WuKong: Effective Diagnosis of Bugs at Large System Scales

Bowen Zhou Milind Kulkarni Saurabh Bagchi
Purdue University
{bzhou, milind, sbagchi}@purdue.edu

Abstract

A key challenge in developing large scale applications (both in system size and in input size) is finding bugs that are latent at the small scales of testing, only manifesting when a program is deployed at large scales. Traditional statistical techniques fail because no error-free run is available at deployment scales for training purposes. Prior work used *scaling models* to detect anomalous behavior at large scales without being trained on correct behavior at that scale. However, that work cannot localize bugs automatically. In this paper, we extend that work in three ways: (i) we develop an automatic diagnosis technique, based on *feature reconstruction*; (ii) we design a heuristic to effectively prune the feature space; and (iii) we validate our design through one fault-injection study, finding that our system can effectively localize bugs in a majority of cases.

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Software/Program Verification—Statistical Methods; D.2.5 [*Software Engineering*]: Testing and Debugging—Diagnostics

General Terms Reliability, Verification

Keywords Scale-dependent Bug, Program Behavior Prediction, Feature Reconstruction

1. Introduction

One of the key challenges of developing large-scale software, intended to run on many processors or with very large data sets, is detecting and diagnosing *scale-dependent* bugs. Most bugs manifest at both small and large scales, and as a result, can be identified and caught during the development process, when programmers are typically working with both small-scale systems and small-scale inputs. However, a particularly insidious class of bugs are those that predominantly arise at deployment scales. These bugs appear far less frequently, if at all, at small scales, and hence are often not caught during development, but only when a program is released into the wild and is deployed at large scales. As a result, these bugs are unlikely to be caught by regular software testing and debugging techniques.

Our previous work Vrisha [4] has attempted to address this particular type of bugs where existing debugging techniques fails. Vrisha is a statistical-model-based debugging tool that exploits *scale-determined* features to detect bugs in large-scale program runs even if the statistical model was only trained on small-scale behavior. Note that in this paper, we adopt the same convention as Vrisha, and use "scale" as a generic term to refer to the number of executing entities (threads, processes, peers, etc.) or input data size (or both). Unfortunately, while the detection of bugs is automatic, Vrisha is only able to identify that the scaling trend has been violated; it cannot determine *which* program feature violated the trend, *i.e.,* where in the program the bug manifested.

1.1 Our approach: WuKong

This paper presents WuKong[1], an *automatic, scalable approach to detecting and diagnosing bugs that manifest at large system scales.* WuKong is based on the same high level concepts as Vrisha, but provides three key contributions over the previous work:

Automatic bug localization WuKong, in contrast to Vrisha, provides an *automatic* diagnosis technique.The key insight driving our localization technique is that a regression model built across multiple training scales can be used to *predict* the expected bug-free behavior at larger scales. By building per-feature scaling models, WuKong is able to infer what the value of each feature *would have been* were a run bug-free, and hence identify which feature(s) deviate most from expected behavior. With carefully chosen program features that can be linked to particular regions of code (WuKong uses calling contexts rooted at conditional statements, as described in Section 2), identifying the most deviant features can pinpoint which lines of code result in particularly unexpected behavior, providing a roadmap the programmer can use in tracking down the bug.

Feature pruning Not all program behaviors are well-correlated with scale, and hence cannot be predicted by scaling models. Examples of such behaviors include truly random conditionals (*e.g.*, `if (x < rand())`) or, more commonly, data-dependent behaviors (where the *values* of the input data, rather than the *size* of that data determine behavior). When *diagnosing* bugs, the existence of such hard-to-model features can dramatically reduce the effectiveness of localization: a feature whose behavior seems aberrant may be truly buggy, or may represent a modeling failure.

To address this shortcoming, we introduce a cross-validation-based *feature pruning* technique. This mechanism provides a "knob" that can effectively prune features that are hard to model accurately from the feature set, allowing programmers to trade off reduced detectability for improved localization accuracy.

2. Bug Localization Via Feature Reconstruction

The fundamental approach of WuKong is to build a statistical model of program behavior that incorporates scale. Essentially, we would like a model that infers the relationship between *control features*, *i.e.,* scale attributes (*e.g.*, number of processes, or input

Copyright is held by the author/owner(s).
PPoPP'13, February 23–27, 2013, Shenzhen, China.
ACM 978-1-4503-1922/13/02.

[1] WuKong, the Monkey King, is the main character in the epic Chinese novel *Journey to the West*, and possesses the ability to recognize evil in any form.

size) and *observational features*, *i.e.,* program behavior (*e.g.*, the number of iterations taken by a particular loop).

WUKONG models application behavior with a collection of base models, each of which characterizes a single observational feature. The base model is a log-transformed regression where all control features are included in a log-transformed regression model to predict each observational feature. The log transformation accounts for higher-order relationships between behavior and scale (consider the many algorithms that are $O(n^2)$).

$$\log(Y) = \beta_0 + \sum_{i=1}^{N} \beta_i \log(X_i) \quad (1)$$

Following the procedure of Barnes *et al.* [2], the regression model in Equation 1 allows us to readily compute the relative prediction error, as in Equation 2.

$$E_i = |e^{\log(Y_i') - \log(Y_i)} - 1| \quad (2)$$

To provide a "roadmap" for developers to follow when tracking down the bug, WUKONG ranks all observational features by relative error.

