

Concurrency Testing Using TestNG Framework

Jafar Inamdar*

Software Development and Test Engineer, Pittsburgh, USA

*Corresponding author: jafar.inamdar@gmail.com

Received September 13, 2025; Revised October 15, 2025; Accepted October 22, 2025

Abstract This document explores the significance of parallel testing in identifying concurrency issues within APIs and proposes the utilization of TestNG framework to facilitate efficient parallel testing. Concurrency issues, often elusive and challenging to detect, can cause critical failures in API systems under high loads. Traditional sequential testing methods may fail to uncover these issues adequately. Parallel testing, when implemented effectively using TestNG, can significantly enhance the detection of concurrency-related defects in APIs.

Keywords: Java, TestNG, Concurrency

Cite This Article: Jafar Inamdar, "Concurrency Testing Using TestNG Framework." *Journal of Computer Sciences and Applications*, vol. 13, no. 1 (2025): 37-40. doi: 10.12691/jcsa-13-1-4.

1. Introduction

1.1. Background

As modern software systems become increasingly complex, ensuring their robustness and reliability under concurrent user loads is imperative. APIs, serving as the backbone of many software applications, are particularly susceptible to concurrency issues. Concurrency issues arise when multiple threads access shared resources concurrently, leading to unexpected behaviors such as race conditions, deadlocks, and data corruption.

1.2. Problem Statement

Modern software systems, particularly APIs, are susceptible to concurrency issues that can cause critical failures under high loads. Traditional testing methods often fail to adequately detect these issues due to their intermittent nature and dependency on specific execution contexts. Sequential testing approaches may overlook concurrency-related defects, leading to undetected vulnerabilities in production environments. Consequently, there is a pressing need for a robust testing methodology capable of efficiently identifying concurrency issues within APIs to ensure their reliability and performance under concurrent user loads.

2. Methodology

2.1. Evaluation

Parallel testing involves executing multiple tests simultaneously, simulating real-world scenarios with concurrent user interactions. By executing tests concurrently, parallel testing exposes APIs to conditions

resembling actual usage patterns, increasing the likelihood of detecting concurrency issues. Additionally, parallel testing reduces the overall test execution time, enhancing the efficiency of the testing process.

TestNG is a powerful testing framework for Java that supports parallel test execution out of the box. Its flexible configuration options enable testers to distribute test cases across multiple threads, maximizing resource utilization and accelerating test execution. TestNG provides annotations such as `@Test` and `@DataProvider` to define test cases and data sources, respectively, making it well-suited for designing comprehensive test suites.

TestNG offers several features that make it well-suited for parallel testing and identifying concurrency issues in APIs:

- **Parallel Execution Modes:** TestNG supports multiple parallel execution modes, including methods, tests, classes, and instances. This flexibility allows testers to tailor parallel execution to specific testing requirements, maximizing efficiency and resource utilization.
- **Thread Pool Configuration:** TestNG allows testers to configure the size of the thread pool used for parallel execution, enabling optimization based on system resources and testing objectives. This ensures efficient parallel execution without overwhelming the system or causing resource contention.
- **Dependency Management:** TestNG provides robust dependency management capabilities, allowing testers to define dependencies between test methods and groups. This ensures that tests are executed in the correct order, maintaining test integrity and consistency in parallel execution scenarios.
- **Data Parallelism:** TestNG supports data-driven testing through data providers, enabling testers to execute test cases with different input data sets concurrently. This facilitates thorough testing of

API functionality under diverse input conditions, enhancing test coverage and defect detection.

- **Reporting and Analysis:** TestNG generates comprehensive test reports with detailed information about test execution, including pass/fail status, execution time, and stack traces for failed tests. This facilitates thorough analysis of test results, aiding in the identification and resolution of concurrency-related defects.

In summary, parallel testing, particularly when implemented using the TestNG framework, offers a powerful approach to uncovering concurrency-related defects efficiently.

2.2. Other Approaches

While TestNG offers robust capabilities for parallel testing and concurrency issue detection, other approaches exist, each with its advantages and limitations:

- **JUnit:** Another popular testing framework for Java, JUnit lacks built-in support for parallel execution, requiring additional plugins or custom implementations for parallel testing. However, JUnit offers simplicity and familiarity for developers already using JUnit for unit testing.
- **Custom Multithreading:** Some teams opt for custom multithreading approaches to simulate concurrent user interactions and test API behavior under load.

While flexible, custom multithreading solutions often lack the reporting and management features provided by testing frameworks like TestNG, making result analysis and defect resolution more challenging.

- **Commercial Testing Tools:** Commercial testing tools such as Selenium and LoadRunner offer comprehensive testing capabilities, including concurrency testing for APIs. However, these tools often come with licensing costs and may require additional training for teams unfamiliar with their usage.
- **Concurrency Testing Libraries:** Libraries like Apache JMeter and Gatling specialize in load testing and can be used for concurrency testing of APIs. While powerful, these libraries may require scripting and configuration overhead, making them less accessible to teams with limited testing expertise.

2.3. Design Overview

The document provides a comprehensive overview of the technical design required for building parallel testing using TestNG. The following flowchart diagram provides a quick overview of the TestNG Design, which will be discussed in further detail in the next section.

