Tad Kellogg
UNI: twk2113
September 28, 2015

COMS W4115: Project Proposal

Introduction

"Describe the language that you plan to implement."

The language proposed for this project falls under the application domain of Big Data. One of the newer infrastructure as a software systems to implement parallel processing on commodity nodes is Apache Spark. The Spark system, along with either Meos or Hadoop for multi-node execution, implements parallel algorithms that can align within the paradigm of a directed acyclic graph programming pattern. Spark uses resilient the distributed dataset (RDD) as the primary data store concept, as such all Spark programs result in a set of transformation and action expressions applied to RDDs. Spark currently supports programs written in Scala, Java or Python using the Spark API. Inspiration for this project comes from the Hadoop based interpreter Pig and the Pig Latin language, where pig latin programs are translated into a set of Map/Reduce java programs for execution in the Hadoop computing platform. The intent of this project is to write a compiler that takes as input source an AWK inspired language, the Spawk language, and produces a as output a target program for execution as a Spark program written in the Python + Spark API language.

Language Highlights

"Explain what sorts of programs are meant to be written in your language"

Just as the AWK language was able to bring programming simplification for the areas of data extraction, manipulation and reporting, so too will Spawk simplify the programming required for similar operations using the Spark cluster computing platform for Big Data. In more broad terms, Spawk could be used as the sole constructor or a component constructor for Big Data Extract Transform Load (ETL) systems.

"Explain the parts of your language and what they do"

Spawk should support most of the same constructs found in the AWK language including:

condition { action } structure of statements with:

BEGIN, END, /regular expression/ and understood empty condition for match every line.

and semicolon (;) as statement separator

variables, constants and operators

control statements: if-else, while, for, …

associative arrays

built in variables (i.e. $0, $1 … NF …) for record processing

user defined functions

built-in functions, such as:

numerical: int, sqrt, exp, log, sin, cos, atan2, rand, srand

string: index, length, match, split, sub, substr, tolower, toupper

input/ouput: file, close, print - supporting Spark streaming, HDFS and local file access for RDD data stores. Also Spark file formats of text, JSON should be supported. Spawk will support both RDD data stores of free form text and pairRDD data stores of key/value pairs. Spawk will also try to determine the optimal use of RDD storage levels via the persist and cache operations.

time: systime, strftime

As in the case of the Hadoop Pig interpreter where many sequences of Map/Reduce jobs are produced for execution of a single pig latin program, the Spawk compiler may have to produce a sequence of Spark programs in order to fully accomplish the desired intent expressed in a single Spawk language program.

Example program

"Include the source code for an interesting program in your language"

Spawk source for counting words in a file, where the main objective is simplification from the Spawk language using built-in commands (file and print) , built-in variables (NF, $i) and an associative array operation to generate all the needed Spark operations for the same functionality.

```
BEGIN { file(testfile,"text") }
{for(i=1;i<=NF;i++) a[$i]++}
END {for(k in a) print(k,a[k],"%s: %i")}
```

Output from compiler will result in issuing the input/output, transformation and action calls in the python + Spark API language. The example code below shows a best possible "translation" of Spawk into PySpark calls.

```
import sys
from operator import add
from pyspark import SparkContext
sc = SparkContext()
A = sc.textFile(testfile, 1)
B = A.flatMap(lambda x: x.split(' ')) \
        .map(lambda x: (x, 1)) \
        .reduceByKey(add)
C = B.collect()
for (D, F) in C:
    print("%s: %i" % (D, F))
sc.stop()
```

References

Karau, H., & Konwinski, A. (2015). Learning Spark. Databricks: O'Reilly Media, Inc.

Aho, A., & Kernighan, B. (1988). The AWK programming language. Reading, Mass.: Addison-Wesley Pub.

Apache Spark™ - Lightning-Fast Cluster Computing. (n.d.). Retrieved 2015, from http://spark.apache.org/

Yadav, R. (2015). Spark cookbook: Over 60 recipes on Spark, covering Spark Core, Spark SQL, Spark Streaming, MLib, and GraphX libraries. Packt Publishing Ltd.