Parikshan: A Testing Harness for In-Vivo Sandbox Testing

Nipun Arora
NEC Research Labs
Princeton, NJ, USA
nipun@nec-labs.com

Franjo Ivancic
Google
New York, NY, USA
ivancic@google.com

Gail Kaiser
Columbia University
New York, NY, USA
kaiser@cs.columbia.edu

ABSTRACT

One of the biggest problems faced by developers testing large scale systems is replicating the deployed environment to figure out errors. In recent years there has been a lot of work in record-and-replay systems which captures traces from live production systems, and replays them. However, most such record-replay systems have a high recording overhead and are still not practical to be used in production environments without paying a penalty in terms of overhead.

In this work we present a testing harness for production systems which allows the capabilities of running test-cases in a sandbox environment in the wild at any point in the execution of an integrated application. The paper levarages, User-Space Containers(OpenVZ/LXCs) to launch test instances in a container cloned and migrated from a running instances of an application. The LXC shell provides a sandbox environment, for safe execution of test-cases provided by the users without disturbing the execution environment. Test cases are initiated using user-defined probe points which launch test-cases using the execution context of the probe point. Our sandboxes provide a seperate namespace for the processes executing the test cases, replicate and copy inputs to the parent application, safetly discard all outputs, and manage the file system such that existing and newly created file descriptors are safetly managed.

We believe our tool provides a mechanism for practical testing of large scale multi-tier and cloud applications. In our evaluation provide a number of use-cases to show the utility of our tool.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

ICSE 2015 Firenze, Italy

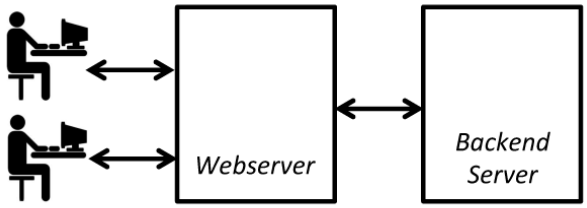

Figure 1: Workflow

Keywords

ACM proceedings, LaTeX, text tagging

1. INTRODUCTION

As application software grows and gets more complicated, testing large scale applications has become increasingly important. However, it is often impossible to recreate realistic workloads in an offline development environment for large scale multi-tier or cloud based applications. Testing in the development environment can be (1). Un-realistic because it may not be possible to faithfully reconstruct the production environment, (2). The test-cases generated may be incomplete, (3) It is infeasible to test all possible configurations given time and cost constraints of releasing the software to the field.

One of the proposed mechanisms of addressing this problem is to "perpetually test" the application in the field after it has been deployed. This is important since testing in a production system enables us to capture previously "unreachable" system states, which are possible only for a long running production configuration.

On the other hand, there has been an equally impressive increase in the scale of computing resources, and distributed scalability of infrastructure. Web based applications are often hosted in cloud environments, this allows for easily scaling up the hardware resources. Leveraging this abundance of resource we present a testing harness which allows the user to dynamically insert test cases in a production environment(we call this in-vivo testing).

In this paper we introduce, `Parikshan`[1], which allows capturing the context of application, and allows for a test-case to be run without effecting the sanctity/and performance of

