Data visualization - Google shared dataset of test suite results

Savyon Fishelovich, Nimrod Busany

Tel Aviv University, Tel Aviv, Israel

{nimrodbusany@post.tau.ac.il,savyonfi@mail.tau.ac.il}

September 16, 2014

1 Introduction

Google recently published the "The Google Dataset of Testing Results" [3]. A dataset containing a summary of 3.5 million tests. The tests include thousands of versions of different products tested by thousands of test suites of different sizes written in different languages. The format of the dataset is a simple comma-separated fields (csv) of 3.5 records.

We decided to focus in this dataset for several reasons:

- The following dataset contains a large sample of test suite execution results from a fast, large scale, and continuous industrial testing infrastructure. This dataset is of interest for the Software Engineering research community, and more specifically The Software Testing and Analysis community, which thrives for well documented and clean real-world data.
- This dataset is unique in size, over 1.5 Gigabytes of data. Therefore, analyzing such amount of data in R poses a computational challenge. Further, creating an interactive tool for a large dataset, should address the long computation time which R may requires.
- The file contains records of ten attributes of different types (integer, continuous, factorial and "free" strings). Analyzing this type of complex data is challenging by itself, and can lead to interesting observation about which tests tend to fail.

1.1 About the data

A schema of the dataset and explanation of the different attributes can be found here [Data Fields - press to view]

1.2 About the visualization in this work

Along the project, we followed Cleveland & McGill work[2]. The authors set a hierarchy of elementary perceptual tasks, that gives superiority to some graphs over the others. For example, a divided bar chart is considered better than a pie chart. As bar chart requires primarily the judging of position along a common scale, while pie chart requires primarily perception of angles [2] (p. 533), which is shown to be a harder visualization task. Furthermore, we applied the principles presented by Talbot [4] and by Cleveland [1]. We modified the shape parameter wisely with fig.hight and fig.weight arguments constructed in knitr. Likewise, we applied the diagrams, theories and tools that were introduced in the course. As such are smoothing methods, vase-plot, mosaic-plot and knitr.

2 Analyzing Google shared dataset of test suite results

To get a better understanding of the dataset at hand, we provide two solutions. The first is Rediant4Google, an online free tool tailored for the analysis of the Google shared dataset. Rediant4Google is designed to assist researchers and other fellow professionals to investigate the data independently. Rediant4Google is an open-source project available through github. The second solution is a thorough investigation of the data using R. This investigation allowed us to gain interesting insights about Google's dataset, such as "which languages is most effective in finding bugs?" We believe that these insights may be generalized to other test suites as well.

2.1 Rediant4Google

Rediant4Google [Rediant4Google - dropbox link - press to view] is based on Radiant [Radiant github link - press to view], an open-source shiny application [1] developed by Dr. Vincent R. Nijs [home page - press to view], from Rady School of Management. We modified the tool to fit the unique needs of the Google shared dataset of test suite results. The main difference between the tools is data management. Our key consideration was creating a responsive application. To achieve this, we broke the full dataset into chucks of data. These chunks can be easily merged into large chucks of data over which the user can perform the analysis. As for visualizations, the features we provide are based on the original Radiant tool and include simple 1/2 dimensional visualizations for different types of data. Finally, we provide a Sampling feature, which is a modified version of the original Rediant sample feature. This feature allows the user to easily sample records and then analyze them. A manual for Rediant4Google can be found in the appendix.

2.2 Executive summary of our analysis

Below we provide a brief summary of the analysis we conducted. For the full please analysis refer to [Full Data Analysis File - press to view].

2.2.1 Single dimension

A simple single dimension analysis shows that large majority of tests end successfully, while only a minor portion of fails. As for the languages used, most tests were written with CC, SH, JAVA,PY and WEB, with CC as the most common language. Three more languages were used in a small portion

[1] About Radiant license: The Radiant package is licensed under the AGPLv3. The help files are licensed under the creative commons attribution, non-commercial, share-alike license CC-NC-SA.

of tests (BORGCFG,GO,GWT). Further, around 60% of tests belong to stage "post" and the rest to the "pre" stage. Shards are used to parallelize the executed tests, most tests were run using a single shard, while small but non-negligible portion of tests are parallelized with 2 to 49 shards. The tests are divided into three size categories. Small tests, Medium tests and Large tests (which possibly represents unit tests, acceptance tests and integration test accordingly). Around half of the tests are small tests, while the other half is (roughly speaking) equally divided between the other tests. Finally, the execution times show that most tests take around 3 seconds with a large majority of test scattered around that value. Nevertheless, one can see the execution times range from a few milliseconds up to several minutes.

