Proposal for a Jenkins plug-in to facilitate benchmarking

Daniel Mercier
Computational Science Group
Autodesk Research

December 4, 2017

Introduction

The Adv. Computation Research team primarily develop solutions in C++. The infrastructure has evolved progressively to reflect this direction. It now includes source control using Git, a custom CMake framework to facilitate operations and an automated platform for Continuous Integration(CI) using Jenkins. The automated platform compiles and tests the solutions and the result are made available through its user interface. However, the display of tests results is currently limited to boolean results, i.e. success or failure. The next stage for the team infrastructure is to acquire a benchmarking capability of numerical test results. The current platform, Jenkins, has a very strong capability for extensions through plug-ins. This document reviews the existing available plug-ins and proposes a new plug-in to benchmark test results with automated notification whenever a result crosses a set threshold.

1 Basics about Jenkins

Jenkins is an open source automation server written in Java. Jenkins is a predominant solution in the software industry to implement Continuous Integration(CI) and Continuous Deployment(CD). It originated from another project called *Hudson* started in summer of 2004 at *Sun Microsystems*. Jenkins was built around the concept of individual *Jobs* and a very simple work-

flow. A *Job* is composed by an abstract sequence of steps. The steps are grouped into work-flow events as follow:

- Install tools (such as JDK, Ant, Maven, etc)
- Perform SCM checkout (such as SVN or Git)
- Perform build step(s) (such as build Ant project, compile code, etc)
- Perform post-build steps(s) (such as Archive Artifacts, send email, etc)

A *Build* is an active execution of the *Job* steps. For this reason, every *Build* of a *Job* is numbered and incrementally increased. During a *Build*, the source files and any related material is placed into a temporary workspace. The steps are executed inside this temporary workspace. After each *Build*, any valuable artifact must be archived as the workspace content is either cleared or overwritten during the following *Build*.

2 Jenkins plug-ins

Jenkins was built around the concept of plug-ins. The intend was to provide a tool-set to expand each segment of the work-flow with new and powerful capabilities. The technology behind plug-ins is based on Java for the logic, Jelly/Groovy for the display and a strong set of Java base classes to describe the work-flow. The association between display and logic is carefully hidden by a system called *Stapler* developed by the creator of Jenkins, Kohsuke Kawaguchi. Because plug-ins are built using Java, developing a plug-in can be done on most Java IDE such as *Eclipse* or *IntelliJ IDEA*.

Jenkins plug-ins are open-source and there is a wide pool of existing plug-ins so contributing to an existing plug-in is usually the preferred way before starting a new plug-in. However there are many challenges to contributing to an existing plug-in: First, find a compatible plug-in; Second, convince the plug-in owner of the validity of the changes; Third, adopt the coding practices developed for the plug-in. In additional, with commercial plug-ins, there is the likely conflict with their commercial finality which explain the opposition of these plug-in owners to contributions.

The intend of this document is to propose a new plug-in to retrieve and process test results. As mentioned before, there are a few plug-ins built for a similar purpose. This review will try to present the current offering and

how much these existing plug-ins match with the intended purpose. Overall, the following six plug-ins emerged as the current best plug-ins for comparing boolean and numerical test results.

The principal plug-in for collecting and processing test results is certainly the JUnit plug-in [1]. Created in part by Kohsuke Kawaguchi, the founder of Jenkins, the plug-in focuses on Java unit tests. The JUnit file format is based on the XML format. The format was designed for unit test; and as such, for boolean results. It is extremely codified and carries only results of test successes/failures, the causes for failures, standard outputs and the execution times. Another noticeable disadvantage of this plug-in is the parsing and filtering of the original JUnit file into a new format composed with only the elements required by the plug-in user interface. This filtering partially prevents the insertion of additional XML information into the JUnit file that could be used by other plug-ins for processing and displaying.

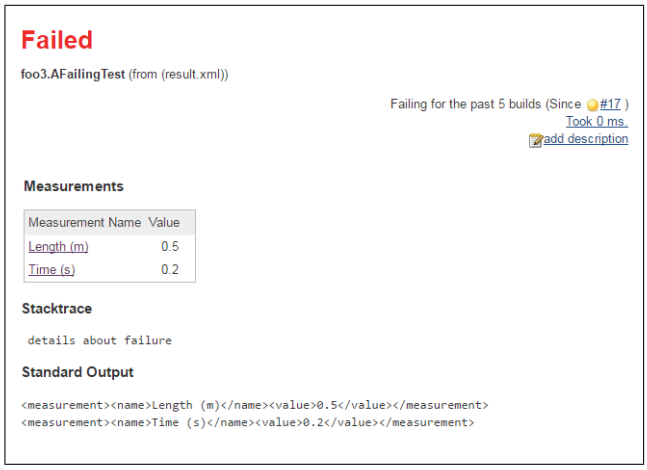

Figure 1: Measurement plot plug-in - Build results

The next noticeable plug-in is the measurement plot plug-in [2]. It is designed to display numerical test results as illustrated by figures 1 and 2. This plug-in works in combination with the JUnit plug-in and requires the user to add formatted XML information in the standard output field of the JUnit file. The standard output is then parsed by the plug-in, the additional information retrieved and each numerical test result displayed in a separate page in both table and graph form .

