## CSC512/495 Course Project: Spark API Converter

## Overview

[Spark](http://spark.apache.org/) is a popular analytics engine for large-scale data processing. It supports various types of tasks, including generic data analysis, machine learning, stream computing, and graph computing. This project involves three different kinds of Spark APIs that all target generic data analysis. They are RDD API, Dataset API, and Dataframe API respectively. RDD API is the most expressive one. Dataset API is less expressive than RDD API but it enjoys better performance in most cases, thanks to its compact intermediate data representation. Dataframe API is a subset of Dataset API, and it is the least expressive one. But Dataframe API could achieve the best performance in general because it can exploit the highly optimized Spark SQL. For more details, please refer to the blog: [A Tale of Three Apache Spark APIs: RDDs, DataFrames, and Datasets](https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html).

Although the three APIs are different in expressiveness, most tasks can be expressed in any of them. In this project, you are going to translate programs written in RDD API to programs written in Dataset API or Dataframe API. You are expected to do the translation automatically via the compiler techniques we learned, and understand the limitations of these techniques.

## Part 0: Playing with Spark (10%)

An easy way to run spark on your computer is using Docker, which is already installed on the VCL image. You can also [install Docker on your computer](https://docs.docker.com/install/#supported-platforms). Use the following command to create a Docker container with Spark:

docker run -it --name spark p7hb/docker-spark:2.2.0 bash

This command creates a container named spark, and then launches a shell inside the container. Next time you can use the following command to start the container:

docker start -i spark

Or launch another shell:

docker exec -it spark bash

Within the container’s shell, you can run the spark-shell command. The spark-shell is built upon [Scala](https://docs.scala-lang.org/tour/tour-of-scala.html) shell, thus it supports [Spark’s APIs for Scala](http://spark.apache.org/docs/2.2.0/api/scala/index.html#org.apache.spark.package). **We will only use spark-shell to run Spark programs for this project**.

In the Spark shell, two contexts are already set up. One is sc and another one is spark. The sc is used for RDD API, and spark is used for Dataset API and Dataframe API. Here are examples that count the number of words in a text file using different APIs:

* RDD API

sc.textFile("yourTextFile.txt")

.map(line => line.split(" ").size)

.reduce((a, b) => a+b)

* Dataset API

spark.read.textFile("yourTextFile.txt")

.map(l => l.split(" ").size)

.reduce((a, b) => a+b)

* Dataframe API

spark.read.text("yourTextFile.txt")

.selectExpr("sum(size(split(value,' ')))")

.as[Long].first

References for Spark APIs:

<http://spark.apache.org/docs/2.2.0/quick-start.html>

<http://spark.apache.org/docs/2.2.0/rdd-programming-guide.html>

<http://spark.apache.org/docs/2.2.0/sql-programming-guide.html>

<https://catalog.lib.ncsu.edu/record/NCSU3500562>

Notes:

* You can mount a directory of your host machine onto the container, and then use the directory to sync files between the host machine and the container. See  
  <https://docs.docker.com/storage/volumes/#start-a-container-with-a-volume>
* In Spark shell, you can type :paste to enter paste mode, and then paste a large chunk of code into it. To exit paste mode, press Ctrl+D.

### Deliverables

In this part, you are expected to submit six programs. Three of them contains a single statement each, and the other three can contain one or more statements.

Each program has three versions that share the same functionality but are written in different APIs (RDD API, Dataset API, and Dataframe API). In your submission, you should include the test input files; in your report, you should include the execution results of your programs on the test inputs.

## Part 1: From RDD API to Dataset API (30%)

In this part, you are expected to submit a program that is able to transform some programs written in RDD API to programs written in Dataset API.

We will use single-statement programs (i.e., the program contains only one single statement) to test your submission. The programs will be composed only with the APIs listed in the following transforming rules. The arguments passed to the APIs could be arbitrary.

### Examples

* RDD API

sc.range(1,10000000)

.map(i=>(i%11, 1))

.reduceByKey((a:Int, b:Int) => a+b)

.collect()

* Dataset API

spark.range(1,10000000)

.map(i=>(i%11, 1))

.groupByKey(\_.\_1)

.agg(reduceByKeyAggregator((a:Int, b:Int) => a+b))

.collect()

### Transforming rules

|  |  |
| --- | --- |
| sc | spark |
| range | range |
| textFile | read.textFile |
| map | map |
| filter | filter |
| reduce(<func>) | select(reduceAggregator(<func>)).collect() |
| reduceByKey(<func>) | groupByKey(\_.\_1).agg(reduceByKeyAggregator(<func>)) |
| sortBy(<func>) | map(row=>((<func>)(row), row)).orderBy("\_1").map(\_.\_2) |
| collect | collect |

The helper functions reduceAggregator and reduceByKeyAggregator are provided in helpers.scala (TBD), and you should paste them into Spark shell before run the transformed program.