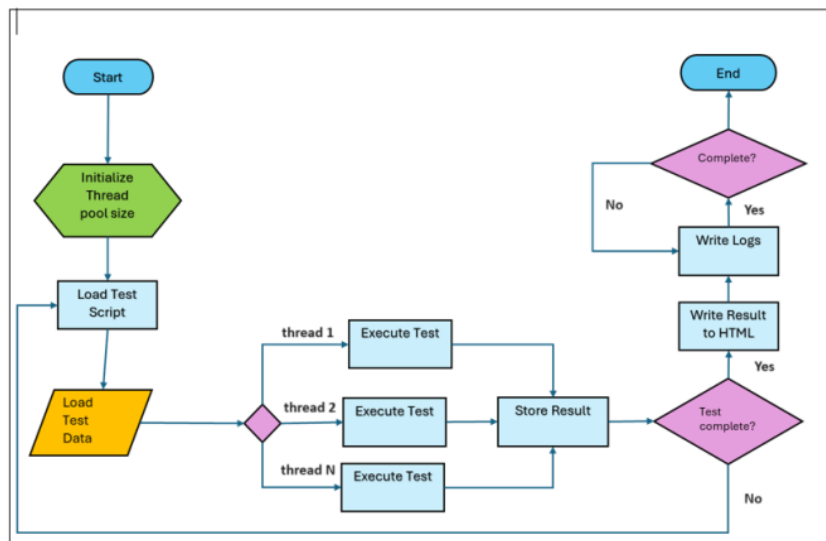


Figure 1. Architecture Design

3. Technical Details

3.1. Prerequisites

To create parallel testing using TestNG, here are some prerequisites:

- **Java programming:** You should have a good understanding of Java programming language, including data structures, control structures, functions, modules, classes, and object-oriented programming (OOP) concepts.
- **Basic Software Testing Knowledge:** Familiarity with core software testing concepts, such as test cases, test execution, and different testing types,

will aid in effectively utilizing TestNG.

- **Build Tool (Recommended):** While not strictly mandatory for very simple projects, using a build tool like Maven or Gradle is highly recommended for managing dependencies, compiling code, and running tests in larger or more complex projects.
- **Selenium WebDriver (if applicable):** If TestNG is being used in conjunction with Selenium for web automation, then Selenium WebDriver and its dependencies also need to be set up.
- **Java Development Kit (JDK):** TestNG is a Java-based testing framework, so a compatible JDK version needs to be installed and configured on the system.
- **Integrated Development Environment (IDE):** An

IDE like Eclipse, IntelliJ IDEA, or Visual Studio Code is recommended for easier development, project management, and execution of TestNG tests.

By mastering these prerequisites, you can start creating your own parallel test with TestNG.

4. Technical Design

The document provides detailed technical design guidelines for building concurrency test in TestNG using Maven.

Step 1: Add necessary Test NG Libraries

```
<dependency>
<groupId>org.testng</groupId>
<artifactId>testng</artifactId>
<version>6.14.2</version>
</dependency>
```

Step 2: Import TestNG classes in your test class

```
import org.testng.annotations.BeforeClass;
import org.testng.annotations.AfterClass;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;
```

Step 3: Using @DataProvider annotation create a method which will supply input to our test

```
@DataProvider(name = "getInputData", parallel=true)
Object[][] getInputData(){
```

```
List<List<Integer>> inputParamList = new
ArrayList<>();
```

```
Random random = new Random();
for(int i =0;i<NO_OF_THREADS; i++) {
List<Integer> testParams = new
ArrayList<Integer>();
testParams.add(random.nextInt());
testParams.add(random.nextInt());
inputParamList.add(testParams);
}
```

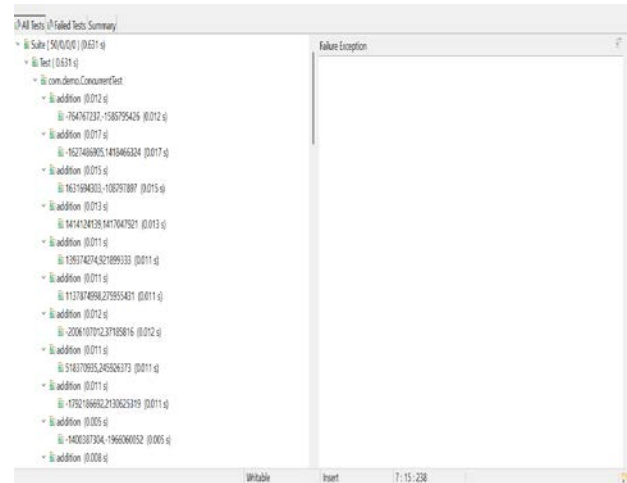
```
return inputParamList
.stream()
.map(l->l.stream().toArray())
.toArray(Object[][]::new);
}
```

