Course Project Writeup - 0

SPARK API Converter

Md Rayhanur Rahman*
Ph.D. Student
Dept. of CSC
NC State University
Raleigh, NC, USA
mrahman@ncsu.edu

Mohammad Maruful Haque†
Ph.D. Student
Dept. of CSC
NC State University
Raleigh, NC, USA
mhaque3@ncsu.edu

Abstract

This report contains the initial concept notes from the SCALA API Converter programming project describing the problem definition, background, related work, methodology, timeline, testing criteria and finally workload distribution.

1 The Problem

In this project, we need to build a source code converter which can transform these following things along with some additional tasks.

1. **RQ1:** Getting introduced with Apache Spark, setting it up in a docker container and writing some piece of code in Spark RDD, Dataset and Dataframe API
2. **RQ2:** Spark RDD API to Spark Dataset API
3. **RQ3:** Spark RDD API to Spark Dataframe API
4. **RQ4:** Implementing some Spark SQL Api calls to Scala functions

2 Background and Motivation

Spark is very popular these days for distributed computing tasks such as data analysis, graph modeling and machine learning etc. There are three APIs in Spark at present named RDD, Dataset and Dataframe. The three different APIs provide different expressive power to achieve the same thing although the performance gain can be different. Moreover, Spark introduced these three APIs in different times so that there are many existing implementations of Spark based systems that needs to be rewritten to gain better performance. So, an API transformer can be handy to migrate Spark APIs to achieve better performance. Moreover, by doing this, a great understanding will be built on compiler techniques such as scanning and parsing, handling CFG grammar and intermediate representation.

*unity id: 200255928
†unity id: 200262103

NCSU, FALL 2018, October 18, 2018, Raleigh, NC, USA 2018.

3 Related Work

To the best of our knowledge after performing an exhaustive search, there is no automatic transformer for this problem. Although there are some good practices and tutorials on how to convert RDD API code to Dataset and Dataframe API, but those have to be done by a programmer and is a manual, time consuming process.

4 Methodology

RQ1 and RQ4 are pretty straightforward. All that is needed to be performed is to install docker in the system, deploy the spark image there and then playing with the spark API to learn how to use it. But RQ2 and RQ3 is a sizable task. Here is the abstract thoughts on how to solve these two problems.

1. **RQ2:** It should not be that much challanging. A tokenizer needs to be developed that can extract all the tokens from a RDD API code sequence. Then, using the RDD to Dataframe transformation rule, the RDD tokens need to be replace with the corresponding Dataframe tokens.
2. **RQ3:** It is not that straightforward as the previous task. Apart from the tokenizer, a parser needs to developed that can build an AST of the RDD API. For parsing, the provided grammar needs to be also converted to a LL1 or LR1 as the provided grammar is ambiguous. After building the AST, UDF part from the RDD API needs to be extracted and replaced with the SQL API. But this is challenging because a custom set of SQL rules and grammars need to be developed to replace UDF tokens with SQL. tokens. Basically, a lambda expression needs to be converted to a declarative SQL expression to achieve the goal.

5 Testing Criteria

The transformer program should be standalone and self packaged. Hence, there should be no need for configuration or settings file. It will be designed in such a way so that the tester can test it just by running the scripts and providing appropriate input files. Custom test cases can be written as well to test some sample inputs and

outputs. The programming language that is preferred to be used is either Java or Python with ANTLR4 for handling the grammar. However, the whole scanner and parser can also be written from scratch. The code samples as input and produced output code samples must preserve the same meaning and produce the same result. That should be the key testing criteria.

6 Workload Distribution

These are the breakdown of the whole project work:

1. Setting up Spark and writing programs in three different API - Maruful
2. Building a Scanner for RDD - Rayhanur
3. Replacing RDD tokens with Dataset tokens - Rayhanur
4. Building a Parser for RDD - Rayhanur
5. Building an AST generator for RDD - Rayhanur
6. Converting the provided grammar to LL1 or LR1 - Rayhanur
7. Developing custom rules for Spark SQL - Maruful
8. Converting RDD UDF to Spark SQL - Maruful
9. Writing Scala functions for several Spark SQL Api - Maruful

7 Timeline

1. Part 0 & 1 is expected to be finished by November 7
2. Part 2 & 3 is expected to be finished by November 28
3. Other miscellaneous task such as report writing, test cases, packaging are expected to be finished by December 1

8 Update

We have made satisfactory progress so far. Here is the list of things we have completed so far.

1. **Part 0:** According to the aforementioned timeline, we have finished setting up Apache Spark environment via docker and writing 6 functions each in RDD, Dataset and Dataframe API.
2. **Part 1:** We have also completed transforming scala sources written in RDD API to Dataset APIs. Initially we planned to build the whole transformer, (i.e. scanner, parser, tree) from scratch. Later, we found out that there are some existing excellent framework for doing these types of tasks with *ANTLR, YACC, JavaCC, Scala Parser Combinators* etc. However, finally we choose a framework named *Lark* which is written in pure Python to generate scanners and parsers for our goal. We didn't choose the De Facto *ANTLR* as it is heavy and initial learning curving is a bit steep despite being a powerful framework for these type of tasks. By using *Lark*, we first had to write a grammar in BNF for RDD API. As there has been no grammar provided for the RDD API in the project details documents given to us, we built that grammar on our own. Then we passed that grammar to the *Lark* and it automatically generated tokenizer and parser for it. Then we simply did the tasks mentioned in Part 1 by just replacing the tokens with corresponding tokens in the parse tree generated by *Lark*. We used Python for these tasks.
3. **Part 2:** We have also partially completed these part as well till generating SQL from UDF codes. By using *Lark* we generated AST from the grammar that has been provided to us. It is mentionworthy that, we needed to modify the grammar a little bit to feed it to *Lark* framework. Now we are studying the structural properties of UDF and SQL. Our UDF to SQL transformer will be built on those difference of structural properties found in the grammar of UDF and SQL.

The milestones and workload distribution we completed so far is still as per the initial plan we submitted. However, it is likely that there would be a slight change in the remaining tasks and workloads. Because, as we planned to build scanner and parser from the scratch, our initial workload distribution reflected that as well. But as we used a tool, so that our workload has been reduced a little bit and so that we will adjust that in the subsequent tasks. However, the timeline is expected to remain same. So, a recap of the tasks is given in the following:

1. **Part 0:** Done by Maruful
2. **Part 1:** Done by Rayhanur
3. **Part 2:** Partially done by Rayhanur. Rest of the tasks will be done according to the timeline.
4. **Part 3:** Not started yet. It will be done according to the timeline.