INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume 4, Issue 3)
Available online at: www.ijariit.com

To study the impact and usage of test parallelization

Priya Gupta
pgup9374@gmail.com
Birla Institute of Technology, Ranchi, Jharkhand

Dr. Anup Kumar Keshri
anup_keshri@bitmesra.ac.in
Birla Institute of Technology, Ranchi, Jharkhand

ABSTRACT

The number of test cases required to ensure the quality of a software system grows hand-in-hand with its complex and henceforth the total test execution time increases proportionally. Traditionally test cases are executed sequentially which takes much time. Thus parallel test execution appeared to be an appealing option to cut over the test execution time.This paper presents our findings on different techniques for test parallelization and its impact with respect to time and test flakiness. Test flakiness is the tests that fail randomly. We have applied the different techniques on a java project containing 623 test cases and observed the effect. Based on the result we have tried to conclude the optimal technique for test parallelization.

Keywords: - *Forking, Test flakiness, Speed off, Ant colony optimization, Sequential testing, Parallel testing*

1. INTRODUCTION

Software testing is a process of executing a program with the intention of finding software bugs. The purpose of testing is to check whether the software satisfies specific requirement. The number of test cases required to ensure the quality of a software system grows hand-in-hand with its complex and henceforth the total test execution time increases proportionally. For large software systems, test execution time becomes increasingly critical in automated regression testing, where a large suite of tests is executed frequently on continuous integration servers [1]. Different approaches like test selection, test prioritization or test case reduction are typically used to speed up test execution, especially in the context of regression testing [4]. Traditionally test cases are executed sequentially which takes lot of time due data dependency. Thus parallel test execution appeared to be an appealing option to cut over the test execution time .Test cases are parallelizable only if all the test cases are Independent which means that there is no dependencies between the test cases. Interference between test cases is created if two or more test cases try to access same resource. For example if one test case needs a data for its input but previously the other test case has already modified the data in that case it produces inconsistency among the test cases. Thus to run test cases parallel we have to ensure that there is no interference among these test cases. Other issues that that can occur is state inconsistency among the test cases. For example a previous test cases has led to a state from which another test cases cannot proceed. One test case requires data from the database that is already deleted by previous test case. To running test cases parallel we need to schedule these test cases properly, so that result of previous test cases cannot affect the current test case. These conditions make test parallelization more challenging.
Dealing with high testing costs have been important.

The problem in software engineering research and industrial practice. Several approaches have been proposed in the research literature to address this problem, with the focus mainly on test suite minimization, prioritization, reduction, and selection [1]. Each single test case requires complex test setups which may involve several hours of human labor, and its execution may occupy e.g. the HiL environment for several hours. Hence, every single opportunity to economize a test case is valuable [3].
To study the effect of test parallelization we have done analysis on various parameters 1. Speed off 2. The rate of test flakiness 4. Usability. *Speed off* examines the impact on an execution time as compared to the sequential execution of test cases.
Test flakiness is test cases failure. A flaky test is which pass or fail under the same configuration. These failures of test cases do not always indicate a bug in the code, therefore it can also tell as harmful to developers. Dependent tests can be affected by the different scheduling of test methods and classes.
Usability measures the rate of usage of parallelism by the developers in the project. I surveyed among the developer in the company where I did an internship. Usually, the projects where test cases run no longer than a minute, there is no requirement of parallelism. But there are few projects which contain a large number of test cases. The whole project requires approximately half hour or an hour to build, thus here we require test parallelization which can reduce the amount of cost of test cases with respect to time.

2. RELATED WORK

We have used the following configuration of a computer
Processor: Intel Core i5 - 4300M CPU, 2.60GHZ
Installed memory (RAM): 16.0 GB

We have performed on a java maven project which consists of 623 test cases. We had performed maven clean install command to build the project.

There is a different configuration for running test cases parallel. Below are mentioned:

1. Sequential Classes Parallel Methods
In this configuration test classes are run sequentially but test methods run concurrently.
2. Parallel classes Sequential Methods
In this configuration test classes are run sequentially but test methods run concurrently.
3. Parallel classes Sequential Methods
In this configuration test classes run concurrently, but test methods run sequentially.
4. Parallel classes Parallel Methods
In this configuration test classes as well as test methods are run concurrently.
5. Only Sequential
In this configuration, no parallelism is involved.

As discussed in paper Test Suite parallelization of open source project by Jeandorson Candido [2], we must have to be careful while choosing the correct configuration for test parallelization. Data sharing can occur, for example, through the state that is reachable from statically-declared variables in the program or through variables declared within the scope of the test class or even through resources available on the file system and the network [5].

For e.g. Condition 1 is preferable over Condition 2when it is clear that test methods do not manipulate on the shared resource. Similarly, Condition 2 is preferable over Condition1 when we know that several test methods in a class perform operations on shared data.