Step 4: Create a Test class with @Test annotation

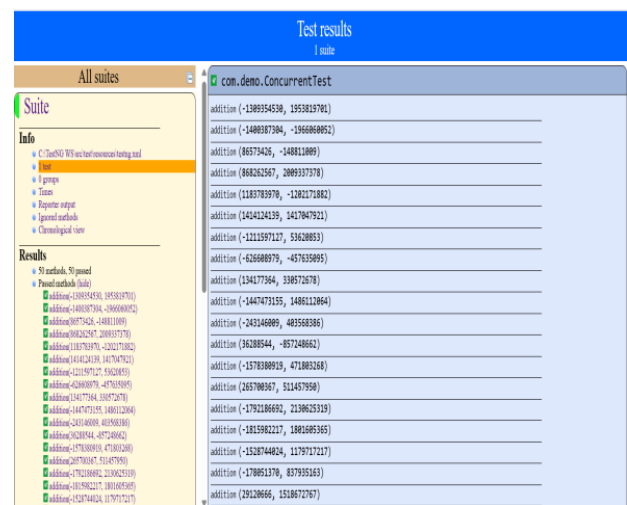
```
dataProvider = "getInputData"
@Test(dataProvider = "getInputData")
public void addition(int a, int b) {
System.out.println("Input Param 1"+a);
System.out.println("Input Param 2"+b);
int sum = a + b;
System.out.println("Sum="+sum);
//return sum;
}
```

Step 5: Build the project

Step 6: Execute test, it will run with multiple threads



Step 7: Finally, view the results



5. Use-Cases

There are numerous use cases for TestNG Concurrency Testing in APIs. Here are some examples:

1. Simulating High Concurrent User Load

- Scenario: An e-commerce API handling thousands of customers checking out simultaneously.
- Value: Parallel TestNG tests can simulate multiple concurrent checkout calls (add to cart, apply discounts, process payments) to uncover race conditions in session handling, cart persistence, or payment validation logic.

2. Testing Shared Resource Access

- Scenario: A financial transaction API where multiple users transfer funds at the same time.
- Value: Parallel tests help detect double spending, balance miscalculations, or deadlocks when two or more requests try to update the same account balance concurrently.

3. Verifying Thread Safety in Caching Layers

- Scenario: An API that uses an in-memory cache (e.g., Redis, Guava, or custom caching).
 - Value: Concurrent requests can be tested to verify if cache reads/writes remain consistent, avoiding stale data or race conditions when multiple threads update cache entries.
4. Testing Idempotent Operations
- Scenario: A REST API where POST /orders or PUT /booking should remain idempotent under retries.
 - Value: Parallel execution validates that repeated concurrent calls don't create duplicate resources or inconsistent states.
5. Concurrent File or Record Updates
- Scenario: An API updating user profiles, documents, or records in a database.
 - Value: Parallel TestNG tests help reveal whether last-write-wins, optimistic locking, or version control mechanisms work as expected under simultaneous updates.
6. Performance Bottleneck Identification
- Scenario: An API endpoint that aggregates data from multiple downstream services.
 - Value: By firing concurrent requests, TestNG can highlight bottlenecks in downstream calls, thread pool exhaustion, or excessive lock contention in service orchestration logic.
7. Microservices Integration Testing
- Scenario: A distributed architecture where multiple microservices interact through APIs.
 - Value: Running TestNG tests in parallel ensures inter-service communication remains reliable under heavy concurrent calls, detecting issues such as service deadlocks or queue overflows.

6. Tools

Implementing concurrency testing with TestNG requires a combination of frameworks, libraries, and supporting tools for execution, monitoring, and analysis. Below are the key components:

- TestNG Framework
- Build and Dependency Management Tools: Maven / Gradle
- API Testing Libraries: RestAssured (commonly

paired with TestNG)

- Logging & Monitoring Tools: Log4j / SLF4J, Dynatrace, New Relic, or Prometheus + Grafana
- CI/CD Integration: Jenkins / GitHub Actions / GitLab CI
- Reporting & Analysis Tools: Allure Reporting (with TestNG adapter)

How These Tools Work Together

- TestNG + RestAssured → Write API concurrency tests.
- Maven/Gradle → Manage dependencies and integrate with build pipelines.
- Logging + Monitoring Tools → Capture execution context, identify concurrency bugs.
- Jenkins/GitHub Actions → Run tests in CI/CD pipelines with parallel execution.
- Allure / TestNG Reports → Analyze concurrency test results visually.

7. Conclusion

Concurrency issues pose significant challenges to the reliability and performance of APIs, necessitating robust testing methodologies for their detection and mitigation. Parallel testing, particularly when implemented using the TestNG framework, offers a powerful approach to uncovering concurrency-related defects efficiently. While other concurrency testing approaches exist, TestNG stands out for its ease of use, flexibility, and comprehensive feature set, making it an ideal choice for API testing in Java environments. By simulating real-world usage patterns and maximizing resource utilization, parallel testing with TestNG enhances the effectiveness and reliability of API testing, ultimately improving software quality and user experience.

References

- [1] Cédric Beust and Hani Suleiman "Next Generation Java Testing: TestNG and Advanced Concepts".
- [2] TestNG – Official Website. <https://www.testng.org/>.
- [3] Rest-assured – Official Website. <https://www.rest-assured.io/>.



© The Author(s) 2025. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).