Another plug-in is the Plot plug-in [6]. This plug-in was specifically designed for comparing numerical data between builds. The data is ex-

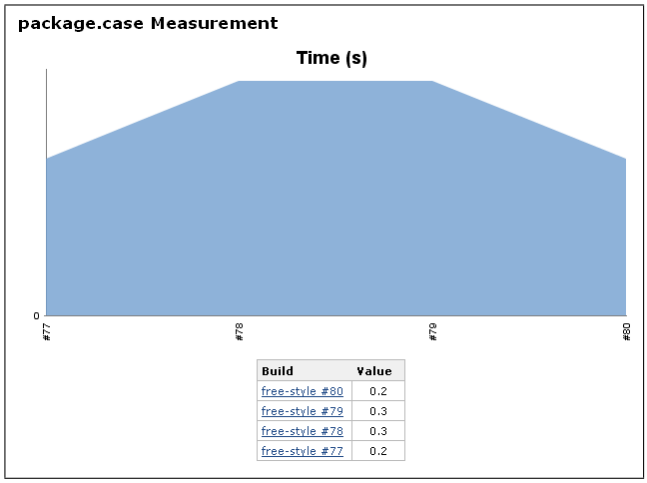

Figure 2: Measurement plot plug-in - Results over builds

tracted from a file generated during the *Build* and displayed in one or several graphs.The plug-in accommodates a range of input formats: XML, CSV & property file. Its disadvantage is that it only provides a graphing capability and a number of details about each graph must be provided beforehand in the job configuration page.

Another plug-in is the Test Analyzer plug-in [7]. This plug-in extend the result display capability of the JUnit plug-in by combining all results in a table. As the JUnit plug-in, the test results are limited to boolean values as well as execution times as illustrated by figure 3. However, the table form is well built to display results for all builds.

The final two plug-ins are the Performance [3] and Performance Publisher plug-ins [4]. They are specifically targeting network performances and are compatible with popular format for cluster monitoring and communication load testing. The Performance plug-in in particular, supports a variety of input formats, has an extended processing of results, displays information in graphic and table forms; and has in the configuration phase, the setting of thresholds for each individual results to set overall build success or failure. The only obstacle to the adoption of this plug-in is its specialization towards performance tools and hard-coded components .

Special mention goes the Performance signature plug-in [5]. The plug-in is designed to work with the Dynatrace software. While the displayed content matches our requirements, the input format is commercial and tied

Chart	See children	Build Number ⇒ Package-Class-Testmethod names ⇓	16	15	14	13	12	11	10	9	8	7	6	5
	⊖	org.common.samplea	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	N/A
	⊖	SampleATest	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	N/A
		testA	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	N/A
		testB	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	N/A
		testC	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	N/A
		testD	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	N/A
	⊕	org.common.sampleb	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	N/A
	⊖	org.common.samplec	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	N/A
	⊖	SampleDTest	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	N/A
		testA	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	N/A
		testB	SKIPPED	SKIPPED	SKIPPED	SKIPPED	SKIPPED	SKIPPED	SKIPPED	SKIPPED	SKIPPED	SKIPPED	SKIPPED	N/A
		testC	SKIPPED	SKIPPED	SKIPPED	SKIPPED	SKIPPED	SKIPPED	SKIPPED	SKIPPED	SKIPPED	SKIPPED	SKIPPED	N/A
		testD	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	N/A

Figure 3: Test result analyzer plug-in - Global table

to its commercial content.

Overall all these plug-ins provide a good indication about what can be achieved using Jenkins technology. As open-source codes, they are also programing material for new plug-ins. The closest plug-in is the Performance plug-in but is developed for computer performance testing. The second best is the measurement plot plug-in but lacks a global table and is limited to the JUnit format for the input file.

