Quantification of Regression Test Suite Execution Time in Parallel Execution Setup with Weighted Test Suite Split Algorithm

Abhinandan H. Patil*
Karnataka, India
E-mail: Abhinandan_patil_1414@yahoo.com
ORCID iD: https://orcid.org/0000-0002-8425-1493
*Corresponding Author

Sangeeta A. Patil
Karnataka, India
E-mail: Sangeetapatil1981@gmail.com

Abstract: Regression test suite execution time study focus is essentially on two aspects. They are execution time reduction and making effective use of available hardware resources and manpower. This paper investigates how the regression test suite can be split into subsets to make use of parallel execution across several machines with identical execution speeds and asymmetrical execution speeds. In the symmetrical execution speed setup, long test execution time test cases are evenly distributed across all the hardware machines. However, in asymmetrical execution speed machines, more test cases are distributed to speed machines to make efficient use of hardware resources. In all the situations where there is automation tool to execute the individual test cases of test suite this approach can be employed to make effective use of hardware resources and to keep the execution time within bounds. The Algorithms can also be used in situations where there are queues involved and serial, fixed time service takes place for each of the entity being served.

Index Terms: Weighted Test Suite Split, Symmetrical Speed Machines, Asymmetrical Speed Machines, Effective Regression Test Execution Time, Regression Test Suite Time Reduction, Test Suite Execution Time Analysis.

1. Introduction

Main focus in Regression test execution is always on two points viz. execution time and resources. Resources could be hardware or manpower. In this paper the discussion is mainly in cases where lab has multiple machines for execution of test cases. The idea is to split the test suite into multiple sub test suites and execute them on different machines. The machines under study could have symmetrical or asymmetrical execution speeds. The paper considers both the cases and suggests how the test suite should be split and then goes ahead to quantify the execution time of whole test suite. In first case the paper considers the case with identical execution speed machines. In the second case paper considers the case with different execution speeds.

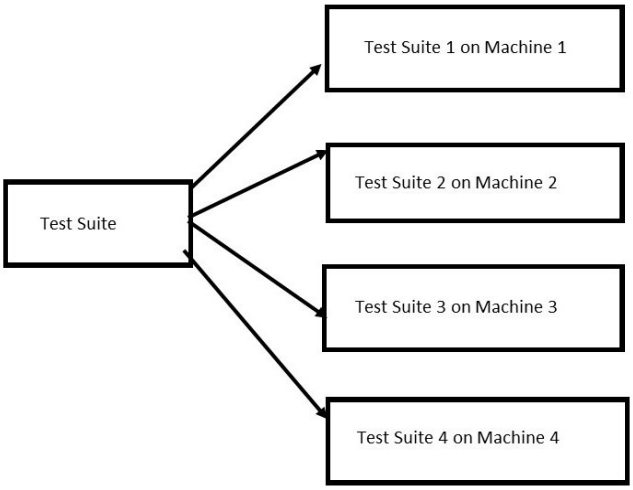

Figure 1.Test Suite Parallel Execution

2. Background Study

When there are multiple machines for execution of test suite, it is a good strategy to split the test suite into multiple sub test suites and execute them on different machines. The idea is to bring down the execution cycle duration by making use of parallel execution setup. Regression test team can maintain the execution time data of each test case which will help further activities. Then comes the strategy of splitting the test suite into suitable sub test suites.

3. Methodology and Algorithms

In this paper two algorithms are used:

3.1 Machines with identical execution speeds.

1. Create a two dimensional data structure to hold the test suite execution times. First index is for machine and second individual test case on that machine.
2. Create a single dimensional data structure to hold the sum of all the test cases execution time for a given machine.
3. Reverse sort the execution time of all the test cases.
4. Distribute the reverse sorted test cases across the machines using simple modulo logic.
5. Once all the test cases are sorted, find the total execution of a given set for a given machine.
6. Now reverse sort the total execution of a given sets across the machines. The first entity in this reverse sorted list gives the longest execution of any set on given machines. Hence is the effective execution time of whole test suite.

3.2 Machines with different execution speeds.

1. Create a two dimensional data structure to hold the test suite execution times. First index is for machine and second individual test case on that machine.
2. Create a single dimensional data structure to hold the sum of all the test cases execution time for a given machine.
3. Don’t sort the execution speeds. Perform the weighted split of the test set in proportion of machine speeds.
4. Step 3 is for ensuring the speed machines execute more test cases than slower machines.
5. While calculating the total execution of a given sub test suite on given machine, take the speed of execution of machine into consideration.

6. Now reverse sort the total execution of a given sets across the machines. The first entity in this reverse sorted list gives the longest execution of any set on given machines. Hence is the effective execution time of whole test suite.

