SPAWK

Language Reference Manual

Tad Kellogg

uni: twk2113

COMS W4115 CVN Fall 2015

1. Introduction

The SPAWK language is a compiler that takes as program code source an AWK inspired syntax and produces a as output a target program for execution as a Spark program written in the Python + Spark API. Apache Spark is one of the newer infrastructure as a software systems to implement parallel processing on commodity nodes. 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. Inspiration for the SPAWK language 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. 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.

2. Lexical Conventions

2.1.Tokens

SPAWK has six types of tokens: identifiers, keywords, constants, operators, separators and white space.

2.2.Comments

SPAWK uses the comment convention of a line starting with the '#' character and ending with a newline character. e.g.

```
# a comment line
```

2.3.Identifiers

In SPAWK in identifier is the name of a variable, array, built-in function or user-defined function. The name of an identifier is composed of a sequence of alphanumeric characters, starting with a letter (A-Za-z0-9, case-sensitive). The identifier name can not match a keyword name.

2.4.Keywords

The following keywords and names of built-in functions are considered reserved words:

BEGIN break continue do else END for function if in print while

index length split map reduceByKey

2.5.Constants

2.5.1. SPAWK has four types of constants: numeric, string, regular expression and pattern statements.

2.5.2.Numbers

Numbers can be an integer, a decimal fraction, or a number in scientific (exponential) notation. Some examples of numeric constants, which all have the same value:

205

2.05e+2

2050e-1

2.5.3.Strings

A string constant consists of a sequence of characters enclosed in double-quote marks. e.g.:

"Ocaml"

represents the string whose contents are ‘Ocaml’.

One uses backslash as an escape sequence to include special character in a string constant.:

\”

Represents a literal double-quote, ‘“’.

\\

Represents a literal backslash, ‘\’.

\a

Represents the "alert" character, control-g, ASCII code 7.

\b

Represents a backspace, control-h, ASCII code 8.

\f

Represents a formfeed, control-l, ASCII code 12.

\n

Represents a newline, control-j, ASCII code 10.

\r

Represents a carriage return, control-m, ASCII code 13.

\t

Represents a horizontal tab, control-i, ASCII code 9.

\v

Represents a vertical tab, control-k, ASCII code 11.

\nnn

Represents the octal value nnn, where nnn are one to three digits between 0 and 7. See unix ascii man page for octal codes.

\xhh...

Represents the hexadecimal value hh, where hh are hexadecimal digits (`0' through `9' and either `A' through `F' or `a' through `f').

2.5.4.Regular Expressions

A regular expression is enclosed in slashes prior to the action statement as the pattern statement, such as

/^start and end of line$/ { action }

AWK Regular expression syntax is supported.

2.5.5. Pattern Statements

The special pattern of BEGIN and END are used in the pattern position to supply start-up or clean-up actions, they are not used to match input records.

2.6.Operators

The operators in SPAWK, in order of increasing precedence, are:

= += -= *= /= %= ^=

Assignment. Both absolute assignment (var=value) and operator assignment are supported.

||

Logical "or".

&&

Logical "and".

< <= > >= != ==

Relational operators.

blank

String concatenation.

+ -

Addition and subtraction.

* / %

Multiplication, division, and modulus.

+ - !

Unary plus, unary minus, and logical negation.

^

Exponentiation.

++ --

Increment and decrement, both prefix and postfix.

$

Field reference.

( )

Grouping

{ }

Action statements, where action is a sequence of statements

2.7.Field separators.

The field separator, which is a single character, controls the way SPAWK splits an input record into fields. SPAWK scans the input record for character sequences that match the separator; the fields themselves are the text between the matches.

2.8.White space

Blanks used as default field separator character.

horizontal and vertical tabs, blanks are ignored except as they separate tokens. Blank space is required to separate otherwise adjacent identifiers, keywords, and constants.

Newlines used as default recored separator. However, SPAWK will ignore newlines after any of the following symbols and keywords:

, { ? : || && do else

3. Arrays

The SPAWK language provides one-dimensional arrays for storing groups of related strings or numbers.

Every array must have a name

Same syntax as variable names

Do not use one name in both ways (as an array and as a variable)

No need to specify the size of an array before use

Any number or string may be used as an array index, not just consecutive integers

4. Variables

4.1.User defined

A variable name is a valid expression by itself; it represents the variable's current value. Variables are given new values with assignment operators, increment operators and decrement operators. A few variables have special built-in meanings, such as FS, the field separator, and NF, the number of fields in the current input record. These built-in variables can be used and assigned just like all other variables, but their values are also used or changed automatically by SPAWK. All built-in variables names are entirely upper-case.

Variables in SPAWK can be assigned either numeric, string values or spark RDD pointers. Resilient Distributed Dataset (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel in the spark environment. By default, variables are initialized to the empty string, which is zero if converted to a number. Pre-initialize of each variable explicitly in SPAWK is not needed.

4.2.Built-in

$1, $2, $3, and so on ($0 is the entire line) are field variables that break a line of text into individual fields.

FS

FS is the input field separator

OFS

This is the output field separator. It is output between the fields output by a print statement. Its default value is " ", a string consisting of a single space.

ORS

This is the output record separator. It is output at the end of every print statement. Its default value is new line “\n”.