2.2.2 Two dimensions

Let us first describe the correlations we found between some of the key attributes and the status attribute. In our analysis we will focus on the proportion of failed tests and not absolute numbers. We then present more correlations between the other variables. First, we discovered that the most of the failed tests were large tests, a small portion medium and the least were small tests. We assume that small tests are mainly Unit tests, which test a single component of the software. These simple tests tend to have very high success rate. In contrast, medium and large tests, which probably correspond to acceptance and integration tests, are designed to test multiple components of the software or even several applications at once. As a result, these complex tests have lower success rate, which is consistent with in the diagram. We also looked for the relation between a failure and the language in which the test was written. We discovered the language which had highest failure frequency was BORGCFG. Java and Web also resulted in a rather high failure rate comparing to the other langauges. To explain the high failure rate of the BORGCFG language, we looked into the size distribution of test written with it. Not surprisingly, we discovered that most of the tests are Large or Medium test. We continued to investigate the shard number, and to our surprise we found that using shard to parallelize the tests led to lower failure rate. Finally, we observed that failed test tend to take around three times as much as successful tests (in terms of execution time). This two interesting insights tell us the execution time and shards number can hint which tests are more likely to fail.

We further searched for other correlations. Looking at the relation between language and size, we discovered that Web,GWT and BORGCFG languages are mainly used for medium and large tests. Java contained around equal number of small and large tests. Finally, SH, PY and CC contained large portion of small tests. Another observation is that as the test size increases, the median of shards number increases, and is more dispersed at high values. It is clear that small tests involve mainly zero shards. It is

also prominent that large tests tend to have multiple shards. Testing for the relation between the stage and the execution time shows that post stage has similar median execution time but larges variability, with around 25% increase in the third quintile. As for the relation between language and execution time, we discovered that the longest tests are GWT and Web tests, while the shortest ones are BORGCFG, GO and SH tests. Further, we observed that increasing the number of shards had led to an increase in the execution time.

2.2.3 Which languages is most effective in finding bugs?

We further investigated the relation between "Language"-"Test Size"-"Status" in order to understand which type of tests pose highest risk or alternatively is most effective in finding bugs. We will use both Radiant4Google and R to visualize the data in order to answer this question. Figure 2 was created using Radiant4Google. It shows the distribution of languages over the test size given the status. The status of a large majority of the tests is "PASSED", therefore it can be viewed a close approximation of the distribution over the full population (i.e. both "PASSED" and "FAILED"). As can be seen, Java dominates the distribution of the large failed tests, with an increase of over four times in its proportion over the full population. As for the medium category. BORGCFG is responsible to a rather large portion of the failed medium tests, even tough it only has a small fraction of the medium tests over the full population. WEB tests are the most common over both the failed and the full population in this category. Finally, C is responsible for most of the failed test over the small category, with proportion that is twice as its proportion over the full population.

Trying to gain even more insights, we decided to focused on the most common languages. We sampled an equal number of failed and passed tests (we did so, to increase the visibility, since the proportion of failed test is too small to gain visual insights). Figure 2 indicates several combinations which should raise a flag to software developers. The most prominent ones are: large Java tests and medium SH and Web tests. This groups contained higher failure rate than the other groups. It is important to note, that the actual failure rate does not represent the true failure rate over the full population, as we took an equal size of failed and passed. Nevertheless, the aim of this diagram, is only to compare between groups, and for that propose it is appropriate.

In summary, using R and Radiant4Google we were able to show that the most prominent faulty combinations are large java tests, medium SH, BORGCFG and Web tests and small C tests. Since the main goal of tests is finding bug, this observation should entourage software developers to use these effective combinations.