3 Proposed architecture

The reviewed plug-in offered a good range of capabilities but have incompatible features for our intended purpose. The proposed architecture is a composition of the different plug-in capabilities detailed above. The idea is not to rebuild everything from scratch but to build the new plug-in using code from the existing plug-ins. This plug-in will be associated with the post-build section of the *Job* work-flow and will rely on the data generated during the *Build* steps. Three aspects of Jenkins tool-set are leveraged:

- The *Configuration page* to activate the plug-in and set the plug-in environment
- A post-build action to collect and process each build information
- The Jenkins *Job* window to display the processed information

Altogether, the aim is to provide a good user experience with just the right amount of functionalities. In short, the ability to activate the plug-in on demand through the *Job* configuration page, clear and detailed display of the results and link to the notification system whenever the test results are incorrect or out-of-bound.

3.1 Configuration page

The configuration page will be the plug-in activator. Its content will take the form of an optional post-build action. Figure 4 provides a mock-up of the configurations for the proposed plug-in.They should provide enough options for the plug-in operations:

- Localize the data inside the build workspace
- Define the number of builds to display
- Entries to add test thresholds

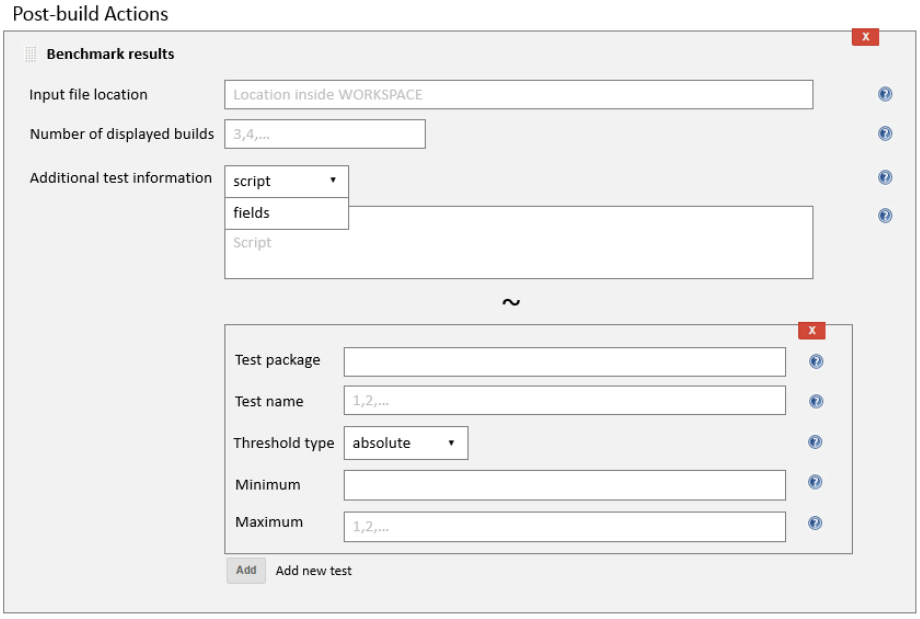

Figure 4: Plug-in configuration mock-up

3.2 Features

Of course, beyond the configuration page, a lot of the logic happens behind the scene.

A key feature is the definition for each *Build* of benchmark success or failure. In the figure 4, the last entry: *Additional test information* was added to allow the user to enter value thresholds and monitor variations between test results. The system will be linked to the *Build* success or failure.

The advantage of setting a *Build* status is a very useful feature when linked to the post-build *Notification plug-in*. It ensures notifications to the right users upon failure or upon test results being out of bounds. The basic scenario is when a test crosses its set threshold, the plug-in fails the *Build*; and the *Notification plug-in* (if activated) will send an email to the registered recipients about it.

The way to define the test threshold method and values for each test will be done through three possible sources and in priority order:

- UI entries for each test
- JSON script
- Content located inside the generated test result file

Here is an initial list of methods for test thresholds:

- Absolute (min/max)
- Relative percentage from last build
- Relative percentage from average
- Relative delta from last build
- Relative delta from average

Another important feature for this plug-in will be the entry file format abstraction which will allow a rich choice of formats. Here are some popular formats: JSON, XML, YAML, JTest, JMeter, ... This architecture will provide an easy way to add new formats.

3.3 Result display

Finally, the most important feature of this plug-in will be the display of test results. The idea is to display results either on the main *Job* window or through a separated window accessible through a button on the *Job* window left menu.

The choice of display follows other popular plug-ins and will combine table and graphs. Figure 5 and 6 gives some indications about the visual content.

