Faculty of Computer Science
March 4, 2015

TN3706 - Bachelor Seminar

Performance Regression Testing
A developer's perspective

Bachelor Seminar Group

M.A. Hoppenbrouwer	4243889
M.J. Otte	4222695
R.S. Sluis	4088816
R. Vink	4233867

M. Loog	Professor
C. Bezemer	Supervisor

Preface

Summary

Contents

List of Figures

List of Tables

1. Detecting and analyzing I/O performance regressions

One of the reason for performance regression testing is to help developers improve their software. Developers can detect an issue in their software and resolve it. For them to be able to do this they have to be able to detect and analyze the data received from the performance regression test. In the following case we will show you the use of I/O performance regression testing. The full report on the study can be read in "Detecting and Analyzing I/O Performance Regressions".
The goal of the study the I/O performance regression testing is used to analyze what effect the changes in source code have on the performance of a system. The performance counter that is used in the study is the amount of I/O writes that are being done. These changes are achieved by monitoring and comparing the execution of two versions of an application. These changes are then analyzed and used to optimalize the application.

creating the profile

To report on the detected data it has to be profiled. During the execution of the application the write actions are logged. The log consist of the number of written bytes, the name of the function and location to which file is being written. To take variation into consideration, the test is executed multiple times. The variation can be caused due to the difference in data content or memory usage. Running the test many times to get a accurate profile also has a tradeoff. Running the test many times can be impractible due to the execution time. Therefore there has to be a compromise between the accuracy and execution time.
The profile is generated by calculating the average number of bytes written per call and also define the lowest and highest values of the average number of bytes written per call. The lowest and highest values can be used to create a window. This window will be the accepted performance range for the current revision of the function. By creating the window, the variation is also being taken into consideration.

analyzing the profile

Now that a profile is created it has to be compared with profiles from different revisions of the application. Comparing these profiles by hand can be a tiring task that is susceptible to mistakes. In order to help developers with this tiring task, the task can be automated. The automation of the comparison is done with a method based on the spectrum-based fault localization. Because the spectrum-based fault localization resembles the human fault detection process, it makes the result easy to interpret.
After making the profile for two revisions (r_1 and r_2) the two will be compared. During the creation of the profile of r_1 a window of the accepted range was made for all functions with write operations. If the value of r_2 is within the window of r_1 it will be saved as a 1, otherwise it will be saved as a 0. By doing this a binary vector is created. Every row in the binary vector represents a function in the application. If a function was not in r_1, but is in r_2, the profile can't be compared. The comparison will be saved for the next revision. The reason a function isn't in r_1 and is in r_2 can be due to it being a new functionality added in the new revision or the function wasn't called due to a bug.

To analyze the performance of the new revision, the binary vector is compared to a 'ideal' vector. The 'ideal' vector consist of only 1's. For the comparison a similarity coefficient is used. If the similarity coefficient is close to 1, the change in performance is low. The similarity coefficient that is being used is the Ochiai similarity coefficient (OSC), which compares two vectors(v_1 and v_2).

"insert math formula"

In this formula a is the number of positions in which both vectors have the value 1. b is the number of positions with value 1 for v_1 and value 0 for v_2. c is the number of positions with value 0 for v_1 and value 1 for v_2.
Even though the similarity coefficient can detect a change in performance effected by an update. It doesn't show how the change effected the performance. To see how the new revision effectes the performance, the average number of bytes outside of the window. This is added to the report as the $Impact$. The $TotalImpact$ is also added to the report, it consist of the $Impact$* the average number of calls. With these new parameters we will be able to see if the performance increased or decreased and by how much. To better understand the ranking, the range, $RangeDiff$, is also added. When the value 500 bytes outside the window, there is a difference if the range was 50 bytes or 50 kilobytes.

When using the method described in "Detecting and Analyzing I/O Performance Regressions", a report will be created. The report consist of the similarity coefficient, number of calls, $Impact$, $TotalImpact$ and $RangeDiff$. With these parameters the developer will be able detect the change in the performance. By using the report the developer will be able to see if the change is positive or negative and how big the effect is. With this knowledge the developers can see what needs to be looked over and what has priority.

Bibliography