An Efficient Code-defect Testing Distributed System

Mingnan Zhou¹, Dahai Jin¹⁺, Songyi Liu², Yunzhan Gong¹

¹Institute of Network Technology, Beijing University of Posts and Telecommunications, Beijing, China
²China Launch and Tracking Control, Beijing, China

*Corresponding author e-mail: jindh@bupt.edu.cn

Abstract. How to improve the efficiency of testing code files has become an increasingly hot topic. This paper focuses on enhancing the efficiency by designing and implementing a distributed system. By showing the detail design of the parts constituting the system framework and the steps of the code project to be handled in the flowchart, this paper tells how to processes the code files with the workload evaluation, labeling, label handling and load balance to guide the system to take the testing work. During the whole procedure, as it is handled in a distributed system, the integrated code project need splitting into several parts, which will bring much problem of the dependency relationship among the files. This paper mainly aims to solve the problem of dependency from several aspects, achieves the goal of splitting the files and finally raising the efficiency of the code-defect testing work.

1. Introduction

Nowadays, the development of information technology has entered a new stage. Whether it is in the area of big data, cloud computing, machine learning or blockchain technology, it needs code programming to implement it. The quality of the code directly affects the actual application effect. Internationally, software testing (software quality control) is a very important engineering job, and also serves as a very independent profession. In the development of large-scale system software companies such as IBM and Microsoft, the amount proportion of developer and tester of many important projects can reach 1:2 or even 1:4[1], and a nice testing tool can help to save a lot of labor costs for testing. As the amount of program code continues to increase [2], the demanding of efficiency of code reliability analysis has been put to a remarkable level. This paper design and implement a distributed system considering the dependency relationship and the system’s load balance, and improve the efficiency of the testing work by enhancing the testing computing ability of the system.

2. Overview

2.1 Framework Introduction

This distributed system is constituted by a cluster of virtual machines which can be divided into two parts: the master node and the slave nodes. The master node plays the role of macro control and command to whole distributed system, and the slave nodes are in charge of the detailed testing work. Moreover, a group of certain slave nodes that are also determined by the master node comprise the Sub Cluster which is responsible for a unique project to be tested. Each Sub Cluster corresponds to a distinct project to be tested. The different slave node in the same Sub Cluster takes the different part of
the testing work of the project, and the parts that are in charge of by all slave nodes in the same Sub Cluster constitute the whole testing work of this project to be tested. When the testing work is finished, the Sub Cluster will be disbanded and the calculation ability will be released.

2.2 Flowchart Introduction
The master node receives the code project to be tested from users, takes the initial analysis, and sends the order to the slave nodes requiring which nodes to deal with which part of testing tasks. Then the Sub Cluster forms and the slave node takes the testing work. The total flowchart can be shown as Figure 1.

When the testing task is submitted, the files of project to be tested will firstly be initialized and classified into collections. In this part, the project will also be analysis and the workload of it will be given. And according to the workload and the available slave nodes, the master node will produce an execution strategy that describes how this project will be split and delivered to the slave nodes. When the split code project arrive at the corresponding slave node, the testing work will be done by the testing software on the slave node. And when the testing work is finished, the result report on each slave node will be generated and then be recovered from each slave node and output the total report.

Figure 1. The Flowchart of This Distributed System.

2.3 Testing Tool on Slave Node
At present, there are many mature detection tools available for use. Internationally representative PMD[3], Findbugs[4] and commercial Klocwork[5], etc. In China, there is Defect Testing System (DTS) developed by the State Key Laboratory of Networking and Switching Technology of Beijing University of Posts and Telecommunications under the support of the National ‘863’ fund project [6].Since 2006, DTS has invested 150 man-years and applied for more than 20 related patents. At present, it has reached the international advanced level in related major technical indicators [7]. The author is one of the member of this laboratory, and is participating in the development of the DTS and is much familiar with DTS. Thus this distributed system deploys the DTS as the testing tool on the slave nodes to take the testing work.

