Homework 1 – CSE239

Mathis AUBERT

Tuesday October 14th, 2025

a) Execution Time

To run the wordcount.py with the enwiki9 file, the elapsed time was 156.31 seconds.

Output of the script:

```
TOP 20 WORDS BY FREQUENCY

1.    quot       : 1875356
2.    id         : 1423225
3.    s          : 1228278
4.    lt         : 1153456
5.    gt         : 1151777
6.    amp        : 919700
7.    was        : 817724
8.    from       : 664337
9.    on         : 630223
10.   title      : 543789
11.   page       : 528100
12.   text       : 515388
13.   revision   : 489476
14.   contributor : 487683
15.   timestamp  : 486951
16.   at         : 446624
17.   census     : 425663
18.   category   : 420522
19.   his        : 414908
20.   http       : 413453
Elapsed Time: 156.31737303733826 seconds
```

b) Performance Optimization

During my analysis, I saw that this step is taking the longer time is the map part. This part takes 150 seconds on the total of 156 seconds.

When I first try to improve the performance by parallelize the map and reduce parts, my performance was degraded by 2.9x/ -290%. The elapsed time was 454 seconds.

Secondly, I tried to only parallelize the reduce part and then I obtain an improvement by 1.12x / +12%

```
TOP 20 WORDS BY FREQUENCY

1.    quot      : 1875356
2.    id        : 1423225
3.    s         : 1228278
4.    lt        : 1153456
5.    gt        : 1151777
6.    amp       : 919700
7.    was       : 817724
8.    from      : 664337
9.    on        : 630223
10.   title     : 543789
11.   page      : 528100
12.   text      : 515388
13.   revision  : 489476
14.   contributor : 487683
15.   timestamp  : 486951
16.   at        : 446624
17.   census    : 425663
18.   category  : 420522
19.   his       : 414908
20.   http      : 413453
Elapsed Time: 138.7230818271637 seconds
```

To run the wordcount.py with the enwiki9 file with the reduce optimization, the elapsed time was 138.72 seconds.

c) Explanation of Improvements

For the parallelization, I used the python package "multiprocessing"[1] that enable us to "parallelize the execution of a function across input values, distributing the input data across processes (data parallelism)." [1]

In the python code, for calling part of the SimpleMapReduce function, I had:

```
with mp.Pool(self.num_workers) as pool:
```

For the lines:

```
map_responses = map(self.map_func, inputs)
```

And

```
reduced_values = map(self.reduce_func, partitioned_data)
```

I added a new variable to the definition of the SimpleMapReduce function,
And affected a value when the object is created:

```
self.num_workers = num_processes
```

I put "None" to the number of processes because, as explained by the documentation, "If processes is None then the number returned by os.process_cpu_count() is used." [1]

When I try to parallelize the map and reduce parts, the time got worst. This is probably due to the I/O part that block the process:

```
with open(filename, 'rt', errors='replace') as f:
```

The process is waiting to finish the following instruction after opening the file, and this cannot be parallelized due to access controls. Two processes cannot access to same file at the same time.

So, I only kept the reduce optimization part that improved the time by 18 seconds.

```
with mp.Pool(self.num_workers) as pool:
            reduced_values = pool.map(self.reduce_func, partitioned_data)
```

The complete wordcount.py code is attached in the Annex part of this document.

[1] Documentation for the multiprocessing python package:
https://docs.python.org/3/library/multiprocessing.html

d) Automation Script

Here is the bash script that I created to download and unzip the enwik9 file, then run the wordcount.py file.

To run the following bash, use the terminal command "./wordcount.sh" inside the folder that include both wordcount.sh and wordcount.py.

