Journal of Computer and System Sciences 80 (2014) 953–972

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Software control flow error detection and correlation with system performance deviation ☆

Atef Shalan *, Mohammad Zulkernine

School of Computing, Queen's University, Kingston, Ontario, K7L 3N6, Canada

ARTICLE INFO

Article history:
Received 23 September 2012
Received in revised form 15 March 2013
Accepted 27 August 2013
Available online 11 February 2014

Keywords:
Control flow error
Error detection
Runtime monitoring
Component-based software
Performance analysis
Connection Dependence Graph (CDG)
Error state parameters
Regression analysis

ABSTRACT

Detecting runtime errors helps avoid the cost of failures and enables systems to perform corrective actions prior to failure occurrences. Control flow errors are major impairments of system dependability during component interactions. Existing control flow monitors are susceptible to false negatives due to possible inaccuracies of the underlying control flow representations. Moreover, avoiding performance overhead and program modifications are major challenges in these monitoring techniques. In this paper, we construct a connection-based signature approach for detecting errors among component interactions. We analyze the monitored system performance and examine the relationship of the captured error state parameters with the system performance deviation. Using the PostgreSQL 8.4.4 open-source database system with randomly injected errors, the experimental evaluation results show a decrease in false negatives using our approach relative to the existing techniques. It also demonstrates a significant ability of identifying the responsible components and error state patterns for system performance deviation.

1. Introduction

With the humongous expansion of the software roles in today's life, ensuring software quality is becoming more essential practice. Practical analysis of failure manifestation in a software system involves tracking the underlying causes (errors) and sources (faults). Detecting runtime errors is important to assure system resilience. It helps avoid the failure costs and enables systems to perform corrective actions prior to failure occurrences. Control flow errors are major impairments of system performance and other quality attributes during component[1] interactions. Operational environments of software systems are the main source of faults that can cause these errors [2]. Software faults with respect to operating systems and hardware faults with respect to computational devices are the main causes of control flow errors in software systems [3]. Recent industry trends of microprocessors aim to manufacture small size products with low supply voltage and high frequency. These trends are responsible for more susceptible transient hardware errors and control flow errors in software systems operating on these microprocessors [4].

☆ The earlier work of this research appeared in [1].

* Corresponding author.
E-mail addresses: atef@cs.queensu.ca (A. Shalan), mzulker@cs.queensu.ca (M. Zulkernine).

[1] *A software component* is a source code and data container that has identified boundaries and specified interfaces with other containers that together construct a software system.

http://dx.doi.org/10.1016/j.jcss.2014.02.004
0022-0000/

Error detection techniques mainly use signature-based approaches to monitor the control transitions among code instructions, system components, or basic blocks and detect their anomalies. These techniques detect errors by the use of watchdog, redundancy, assertion, or monitoring approaches. These approaches vary among each other by trading-off among four main goals: avoiding performance overhead, avoiding program modifications, avoiding the use of extra hardware, and increasing error coverage by detecting more errors of different types or among different scopes of control transitions. Signature-based control flow monitors can save more performance overhead than the other techniques and at the same time they do not require any additional hardware or software redundancy. Moreover, based on the signature structure, these techniques can also increase the error coverage. However, in these techniques, a unique signature is assigned to and injected into each block. Then, errors are detected by validating runtime signatures of the blocks against the program architecture. Some signature-based monitors use finite state automata [5] or branch traces [6] to represent a software architecture. However, the majority of these techniques rely on Block-based CFGs (BCFGs) for this purpose [7,4,8].

A BCFG is a CFG structure where the graph nodes are basic blocks[2] and edges are the control transitions among these blocks. However, the BCFG structure is remarkably complex and it lacks accuracy by allowing some illegal control branches in its representations. Due to the possible inaccuracies of control flow representations, the error detection techniques may encounter improper information about the internal system states and other consequent flaws in their results. Therefore, in these techniques, validating the control flow transitions may not be feasible for large software systems and may be susceptible to false negatives due to the inclusion of some illegal control branches in the BCFG. Moreover, maintaining simple control flow representation is important to practically implement these analysis techniques in large software systems. In addition, the existing techniques do not differentiate between connection invocations and control transitions based on the language constructs. As a result, these techniques are hard to use for backtracking program execution traces and validating their successful terminations. Moreover, the basic block partitioning in these techniques decreases the precision of error localization at the statement level inside these blocks.

System performance is an important quality attribute and of high concern with respect to several software systems (*e.g.*, production, control, and multimedia). Performance analysis techniques (*e.g.*, load test) can be time consuming. Expensive efforts are required to set up the test environment, prepare the test data, execute the test, and analyze the results [11–13]. In general, these techniques utilize selective performance log variables or execution state variables to reason about system performance. Control flow monitors can perceive details about system execution states that can be used to reason about system performance and other quality attributes. The Connection Dependence Graph (CDG) [14] can also enrich the error detection with useful error state parameters that correlate to system performance. Utilizing the distribution of errors and error state parameters among components in the performance analysis can help identify responsible components and error state patterns (specific combination of the error state parameter values) for performance deviations among system runs.

By extending our previous work [1], this paper utilizes a simple and accurate CFG representation based on the CDG to construct a connection-based signature approach for control flow error detection. We also correlate the detected error parameters to system performance deviation. We first describe our connection-based control flow signature structure in which, we partition the program components into code blocks. Each partition (or block) is associated with a CDG to represent the control structure of its code content. Next, we provide the control flow monitor structure and error checking algorithm based on these CDGs. We use the Multiple Correlation Coefficient (MCC) to examine the relationships of the captured error state parameters with the system performance deviation and identify the responsible components and error state patterns. The error detection approach is evaluated using PostgreSQL open-source database [15] and results of detecting control flow errors in different software versions with variable numbers of randomly altered code statements are rendered. The results show a decrease in false negatives using our approach relative to the existing signature-based techniques that use the BCFG representation. It also demonstrates the ability to identify highly correlated components and error state patterns to the system performance deviation with high "significance test"[3] values.

Fig. 1 presents an overview of the proposed technique in two steps: preparation and monitoring. In the preparation step, we instrument the software system in order to enable the generation of runtime signatures during its execution. A *connection-based control flow signature* is a runtime state describing the currently invoked connection during a system runtime. A program is partitioned into Connection Implementation Blocks (CIBs) (a code block). Each CIB is represented using a CDG. These CDGs are used to compute the runtime signatures. Signatures are injected in the program code to allow exporting the actual runtime transitions in a form of runtime signatures. In the monitoring step, we capture these runtime signatures and use them to verify the system control flow transitions. The monitor validates these runtime signatures by using the valid control flow transitions specified by the program CDGs and finally, an error report is generated. Using log data of a number of system runs, the errors and their parameters are correlated to the system performance at the component level and at the error state pattern level. The results of this process describe the correlation of each component and error state pattern with the system performance deviation.

The main contributions of this work include a control flow monitor using connection-based signatures. Our signature-based control flow monitor does not require any additional hardware or software redundancy. The underlying control flow

[2] A *basic block* is a maximal set of ordered code statements in which the execution starts at the first statement and ends at the last one. With the exception of the last statement, a block cannot include any control flow branching instruction [9,10].

[3] The significance test (F-test) checks if the resulting correlation coefficients have a Fisher distribution under the null hypothesis.