Laboratory Work 3: MapReduce with Phoenix++

Anita de Mello Koch (1371116), Nicholas Kastanos (1393410),
Brendon Swanepoel (601949)
School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

ELEN4020 Data Intensive Computing in Data Science
15 April 2019

I. Introduction

Phoenix is a MapReduce framework for shared memory systems implemented using C, and reimplemented in Phoenix++ for C++. It uses an inheritance-based model, which when the `split()`, `map()`, and `reduce()` functions are overwritten, can implement MapReduce functionality.

Three different algorithms were implemented using Phoenix++: a word-count algorithm which counts the words in the input text; a Top-K algorithm which outputs the top K elements of the word-count; and a reverse index of words in the text. These algorithms were run on two text inputs, and a list of stopwords is supplied to exclude them from the operations.

II. Results

A long and short set of data were used to run the algorithms. *Dracula* by Bram Stoker is used as the short text, as it is $820\ KB$ in size. The large file of data is a compilation of texts (which can be downloaded from the Phoenix Github page), and in $100\ MB$ in size.

The running times of the MapReduce section of each algorithm can be seen in Table I.

TABLE I: Running Times in Seconds

Algorithm	Small Data	Large Data
Word Count	0.0091722	0.553257
Top-10	0.0092126	0.549055
Top-20	0.0122066	0.535806
Reverse Index	0.0686829	3.60311

III. Algorithm Data Types

In order to count the number of appearances of a single word, the word-count and top-K algorithms use a key-pair which uses the word as a key, and the number of appearances as the value. By specifying that the `WordCounter` class inherits from the `MapReduceSort` class using a `sum_combiner`, the MapReduce operation sums the values for each unique key during combination, and sorts the list. The mapping function used for these algorithms can be seen in Algorithm 1.

The inverted indexing algorithm makes use of different key-value pairs. The word is used as the key, and the line number which the word appeared is used as the key. The inverted indexing algorithm also inherits from the `MapReduceSort` class, but makes use of a `buffer_combiner` to list all of the unique values of line number for each key. The mapping function used for these algorithms can be seen in Algorithm 2.

The split function used in all of the algorithms can be seen in Algorithm 3.

IV. Conclusion

Phoenix++ is can successfully solve MapReduce problems, and has a simple method of utilising the framework. By making use of C++, one is able to manage the memory more closely and increase the performance of the code.

Phoenix++ was sucessfully used to solve the three MapReduce problems, on both small and larger input data.

APPENDIX A
PSEUDOCODE FOR WORD COUNT

```
input : Section of text in each bin.
output: Text separated into single words and count of appearance.
foreach character in input do
  | Change character to lower-case;
end
while not at end of input do
  | while not at end of input & character is not aphabetic do
  |   | Increase iterator;
  | end
  | Set beginning of word to iterator;
  | while not at end of input & (character is alphabetic or apostrophe) do
  |   | Increase iterator;
  | end
  | if not beginning of word then
  |   | Output word and count 1;
  | end
end
```

Algorithm 1: Map function of the word count and top-K programs.

```
input : Section of text in each bin.
output: Text separated into single words and line number.
foreach character in input do
  | Change character to lower-case;
end
while not at end of input do
  | while not at end of input & character is not aphabetic do
  |   | Increase iterator;
  | end
  | Set beginning of word to iterator;
  | while not at end of input & (character is alphabetic or apostrophe) do
  |   | Increase iterator;
  | end
  | if not beginning of word then
  |   | Output word and line number;
  | end
end
```

Algorithm 2: Map function of the reverse index program.

```
input : Chunk which receives the data.
output: success or failure to allocate data to chunk
if at end of data then
  | return 0;
end
while iterator is less than the chunk size & not the end of the file do
  | increase iterator;
end
return 1;
```

Algorithm 3: Split function for the Phoenix++ programs.

APPENDIX B
STOPWORDS

a	below	from	how	most	over	them	we	won't
about	between	further	how's	mustn't	own	themselves	we'd	why
above	both	had	i	my	same	then	we'll	why's
after	but	hadn't	i'd	myself	shan't	there	we're	would
again	by	has	i'll	needn't	she	there's	we've	wouldn't
against	can	hasn't	i'm	no	she'd	these	were	you
ain't	can't	have	i've	nor	she'll	they	weren't	you'd
all	could	haven't	if	not	she's	they'd	what	you'll
am	couldn't	having	in	now	should	they're	what's	you're
an	did	he	into	of	should've	they've	when	you've
and	didn't	he'd	is	off	shouldn't	this	when's	your
any	do	he'll	isn't	on	so	those	where	yours
are	does	he's	it	once	some	through	where's	yourself
aren't	doesn't	her	it's	only	such	to	which	yourselves
as	doing	here	its	or	than	too	while	
at	don't	here's	itself	other	that	under	who	
be	down	hers	just	ought	that'll	until	who's	
because	during	herself	let's	our	that's	up	whom	
been	each	him	me	ours	the	very	why	
before	few	himself	mightn't	ourselves	their	was	will	
being	for	his	more	out	theirs	wasn't	with	