The complete package that includes: wordcount.sh and wordcount.py, is attached to this homework.

```bash
WORKDIR="$(pwd)"

# Create the txt folder to store downloaded wikipages
mkdir -p "$WORKDIR/txt"
cd "$WORKDIR"

# Download the wikipages if not already present
echo "Downloading enwik9.zip..."
curl -L -o enwik9.zip https://mattmahoney.net/dc/enwik9.zip


# Unzip the file
echo "Extracting enwik9.zip..."
unzip -oq enwik9.zip

# Move enwik9 to the txt folder
if [ -f "enwik9" ]; then
    mv enwik9 txt/enwik9.txt
else
    echo "Error: enwik9 file not found after unzip."
    exit 1
fi

# Remove enwik9.zip that is unnecessary now
rm -f enwik9.zip

# Run the wordcount.py file
echo "Running word count using Python..."
python3 wordcount.py
```

Reference

[1] Python documentation, *multiprocessing — Process-based parallelism*, https://docs.python.org/3/library/multiprocessing.html

Annex

Wordcount.py

```python
import string
import collections
import itertools
import time
import operator
import glob
import multiprocessing as mp

class SimpleMapReduce(object):
    def __init__(self, map_func, reduce_func, num_processes=None):
        """
        map_func
            Function to map inputs to intermediate data. Takes as
            argument one input value and returns a tuple with the key
            and a value to be reduced.
        reduce_func
            Function to reduce partitioned version of intermediate data
            to final output. Takes as argument a key as produced by
            map_func and a sequence of the values associated with that
            key.
        """
        self.map_func = map_func
        self.reduce_func = reduce_func
        self.num_workers = num_processes

    def partition(self, mapped_values):
        """Organize the mapped values by their key.
        Returns an unsorted sequence of tuples with a key and a sequence of
values.
        """
        partitioned_data = collections.defaultdict(list)
        for key, value in mapped_values:
            partitioned_data[key].append(value)
        return partitioned_data.items()

    def __call__(self, inputs):
        """Process the inputs through the map and reduce functions given.
        inputs
        An iterable containing the input data to be processed.
        """
        map_responses = map(self.map_func, inputs)
        partitioned_data = self.partition(itertools.chain(*map_responses))

        with mp.Pool(self.num_workers) as pool:
            reduced_values = pool.map(self.reduce_func, partitioned_data)

        return reduced_values
```

```python
def file_to_words(filename):
    """Read a file and return a sequence of (word, occurances) values.
    """
    STOP_WORDS = set([
    'a', 'an', 'and', 'are', 'as', 'be', 'by', 'for', 'if', 'in',
    'is', 'it', 'of', 'or', 'py', 'rst', 'that', 'the', 'to', 'with',
    ])
    TR = "".maketrans(string.punctuation, ' ' * len(string.punctuation))
    print('reading', filename)
    output = []
    with open(filename, 'rt', errors='replace') as f:
        for line in f:
            if line.lstrip().startswith('..'): # Skip rst comment lines
                continue
            line = line.translate(TR) # Strip punctuation
            for word in line.split():
                word = word.lower()
                if word.isalpha() and word not in STOP_WORDS:
                    output.append( (word, 1) )
    print('Done', filename)
    return output

def count_words(item):
    """Convert the partitioned data for a word to a
    tuple containing the word and the number of occurances.
    """
    word, occurances = item
    return (word, sum(occurances))



if __name__ == '__main__':
    start_time = time.time()
    input_files = glob.glob('txt/*')
    mapper = SimpleMapReduce(file_to_words, count_words, num_processes=None) #None
number of processes will all prossible cores
    word_counts = mapper(input_files)
    #print(list(word_counts))
    word_counts = sorted(word_counts, key=operator.itemgetter(1))
    word_counts.reverse()
    print('\nTOP 20 WORDS BY FREQUENCY\n')
    top20 = word_counts[0:20]
    longest = max(len(word) for word, count in top20)
    i = 1
    for word, count in top20:
        print('%s.\t%-*s: %5s' % (i, longest+1, word, count))
    i = i + 1
    end_time = time.time()
    elapsed_time = end_time - start_time
    print("Elapsed Time: {} seconds".format(elapsed_time))
```