As a final remark, we hope that our analysis shows how combining Ra-

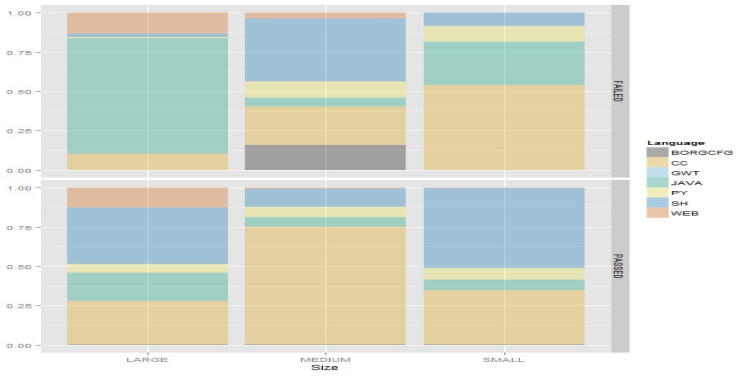


Figure 1: Radiant4Google: Status ∼ Size ∼ Language

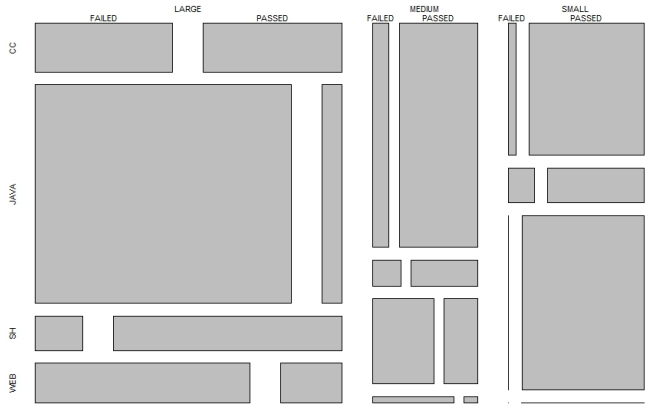


Figure 2: RSample: Status ∼ Size ∼ Language

diant4Google with R can help in the data analysis. While Radiant4Google is not as powereful as R, it allows one to promptly generate insightful plots which can focus the use of R and reduce precious coding time.

References

[1] W. S. Cleveland, M. E. McGill, and R. McGill. The shape parameter of a two-variable graph. *Journal of the American Statistical Association*, 83:289–300, 1988.

[2] William S. Cleveland and Robert McGill. An experiment in graphical perception. *International Journal of Man-Machine Studies*, 25(5):491 – 500, 533, 1986.

[3] Andrew Mclaughlin Sebastian Elbaum and John Penix. The google dataset of testing results.

[4] J. Talbot, J. Gerth, and P. Hanrahan. An empirical model of slope ratio comparisons. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2613–2620, Dec 2012.

Appendices

A User manual

In this section, we will walk through the features of Rediant4Google. Figure 3, shows Rediant4Google main page. It consists of two navigational panels. The main panel, lets the user scroll between feature:

- Data management - load and visualize data
- Sample - create a sample from data
- State - save application's state
- About - basic information about the Rediant4Google

