Testing and Validation plugin for Spark Clusters?

Kevin Kim, Yan Liu
Faculty of Engineering and Computer Science
Concordia University
Montreal, Canada
kevin@concordia.ca
yan.liu@concordia.ca

***Abstract*—This paper presents a simple, yet efficient solution for testing and validating metrics at different level of a spark job. The solution presented leverages the Log4j library which requires low overhead and straightforward for integration with your spark application.**

I. Introduction

II. Problem Overview

Application development on a big data processing framework such as Apache Spark involves many challenges such as scalability, efficiency, accuracy, etc. Scalability means that the application should continue achieving its goal under a growing input data size. Efficiency describes an application which is completed under an reasonable amount of time depending set by the use case. Finally, the accuracy of means that multiple executions of the application will generate the same output. This paper focuses on the accuracy evaluation of an application.

On a Spark distributed environment, data shuffling and data partitioning are transparently handled by the framework unless manually specified by the user. These technicalities of parallel processing of data must carefully be considered during application development. The workflow should be tested and validated for accuracy of every stages of the spark application.

From the developer's perspective, it can be a time consuming task to manually validate the data after each stage of his application. This is usually achieved by logging to the console or to an output location where he can then access their consistency.

A. The contribution

The contribution of this paper is a prototype at automating the testing and validation process. This prototype extends the widely used Log4j library. By using this library, we have a lightweight troubleshooting tool which simplifies and reduces the time spend on testing and validation. This approach has a low integration overhead because of the commonly used Log4j library.

III. Proposed Solution

This section will present the Log4j custom module prototype, which we will simple name "Validator" module. After considering other alternatives, this solution was beneficial because it has a low overhead integrating with most applications, is based of a popular logging library with a strong community and incurres negligible latency overhead.

A. Validator Module Description

The Validator module is an extention of the Log4j appender. The default Log4j appender allows logging events to a target destination. The target location can a be a simple text file, a simple database, or many other various systems. In the current prototype, we are logging to a text file for simplicity.

The propose solution allows the user to use Log4j as he normally would, to log events of different levels: Error, Info, Debug, etch. In addition, the Validator module is different behind the hood where those log events are analyzed at the end of the spark application against historical data from previous executions.

At the end of the application, a method from this custom module is manually called to validate the loggings of interest with the ones of the previous executions of the application. This method is "validateAll()". The purpose of this method is to compare to read the target logging destination and to analyze if the same log events from different application execution are equal in value. At the end of the execution of the spark job, we will see in the console if the logging events are consistent or not. This insight is crucial for application development in a distributed environment.

B. Using the module

The module is currently a prototype. One of the key ideas of this Proof of Concept was for the simple integration with spark applications. This is achieved in the following steps. (We are assuming here that the user is user a Java environment, but we are expecting the reading to adapt the steps to their log4j compatible environment as necessary).

1) Include the Validator Dependency in the pom.xml
2) Create or add the log4j.properties file to your project with the following content
3) Instantiate the logger in the code using the following lines ....
4) Log desired values of interest using the follow pattern...
5) Call the following method at the end of the program to start the validation process

IV. Experimentation

For experimental purpose, we are using the Validator module on two in-house applications "smart-grid-analysis" and "Trajectory-Companion-Finder".

A. Experiments on application 1

The first application is "smart-grid-analysis" which uses the Spark framework to analyse a household energy consumption dataset to generate insights. The application is presents 3 different RDD transformations. An example of using the Validator was to output the count() of the RDD after those three transformations. By doing so, we were able to identify inconsistency between every execution of the application. We were able to see that for the specific transformation, the rdd count would always fluctuate.

B. Experiments on application 2

The second application is "Trajectory-Companion-Finder" which uses the Spark framework to analyse gps trajectory data to identify objects that are moving along the same trajectory. This application uses the DBSCAN unsupervised learning technique. Similar to previous experimentation, we identified inconsistency in the output results. By adding the Validator to the application and logging key metrics at different transformation of the spark job, we were able to identigy which transformation was the source of the inconsistency and quickly zone its root cause.

V. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.