Lab Assignment 6

Course: CS202 Software Tools and Techniques for CSE
Lab Topic: Python Test Parallelization
Date: 13th February 2025

Objective
This lab aims to explore and analyze the challenges of test parallelization in Python. Students will work with multiple open-source Python repositories to evaluate the effectiveness and potential issues of parallel test execution. The assignment focuses on understanding different parallelization modes, identifying flaky tests, and documenting the readiness of open-source projects for parallel testing.

Learning Outcomes
By the end of this lab, students will be able to:

- ✓ Understand and apply different parallelization modes in `pytest-xdist`.
- ✓ Understand and apply different parallelization modes in `pytest-run-parallel`.
- ✓ Analyze test stability and flakiness in parallel execution environments.
- ✓ Evaluate the challenges and limitations of parallel test execution.
- ✓ Document and compare the parallel testing readiness of various open-source projects.

Pre-Lab Requirements

- ➢ Operating System: Windows/Linux/MacOS
- ➢ Programming Language: Python (setup a venv for this)[1]
- ➢ Required Tools:
  - `pytest` (test execution)
  - `pytest-xdist` (process level test parallelization[2])
  - `pytest-run-parallel` (thread level test parallelization[3])
- ➢ **Read**: https://pytest-xdist.readthedocs.io/en/stable
- ➢ **Read**: https://pypi.org/project/pytest-run-parallel

Lab Activities:

1. Clone the keon/algorithms repository to your local and setup the required dependencies in a different environment. Report the current commit hash in the documentation for this lab.
2. Set up a Python virtual environment for the repository and install the required dependencies.
3. For the repository, perform:

[1] The experiments should be performed after creating a virtual environment (VE). However, this is not the SET-IITGN-VM. In virtual machine (VM), you will observe very little or no speedup. VE is recommended.
[2] Test execution in parallel.
[3] You will observe interesting cases!

(a) Sequential Test Execution
- ★ Execute the existing full test suite sequentially ten times. Identify failing test cases and test cases with unstable verdicts, i.e., passes in one run but fails in the other. The latter ones are non-deterministic tests and termed as flaky. Eliminate both failing+flaky tests.
- ★ When no failing+flaky test remains in the test suite, repeat sequential execution of the full test suite three times, and denote the average execution time of five repetitions as T_{seq}.

(b) Parallel Test Execution
- ★ Setup parallel configuration:

  {`-n <1,auto>`}. //process (worker) level (**`pytest-xdist`**)
  {`--parallel-threads <1,auto>`} //thread level (**`pytest-run-parallel`**)
- ★ Execute the test suite with the selected configuration using different **`pytest-xdist`** parallelization modes[4]:

  ```
  --dist load
  --dist no
  ```
- ★ For each combination of the previous two steps **jointly considered**[5], perform three executions (repetitions) and denote the average execution time as T_{par}.
- ★ Record test failures (count, name of failed test) at each repetition. These are flaky tests due to test parallelization, different from those eliminated (if any) during the sequential execution.

4. Analyze the results:
   (a) Identify new flaky tests failed in parallel test executions. These are problematic for test parallelization and highlight that the test suite is not prepared to be executed in parallel.
   (b) Inspect and document the causes of test failures in parallel runs (e.g., shared resources, timing issues, i.e., timeout).
   (c) Calculate and compare speedup ratios for different parallelization modes and worker counts.
5. Create a comprehensive report including:
   (a) An execution matrix for the repository, detailing parallelization mode, worker count, average execution time, and failure rates (parallel execution). Show speedup plots as necessary.
   (b) Analysis of parallelization success/failure patterns for the repository.
   (c) Discussion on the parallel testing readiness of the project, including potential improvements.
   (d) Any suggestions for **`pytest`** developers to ensure thread safety (if any)?

---

[4] use `--dist=<...>` if needed.
[5] An example: **`pytest -n auto --dist load --parallel-threads auto`**

Resources

- https://pypi.org/project/pytest
- https://docs.pytest.org/en/stable
- https://pypi.org/project/pytest-xdist
- https://pytest-xdist.readthedocs.io/en/stable
- https://pypi.org/project/pytest-run-parallel
- https://github.com/Quansight-Labs/pytest-run-parallel