    - Run 10: #17113 Flaky Test in GlobalStreamThreadTest#shouldThrowStreamsExceptionOnStartupIfExcepti
    - Run 9: #17114 DefaultStateUpdater::handleRuntimeException should update isRunning before calling `addToExceptionsAndFailedTasksThenClearUpdatingAndPausedTasks`
    - Run 8: #17162 DefaultTaskManagerTest may leak AwaitingRunnable thread
    - Run 7: #17354 StreamThread::setState race condition causes java.lang.RuntimeException: State mismatch PENDING_SHUTDOWN different from STARTING
    - Run 6: #17371 Flaky test in DefaultTaskExecutorTest.shouldUnassignTaskWhenRequired
    - Run 5: #17379 KafkaStreams: Unexpected state transition from ERROR to PENDING_SHUTDOWN
    - Run 4: #17394 Flaky test in DefaultTaskExecutorTest.shouldSetUncaughtStreamsException
    - Run 3: #17402 Test failure: DefaultStateUpdaterTest.shouldGetTasksFromRestoredActiveTasks expected: <2> but was: <3>
    - Run 2: #17929 `awaitProcessableTasks` is not safe in the presence of spurious wakeups.
    - Run 1: #17946 Flaky test DeafultStateUpdaterTest::shouldResumeStandbyTask due to concurrency issue
    - Run 0: #18418 Flaky test in KafkaStreamsTest::shouldThrowOnCleanupWhileShuttingDownStreamClosed
- Lucene
    - Run 1: #13547 Flaky Test in TestMergeSchedulerExternal#testSubclassConcurrentMergeScheduler
    - Run 4: #13552 Test TestIndexWriterWithThreads#testIOExceptionDuringWriteSegmentWithThrea Failed
    - Run 0, 3: #13571 DocumentsWriterDeleteQueue.getNextSequenceNumber assertion failure seqNo=9 vs maxSeqNo=8
    - Run 2: #13593 ConcurrentMergeScheduler may spawn more merge threads than specified
- Guava
    - Run 0, 1, 2: #7319 Lingering threads in multiple tests

Reusability Guide

The Fray repository contains the main documentation of Fray design and implementation.

- Usage Guide provides instructions on how to use Fray in normal applications as JUnit tests.

- Debugger Plugin provides instructions on how to use the Fray dubugger plugin.
- Architecture provdes an overview of the Fray architecture and design.
- Debugging Guide provides instructions on how to debug Fray and its components.
- We also wrote blog posts showing how to use Fray to find real-world bugs[1, 2, 3].

(Optional) Reproducing a Bug Found by Fray

To reproduce a bug found by Fray, you can use the following command:

```
bash python -m fray_benchmark replay RUN_DIR
```

For example, you can run `python3 -m fray_benchmark replay ./output/benchmark/sctbench/random/iter-0/0` to reproduce the `StringBufferJDK` bug found in SCTBench.

Note that real-world applications have randomness other than concurrency, so you may not be able to reproduce the bug deterministically. This commit shows the changes required to reproduce the bug found in Lucene.

(Optional) Using Fray in Real-World Applications

Please follow the Fray documentation to use Fray in any real-world applications. Good luck finding real-world bugs!

Other Notes

RR is very sensitive to the CPU and kernel version. You may have slightly different result if you are using a different CPU or kernel version.

Changes for Revision

All required revision changes concern paper writing only and do not affect Fray's implementation or evaluation, so we expect no changes to the evaluation results.