### Implementation of your transformer

Your transformer needs to tokenize the program first, and then replace RDD APIs to corresponding Dataset APIs. The token definitions are given as follows:

### Token definitions

<letter> --> a | b | ... | y | z | A | B | ... | Z | **\_**

<digit> --> 0 | 1 | ... | 9

**<number>** --> <digit>+

**<identifier>** --> <letter> (<letter> | <digit>)\*

**<string>** --> any string between (and including) the closest pair of double quotation marks.

**<char>** --> a character between (and including) a pair of single quotation marks.

**<symbol>** --> any non-space character that is not a part of other tokens

### Deliverables

A program that is able to transform some programs written in RDD API to programs written in Dataset API. In your submission, you should include the test input files; in your report, you should include the execution results of your programs on the test inputs.

## 

## Part 2: From RDD API to Dataframe API (30%)

In this part, you are asked to create a transformer that transforms a program written in RDD API to a program in DataFrame API. And if the input program contains UDFs (user defined functions), they should be replaced with Spark SQL expressions. Such a replacement can usually improve the performance of the program significantly as it avoids some overhead in calling UDFs.

A short explanation of UDF: A UDF is a function defined by users. They are often used as parameters in RDD API calls (e.g., map). They are often frequently called during the execution of a program, calling large runtime overhead. UDF in general can be complex; in this project, we restrict the forms of UDF. Below is the grammar specifying the possible forms of an input program:

### Grammar

<Program> ::= sc.range(<number>,<number>)<MapOps>.collect()

<MapOps> ::= ∅ | <MapOps>.map(<UDF>)

<UDF> ::= <identifier> => <Expression>

<Expression> ::= {<ComplexExpr>} | <SimpleExpr>

<SimpleExpr> ::= <PureExpr> | (<TupleExpr>)

<TupleExpr> ::= <PureExpr>, <PureExpr> | <TupleExpr>, <PureExpr>

<ComplexExpr> ::= <SimpleExpr> | <AssignExprs>;<SimpleExpr>

<AssignExprs> ::= <AssignExpr> | <AssignExprs>;<AssignExpr>

<AssignExpr> ::= val <identifier> = <PureExpr>

<PureExpr> ::= <identifier> |

<identifier>.<identifier> |

(<PureExpr>) |

<PureExpr> <Op> <PureExpr> |

if ( <CompExpr>) <PureExpr> else <PureExpr>

<CompExpr> ::= <PureExpr> <Comp> <PureExpr>

<Op> ::= + | - | \* | %

<Comp> ::= == | < | > | != | >= | <=

Notes:

* Token definitions are the same as in Part 1.
* Only operation map and the lambda form of UDF are required to support.
* A UDF only returns Int, Long, or a tuple composed by Int or Long.
* Since the semantic of division is different in Scala and SQL, it is excluded for simplicity.
* All assignment statements begin with val, which means the value cannot be modified.

### Implementation of your transformer

For an input program, your transformer needs to generate the AST of the program and then apply the following translation rules to generate a new version of the program written in DataFrame APIs. Rules for transforming APIs are provided, but you need to figure out the rules for translating UDF to SQL. An example is provided at the end of this part for your reference.

### Translation rules

* From RDD API to DataFrame API

|  |  |
| --- | --- |
| sc | spark |
| range(m,n) | range(m,n).selectExpr("id as \_1") |
| map(UDF) | selectExpr(SQL) |
| collect | collect |

* From UDF to SQL translation

You need to create the rules for translating UDF to SQL on your own.

### Example

* RDD API

sc.range(10,100)

.map(i=>{val j=i%3;(i, if(j==0)i\*10 else i\*2)})

.map(r=>r.\_1+r.\_2)

.collect()

* Dataframe API

spark.range(10,100).selectExpr("id as \_1")

.selectExpr("\_1 as \_1", "if(\_1%3==0,\_1\*10,\_1\*2) as \_2")

.selectExpr("\_1+\_2 as \_1")

.collect()

### Deliverables

A transformer that translates programs written in RDD API to programs written in DataFrame API. The input programs may use UDF. In your submission, you should include three test programs that use UDF; in your report, you should include the transformed code of the test programs.

## Part 3: Identify the limitation (30%)

Although in Part 2 the translation only applies to very simple Spark programs, it is possible to expand the grammar and translation rules to support more complex programs.

### Deliverables

In this part, you are required to write Scala code that implements some Spark SQL APIs; an example is given below. Each team picks 10 APIs from a set that we will provide. In your report, explain your thoughts on how a transformer could be potentially created such that it can automatically replace the Scala code with the corresponding Spark SQL API. If you feel that such a transformer is very difficult to create in general, identify the difficulties.

### Example

* SQL API

trim()

* Scala implementation (variant 1)

def trim(str: String) = {

var (s, e) = (0, str.size)

while (s < e && str(s) == ' ') s += 1

while (e > s && str(e-1) == ' ') e -= 1

str.substring(s, e)

}

* Scala implementation (variant 2)

def trim(str: String) = {

var (s, e) = (0, str.size)

for (i <- 0 until str.size) {

if (s == i && str(i) == ' ') s += 1

else if (e == str.size && str(i) == ' ') e = i

else if (str(i) != ' ') e = str.size

}

str.substring(s, e)

}

* Scala implementation (variant 3)

def trim(str:String) = {

var ret = ""

var s = 0

while (s < str.size && str(s) == ' ') s += 1

for (i <- s until str.size) {

if (str(i) != ' ') {

ret += str.substring(s, i+1)

s = i+1

}

}

ret

}