ISSN: 2994-6433

Research article

Journal of Sensor Networks and Data Communications

Algorithms for Test Suite Split in Multi Machine Setup with Symmetrical and Asymmetrical Machine Execution Speeds

Abhinandan H. Patil* **and Sangeeta A. Patil**

Educational Content Creators at 14AISS, Karnataka, India

***Corresponding Author**
Abhinandan H. Patil, Senior IEEE Member and Fellow IEI, Karnataka, India.

Submitted: 2024, May 31; **Accepted**: 2024, Jul 01; **Published**: 2024, Jul 04

Citation: Patil, A. H., Patil, S. A. (2024). Algorithms for Test Suite Split in Multi Machine Setup with Symmetrical and Asymmetrical Machine Execution Speeds. *J Sen Net Data Comm, 4*(2), 01-07.

Abstract

Regression test suite study can be on various parameters such as code and requirement coverage, test suite execution time reduction. The focus of the article is on reduction of test suite execution in multi machine setup. Two cases are possible. Test setup with symmetrical execution speed of machines and asymmetrical execution speeds. This article proposes four algorithms for both symmetrical and asymmetrical execution combined. Our previous published article explains the case of identical split for identical execution speed machines and weighted split for asymmetrical execution speed cases. We build on those two algorithms and total four algorithms are proposed. Two new algorithms are being proposed in this article. The logic underlying these algorithms is efficient usage of execution speeds of the machines. Although the article is for regression test suite execution, the algorithms can be beneficial in all cases where queuing is involved and service time of each entity is known prior. The algorithm and python version of the code is shared in this article for ready reference. While the algorithms can be beneficial for the test suite execution reduction for edge cases where order of test cases execution is must these algorithms should not be used. As mentioned earlier 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.

Keywords: 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

Regression test execution study focuses on how to make efficient usage of the lab resources aka hardware resources. This article assumes multi machine test setup at the disposal of test team. This article addresses the problem definition of splitting the large test suite across various machines using algorithms. The test suite split algorithm decides which test case should be executed on which machine. Further these algorithms can be beneficial for cases where historical test execution data is maintained by the test teams. While we touch upon case of lab with symmetrical execution speeds, three algorithms are dedicated to more realistic case of asymmetrical execution speed machines. The two algorithms already published are re-referenced for the benefit of the readers and can be skipped if the reader has already gone through them. However, two new algorithms are introduced for asymmetrical execution speed test setup. The following sections discuss the algorithms and python version of the algorithms.

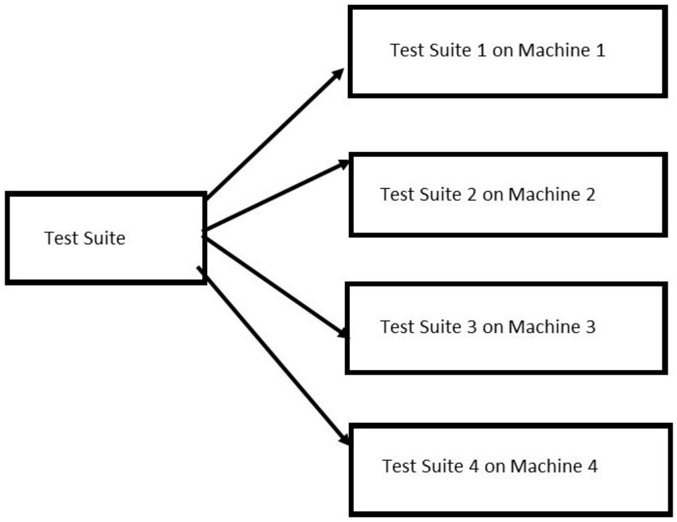
Test Suite
Test Suite 1 on Machine 1
Test Suite 2 on Machine 2
Test Suite 3 on Machine 3
Test Suite 4 on Machine 4

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. Weighted Test Suite Split for 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.

3.3. Identical Test Suite Split for Machines with Different Execution Speeds Followed by Distribution on Machines Sorted with Respect to Execution Speeds

1. Create a single dimensional data structure to hold the sub suites post regression test suite split.
2. Create a single dimensional data structure to hold the sum of all the test cases execution time for a given machine.
3. Sort the test execution time of the test cases so that the long execution test cases appear at the front of the list meant to hold the test execution time of all the test cases.
4. Sort the test machines based on their execution speeds.
5. Now split the test suite into identical sub suites with respect to test cases count.
6. Calculate the total execution time of given sub suite considering the absolute test execution into consideration.
7. 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.

3.4. Identical Test Suite Split for Machines with Different Execution Speeds Followed by Round Robin Distribution on Machines Sorted with Respect to Execution Speeds

1. Create a two dimensional data structure to hold the sub suites post regression test suite split.
2. Create a single dimensional data structure to hold the sum of all the test cases execution time for a given machine.

