HU Extension Assignment 11 E-90 Cloud Computing

Handed out: 11/15/2015 Due by 11:59 PM on Friday, 11/20/2015

Place all of your narratives and illustrations in a single Word or PDF document named E90_LastNameFirstNameHW11.docx [.pdf]. Use this assignment as the initial template. Please implement code solutions as a separate class in one (Java, C#, Ruby, Python,..) project. Add description of your steps and your code below problem statements used for each problem.
Upload your homework file and your working code (e.g., Filename.java) into your Assignment 11 folder. Do not include executables. Please also do not zip your files.

Please use any programming language you are comfortable with.

Problem 1: Implement the Monte Carlo algorithm for determining the value of number *Π* (pi) as described on pages 12 to 14 of the lecture slides on Parallel Programming. Make sure that your code accepts as input parameters a seed for your random number generator and the number of darts you are shooting on the square. Once you have the basic algorithm working organize execution of your code in a master-worker arrangement. The master would provide the worker with the seed value for the random number generator and the number of dart throws. The worker would return an approximate value of *Π*. In a parallel universe the master would instantiate 10 or 100 workers, instruct each of them to do the work, collect results and perform final aggregation of those results. Do not attempt to do that. Simply have your master repeatedly call the same worker 10 or 100 times. Compare the error of the aggregate value produced by the "final" master's result with the error each worker made. Experiment with the number of dart throws to judge how big that number has to be to achieve 1% accuracy.

Total Points: 50

```
import java.util.Random;

public class PiCalculator {
	/*
	 * The Worker thread. Take a random seed and number of darts as input
	 * parameters. Calculate an approximate value of Pi and return it to
	 * Master thread.
	 */
	private double workerThread(long randomSeed, int numOfDarts) {
		int countInCircle = 0;

		Random random = new Random(randomSeed);

		for (int i = 0; i < numOfDarts; i++) {
			double x = random.nextDouble();
			double y = random.nextDouble();

			double val = x * x + y * y;
			if (val < 1.0) {
				countInCircle++;
			}
		}

		double pi = (double)countInCircle / (double)numOfDarts * 4;
		return pi;
	}

	/*
	 * The Master thread. Take a random seed, number of darts and number of workers
	 * as input parameters. Initiate a number of worker threads to work and collect
	 * the results from them. Aggregate the results and calculate the error rate and
```

```
         * accuracy.
         */
        private double masterThread(long randomSeed, int numOfDarts, int numOfWorkers) {
                double sumPi = 0.0;
                double sumDiff = 0.0;

                Random random = new Random(randomSeed);

                for (int i = 0; i < numOfWorkers; i++) {
                        // Generate a random seed for each worker. Otherwise they'll get
                        // the same results (Pi value) with the same seed.
                        int workerRandomSeed = random.nextInt();

                        // Initiate the worker threads. Gather the value of Pi.
                        double CalculatedPi = workerThread(workerRandomSeed, numOfDarts);
                        System.out.println("The calculated Pi value is: " + CalculatedPi);
                        sumPi += CalculatedPi;

                        // Calculate the error rate for each worker thread.
                        double diffInPercent = (Math.PI - CalculatedPi) / Math.PI * 100;
                        System.out.println("The calculated diff in percentage is: " + diffInPercent);
                        sumDiff += diffInPercent;
                }

                double meanPi = sumPi / numOfWorkers;
                System.out.println("\nThe calculated mean Pi value is: " + meanPi);

                // Two ways to calculate the mean difference. Another way can be:
                // double meanDiffInPercent = Math.abs(sumDiff / numOfWorkers);
                // They get the same results.
                double meanDiffInPercent = Math.abs((Math.PI - meanPi) / Math.PI * 100);
                System.out.println("\nMean difference in percentage: " + meanDiffInPercent +"\n");

                return meanDiffInPercent;
        }

        public static void main(String[] args) {
                long randomSeed = 0;
                int numOfDarts = 0;
                int numOfWorkers = 0;

                if(args.length == 0) {
                        System.out.println("Usage: ./CalculatePi <randomSeed> <numOfDarts> <numOfWorkers>");
                        System.exit(0);
                } else {
                        randomSeed = Long.parseLong(args[0]);
                        numOfDarts = Integer.parseInt(args[1]);
                        numOfWorkers = Integer.parseInt(args[2]);
                        System.out.println("Cmd: ./CalculatePi " + randomSeed + " " + numOfDarts + " " +
numOfWorkers + "\n");
                }

                PiCalculator piCal = new PiCalculator();
                // Initiate the master thread.
                double meanDiffInPercent = piCal.masterThread(randomSeed, numOfDarts, numOfWorkers);

                if (meanDiffInPercent < 1.0) {
                        System.out.println("Achieve 1% accuracy!");
                } else {
                        System.out.println("Fail to achieve 1% accuracy!");
                }
        }
}
```