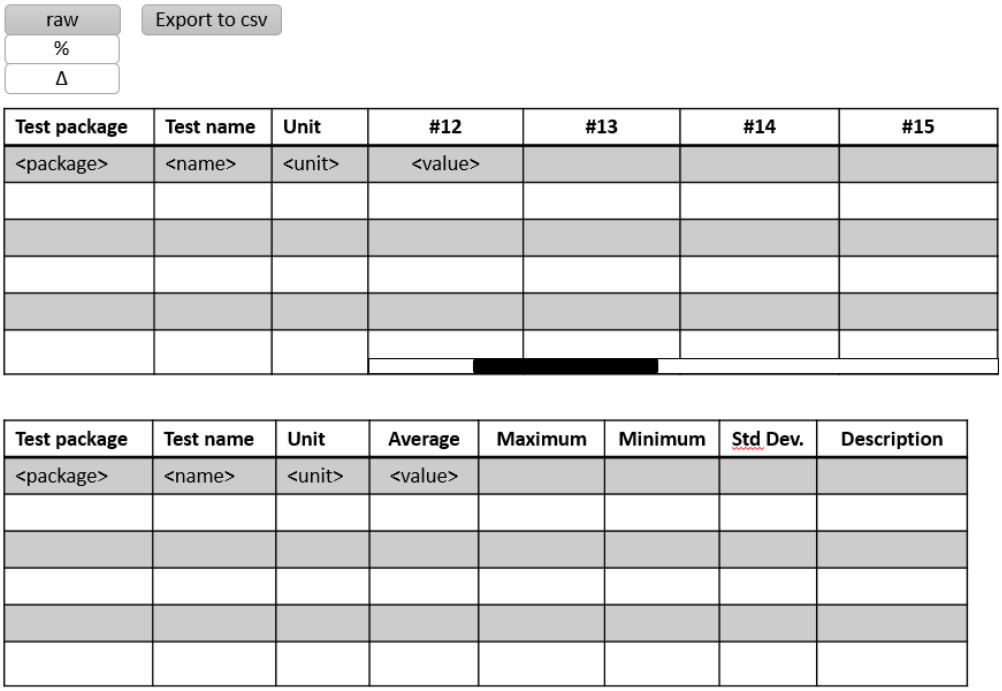

Figure 5: Table representation of test results

The table form as illustrated by figure 5, is split in two tables to display raw [top] and condensed [bottom] results. The top left button will allow the user to display results either as raw values or as relative delta or finally, as relative percentages compared to the last build. The second top left button will allow the user to export the condensed results to a *CSV* file for further post-processing.

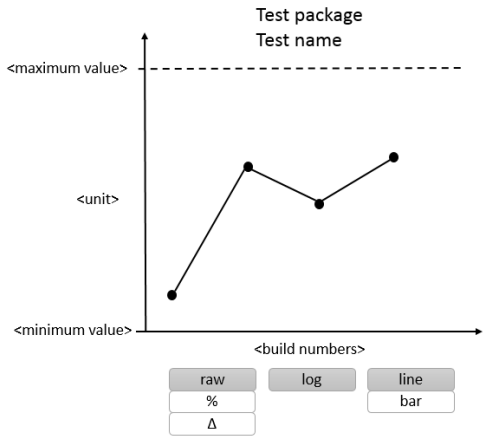

Figure 6: Graph representation of test results

The second type of display, the graph form as illustrated by figure 6 is very common and used by many other plug-in such as [3] or [6]. The way to access the graph will be as follow: Each line in the table of figure 5 will be associated with a test result and linked to a separate graph.

Conclusion

Jenkins offers many plug-ins for test result comparisons. Many such plug-ins are in comparison with our intend either partial solutions or are too specialized to be converted into a generic form. However, the advantage of the wide pool of existing open-source code is that it can easily be converted and combined into this new plug-in. This is the proposed solution. This new plug-in will combine existing code in order to target boolean and numerical test results alike, a powerful benchmarking capability with value thresholds and automated notifications, offer standard result displays and enough room to add new capabilities. Overall, this new plug-in should also make a fine contribution to the field of development operations.

References

[1] *Jenkins JUnit plugin*. URL: `https://wiki.jenkins-ci.org/display/JENKINS/JUnit+Plugin` (visited on 05/11/2017).

[2] *Jenkins Measurement plot plugin*. URL: https://wiki.jenkins-ci.org/display/JENKINS/Measurement+Plots+Plugin (visited on 05/11/2017).

[3] *Jenkins Performance Publisher plugin*. URL: https://plugins.jenkins.io/performance (visited on 05/11/2017).

[4] *Jenkins Performance Publisher plugin*. URL: https://plugins.jenkins.io/perfpublisher (visited on 05/11/2017).

[5] *Jenkins Performance Signature plugin*. URL: https://plugins.jenkins.io/performance-signature-dynatrace (visited on 05/11/2017).

[6] *Jenkins Plot plugin*. URL: https://wiki.jenkins-ci.org/display/JENKINS/Plot+Plugin (visited on 05/11/2017).

[7] *Jenkins Test Analyzer plugin*. URL: https://wiki.jenkins-ci.org/display/JENKINS/Test+Results+Analyzer+Plugin (visited on 05/11/2017).