Fault Localization Evaluation Framework

IN4MATX 215

Milestone 3

Team Member:

Chiyu Cheng

Yi Zhou

Introduction

Testing is one of the most critical and expensive tasks in the development of software. The process of detecting and locating bugs in a software system is called fault localization. Manually fault localization is acceptable for the small software system but unbearable for the large-scale system. Therefore, an automated fault localization technique is necessary for the real-world complex software system. There are many fault localization techniques like Tarantula, Jugular, and etc. However, there is no a comprehensive evaluation framework to evaluate and compare those techniques. An evaluation framework will guide the researchers to choose the right direction and method in the fault localization field. In addition, the process of evaluating different fault localization techniques will help us to understand this field better, and also help us to answer our research question - what is the best fault localization technique to locate the faults in the large-scale system?

Challenges

There are many challenges to implement this evaluation framework. The biggest challenge is how to build a runnable program of each technique we evaluate. It is impossible for us to re-implement the techniques by ourselves within only one month. Therefore, we are planning to use the existed code provided by the paper to test it. Another big headache is how to evaluate those techniques under the same standing since each technique has different context or definition. For the runnable techniques, we will try to run it on several open source programs to check their performance based on our criteria.

Approach

In order to answer the research question, what is the best method to locate the concurrent faults in the large-scale system, we will develop an evaluation framework to study and evaluate the current state of the art in fault localization. The plan is surveying 8-10 papers in this topic and evaluating the methods provided in those papers under the same standard. For each of the paper, we will run the corresponded program on several open source programs, including the AsterixDB, Elasticsearch, and more if we have enough time. We will collect the data from our experiments and analyze it based on our evaluation criteria. The basic evaluation criteria are feasibility, efficiency, and accuracy.

In this project, feasibility means the easiness to implement and deploy one technique to existing programs. It is hard to convince a programmer to adopt one testing technique when it is hard to implement or integrate it on their programs. Therefore, we believe that the feasibility of one technique is worth to evaluate. We will collect the total amount of time we spend to configure the development environment and deployment for each fault localization technique, and use this time as the measurement of the feasibility of each technique.

Efficiency includes the time complexity of the algorithm for each technique, and the resource usage like CPU, memory, or disk. Efficiency is important for most programmers since they might don't have enough time or computing resource to adopt one new technique. In order to measure the efficiency of each technique, we will run those techniques for the same open source program in the virtual machine with same configuration, and monitor the CPU and memory usage under the same amount of time.

Accuracy stands for the success rate to find the real concurrent bugs under a certain number of executions. Without accuracy, A technique is useless even if it is feasible and efficient. For now, we are planning to use those three basic criteria to evaluate different techniques, and we might add more criteria after we finish reading more related papers. In this evaluation frame, we will run those techniques for the same open source program in the virtual machine with same configuration many times, and record the average number of real fault they find.

Evaluation

The Evaluation of an evaluation framework is always tricky since it is hard to provide any data or charts. One of the most important purposes of this project is helping the author and reader to learn and understand the field of fault localization. Another essential purpose is helping researchers or programmers to find the best localization technique based on their specific requirement. Therefore, if the key reader, professor Ahmed, or the majority of the reader, agrees that this paper is helpful for achieving the two major purposes mentioned above, then this paper is qualified.

Motivation Example

Fault localization could be a tricky problems and different techniques might give different results to the same program. The results given by those fault localization program is suspiciousness, and how to design a framework or a reliable metrics could be a great challenge.

For example, look at the following program:

```
mid(){
        int x,y,z,m;
        read("Enter 3 numbers:", x, y, z);
        m = z;
        if(y < z)
                if(x<y)
                m=y;
        else if(x<z)
        m=y;//***bug***
        else
                if(x>y)
                        m = y
                else if(x>z)
                        m=x;
        print(Middle number is:", m);
}
```

As we can see, it's a quite simple program, which just calculate the middle value of three numbers, and the obvious bug has been flagged. however, according to our previous survey, different fault localization tools might differs when give suspiciousness to different line of code. So how to compare their effiency according to their given-out suspiciousness could be challenging, especially when it comes to a larger project with more than ten thousands line of code. Besides, how to compare the effiency of those fault-localization tools is also a critical problem that we will try to solve.

Example Solution

In this section, we will illustrate how we are going to evaluate the technique named Tarantula as an example. First, we pull the source code from the GitHub https://github.com/spideruci/Tarantula, add dependencies and required libraries.

Tarantula is one of the popular fault localization tools which based on simple intuition that entities in a program that are primarily executed by failed test cases are more likely to be faulty than those that are primarily executed by passed test cases. It utilizes the information readily

available from standard testing tools. Such information includes: the pass/fail information about each test case, the entities that were executed by each test case and the source code for the program under test. Unlike the techniques based on coverage information, Tarantula allows some tolerance for the fault to be occasionally executed by passed test cases. The central part of code calculating score is following:

```
double[][] calculateSuspiciousnessAndConfidence(
      int numStmts, int totalLivePass, int totalLiveFail,
      double[] passRatio, double[] failRatio) {
    double[] suspiciousness = new double[numStmts];
    double[] confidence = new double[numStmts];

    for (int i = 0; i < numStmts; i++) {

      if ((totalLiveFail == 0) && (totalLivePass == 0)) {
        suspiciousness[i] = -1d;
        confidence[i] = -1d;
      } else if ((failRatio[i] == 0d) && (passRatio[i] == 0d)) {
        suspiciousness[i] = -1d;
        confidence[i] = -1d;
      } else {
        suspiciousness[i] = failRatio[i]
            / (failRatio[i] + passRatio[i]);
        confidence[i] = Math.max(failRatio[i], passRatio[i]);
      }
    }

    return new double[][] { suspiciousness, confidence };
  }
```

Then we compile and build the code with Maven. During these steps, it almost cost me to 30 mins to fix the minor bugs to successfully compile it, which will be used as the metric to measure the feasibility of this project. Since we don't have enough time to test it on the large-scale system, I have to use the test to simulate this process. The code is:

```
public void expect_MaximumSuspiciousness_WithAllTestsFailing() {
  //given
  final TarantulaFaultLocalizer localizer = new TarantulaFaultLocalizer();
  final int numTests = 10;
  final int numStmts = 100;
  final int totalLivePass = 0;
  final int totalLiveFail = 10;
  final double[] passRatio = getSimpleUniformDoubleArray(0.0, numStmts);
  final double[] failRatio = getSimpleUniformDoubleArray(1.0, numStmts);
  //when
```

```
11.   double[][] snc =
12.       localizer.calculateSuspiciousnessAndConfidence(
13.           numTests, totalLivePass, totalLiveFail, passRatio, failRatio);
14.   //then
15.   assertThat(snc[0], everyDouble(equalTo(1.0)));
16. }
```

This test is simulating the case that the suspicious score calculated based on Tarantula should be the highest level when all the test case fails. We record the test result as accuracy, and the time spend to run this program as efficiency. This is an example of how we are going to do evaluate an fault localization technique on the large-scale open source program.