3. Detailed Design
The whole system aims to solve the problem of dependency relationship and load balance. The dependency relationship means that one code file may need the information, like the variable or method, in another code file. As project files will be split, these files with referenced relationship may be not in the same testing environment, and the necessary information may not be called as normally as it is in the single system. These kinds of problem caused by the dependency relationship among files this paper defines it as Dependency Problem. For the dependency problem, this paper mainly gives two ways to deal with it based on two steps of before and during the testing: (1) before the testing, pretreat the source file to reduce the dependency relationship among the files; (2) during the testing, for those files whose dependency relationship cannot be canceled, this system try to allocate these files into the same slave node to ensure the necessary information they can get when these files are tested in the slave nodes.

3.1 Initialization
The Initialization part mainly takes the preprocessing work to the code files aiming to reduce the dependency relationship among the code files, is also a method to solve the dependency problem before the testing work. The C and C++ source code files should firstly be handled with the following
procedures then can we get the executable files[8]: get the middle file (‘.i’ file) with the preprocessing, get the assembly file (‘.s’ file) with the compiling, get the object file (‘.o’ file) with assembling and get the executable file (‘.exe’ file) with linking.

3.2 Label Design
Firstly this system will take the analysis of abstract syntax tree and symbol table to the project files, and the dependency relationship can be analysis with the module. The files in the same module are within the same group of the dependency relationship, and the group of files in other module will not influence the files in this module by dependency. Hence this paper takes the Labeling module by module, taking the L collection to describe a certain dependency relationship. Taking a series of L collections to show the different kinds of dependency, the files in the project to be tested are classified into different collections by the kind of dependency relationships. The Labeling Design is the basic work to reduce the influence among the procedure of testing by the distributed system.

3.3 Load Balance
This distributed system is built based on the virtual machine, and each salve node in the cluster is set with the same configuration, hence each of them should have the same system performance and the same calculation ability. And the cost of time depends on the node’s task to do and node’s calculation performance, so the key factor to achieve the load balance of the system is to allocate the similar volume of the testing work to the slave node.

3.3.1 Label Collation. As the code files with the same label is in the same dependency relationship, they should be tested on the same slave node considering the dependency problem. Although each file is labelled differently after dependency processing, it can theoretically be delivered to different testing nodes for testing according to the label's classification (that is, a label or a dependency corresponds to a testing node), however, the actual operation needs to consider the optimization of the system and the maximum use of resources. The most common situation is that the files of different labels has high similarity and overlap.

![Figure 2. The Label Collation](image)

As Figure 2 (a) describes, the files of file1, file2, file3 and file 4 are all with Label A and file1, file2, file3 and file5 are with Label B. If we put the files with Label A into a testing slave node and files with Label B into another testing slave node, the files of file1, file 2 and file3 will be tested twice. Under normal circumstances (that is, the test costs for file4 and file5 are in one order of magnitude, and not in the same magnitude will be discussed later), such a test strategy will result in a waste of system resources. However, if we put files with Label A and Label B into the same testing node as the Figure 2 (b) shows, although it seems that one testing node needs to handle two label kinds of files, the total account of files that are tested actually is reduced.
3.3.2 Unit Collation. In order to solve the above problem and considering the calculation efficiency of this distributed system, this paper collates these labels into a new collection called ‘Unit’ (abbreviated as ‘U’). We put the labels with high overlap of files into the same unit and take the union testing. Each U collection (U1, U2, U3…Un) contains a group of label kinds of files, and each U(x) collection can also be considered as a result of processing ‘reduced repetition’ to the label on behalf of the files. This progress is called the Unit Collation. Before Unit Collation, the files are classified with different labels, and after it takes the Unit Collation, the files are classified into the Unit level according to their Labels. The treatment we put Label A and Label B together in Figure 4 (b) can be though as a kind of Unit Collation.