[1]Parikshan is the sanskrit word for testing

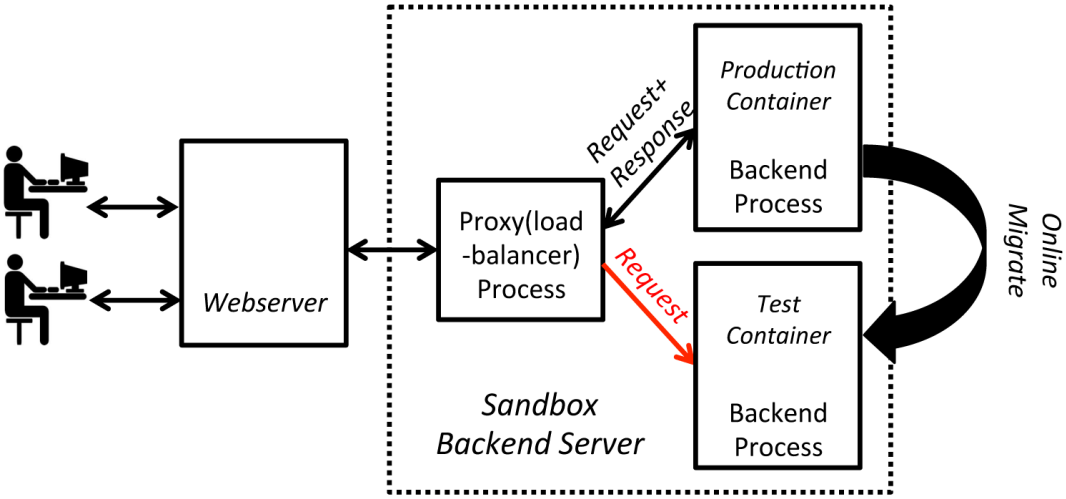

Figure 2: Workflow

the actual application. This is done by cloning a production server and creating two containers: a production container, and a testing container. We duplicate the incoming traffic to both the production container and the test container using a custom proxy, which ignores the responses from the test-container. The testing on the test-container is done on the fly using dynamic instrumentation, hence any set of test-cases can be turned on whenever required. The user can pre-define probe points for dynamically inserting test-cases (by default the entry and exit of each function is considered a probe point). Since the test is executed in a VM it acts like a sandbox which restricts it from causing any perturbation to the state of the parent process, or effecting the sanity of the responses to the production client. We synchronize the production and test containers using a variant of live migration without suspending the services of the production server. Frequent synchronization is necessary because the test container can potentially go out of sync with the production because of non-determinism or because a test-case changes the state of the container, and effects future problems.

The key contributions of this paper are:

- Our tool provides a sandbox environment to execute test cases in the production environment. This allows for a safe and secure test harness which does not effect the production state, and allows the application to proceed in it's execution.
- We allow for dynamic insertion of the test case, and safetly capturing the context of the application. Dynamically inserting test-cases is important to avoid relaunching binaries in the test-container with the required test-cases. Restarting binaries is not possible, because it would break active network connections, and destroy the state of the test container
- Language and Platform agnostic: One of the key advantages of our approach is that it is language and platform agnostic. Since the underlying mechanism takes advantage of containers as a platform to do the cloning, the language or interface does not matter as far as cloning is concerned. Of-course testing mechanims may differ depending upon different languages.

Traditional testing approaches break states and are unable to

1.1 Contributions

The impact of sandbox testing can be seen in several different ways

- **Sandbox Live Testing** One of the key motivations leading to *Parikshan* is to provide a harness to allow the user to test real, live implementations.
- **Fault Tolerance**
- **Verification**
- **Integration Testing**

2. MOTIVATION

The key intuition behind our approach is that

Q1: Is it important to sandbox test-cases?
Q2: Is recreating production environment difficult?
Q3: Is redundant computing available?
Q4: How would executing test-cases in a production server effect user-experience?

2.1 Q1: Is it persistent testing important?

2.2 Q2: Is recreating production environment difficult?

2.3 Q3: Can redundant computing be utilized for testing?

2.4 Q4: How would executing test-cases in a production server effect user-experience?

3. RELATED WORK

There have been several existing approaches that look into testing applications in the wild. The related work can be divided in several categories:

- **Perpetual Testing** We are inspired by the notion of perpetual testing[**?**] which advocate that software testing should be key part of the deployment phase and not just restricted to the development phase.
- **Record and Replay**
- **Alpha-Beta Testing**

4. DESIGN AND IMPLEMENTATION

In this section we begin with a system overview of `Parikshan`. We then explain how it inserts test cases, into the test harness, and finally we explain how a user can use the `Parikshan` api to insert test cases in the test harness.

4.1 System Overview

There are two key components of `Parikshan` : (1) proxy network request duplicator, (2) container clone manager

5. TRIGGERING AND INSERTING TEST CASES

6. NETWORK ISSUES

There are several problems that can effect the execution of sandbox testing.

- **Stateful Connections**
- **Time Lag**

7. IMPLEMENTATION

8. CONCLUSION

9. REFERENCES