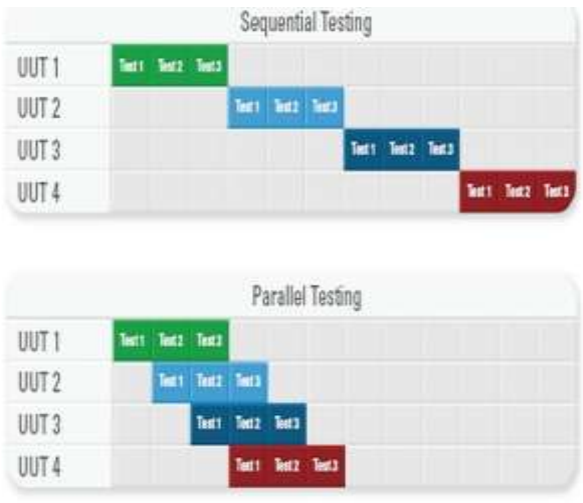

Fig 1: Comparison of Execution time of Sequential Configuration and parallel configuration

Related Work for sequential Execution
- First of all, we measure the execution time for configuration 5 that is only sequential.
- We have put the print statement on each of the beginning and end section of test cases to observe whether the test cases are running sequentially or parallel.
- We noted that all first it prints the “begin of test case no. i” and then followed by print “end of test case no. i”. The index ‘I’ is the index no. of the test case.
- We had performed the command maven clean install, noted the time taken by the project to build.
- The average time taken to build the project is approx. 14 minutes, which is very costly in terms of time.
- Thus we can apply test parallelization technique to reduce time.
- For a large project, when test cases require a large amount of time for e.g. Hours in such cases sequential execution does not satisfy any more industrial demand, as they require fast feedback cycle.

2.1 Related work with Application of Test parallelization technique:
Here we apply test parallelization technique and observed the related effect.
Use of Test parallelization without forking
- In the ***1st*** *case*, we use parallelization by adding the plugin in pom.xml file of maven project.
- Plugins perform a real action in a maven project. It can also be said that Maven is a core framework which is a collection of maven plugins. They are used to create jar files, creates war files, compile code, unit test code and create project documentation, Plugins executes an action in the context of a project description in project object model (POM).

We have used the following plugins
```
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-surefire-plugin</artifactId>
<version>2.7.1</version>
<configuration>
<parallel>methods</parallel>
<threadCount>5</threadCount>
</configuration>
</plugin>
</plugins>
```

2.1.2 Results:
As we mentioned earlier we have to be very careful in choosing the above-mentioned configuration.
- We have applied both the configuration 1 and configuration 2, as due to a large number of test cases it is really challenging to detect dependencies.
- If we apply configuration 2 that is parallel class and sequential method we observed inconsistent state that is no test classes are allowed to run due to data dependency between the classes mainly before in the setup configuration of before class.
- If we apply configuration 1 that is sequential class and parallel methods, we can observe a high number of test flakiness but we found a remarkable improvement in execution time, earlier in the sequential configuration the average time was 9 min, but here we found the execution time with 3 min, though we have a test cases failure.
- We run the project multiple time. We found that some set of test cases failing repeatedly. This happens due to test dependency. We manually try to remove the data cases dependency by modification in the code. Doing so, the test cases pass.
- But as there is a large number of test cases it is not possible to detect and remove the data dependencies manually.
- Thus we require proper scheduling of test cases so that there should not any interference between the test cases.

- Various research is going on this field to plan and schedule the execution of complex test suites in order to avoid undesired interference between tests.
- According to Research paper "An optimization of test parallelism with constraint" by Masoumeh Parse, he suggested an algorithm based on Ant Colony Method to reduce the constraint between test cases so that it can be implemented parallel. The output of the "ACS-TSO" algorithm is a test case schedule plan, which, when enforced, would result in a reduced overall test execution time.
- Ant Colony Optimization technique is a set of instruction based on search algorithm of Artificial Intelligence near to an optimal solution. It mainly deals with two processes pheromone deposition and trail pheromone deposition.
- Thus if we can schedule the test case plan, then we easily save time
- Non-forked JVMs can achieve impressive speedups at the expense of sometimes impressive rates of flakiness. Breaking test dependencies to avoid flakiness and take full advantage of those options is advised for developers with a greater interest in efficiency [9].

2.2 Use of parallelization techniques with forking

The *2nd* methods that are required to run test cases parallel are the use of forking. Forking OS processes to run test jobs is the basic mechanism of build systems to obtain parallelism at the machine space [2].

The list below shows the choices to control parallelism through the build system (e.g., Maven).

- **Forked JVMs with sequential methods (FC0)** the build system spawns multiple JVMs with this configuration, assigning a partition of the set of test classes to each JVM. Test classes and methods run sequentially within each JVM [2].
- **Forked JVMs with parallel methods (FC1)** With this configuration, the build system forks multiple JVMs, as FC0 does, but it runs test methods concurrently, as C1 does[2]

We have used the following plugins:

