Lab 6 Report: Analysis of Test Parallelization in Python

Course: CS202 Software Tools and Techniques for CSE
Date: 13th February 2025
Author: Pathan Mohammad Rashid (22110187)

---

1. Introduction, Setup, and Tools

1.1 Objective

In this lab, I explored and analyzed the challenges of test parallelization in Python. My goal was to evaluate different parallelization modes using an open-source repository, identify flaky tests, and assess whether the project is ready for parallel test execution.

1.2 Learning Outcomes

By performing this lab, I aimed to:

- Apply parallelization modes using pytest-xdist and pytest-run-parallel.
- Analyze test stability and identify flaky tests in a parallel execution environment.
- Evaluate the challenges and limitations of running tests in parallel.
- Document and compare the parallel testing readiness of the open-source project.

1.3 Environment Setup

I set up my environment as follows:

- **Operating System:** Windows 11
- **Programming Language:** Python (using a dedicated virtual environment)
- **Tools and Versions:**
    - pytest (v8.3.4)
    - pytest-xdist (v3.6.1)

- pytest-run-parallel (v0.3.1)
- Python version: 3.12.3

1.4 Repository Commit Hash

I cloned the keon/algorithms repository and verified its commit hash. The commit hash I recorded is:

`cad4754bc71742c2d6fcbd3b92ae74834d359844`

---

2. Methodology and Execution

2.1 Repository Setup

I began by cloning the keon/algorithms repository to my local machine. Then, I set up a Python virtual environment and installed all required dependencies. To verify the commit hash, I ran the following command:

```
git rev-parse HEAD
```

which returned the hash noted above.

2.2 Sequential Test Execution

Procedure:

- I executed the full test suite sequentially ten times.
- I identified and eliminated tests that were failing or showing non-deterministic (flaky) behavior.
- Once I had a stable test suite, I repeated the sequential execution three times to compute the average execution time (Tseq).

Example Command and Output:

```
python run_sequential_tests.py
```

The sequential run times I observed were:

- Run 1: 6.77 s
- Run 2: 5.23 s
- Run 3: 5.66 s
- **Average Tseq:** Approximately 5.89 s

2.3 Parallel Test Execution

Configurations:
I evaluated parallel execution using both pytest-xdist and pytest-run-parallel.

Process-Level Parallelization:

Command using pytest-xdist with distribution mode load:

```
pytest -n auto --dist=load
```

- *Observation:* All tests passed in approximately 9.21 seconds.

Command using pytest-xdist with distribution mode no:

```
pytest -n auto --dist=no
```

- *Observation:* The tests completed in around 9.84 seconds.

Thread-Level Parallelization:

Command using pytest-run-parallel (thread-level):

```
pytest -n auto --parallel-threads=auto --dist=load
```

- *Observation:* This execution took significantly longer (about 54.20 seconds), and I encountered several test failures due to parallelization issues.

2.4 Code Snippets and Error Handling

During the lab, I encountered several test failures under parallel execution. For instance:

Heap Test Failures:

- **Test Insert:** I expected the heap after insertion to be:
  `[0, 2, 50, 4, 55, 90, 87, 7]`,
  but the actual output was:

`[0, 2, 2, 4, 50, 90, 87, 7, 55]`.

- **Test Remove min:** The expected return value was `4`, yet I observed `7`.

Other Failures:

- I also encountered failures in the linked list palindrome detection test and Huffman coding test. These issues indicated problems likely caused by shared resources and timing conflicts when tests were run concurrently.

---

3. Results and Analysis

3.1 Execution Matrix

I compiled an execution matrix to compare the different configurations:

Configuration	Average Time (s)	Speedup (Tpar/Tseq)
Sequential Execution	5.89	1.000
pytest-xdist (--dist=load)	9.21	0.639
pytest-xdist (--dist=no)	9.84	0.598
pytest-run-parallel (--parallel-threads=auto)	54.20	0.109

Table 1: Execution Matrix for Different Parallelization Modes

3.2 Key Observations

- **Speedup Ratios:** I noted that the speedup for parallel execution using pytest-xdist was less than 1. This indicates that the overhead introduced by process-based parallelization reduced efficiency compared to sequential runs. In contrast, thread-level parallelization using pytest-run-parallel resulted in much longer execution times.