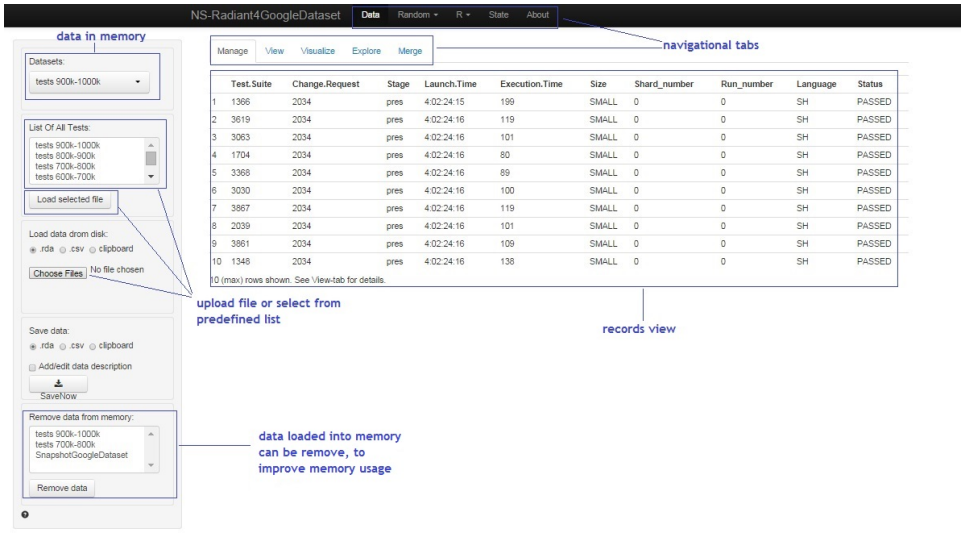


Figure 3: Main page

The second navigational panel allows navigation between each feature.

Figure 3 also shows the data management feature. The user can upload a file from disk (csv,dat) or from the clipboard. Further the shiny server hosting Rediant4Google includes the Google dataset spread over several data chunks. Each chunk contains a 100K of records and can be loaded into memory using the dedicate button. The user may also save the data into a file, or remove upload files from memory. Removing data from the memory is crucial when working with such large data, as it allows the user to improve memory usage (and not to exceed application memory capacity).

The view screen is shown in Figure 4. This tab allows the user to view the data, or selected attributes from the data. Further, it allows the user to

search for specific values in the records. It is also possible to include a filter criteria.

The view screen is shown in Figure 4. This tab allows the user to view the data, or select attributes from the data. Further, it allows the user to search for specific values in the records. Finally, may also include a filtering criteria.

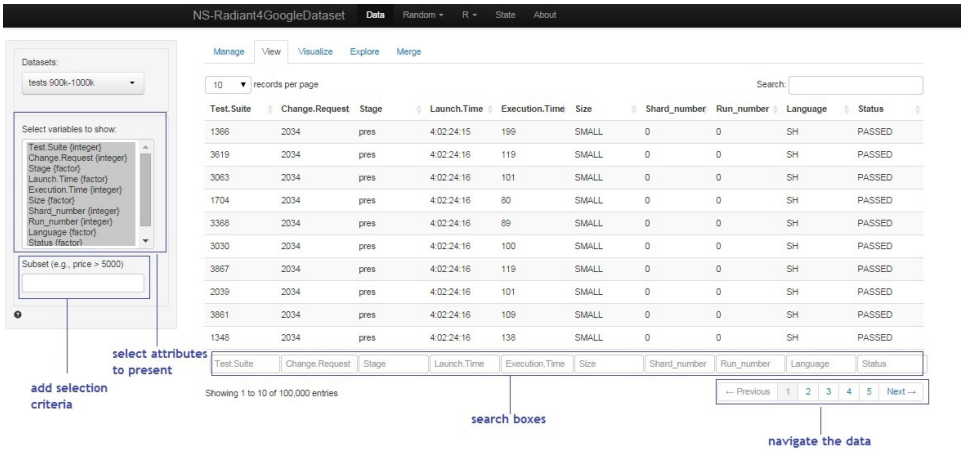


Figure 4: View

Figure 5 shows the visualization feature. This feature lets the user create one or two dimensional diagrams. It is also possible to add a column and a row facet. This feature supports all types of data and creates scatterplot, boxplot, histogram or mosaic plots according to the variables selected (i.e. their types). It is also possible to mark the Multiple button. Multiple plots several graphs of x variables against a single y variable. For example, in Figure 6 languages distribution is plotted as function of status and stage. Other options that applicable for scatter plots adding Line, Loess curve and Jitter (which add a small amount of noise to numeric vectors).

