3

An Agent Based Framework for the Distribution of Test Cases

[redacted]

Abstract

Software correctness is rapidly becoming one of the most important an researched aspects of computer science. In order to ensure correctness a large number of test cases must be run to have confidence that we have adequately tested the software. Running these tests on a single computer may be prohibitive because of the time involved in execution. In order to decrease the execution time of the test cases we will investigate a framework where mobile agents will be used to carry test cases to the most appropriate host computer where the test cases will be executed. An agent based framework will be capable of determining if a host computer is appropriate for a specific test instead of placing that burden on the host computer. In order to test the effectiveness of an agent based framework as opposed to a tuple space implementation we will run the same set of test cases on the same set of host computers with each implementation and compare the total time needed to execute the test cases.

1 Introduction

Regression Testing provides a way to practically and cost effectively test a changed program for errors. Regression testing is defined by Haritaoglu as "the process of validating modified parts of the software and ensuring that no new errors are introduced into previously tested code" [1]. Because regression testing is conducted on a piece of code that was tested during development it is possible to use the same test cases that were run on the original code to determine if there have been any changes in the way the software executes. One important consideration about regression testing is the amount of time that it takes to execute a suite, or collection, of test cases. The complexity of many programs today requires a large number of test cases to ensure, with a high degree of confidence, that the software executes correctly from version to version. One way to reduce the time needed to execute a suite of test cases is to distribute the execution across a network of computers.

2 Considerations for Distributed Test Execution

There are five considerations that must be taken into account when designing a framework to distribute the execution of test cases across a network of computers [2].

- **Transparent and Automatic Distribution**: The distribution of the n test cases across the m machines should be as transparent and automatic as possible.
- **Test Case Contamination Avoidance** When the n test cases are distributed across the m machines, it must be done in a manner that attempts to avoid test case contamination.
- **Test Load Distribution** When the n test cases are distributed across the m machines, it must be done in a manner that appropriately distributes that load during the test execution process.
- **Test Suite Integrity** The process of distributing the n test cases across the m machine should not affect the correctness of test case results or impede the proper execution of the test suite.
- **Test Execution Control** It must be possible to control the execution of the m test cases and view the results of each test from a centralized testing environment.

3 Joshua

Joshua is a regression test suite developed by Greg Kapfhammer [2]. Joshua is an extension of the `JUnit` testing framework that uses Jini and JavaSpaces to effect the distribution of the test cases. A high level view of the Joshua architecture can be found in Figure 1.

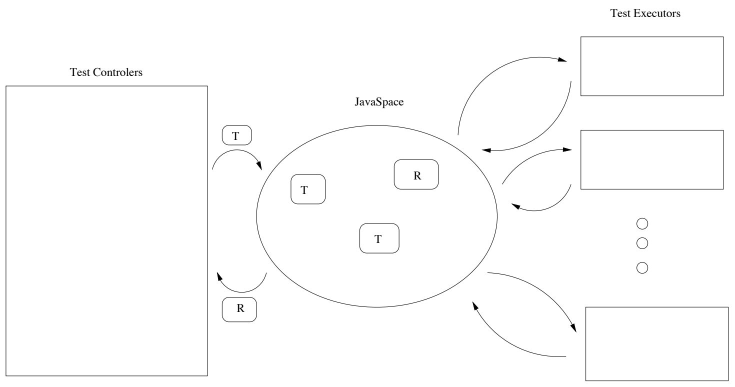

Figure 1: Architecture of Joshua

- **Test Controller** The test controller is responsible for two things. Interacting with `JUnit` to create and format the test cases so that they can be written to the test space and collecting the results after each test case has been executed.

- **Test Space** The test space provides a repository where both the test cases and the results of executing the test cases can be stored temporarily until the test controller or test executors can process them.

- **Test Executors** The test executors are responsible for consuming the test case from the test space, executing the test cases, and then writing the results of executing the test cases back into the test space for the controller to collect.