3. Sort the test execution time of the test cases so that the long execution test cases appear at the front of the list meant to hold the test execution time of all the test cases.
4. Sort the test machines based on their execution speeds.
5. Now split the test suite into identical sub suites with respect to test cases count in round robin manner so that test cases with comparable test execution time are evenly distributed.
6. Calculate the total execution time of given sub suite considering the absolute test execution into consideration.
7. 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
identical_split_of_test_exec_time_list_assymetrical_speed_round_robin_post_sort(original_test_exec_time_list,
absolute_machine_speeds):
    print('identical_split_test_exec_time_list_assymetrical_speed_round_robin_post_sort at work')
    machine_i_test_set = []
    machine_i_test_set_exec_time = []
    original_test_exec_time_list = sorted(original_test_exec_time_list, reverse=True)
    absolute_machine_speeds = sorted(absolute_machine_speeds, reverse=True)


    n = len(absolute_machine_speeds)

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

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


    for i in range(len(absolute_machine_speeds)):
        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(absolute_machine_speeds)):
        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_split_of_test_exec_time_list_assymetrical_speed(original_test_exec_time_list,
absolute_machine_speeds):
    print('identical_split_test_exec_time_list_assymetrical_speed at work')
    machine_i_test_set = []
    machine_i_test_set_exec_time = []
    prev_index = 0
    original_test_exec_time_list = sorted(original_test_exec_time_list, reverse=True)
    absolute_machine_speeds = sorted(absolute_machine_speeds, reverse=True)
```

```
        for i in range(len(absolute_machine_speeds)):
            next_index = prev_index + len(original_test_exec_time_list) // len(absolute_machine_speeds)
            machine_i_test_set.append(original_test_exec_time_list[prev_index: next_index])
            prev_index = next_index

        for i in range(len(absolute_machine_speeds)):
            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(absolute_machine_speeds)):
            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 weighted_split_of_test_exec_time_list_assymetrical_speed(original_test_exec_time_list, weight_list,
absolute_machine_speeds):
        print('weighted_split_test_exec_time_list_assymetrical_speed at work')
        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_split_of_test_exec_time_list_symetrical_speed(x, n):
        print('identical_split_test_exec_time_list_symetrical_speed at work')
        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(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(), "Algorithms\\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_split_of_test_exec_time_list_symetrical_speed(data_set, len(identical_machine_speeds))

        weighted_split_of_test_exec_time_list_assymetrical_speed(data_set, weighted_machine_speeds,
absolute_machine_speeds)
        identical_split_of_test_exec_time_list_assymetrical_speed(data_set, absolute_machine_speeds)
        identical_split_of_test_exec_time_list_assymetrical_speed_round_robin_post_sort(data_set,
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]
identical_split_test_exec_time_list_symetrical_speed at work
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
weighted_split_test_exec_time_list_assymetrical_speed at work
[[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
identical_split_test_exec_time_list_assymetrical_speed at work
[[60.0, 50.0, 40.0], [30.0, 24.0, 20.3], [20.2, 20.1, 15.0], [13.0, 12.0, 10.0]]
Machine 0 Will execute the following test cases [60.0, 50.0, 40.0] in 75.0 unit time
Machine 1 Will execute the following test cases [30.0, 24.0, 20.3] in 37.15 unit time
Machine 2 Will execute the following test cases [20.2, 20.1, 15.0] in 55.3 unit time
Machine 3 Will execute the following test cases [13.0, 12.0, 10.0] in 35.0 unit time
Longest time is 75.0 Which is effective execution time
All Machines will put together will be busy for 202.45
identical_split_test_exec_time_list_assymetrical_speed_round_robin_post_sort at work
[[60.0, 24.0, 15.0], [50.0, 20.3, 13.0], [40.0, 20.2, 12.0], [30.0, 20.1, 10.0]]
Machine 0 Will execute the following test cases [60.0, 24.0, 15.0] in 49.5 unit time
Machine 1 Will execute the following test cases [50.0, 20.3, 13.0] in 41.65 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 72.2 Which is effective execution time
All Machines will put together will be busy for 223.45000000000002
```

5. Execution Results Analysis

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 involving identical execution speeds, test team has four identical speed test machines. For the asymmetrical 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 identical test execution speed case, test cases with long execution time are evenly distributed across all the four machines. In asymmetrical test execution speed setup, we employ three different algorithms viz. weighted split, identical split, identical split with round robin distributions. The observed results are very much supplied data specific. However, they demonstrate the working of algorithms for small set of test cases data supplied. 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.

6. Conclusion and Future Works

The Algorithms are able to achieve the intended functionality for both the cases viz. Symmetrical speed execution setup and asymmetrical speed execution setups involving more than one execution machines. Although the Algorithms are being proposed for test execution, in general the Algorithms can be used for any queueing scenario where service time is known apriori. The proposed Algorithms can be used in Industrial setups. As part of the future work the Algorithms will be proposed to appropriate Industry counterparts and feedback will be incorporated appropriately [1-11].

References

1. https://github.com/Abhinandan1414/Parallel Execution Machines Setup Algorithms For Regression Testing
2. Patil, A. H., & Patil, S. A. (2024). Quantification of Regression Test Suite Execution Time in Parallel Execution Setup with Weighted Test Suite Split Algorithm. *J Sen Net Data Comm, 4*(1), 01-04.
3. Patil, A. H. (2020). Computer System Performance Analysis an Informal Approach. Text Book.
4. Patil, A. H. (2020). Mathematics Part 6: Mathematics Learning with Aid of Software. Text Book.
5. Patil, A. H. (2020). Learning Python Through LAB Based Approach. Text Book.
6. Patil, A. H. (2020). *Regression Testing in Era of Internet of Things and Machine Learning. Lulu*. com.
7. Anthony Croft et al. (2021). Engineering Mathematics. Text Book.
8. Bird, J. (2018). *Bird's comprehensive engineering mathematics*. Routledge.
9. Van Rossum, G., & Drake Jr, F. L. (1995). Python tutorial.
10. Test Environment Management Best Practices = http://www.softwaretestinghelp.com/test-bed-testenvironment-management-best-practices/
11. Patil, A. H., Goveas, N., & Rangarajan, K. (2016). Regression Test Suite Execution Time Analysis using Statistical Techniques. *IJ Education and Management Engineering, 6*(3), 33-41.