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-surefire-plugin</artifactId>
<configuration>
<forkCount>1C</forkCount>
<reuseForks>true</reuseForks>
<parallel>methods</parallel>
<threadCount>5</threadCount>
</configuration>
</plugin>
```

The parameter **forkCount** is defined as the maximum number of JVM processes that maven-surefire-plugin will spawn concurrently to execute the tests. The parameter **reuseForks** is defined as whether to terminate the spawned process after one test class and to create a new process for the next line in the test.

2.2.2 Results

We have used the configuration sequential class and parallel methods.

- We have observed that the project build successfully.
- There are no test failures which can also be told as test flakiness.
- However, we have not achieved a drastic saving of time.
- The table for execution time is given below:
- We have performed 3 trial. The duration of each trial is a half hour.

Table- 1 Execution time for parallel execution

Trial 1 (in min)	Trial 2 (in min)	Trial 3 (in min)	Mean Execution time min)
9.897	10.054	9.598	9.840
11.098	9.897	13.876	11.623
10.987	11.876	8.098	10.320
13.987	12.876	9.876	12.240
12.991	13.567	10.675	12.412
9.678	11.765	12.657	11.366
9.876	12.876	11.123	11.291
8.990	10.879	13.987	11.285
11.990	12.567	14.867	13.141
12.887	13.567	12.432	12.962
10.556	14.332	13.665	12.851

The cache memory manipulates the run time of the algorithm as the compiler first try to look in the cache. And as we run the same input, again and again, the time consumed to perform a process gets minimized. So, we took 3 trials at the interval of half hour.

- After finding the average execution time of parallel method level configuration we will compare it with the execution time of serial configuration.
- The following table shows the speed off.
- T_{seq} is the execution time for sequential configuration.
- $T_{parallel}$ is the execution time for parallel configuration.

Table- 2: Speed off the table

T_{seq}	$T_{parallel}$	$T_{seq}/ T_{parallel}$
14.054	9.840	1.428
13.890	11.623	1.195
11.987	10.320	1.160
14.870	12.240	1.214
13.768	12.412	1.124
13.897	11.366	1.122
11.087	11.291	0.981
14.099	11.285	1.241
13.897	13.141	1.057
14.567	12.962	1.123
12.987	12.851	1.010

- Thus the average speed off result= 1.150
- The result shows that that parallel configuration is 1.150 faster as compared with serial configuration.
- In next part we had also used a different number of threads, so to achieve maximum speed off.
- The following graph shows the difference in execution time.
- The result shows that that parallel configuration is 1.150 faster as compared with serial configuration.
- In next part we had also used a different number of threads, so to achieve maximum speed off.
- The following graph shows the difference in execution time.

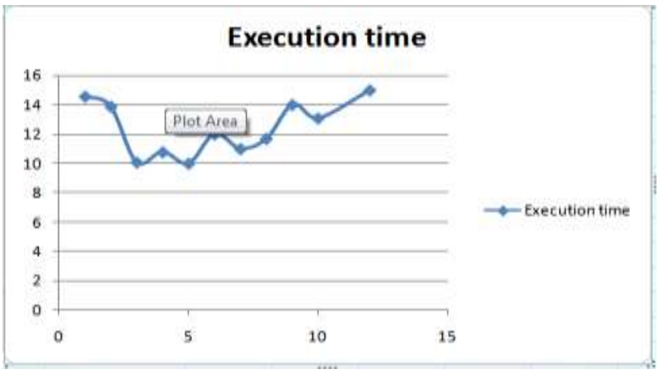

Fig 2: Graph between Thread Count and Execution time

Here X-axis = number of threads used in threads count
Y-axis = Execution time of the project in minutes.

Minimum execution time is achieved in when we use a thread count between 4 and 6. We are using multicore processors with a number of cores= 6. Thus we can conclude that an optimal number of thread count is equal to the number of cores.

3. CONCLUSION

As we know that testing is a crucial phase for software engineering. Traditionally test cases are run sequentially, due to the dependency of data between test cases. If the test cases of project need less than a minute then there is no requirement for test parallelization, but if the test cases require a significant amount of time then it is required for test parallelization. As there are many configurations for running test parallelization suites, we have to be careful while choosing the test parallelization to select correct one. Though test parallelization techniques are very less used it can be used without affecting reliability. In case of Without forking test parallelization technique, we can save more time if we schedule the test cases so that it can be run without interference. Thus more research should be done to the automatic scheduling of test cases to reduce the interdependency among test cases. With forking we can achieve success as shown in our paper it achieves a speed off of approx. 1.

4. REFERENCES

[1] Adnan Ashraf, Dragos Truscan, Ivvan Porres,”An optimization of Test Parallelization with Constraint”, ResearchGate, Feb 2016

[2] Jeanderson Candido ,Luis Melo, Marcelo d’Amorim ,”Test Suite Parallelization in open Source project :A study on its Usage and Impact”,

[3] A. Gotlieb and D. Marijan. FLOWER: optimal test suite reduction as a network maximum flow. In ISSTA, pages 171–180, 2014.

[4] S. Yoo und M. Harman. Regression Testing Minimization, Selection and Prioritization:A Survey. Softw. Test. Verif. Reliab., 22(2):67–120, Marz 2012.

[5] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. An empirical analysis of flaky tests. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software.

BIOGRAPHY

Priya Gupta
M. tech.
Computer Science and Engineering,
Birla Institute of Technology, Mesra, Ranchi, India.

Dr. Anup Keshri
Assistant Professor
Computer Science and Engineering,
Birla Institute of Technology, Mesra, Ranchi India.