ISSN: 2994-6433

Research Article

Journal of Sensor Networks and Data Communications

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

Abhinandan H. Patil* and Sangeeta A. Patil

Educational Content Creators at 14AISS, Karnataka, India

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

Submitted: 2023, Dec 22; **Accepted**:2024, Jan 29: **Published**: 2024, Feb 02

Citation: 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.

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.

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

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.

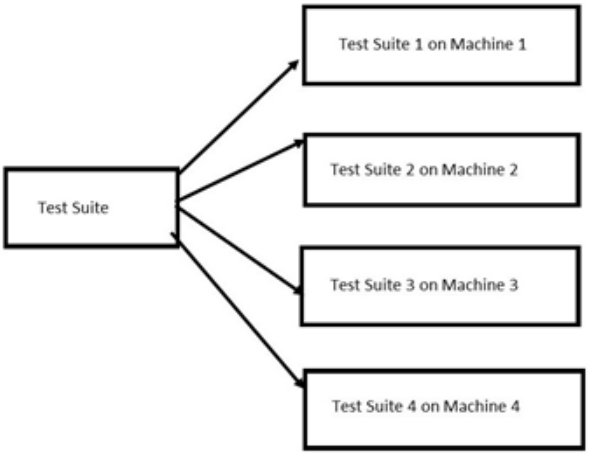

Figure 1: Test Suite Parallel Execution

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

• 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.
• Create a single dimensional data structure to hold the sum of all the test cases execution time for a given machine.
• Reverse sort the execution time of all the test cases.
• Distribute the reverse sorted test cases across the machines using simple modulo logic.
• Once all the test cases are sorted, find the total execution of a given set for a given machine.
• 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

• 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.
• Create a single dimensional data structure to hold the sum of all the test cases execution time for a given machine.
• Don't sort the execution speeds. Perform the weighted split of the test set in proportion of machine speeds.
• Step 3 is for ensuring the speed machines execute more test cases than slower machines.
• While calculating the total execution of a given sub test suite on given machine, take the speed of execution of machine into consideration.
• 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.

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 [1-18].

References

1. Abhinandan H. Patil. (2020). Computer System Performance Analysis an Informal Approach. Text Book.
2. Abhinandan H. Patil. (2020). Mathematics Part 6: Mathematics Learning with Aid of Software. Text Book.
3. Abhinandan H. Patil. (2020). Learning Python Through LAB Based Approach. Text Book.
4. Patil, A. H. (2020). *Regression Testing in Era of Internet of Things and Machine Learning*. Lulu. com.
5. Anthony Croft et al. (2021). Engineering Mathematics. Text Book.
6. JVM Hotspot command line arguments.
7. Aranha, E., & Borba, P. (2009). Estimating manual test execution effort and capacity based on execution points. *International Journal of Computers and Applications, 31*(3), 167-172.
8. da Silva Aranha, E. H. (2009). *Estimating Test Execution Effort Based on Test Specifications* (Doctoral dissertation, thesis).
9. Muneer, I. (2014). Systematic Review on Automated Testing (Types, Effort and ROI).
10. Nageswaran, S. (2001, June). Test effort estimation using use case points. *In Quality week* (Vol. 6, pp. 1-6).
11. Di Leo, D., Natella, R., Pietrantuono, R., & Ovilio, B. (2012). Test effort and test coverage: correlation analysis in a safety critical operating system.
12. Test environment management best practices.
13. Hsieh, T. Y., Lee, K. J., & You, J. J. (2007, October). Test efficiency analysis and improvement of SOC test platforms. In *16th Asian Test Symposium (ATS 2007)* (pp. 463-466). IEEE.
14. Tahat, L., Korel, B., Harman, M., & Ural, H. (2012). Regression test suite prioritization using system models. *Software Testing, Verification and Reliability, 22*(7), 481-506.
15. Mathur, A. P. (2013). *Foundations of software testing, 2/e.* Pearson Education India.
16. Jorgensen, P. C. (2013). *Software testing: a craftsman's approach.* Auerbach Publications.
17. 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.
18. Oenen, J. V. (2010). *Improving regression test code coverage using meta heuristics* (Doctoral dissertation, Thesis).