- **Flaky Tests:** New flaky tests emerged during parallel executions, such as in the heap and linked list tests. Additionally, failures in the Huffman coding test pointed to issues with concurrent file operations or timing conflicts.

3.3 Speedup Plot

Below, I have placed a placeholder for the speedup plot that visually compares the execution times for different configurations.

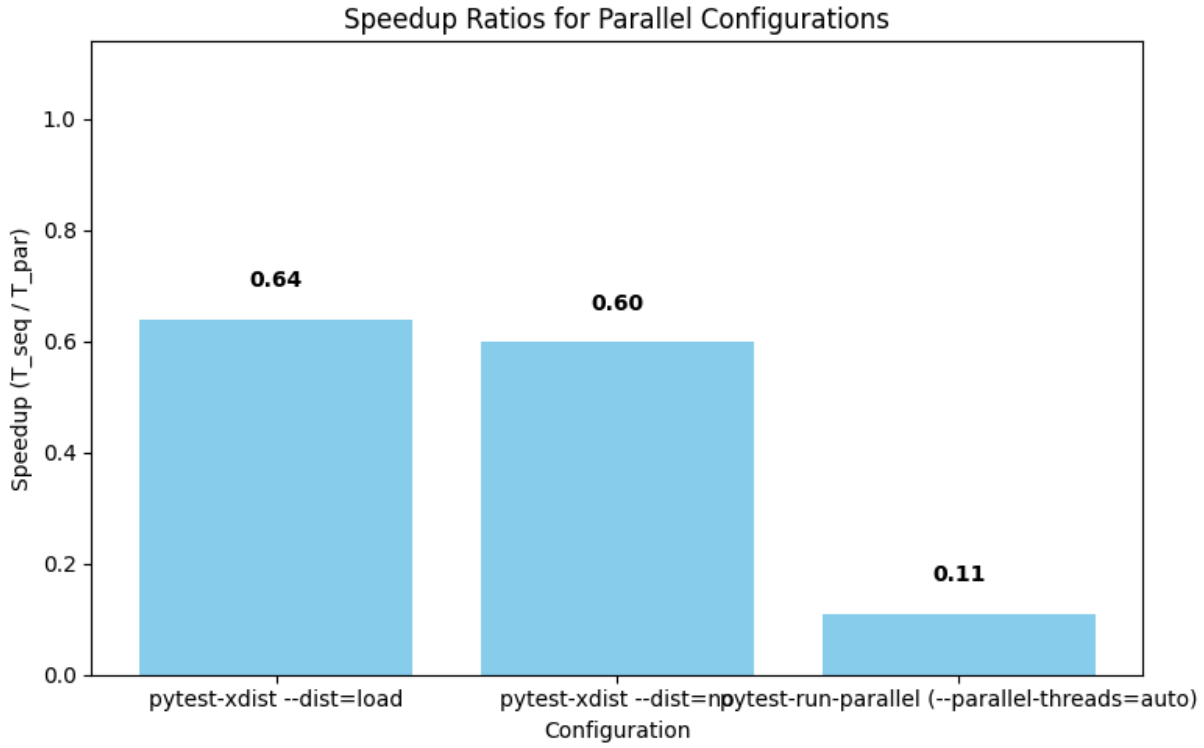

Figure 1: Speedup plot comparing execution times for different configurations.

---

4. Discussion and Conclusion

4.1 Challenges Encountered

I faced several challenges during this lab:

- **Test Stability:** Although the sequential tests became stable after removing flaky tests, the parallel executions introduced new issues such as shared resource conflicts and race conditions.

4.2 Reflections and Lessons Learned

- I learned that the repository is not fully ready for parallel execution without further modifications.

- I realized that test suites must be designed with thread safety in mind to avoid side effects during parallel execution.

5. GitHub Repository

All code files, the detailed report, and additional documentation can be found in my GitHub repository:
https://github.com/Pathan-Mohammad-Rashid/STT_Labs.git

Onedrive Link:
https://iitgnacin-my.sharepoint.com/:f:/g/personal/22110187_iitgn_ac_in/EnECtzFbUNZKkBWY6aa8QJYBp6liTXfpuYRtJNMd1TwR6A?e=nsND4y