RS

This is the input record separator. Its default value is a string containing a single newline character, which means that an input record consists of a single line of text.

FILENAME

This is the name of the file that SPAWK is currently reading and subsequently the name of the default RDD variable. When no data files are listed on the command line, SPAWK reads from the standard input, and FILENAME is set to "STDIN". Inside a BEGIN pattern, the value of FILENAME is "NORDD"

FNR

FNR is the current record number in the current file. FNR is incremented each time a new record is read. It is reinitialized to zero each time a new input file is started.

NF

NF is the number of fields in the current input record. NF is set each time a new record is read.

NR

This is the number of input records SPAWK has processed since the beginning of the program's execution. NR is set each time a new record is read.

5. Statements

5.1.Action Statements

Most often, each line in an SPAWK program is a separate statement or separate rule, like this:

```
/42/  { print $0 }
```

To split a single statement into two lines at a point where a newline would terminate it, you can continue it by ending the first line with a backslash character (‘\’):

```
BEGIN { \
 print \\
```

```
    "hello, world" \
}
```

5.2.Selection Statements

If statement is used to check the conditions, if the condition returns true, it performs its corresponding action(s), where {} are needed:

```
if (conditional-expression)
{
        action1;
        action2;
}
```

If Else statement you can give the list of action to perform if the condition is false:

```
if (conditional-expression)
        action1
else
        action2
```

5.3.Iteration Statements

While statement is the simplest looping statement. It repeatedly executes a statement as long as a condition is true. It looks like this:

```
while (condition)
 {
        action1;
        action2;
}
```

The do loop executes the action statement once, and then repeats action statement as long as condition is true. It looks like this:

```
do
 {
        action1;
```

```
        action2;
}
```

while (condition)

The for statement looks like this:

```
for (initialization; condition; increment)
 {
        action statements
 }
```

The initialization, condition and increment parts are arbitrary expressions.

The for statement starts by executing initialization. Then, as long as condition is true, it repeatedly executes action block and then increment.

The for in loop, for iterating over all the indices of an array:

```
for (i in array)
   do { action statements with with array[i] }
```

6. Functions

6.1.Built-in

print or print(item1,item2,..)

The print function does output with simple, standardized formatting. You specify only the strings or numbers to be printed, in a list separated by commas. They are output, separated by single spaces, followed by a newline. The statement looks like this:

print item1, item2, ...

The entire list of items may optionally be enclosed in parentheses. The items to be printed can be constant strings or numbers, fields of the current record (such as $1), variables, or any expressions. Numeric values are converted to strings, and then printed.

index(s, t)

Return the position, in characters, numbering from 1, in string s where string t first occurs, or zero if it does not occur at all.

length[([s])]

Return the length, in characters, of its argument taken as a string, or of the whole record, $0, if there is no argument.

split(s, a[, fs])

Split the string s into array elements a[1], a[2], ..., a[n], and return n. All elements of the array shall be deleted before the split is performed. The separation shall be done with fs or with the field separator FS if fs is not given. Each array element shall have a string value when created and, if appropriate, the array element shall be considered a numeric string.

reduceByKey(A,func)

Merge the values for each key, here SPAWK assumes the key value is $1 in the record of the RDD variable A, using an associative reduce function(func). This is a call to python spark API and returns an array of collected values.

map(A,func)

Spark call for applying a function(func) to each record of the given RDD variable A. Returns a variable to new RDD.

6.2.User-defined

functions can be defined as:

function name([parameter, ...]) { statements }

A function can be referred to anywhere in an SPAWK program; in particular, its use can precede its definition. The scope of a function is global.

Function parameters, if present, can be either scalars or arrays; the behavior is undefined if an array name is passed as a parameter that the function uses as a scalar, or if a scalar expression is passed as a parameter that the function uses as an array. Function parameters shall be passed by value if scalar and by reference if array name.

7. Scope

7.1. User-defined Functions

In SPAWK, there is no way to make a variable local to a { … } block/compound statement. However, one can make a variable local to a user-defined function.

To make a variable local to a function, simply declare the variable as an argument after the actual function arguments (see Definition Syntax). Look at the following example, where variable i is a global variable used by both functions foo() and bar():

```
function bar()
{
   for (i = 0; i < 3; i++)
      print "bar's i=" i
}
```

8. Example Program

SPAWK programs take the form:

pattern { action }

structure with:

where pattern is one of: BEGIN, END, /regular expression/ and “understood empty condition for match every line”.

On the unix command line the name of the text file is given:

% spawk programFile.spawk inputFileName

where programFile.spawk contains the following lines (example program for word count):

```
function mapper(x)
{
        for (i=0; i< NF ; i++)
                print $i,1
}
function reducer(x,y)
{
```

```
        print (x+y)
}
{
A=map(FILENAME,mapper);
a=reduceByKey(A,reducer);
}
END {for(k in a) print(k,a[k])}
```

9. References

9.1. Kernighan, Brian W.; Ritchie, Dennis (1988-03-22). C Programming Language (p. 191). Pearson Education.

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

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

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

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

9.6. Robbins, A. (2003). GAWK: Effective AWK programming : A user's guide for GNU Awk (Ed. 3. ed.). Boston, MA: Free Software Foundation.