4. Python Version of Algorithms with Execution Results.

```
import math
import pandas
import os


def weighted_split_test_exec_time_list(original_test_exec_time_list, weight_list, absolute_machine_speeds):
    machine_i_test_set = []
    machine_i_test_set_exec_time = []
    prev_index = 0

    for weight in weight_list:
        next_index = prev_index + math.ceil((len(original_test_exec_time_list) * weight))
        machine_i_test_set.append(original_test_exec_time_list[prev_index: next_index])
        prev_index = next_index

    for i in range(len(weight_list)):
        machine_i_test_set_exec_time.append(
            sum(machine_i_test_set[i])/absolute_machine_speeds[i])

    print(machine_i_test_set)
    for i in range(len(weight_list)):
        print("Machine",i,"Will execute the following test cases",machine_i_test_set[i],"in",machine_i_test_set_exec_time[i],"unit time")


    local_sorted_execution_time_on_machines = sorted(machine_i_test_set_exec_time,reverse=True)
    print("Longest time is", local_sorted_execution_time_on_machines[0],"Which is effective execution time")

    print("All Machines will put together will be busy for",sum(machine_i_test_set_exec_time))



def identical_machines_total_execution_time(x, n):
    machine_i_test_set = []
    machine_i_test_set_exec_time = []
    x = sorted(x, reverse=True)

    for i in range(n):
        machine_i_test_set.append([])

    for i in range(len(x)):
        machine_i_test_set[i % n].append(x[i])

    for i in range(n):
        machine_i_test_set_exec_time.append(sum(machine_i_test_set[i]))

    #print(machine_i_test_set)

    for i in range(n):
        print("Machine",i,"Will execute the following test cases",machine_i_test_set[i],"in",machine_i_test_set_exec_time[i],"unit time")
        #print("Test Suite on Machine", i, "Will run for", machine_i_test_set_exec_time[i],"Units of Time")
```

```
    local_sorted_execution_time_on_machines = sorted(machine_i_test_set_exec_time,reverse=True)
    print("Longest time is", local_sorted_execution_time_on_machines[0],"Which is effective execution time")

    print("All Machines will put together will be busy for",sum(machine_i_test_set_exec_time))


def main():
    df = pandas.read_csv(os.path.join(os.getcwd(), "Regression\\testexecutiondata.csv"),
                         sep=',')

    data_set = df["execution_time"].to_list()

    print("Data set is",data_set)

    absolute_machine_speeds = [1, 2, 2, 1]
    weighted_machine_speeds = [.16, .32, .32, .16]
    identical_machine_speeds = [1, 1, 1, 1]

    identical_machines_total_execution_time(
        data_set, len(identical_machine_speeds))

    weighted_split_test_exec_time_list(data_set, weighted_machine_speeds, absolute_machine_speeds)


main()
```

To this code supply the following data.

```
test_case_no,execution_time
T1,20.1
T2,30
T3,40
T4,50
T5,13
T6,10
T7,12
T8,60
T9,15
T10,20.2
T11,24
T12,20.3
```

Results of execution are as follows:

```
Data set is [20.1, 30.0, 40.0, 50.0, 13.0, 10.0, 12.0, 60.0, 15.0, 20.2, 24.0, 20.3]

Machine 0 Will execute the following test cases [60.0, 24.0, 15.0] in 99.0 unit time
Machine 1 Will execute the following test cases [50.0, 20.3, 13.0] in 83.3 unit time
Machine 2 Will execute the following test cases [40.0, 20.2, 12.0] in 72.2 unit time
Machine 3 Will execute the following test cases [30.0, 20.1, 10.0] in 60.1 unit time
Longest time is 99.0 Which is effective execution time
All Machines will put together will be busy for 314.6

[[20.1, 30.0], [40.0, 50.0, 13.0, 10.0], [12.0, 60.0, 15.0, 20.2], [24.0, 20.3]]
Machine 0 Will execute the following test cases [20.1, 30.0] in 50.1 unit time
Machine 1 Will execute the following test cases [40.0, 50.0, 13.0, 10.0] in 56.5 unit time
Machine 2 Will execute the following test cases [12.0, 60.0, 15.0, 20.2] in 53.6 unit time
Machine 3 Will execute the following test cases [24.0, 20.3] in 44.3 unit time
Longest time is 56.5 Which is effective execution time
All Machines will put together will be busy for 204.5
```

5. Execution Results Analysis

First Algorithm is able to sort the longest test cases evenly across the identical execution speed machines. The second algorithm executes more test cases on speed machines and takes the speed of machines while calculating the effective speed of total test suite. Test cases and their execution time can be maintained in comma separated value files. This historical data can be maintained between successive test executions. For generating the data programming language features can be used with execution start and end time stamps. The test execution time is the difference between end time and start time. For the first hypothetical case, test team has four identical speed test machines. For the second case, there are four machines. Two machines are twice as speed as the rest of the two machines. Therefore, the weighted speed data is assigned the weights weighted_machine_speeds = [.16, .32, .32, .16] according to the setup. As can be seen from the results in the first case test cases with long execution time are evenly distributed across all the four machines. In the second case, more test cases are assigned to fast execution machines, while a smaller number of test cases are assigned to slow machines. This uneven distribution is to ensure that test cases make use of computational speeds appropriately.