After optimizing the problem of high ‘duplicate’ files in different Labels, we got the ‘Unit’ level collections. Theoretically, we can directly deliver the set of files in the collection of Unit level to different testing nodes to test, but this is not conducive to the balanced call of the overall resources of the distributed system. After the label processing we have three units, namely U1, U2, and U3, which respectively contain file1 with label A, file2 with label B, and file3 with label C. If delivery is performed on a unit basis, U1, U2, and U3 are delivered to slave1, slave2, and slave3, respectively. If the time of processing U1 is t1, U2 is t2, U3 is t3, and t1 + t2 < t3, such a task delivery scheme is regarded as unreasonable. Because slave1 and slave2 cannot be released although their testing tasks are completed. As they and slave3 are in the same sub-cluster, slave1 and slave2 need to keep waiting until slave3 finishes its work. Here if we deliver U1 and U2 to the same testing node, on the one hand it will not extend the overall testing time (still t3), on the other hand will save the available resources in the system (saving the computing ability of slave2). This can be seen as an improvement on task delivery. Therefore, we need to continue to perform similar optimization on Unit-level collections in order to make better use of the computing resources of each testing node in the distributed system.

3.3.3 Task Collation and Load Balance. After taking the Unit Collation to the files, we get several collections of the unit level. As discussed earlier in this paper, in order to make the system load balance, the key factor is to allocate the similar workload to each slave testing node. Thus we need to handle these existing collections into a group of new dimension of collection Task (abbreviated as ‘T’) mainly considering about the workload.

When we get the unit level collections, we sign the workload to each unit by adding the workload of each file in the unit. This paper firstly calculate the total workload of all unit collections and get the account of available slave testing node, and then get the weighted average workload that should be sent to each slave node with the algorithm of Bin Packing [9]. By achieving this algorithm, the units are into new groups and form different T collections and each T collection has the similar workload. The above progress is called Task Collection. As Figure 3 shows, the WL(A) is U(a)’s workload, and WL(B) is U(b)’s and so on. After the Task Collation, T collections are formed. Each T collection (T(a) or T(b) or….T(s)) is formed by the several unit collections.

As each T collection has the similar workload and deals with the Dependency Problem of the code files in itself, each of them can be sent to different slave testing node to take the testing work.

![Figure 3. The Task Collation.](image-url)
3.3.4 Execution Strategy. Execution Strategy is a field that is sent from the master node to the corresponding slave nodes containing the necessary information of one time execution of testing the code project. This execution strategy is constituted by 5 parts with the key words of Tid, Pid, subCluster ifLeader and taskList. Tid (short for Task-id) identifies the basic information of the task including how many times is this project to be submitted to take the testing work, when is this project submitted. Pid holds the information of the project to be tested including the position the file is saved, the size of file and the workload of the file and so on. subCluster gives the constitution of the sub cluster for this project to be tested. ifLeader gives the identification of each slave node in the sub cluster (will be discussed in the part of 3.2 Allocation and Recovery). And taskList is mainly from the T collection showing the catalog of which files are tested on which slave testing node.

3.4 Allocation and Recovery

3.4.1 CAP theory and the satisfaction of ‘availability. Each distributed system must consider the CAP principle, which is a basic feature of all distributed systems. The CAP principles, namely Consistency, Availability, and Partition tolerance, cannot be combined with the three characteristics [10]. The basic characteristic of an efficient distributed defect detection system platform for DTS is that each slave node in the cluster that is responsible for a specific test task has a complete set of DTS installed. The test behavior of DTS can be seen as an atomization and the operation can now be operated stably. This is the most basic condition that this distributed system needs to meet, therefore, the principle of ‘usability’ of the distributed system must be satisfied in the CAP theory.

3.4.2 Priority of ‘consistency’ and ‘partitioned fault tolerance. In this distributed system, the master distributes tasks according to the execution strategy. With the progress of test tasks on different nodes in the sub-cluster, the system considers that the test results brought by the file dependencies are not accurate enough, so that the test nodes are adjusted accordingly. Therefore, the consistency of the information set in the execution strategy is very important. At the same time this approach also takes into account the ‘partition fault tolerance’ situation, that is, a node downtime, the test tasks undertaken by it can find backups in other nodes.

