Centralized and distributed testing

Simon Kers
skers@kth.se
School of Technology and Health
KTH Royal Institute of Technology
February 2013

David Wikmans
wikmans@kth.se
School of Technology and Health
KTH Royal Institute of Technology
February 2013

Abstract

This paper reviews some select methodologies and techniques used for testing the behaviour and reliability of software applications, and how testing differs between distributed and centralized systems. Application operability, uptime and availability are concerns for large systems and services but testing these areas can be tricky. In the software industry testing is not only necessary but common practice and with the emergance of distributed systems and the internet, testing faces new challanges. By reducing dependencies[1] of the distributed parts of the system and purposely subjecting the network to high load, the distributed system can be hardened and better protected against bugs.

1 Introduction

Software testing has been a part of software-development as long as there have been computers. To assess the quality of computer software testing is very important. Testing typically takes upwards 40% – 50% of the development time, and even more for systems that require higher levels of reliability. [3]

Software testing is divided into two schools of thought. Static testing which is to analyze the source code by inspections, peer reviews and walkthroughs. Dynamic testing where the application is executed and the behaviour of the program, or rather, the systems response is tested.

Testing can be done after the development process by an independent group of testers, which is the traditional or *Waterfall* approach. Writing tests before the development process begins is known as *Test Driven Development*.

2 Evaluation

2.1 Centralized systems

Unit tests are tests for select code segments. Normally these tests are on a functional level. In object-oriented languages unit tests means tests for classes.

The *box approach* is where you put something in the box and then return a result. A black-box is where you do not care about the inner workings of the box (implementation) only the return value. A white or clear-box is when you test the interal structure of the object. A gray-box test is a combination of both.

Another is the *bottom-up top-down approach*, where bottom-up means writing tests for each component and top-down only the functions of the high level objects.

[1]Low coupling

Mentioned techniques are not exclusive to centralized systems.

2.2 Distributed systems

2.2.1 States

Distributed applications, effectively increases the number of possible states the system can have. One meassure against this is employing designs which reduce dependencies between nodes, that the state of one node affects the state of another as little as possible.

Distributed systems rely on communication between nodes, this could lead to an increased number of states the system as a whole can take, increasing the possibility of bugs.[1]

2.2.2 Chaos monkey

When testing distributed systems a new class of software bugs emerge which conventional testing procedures may not catch. One inconvenient property of such bugs are infrequent occurrences and are difficult to reproduce, especially with code not in production[2].

One counterintuitive approach would be Netflix *Chaos Monkey* implementation, a disaster testing system, which operates on Amazons cloud computing platform. It randomly causes failures of the virtual machine instances, know as *Auto Scaling Groups*.[2]

This puts the distributed systems resiliency at test and enforces security measure against netsplits, erratic behavior and failure caused by high latency and bugs of such distributed nature. It enables software developers to discover rarely occuring bugs and resolve them before the code goes live.

3 Conclusions

A whole set of new problems arises when developing distributed systems, the proneness to error that occurs in centralized software are still there in distributed systems. One way of decreasing the need for distributed software testing is to reduce coupling between the different nodes. This isolates the test cases to a local level and avoids some aspects of distributed testing procedures and simplifies code by treating the nodes as resources.

[2]Code that is currently in development

External occurences such as power outages and network failure are hard to predict, therefore new ways of simulating failure and load has gained ground. Distributed system often grow large and produce detailed logs of programmatic operation and network status, analyzing large quantities of log data becomes more important as these systems grow and becomes more prevalent.

Though tests serve an important purpose, they are only as good as their implementation and can not guarantee a faultless system. Testing distributed aspects will become an important part of developing large scale systems.

4 References

[1] *Erlang '11: Proceedings of the 10th ACM SIGPLAN workshop on Erlang*, New York, NY, USA, 2011. ACM. 553112.

[2] Cory Bennett and Ariel Tseitlin. Chaos Monkey Released Into The Wild. `http://techblog.netflix.com/2012/07/chaos-monkey-released-into-wild.html`, 2008. [Online; accessed 19-July-2008].

[3] Lu Luo. Software testing techniques.