The Journal of Systems and Software 90 (2014) 18–28

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A dynamic code coverage approach to maximize fault localization efficiency

Alexandre Perez*, Rui Abreu, André Riboira

Department of Informatics Engineering, Faculty of Engineering, University of Porto, Porto, Portugal

ARTICLE INFO

Article history:
Received 31 January 2013
Received in revised form 9 December 2013
Accepted 16 December 2013
Available online 4 January 2014

Keywords:
Dynamic coverage
Software diagnosis
Spectrum-based fault localization

ABSTRACT

Spectrum-based fault localization is amongst the most effective techniques for automatic fault localization. However, abstractions of program execution traces, one of the required inputs for this technique, require instrumentation of the software under test at a statement level of granularity in order to compute a list of potential faulty statements. This introduces a considerable overhead in the fault localization process, which can even become prohibitive in, e.g., resource constrained environments. To counter this problem, we propose a new approach, coined dynamic code coverage (DCC), aimed at reducing this instrumentation overhead. This technique, by means of using coarser instrumentation, starts by analyzing coverage traces for large components of the system under test. It then progressively increases the instrumentation detail for faulty components, until the statement level of detail is reached. To assess the validity of our proposed approach, an empirical evaluation was performed, injecting faults in six real-world software projects. The empirical evaluation demonstrates that the dynamic code coverage approach reduces the execution overhead that exists in spectrum-based fault localization, and even presents a more concise potential fault ranking to the user. We have observed execution time reductions of 27% on average and diagnostic report size reductions of 77% on average.

1. Introduction

Automatic fault localization techniques aid developers/testers to pinpoint the root cause of failures, thereby reducing the debugging effort. Amongst the most diagnostic effective techniques is spectrum-based fault localization (SFL). SFL is a statistical technique that uses abstraction of program traces (also known as program spectra) to correlate software component (e.g., statements, methods, and classes) activity with program failures (Abreu et al., 2009c; Liu et al., 2006; Wong et al., 2008). As SFL is typically used to aid developers in identifying the root cause of observed failures, it is used with low-level of granularity (i.e., statement level).

Statistical approaches are very attractive due to the relatively small overhead with respect to CPU time and memory requirement (Abreu et al., 2009b,c). However, gathering the input information, per test case, to compute the diagnostic ranking may still impose a considerable (CPU time) overhead. This is particularly the case for resource constrained environments. The effort required to inspect SFL's diagnostic report is also noteworthy.

As said before, typically, SFL is used at development-time at a statement level granularity (since debugging requires to locate the faulty statement). But not all components need to be inspected at such detailed granularity. In fact, components that are unlikely to be faulty do not need to be inspected. With this reasoning in mind, we propose a technique, coined dynamic code coverage (DCC), that automatically adjusts the granularity per component. First, our approach instruments the source code using a coarse granularity (e.g., package level in Java), and then decides which components to *expand* based on the output of the fault localization technique. With expanding we mean changing the granularity of the instrumentation (e.g., in Java, for instance, instrument classes, then methods, and finally statements). This expansion can be done in different ways, for instance, by selecting the top ranked components, according to a set percentage.

Our empirical evaluation demonstrates that DCC has the potential to drastically reduce the execution overhead, while still maintaining the diagnostic effectiveness of statement-based spectrum-based fault localization. In our experiments, we have observed a time reduction of 27% on average. Furthermore, a 77% reduction of the diagnostic report size was observed in our empirical evaluation, lessening the effort required by developers to perform an inspection.

In particular, this paper makes the following contributions:

- We propose DCC, a technique that automatically decides the instrumentation granularity per module in the system.

* Corresponding author. Tel.: +351 960002243.
E-mail addresses: alexandre.perez@fe.up.pt (A. Perez), rui@computer.org (R. Abreu), andre.riboira@fe.up.pt (A. Riboira).

0164-1212/$ – see front matter © 2014 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.12.036

- We provide an implementation of the DCC approach within the GZoltar (Campos et al., 2012) testing framework.
- An empirical study to validate the proposed technique, demonstrating its efficacy and efficiency using real-world, large programs. The empirical results show that DCC can indeed decrease the overhead imposed in the software under test, while still maintaining the same diagnostic accuracy as current approaches to fault localization. DCC also decreases the diagnostic report size when compared to traditional SFL.

This work builds on top of previous work (Perez et al., 2012), where we proposed a lightweight topology-based model to estimate the diagnostic efficiency of fault localization techniques, extending it as follows. First we provide a motivation for a hierarchical approach to fault localization. Second, we detail our proposed technique, coined DCC. Finally, we provide an empirical evaluation of the efficacy and efficiency of both DCC and our topology-based analysis model by injecting single and multiple faults into real-world, large applications.