3. Feature Pruning

To eliminate bad features, *i.e.,* features that cannot be modeled accurately, WUKONG employs cross validation. Cross validation uses a portion of the training data to test models built using the remainder of the training data. More specifically, WUKONG employs k-fold cross validation. It splits the original training data by row into k folds, treats each one of the k folds in turn as the test data and the remaining $k-1$ folds as the training data, then trains and evaluates a model using each of the k sets of data. In this way, cross validation is able to estimate the mean reconstruction error for each feature using the original training data.

If a particular feature cannot be modeled well during cross validation, WUKONG assumes that the feature is unpredictable and will filter it out from the roadmaps generated during the localization phase. Those features with an error higher than a preset threshold h are removed from the input. The threshold is configurable by users and usually set to be a positive number less than 100%, as it is trivial to achieve 100% relative error by using 0 as the prediction.

4. Evaluation

We conducted a large scale fault injection experiment on a benchmark applications from the Sequoia benchmark suite [1]: AMG2006. AMG2006 is a parallel algebraic multigrid solver for linear systems arising from problems on unstructured grids.We began by building a model for each observational feature of AMG2006, using as training runs program executions ranging from 8 to 128 nodes. The control features were the X, Y, Z dimension parameters of the 3D process topology, and the observational features were the number of times each branch is taken at a unique calling context, resulting in 3 control features and 1633 observational features. Then we prune all observational features whose reconstruction error during cross validation is greater than 0.5. We are left with 1466 features for which WUKONG builds the final models. We use Pin [3] to collect the observational features from our training runs and testing runs.

We ran 100 instances of the 1024-node execution, each time injecting a fault into rank 0 process by choosing (randomly) one conditional test (selected from the set of all observational features) to "flip" throughout the rest of the execution. Of the 100 injected runs, 47 resulted in non-crashing bugs.

We found that in 70.2% of cases, the source of the bug (the conditional that was flipped) was among the top 10 aberrant features. Furthermore, 46.8% of the time the faulty feature was the first feature identified by WUKONG. We then examined the sensitivity of WUKONG's localization to the filtering threshold used by the feature pruning algorithm described in Section 3. We used four different pruning thresholds: none, 1.0, 0.5 and 0.2. Figure 1 shows the percentage of buggy runs where the faulty feature appears in the top x features for different x. We see that performing a small amount of feature pruning can dramatically improve the quality of WUKONG's roadmap: at a threshold of 0.5, over 66% of the faulty features appear in the top 5 features suggested by WUKONG.

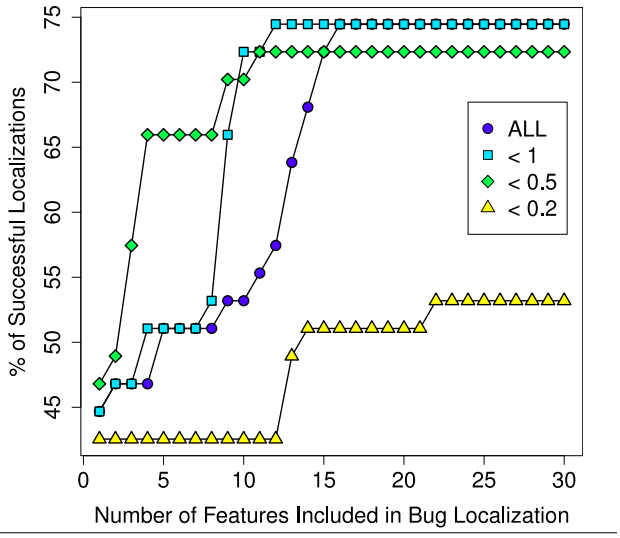

Figure 1. Comparison of different feature pruning thresholds. "ALL" means no pruning. "$< \eta$" means only keeping features with reconstruction error less than η.

5. Conclusions

With the increasing scale at which programs are being deployed, both in terms of input size and system size, techniques to automatically detect and diagnose bugs in large-scale programs are becoming increasingly important, especially if bugs are scale-dependent. To address these problem, we developed WUKONG, which leverages novel statistical modeling and feature reconstruction techniques to automatically diagnose bugs in large scale systems, even when trained only on data from small-scale runs. This approach is well-suited to modern development practices, where developers may only have access to small scales, and bugs may manifest only rarely at large scales. With a fault injection study, we showed that WUKONG is able to automatically, scalably and effectively diagnose bugs.

References

[1] ASC Sequoia Benchmark Codes. `https://asc.llnl.gov/sequoia/benchmarks/`.

[2] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. de Supinski, and M. Schulz. A regression-based approach to scalability prediction. In *Proceedings of the 22nd annual international conference on Supercomputing*, pages 368–377, 2008.

[3] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, pages 190–200, 2005.

[4] B. Zhou, M. Kulkarni, and S. Bagchi. Vrisha: using scaling properties of parallel programs for bug detection and localization. In *Proceedings of the 20th ACM international symposium on High performance distributed computing*, pages 85–96, 2011.