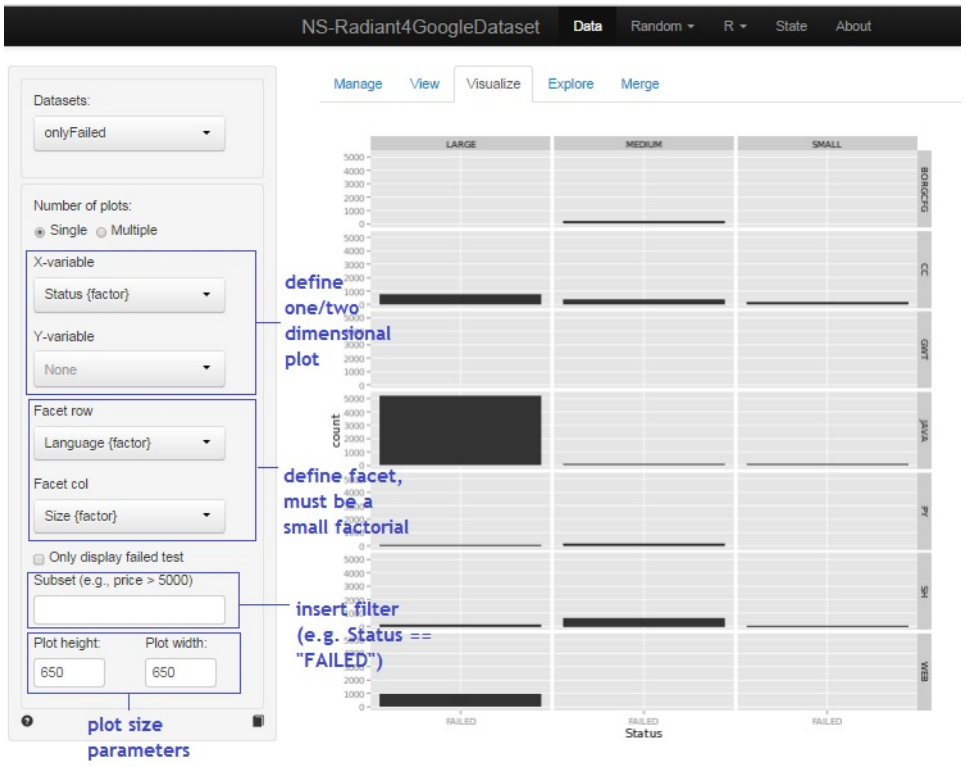


Figure 5: Visualize

Figure 7 shows the explore feature. This feature allows the user to easily group the data according to factorial attributes. It then generates functions over the numerical attributes such as computing the average, median and so on.

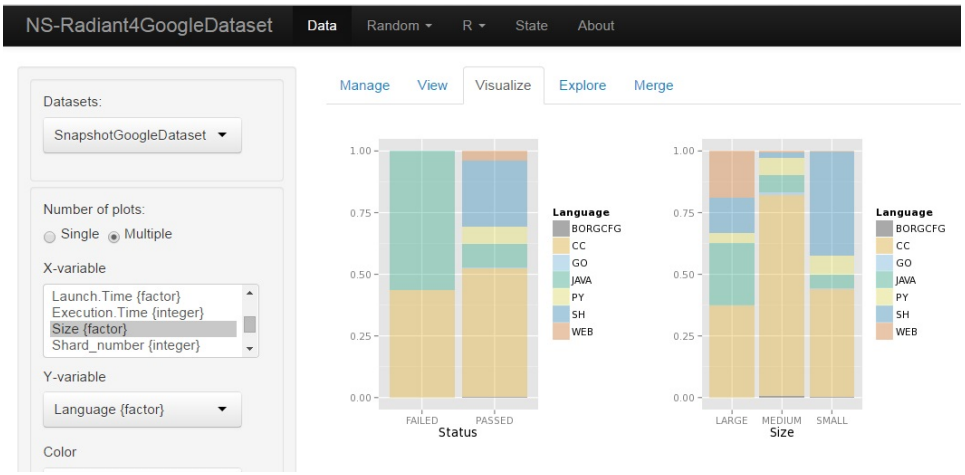
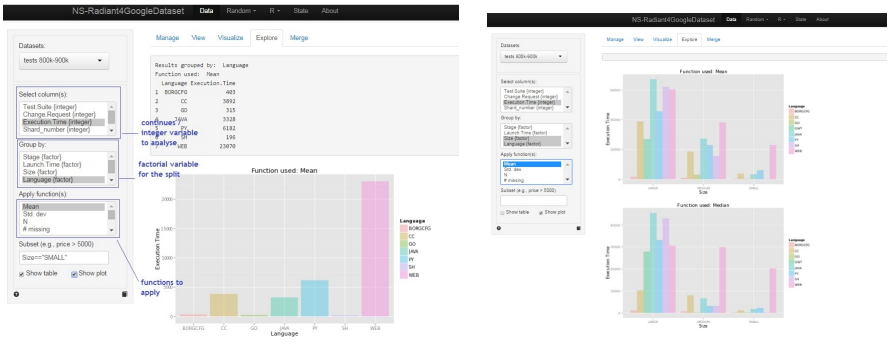