After the test is over, the test results on each Follower will be returned to the Leader, and the Leader will return to the Master for integration. At this point, the system has basically met the requirements for consistency.

4. Experiment and result analysis

This paper takes 5 open-source code projects named spell-1.0, samba-3.0.23, barcode-0.98, playa and SpGateWay to test originally and with this distributed system, respectively. The configuration of the single machine and the node of the clusters (one Master node and three Slave nodes) on virtual machine are as Table 1 shows:

| TABLE 1. THREE SCHEME COMPARING. |
|----------------------------------|
| System Information               | Software environment          |
| OS: Windows 7 (32 bit)           | JDK: 1.7.0_02.                |
| Memory: 4.00GB.                  | DTS: 9.0                      |

The execution time of the centralized system and distributed system are shown as Table 2. According to the comparison the execution time, we can find the time spent on distributed systems is about one-tenths that of a centralized system, which shows this distributed system really improves the efficiency of the code-defect testing.

The statistical results show that the standard C engineering implementation of the DTS project is analyzed. The comparison of the analysis time and the use of the intermediate file reduction strategy is the result of the analysis of the simple strategy. After dealing with the intermediate files, and simplify the intermediate files. Defect analysis time was shortened obviously and defect analysis efficiency got obvious improvement.
TABLE II. ANALYSIS TIME COMPARISON

| Project name   | Time Spent on Single Machine(s) | Time Spent on Distributed System(s) |
|----------------|---------------------------------|-------------------------------------|
| spell-1.0      | 587                             | 525                                 |
| samba-3.0.23   | 790                             | 709                                 |
| barcode-0.98   | 1198                            | 1072                                |
| playa          | 989                             | 857                                 |
| SpGateWay      | 1953                            | 1592                                |

5. Conclusion and Further Work
This paper focuses on the system level to improve the testing efficiency by taking the distributed system rather than centralized one. Taking the Master – Slave model and ZAB principle, each node inside this distributed system has a clear role assignment and a stable operating mechanism. After the project to be tested is submitted, it is firstly taken the initialization work in the Master node. During the initialization step, the system analyzes the project and gives its corresponding workload as the basis for judgment for load balance, and then label the certain files with dependency relationship. By handling these labels into Unit collaboration and Task collaboration, the system continuously optimizes the task’s execution strategy and leads the slave nodes work with the established strategy, and enhance the code-defect efficiency by nearly 10% as a result.

For system-level improvements, the future research should pay much attention to the design of the task scheduling, for now this distributed system can just deal with submitting only one task at a time. The ideal situation is that the users can continuously submit the code project to be tested and the system can make reasonable scheduling arrangements for the submitted tasks, and then completes the execution strategy according to this arrangement and the experience of the distributed system will also be better.

Acknowledgments
This work was financially supported by the National Natural Science Foundation of China (No. U1736110) and the National Natural Science Foundation of China (No.61702044)

References
[1] Yi Jieming, ‘Research on the Development Status and Trend of Software Testing at Home and Abroad[J]’. Computer Knowledge and Technology, 2013(26):6020-6022.
[2] Xiao Yifei, Jin Dahai, ‘A Method of Tainted Data Detection Based on Static Code Analysis’, International Conference on Mechatronics, 2015
[3] PMD, PMD official website, https://pmd.github.io/
[4] Findbugs, Findbugs official website, http://findbugs.sourceforge.net/
[5] Klocwork Insight, Klocwork Insight official website,
[6] Jin Dahai, Gong Yunzhan, Wang Yawen, et al. ‘Software Code Testing Technology[J]’. Information and Communication Technology. 2015:33-39
[7] Wu Xin, ‘Design and Implementation to Job Scheduling of Code Defect Test System’, vol 2, pp. 2, 2016
[8] Alex Smith, ‘Building C Projects’, http://nethack4.org/blog/building-c.html
[9] Wikipedia editor, Wikipedia website, https://en.wikipedia.org/wiki/Bin_packing_problem
[10] Nancy Lynch, Seth Gilbert, ‘Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services’, ‘Acm Sigact News’, 2002, 33 (2) :51-59