Joshua fulfills all five considerations for the effective distribution and execution of regression test cases. However there are possibilities for improvement on the way that it deals with Consideration three. Joshua uses load sharing when dividing the test cases among the test executors. Load sharing simply attempts to initiate the execution of a test suite on a lightly loaded test machine [2]. While this is acceptable it does not ensure that a test case is executed on the most appropriate machine. When a test executor has completed execution of a test case it consumes a random test case from the test space and begins executing the new test case. A different approach would be to intelligently schedule the execution of the test cases so that they are executed on the machine that is the most appropriate. A distributed execution framework based on mobile agents will be able to use load balancing instead of load sharing and achieve better execution of the test cases.

4 Agent Based Architecture

A diagram of the architecture of the mobile agent framework can be seen in Figure 2.

- **Test Controller** The test controller is responsible for three things. Interacting with `JUnit` to create and format the test cases, creating

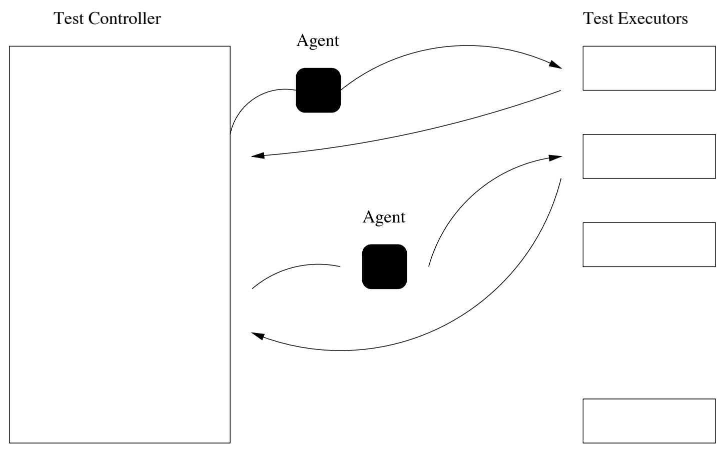

Figure 2: Architecture of a Mobile Agent Framework

the mobile agents, passing them the test cases and telling them the appropriate environment to search for, and collecting the results after each test case has been executed.

- **Test Executors** The test executors are responsible for accepting the mobile agents and allowing them to execute the test cases that they carry.

- **Agents** The agents are created by the test controller and migrate across the network of test executors until they find an computer that has the appropriate environment for them to execute the test cases. The agents then execute the test cases on the test executors and then carry the test results back to the test controller for compilation.

5 Addressing Distributed Considerations

A framework based on mobile agents would still be able to implement the five considerations set forth in Section 2. The distribution of the test cases would be even more transparent to the users of the test executors because all that is required of the test executors is a running thread that allows the agent to execute the test case. The report of the results and the arrival of the test case is now taken over by the agent. The considerations dealing with test case contamination and test suite integrity can be handled by the same means that Joshua uses to address these concern and the test controller will still create a central location where the execution of the test cases and the collection of the results can be viewed.

6 Additional Benefits

In addition to the decreased execution time for a test suite an agent based distributed test framework potentially has several other benefits that will increase the cost-effectiveness of distributed regression testing. If it is possible to create mobile agents that support strong mobility it will be possible to implement check-pointing with our framework. Check-pointing when then agent would record its current state. This has two main benefits in the context of a distributed agent framework. If a test executor would crash it would be possible for the agent to restart itself after the test executor was brought back up. This would save large amounts of time since the test would not need to be rerun from the start. The second benefit that this would create has to deal with the ability of other users to use the test executors while test cases are being run. If a user needs to use a workstation where a test is bring run it is possible for the mobile agent to simply move its execution to another machine on the network and save all of the computational work that has been done on the test.

7 Conclusion

A distributed testing framework based on mobile agents will allow the regression testing process to be speed up for large software systems. It will use load balancing to intelligently distribute the test cases to the test executor to minimize the over all time that is need to execute a suite of test cases.

References

[1] Ismail Haritaoglu. Hierarchical regression testing with the omega test. Available online at `http://www.umiacs.umd.edu/~hismail/Omega/projereport.html`.

[2] Gregory M. Kapfhammer. Automatically and transparently distributing the execution of regression test suites. *18th International Conference and Exposition on Testing Computer Software*, 2001. Avail-

able online at `http://alden29.allegheny.edu/~gkapfham/research/publish/index.html`.

[3] The Free On line Dictionary of Computing. Definition of an agent. Available oneline at `http://work.ucsd.edu:5141/cgi-bin/http_webster?agent`.