Figure 6: Visualize - multiple graphs example



(a) table & plot

(b) several functions applied

Figure 7: Explore

Figure 8 shows the merge feature. This feature allows the user to merge datasets. The merged datasets are required to have the same table structure (i.e. same number of rows, same column names). After selecting the required dataset, a merged data chunk is added to in-memory datasets. For a simple merge, select full to be the merge type and match all as the policy (as in Figure 8). The other combinations correspond to join logic in relational databases.

Figure 9 shows the random feature. This feature allows the user to create a random sample from selected datasets. The dataset from which the data is sampled is defined as the current selection in the Data feature. After generating a sample, a new dataset will be added to the in-memory datasets. Multiple samples can be taken from the same data chunk (samples have consecutive counters near their name).

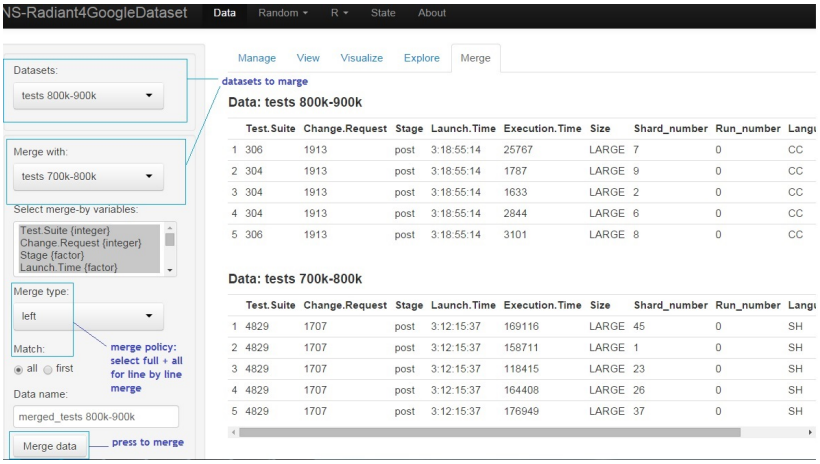


Figure 8: Merge

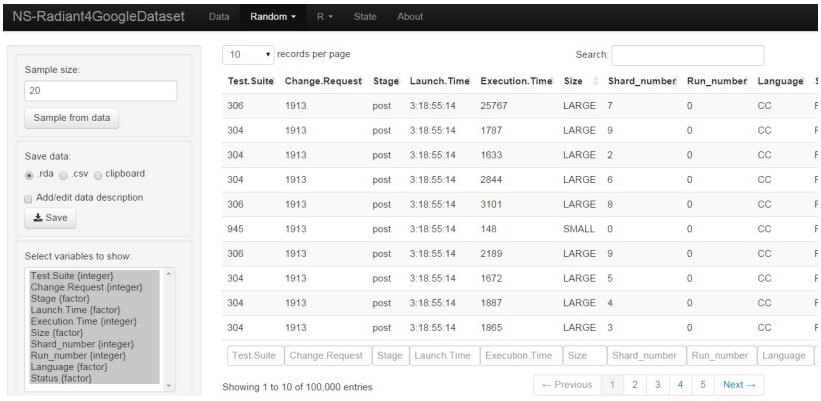


Figure 9: Random

As a final note, we left "R", "State" and "About" tabs from Radiant. This features allow the user to include his own R code, save and load the application state and provide basic information about Radiant4Google.