The remainder of this paper is organized as follows. In Section 2 we present concepts relevant to this paper as well as a motivational example for our work. In Section 3 the dynamic code coverage approach, DCC, is described. In Section 4, a topology-based analysis to assess whether to use spectrum-based fault localization (SFL) or DCC is detailed. In Section 5 the findings of our empirical evaluation are presented. We compare DCC with related work in Section 6. In Section 7 we conclude and discuss future work.

2. Concepts and motivational example

In this section, we introduce the concept of program spectra, and its use in fault localization. Throughout this paper, we use the following terminology (Avižienis et al., 2004):

- A *failure* is an event that occurs when delivered service deviates from correct service.
- An *error* is a system state that may cause a failure.
- A *fault* (defect/bug) is the cause of an error in the system.

In this paper, we apply this terminology to software programs, where faults are bugs in the program code. Failures and errors are symptoms caused by faults in the program. The purpose of fault localization is to pinpoint the root cause of observed symptoms.

Definition 1. A software program Π is formed by a sequence M of one or more statements.

Given its dynamic nature, central to the fault localization technique considered in this paper is the existence of a test suite.

Definition 2. A test suite $T = \{t_1, \ldots, t_N\}$ is a collection of test cases that are intended to test whether the program follows the specified set of requirements. The cardinality of T is the number of test cases in the set $|T| = N$.

Definition 3. A test case t is a (i, o) tuple, where i is a collection of input settings or variables for determining whether a software system works as expected or not, and o is the expected output. If $\Pi(i) = o$ the test case passes, otherwise fails.

2.1. Program spectra

A program spectrum is a characterization of a program's execution on a dataset (Reps et al., 1997). This collection of data, gathered at runtime, provides a view on the dynamic behavior of a program. The data consists of counters or flags for each software component. Various different program spectra exist (Harrold et al., 2000), such as path-hit spectra, data-dependence-hit spectra, and block-hit spectra.

$$N \text{ spectra} \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1M} \\ a_{21} & a_{22} & \cdots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NM} \end{bmatrix} \quad \begin{matrix} M \text{ components} \\ \\ \\ \\ \\ \end{matrix} \quad \begin{matrix} \text{error} \\ \text{detection} \\ \\ \\ \\ \end{matrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix}$$

Fig. 1. Input to SFL.

In order to obtain information about which components were covered in each execution, the program's source code needs to be instrumented, similarly to code coverage tools (Yang et al., 2006). This instrumentation will monitor each component and register those that were executed. Components can be of several detail granularities, such as classes, methods, and lines of code.

2.2. Fault localization

A fault localization technique that uses program spectra, called SFL, exploits information from passed and failed system runs. A passed run is a program execution that is completed correctly, and a failed run is an execution where an error was detected (Abreu et al., 2009c). The criteria for determining if a run has passed or failed can be from a variety of different sources, namely test case results and program assertions, among others. The information gathered from these runs is their hit spectra (Abreu et al., 2009c).

The hit spectra of N runs constitutes a binary $N \times M$ matrix A, where M corresponds to the instrumented components of the program. Information of passed and failed runs is gathered in a N-length vector e, called the error vector. The pair (A, e) serves as input for the SFL technique, as seen in Fig. 1.

With this input, fault localization consists in identifying what columns of the matrix A resemble the vector e the most. For that, several different similarity coefficients can be used (Jain and Dubes, 1988). One of the most effective is the Ochiai coefficient (Abreu et al., 2007), used in the molecular biology domain:

$$s_O(j) = \frac{n_{11}(j)}{\sqrt{(n_{11}(j) + n_{01}(j)) \times (n_{11}(j) + n_{10}(j))}} \quad (1)$$

where $n_{pq}(j)$ is the number of runs in which the component j has been touched during execution ($p = 1$) or not touched during execution ($p = 0$), and where the runs failed ($q = 1$) or passed ($q = 0$). For instance, $n_{11}(j)$ counts the number of times component j has been involved in failed executions, whereas $n_{10}(j)$ counts the number of times component j has been involved in passed executions. Formally, $n_{pq}(j)$ is defined as

$$n_{pq}(j) = |\{i \mid a_{ij} = p \wedge e_i = q\}| \quad (2)$$

SFL can be used with program spectra of several different granularities. However, it is most commonly used ad the line of code (LOC) level and at the basic block level. Using coarser granularities would be difficult for programmers to investigate if a given fault hypothesis generated by SFL was, in fact, faulty. Throughout this work, we will be using a LOC level as the instrumentation granularity for the fault localization diagnosis report.

2.3. Motivational example

Suppose a program responsible for controlling a television set is being debugged. Consider that such program has three main high-level modules:

1. Audio and video processing.
2. Teletext decoding and navigation.