When the test cases count is in excess of 1000, the methodology can be still employed as long as proper comma separated values are maintained properly. Using file read and write operation, additional value can be maintained in the comma separated value file which will tell which machine the test case is being assigned and then the test case suite can be split according to the additional field in the comma separated file. However, this methodology assumes there is no preset execution order of the test cases. If there is a particular order of test case execution, the test suite cannot be split using the algorithms just mentioned in the paper.

References

[1] Abhinandan H. Patil, “Computer System Performance Analysis an Informal Approach”, text book, 2020.
[2] Abhinandan H. Patil, “Mathematics Part 6: Mathematics Learning with Aid of Software”, text book, 2020.
[3] Abhinandan H. Patil, “Learning Python Through LAB Based Approach”, text book, 2020.
[4] Abhinandan H. Patil, “Regression Testing in Era of Internet of Things and Machine Learning”, text book, 2020.
[5] Anthony Croft et al. “Engineering Mathematics”, textbook, 2021.
[6] JVM Hotspot command line arguments =http://www.oracle.com/technetwork/java/javase/tech/vmoptionsjsp-140102.html
[7] E. Aranha and P. Borba, “Estimating Manual Test Execution Effort and Capacity Based on Execution Points”, International Journal of Computer and application, .2009.
[8] Eduardo Henrique da Silva Aranha, “Estimating Test Execution Effort Based on Test Specifications”, thesis, 2009.
[9] Imran Muneer, “Systematic Review on Automated Testing Types, Effort and ROI”, thesis, 2014.
[10] Suresh Nageswaran, “Test Effort Estimation Using Use Case Points”, Quality week, 2001.
[11] Bartolomeo Ovilio, “Test effort and test coverage: correlation analysis in a safety critical operating system”, thesis, 2012.
[12] Test environment management best practices = http://www.softwaretestinghelp.com/test-bed-testenvironment-management-best-practices/
[13] Tong-Yu Hsieh, Kuen-Jong Lee and Jian-Jhih You, “Test Efficiency Analysis and Improvement of SOC Test Platforms”, 16th IEEE Asian Test Symposium, 2007.
[14] [Luay Tahat, Bogdan Korel, Mark Harman and Hasan Ural, “Regression Test Suite Prioritization using System Models”, Wiley online Library, 2011.
[15] Aditya P. Mathur, “Foundations of software testing”, text book, 2013.
[16] Paul C. Jorgensen, “Software testing a craftman’s approach”, text book, 2013.
[17] Abhinandan H. Patil, Neena Goveas, Krishnan Rangarajan,"Regression Test Suite Execution Time Analysis using Statistical Techniques", International Journal of Education and Management Engineering(IJEME), Vol.6, No.3, pp.33-41, 2016.DOI: 10.5815/ijeme.2016.03.04
[18] J. V. Oenen, “Improving regression test code coverage using meta heuristics”, Thesis, 2010.

Authors’ Profiles

Abhinandan is the Author of 15 Books and 14 Scientific Articles. He is Life-Long learner with many areas of interests. Earlier, he worked in Wireless Network Software Organization as Lead Software Engineer for close to a decade. He was in the USA for two long stints and was instrumental in Releases of Mobility Manager at Motorola USA as Single Point of Contact for Network Simulator Tool. His Research is available as Books and Thesis in IJSER, USA. His thesis published as Book is rated as one of the best Books of all time for Regression testing by BookAuthority.org. He is Active Researcher in the field of Computational Mathematics, Machine Learning, Deep Learning, Data Science, Artificial Intelligence, Regression Testing applied to Networks, Communication, and Internet of Things. He is an active contributor of Science, Technology, Engineering and Mathematics. He has started Blogging recently on Technology and Allied Areas. He is National and International awardee and also Senior IEEE member since 2013 and member of Smart Tribe and Cheeky Scientists Association. He is UGC-NET Qualified (2012). Recipient of several Bravo awards for deserving work at Motorola. He is on the Editorial Board of a few Scientific Journals. He is an ardent reader of STEM (Science, Technology, Engineering and Mathematics). He has a desire to contribute more to STEM.

Sangeeta is an Engineer by qualification and is a professional software tester. She has keen interest in software testing.