Test Sharding and Parallelization in Large Monorepos

In large-scale engineering organizations, monolithic repositories (monorepos) are often used to maintain codebases for multiple applications, libraries, or services. Running tests efficiently in such repositories becomes a major challenge, especially as the number of test cases and their execution time grow. **Test sharding** and **parallelization** are two critical strategies to scale test execution, reduce feedback cycles, and improve developer productivity.

🔶 Test Sharding: Divide and Conquer

Test sharding refers to splitting the total test suite into smaller subsets (shards) which can be executed independently and in parallel.

Key Strategies:

- **Static Sharding**: Predefined test splits based on test file names, directories, or modules.
- **Dynamic Sharding**: Tests are split at runtime based on historical duration data to balance execution time across shards.

Benefits:

- Reduced test wall time by distributing work.
- Easier to scale horizontally by increasing workers.
- Enables large suites to fit within CI time limits.

Example (Pytest):

```
pytest --dist=loadscope --tx 4*popen
```

🔷 Tools and Frameworks

- **Bazel**: Native support for test parallelization and caching
- **pytest-xdist**: Parallel test runner plugin
- **JUnit + Gradle**: `maxParallelForks`, `testLogging`, etc.
- **TestNG (Java)**: Allows defining test groups and parallelism at suite/class/method level
- **GitHub Actions / GitLab CI**: Matrix builds for sharding

🔷 Best Practices

- Tag tests for selective runs (e.g., `@smoke`, `@integration`)
- Collect and analyze test duration metrics
- Use isolated environments per shard (e.g., containers)
- Combine sharding with change-based test selection
- Monitor and retry flaky shards independently

🔶 Conclusion

Test sharding and parallelization are essential for maintaining agility and scalability in large monorepos. By carefully splitting test execution and using modern tooling, teams can drastically cut down CI times, reduce flakiness, and deliver faster, more reliable code.