Output from the console:

```
Cmd: ./CalculatePi 42 80 60

The calculated Pi value is: 2.8
The calculated diff in percentage is: 10.873231868538614
The calculated Pi value is: 3.15
The calculated diff in percentage is: -0.2676141478940626
The calculated Pi value is: 3.05
The calculated diff in percentage is: 2.915484713943847
The calculated Pi value is: 3.35
The calculated diff in percentage is: -6.633811871569882
The calculated Pi value is: 3.05
The calculated diff in percentage is: 2.915484713943847
The calculated Pi value is: 3.3
The calculated diff in percentage is: -5.04226244065092
The calculated Pi value is: 3.1
The calculated diff in percentage is: 1.3239352830248852
The calculated Pi value is: 2.85
The calculated diff in percentage is: 9.281682437619653
The calculated Pi value is: 3.1
The calculated diff in percentage is: 1.3239352830248852
The calculated Pi value is: 3.05
The calculated diff in percentage is: 2.915484713943847
The calculated Pi value is: 2.95
The calculated diff in percentage is: 6.098583575781743
The calculated Pi value is: 3.2
The calculated diff in percentage is: -1.8591635788130243
The calculated Pi value is: 3.25
The calculated diff in percentage is: -3.4507130097319725
The calculated Pi value is: 2.9
The calculated diff in percentage is: 7.690133006700704
The calculated Pi value is: 3.15
The calculated diff in percentage is: -0.2676141478940626
The calculated Pi value is: 3.4
The calculated diff in percentage is: -8.22536130248883
The calculated Pi value is: 3.45
The calculated diff in percentage is: -9.816910733407791
The calculated Pi value is: 3.1
The calculated diff in percentage is: 1.3239352830248852
The calculated Pi value is: 3.2
The calculated diff in percentage is: -1.8591635788130243
The calculated Pi value is: 2.8
The calculated diff in percentage is: 10.873231868538614
The calculated Pi value is: 3.05
The calculated diff in percentage is: 2.915484713943847
The calculated Pi value is: 3.2
The calculated diff in percentage is: -1.8591635788130243
The calculated Pi value is: 3.0
The calculated diff in percentage is: 4.507034144862795
The calculated Pi value is: 3.05
The calculated diff in percentage is: 2.915484713943847
The calculated Pi value is: 3.05
The calculated diff in percentage is: 2.915484713943847
The calculated Pi value is: 3.45
The calculated diff in percentage is: -9.816910733407791
The calculated Pi value is: 3.15
The calculated diff in percentage is: -0.2676141478940626
The calculated Pi value is: 3.3
The calculated diff in percentage is: -5.04226244065092
The calculated Pi value is: 3.3
The calculated diff in percentage is: -5.04226244065092
The calculated Pi value is: 3.2
The calculated diff in percentage is: -1.8591635788130243
```

```
The calculated Pi value is: 3.15
The calculated diff in percentage is: -0.2676141478940626
The calculated Pi value is: 2.85
The calculated diff in percentage is: 9.281682437619653
The calculated Pi value is: 3.0
The calculated diff in percentage is: 4.507034144862795
The calculated Pi value is: 3.1
The calculated diff in percentage is: 1.3239352830248852
The calculated Pi value is: 3.0
The calculated diff in percentage is: 4.507034144862795
The calculated Pi value is: 3.3
The calculated diff in percentage is: -5.04226244065092
The calculated Pi value is: 2.95
The calculated diff in percentage is: 6.098583575781743
The calculated Pi value is: 3.25
The calculated diff in percentage is: -3.4507130097319725
The calculated Pi value is: 3.0
The calculated diff in percentage is: 4.507034144862795
The calculated Pi value is: 3.0
The calculated diff in percentage is: 4.507034144862795
The calculated Pi value is: 3.35
The calculated diff in percentage is: -6.633811871569882
The calculated Pi value is: 3.3
The calculated diff in percentage is: -5.04226244065092
The calculated Pi value is: 2.95
The calculated diff in percentage is: 6.098583575781743
The calculated Pi value is: 3.35
The calculated diff in percentage is: -6.633811871569882
The calculated Pi value is: 3.35
The calculated diff in percentage is: -6.633811871569882
The calculated Pi value is: 3.25
The calculated diff in percentage is: -3.4507130097319725
The calculated Pi value is: 3.55
The calculated diff in percentage is: -13.000009595245688
The calculated Pi value is: 3.3
The calculated diff in percentage is: -5.04226244065092
The calculated Pi value is: 3.35
The calculated diff in percentage is: -6.633811871569882
The calculated Pi value is: 3.3
The calculated diff in percentage is: -5.04226244065092
The calculated Pi value is: 3.25
The calculated diff in percentage is: -3.4507130097319725
The calculated Pi value is: 3.3
The calculated diff in percentage is: -5.04226244065092
The calculated Pi value is: 3.0
The calculated diff in percentage is: 4.507034144862795
The calculated Pi value is: 3.2
The calculated diff in percentage is: -1.8591635788130243
The calculated Pi value is: 3.15
The calculated diff in percentage is: -0.2676141478940626
The calculated Pi value is: 2.95
The calculated diff in percentage is: 6.098583575781743
The calculated Pi value is: 3.0
The calculated diff in percentage is: 4.507034144862795
The calculated Pi value is: 3.2
The calculated diff in percentage is: -1.8591635788130243
The calculated Pi value is: 3.15
The calculated diff in percentage is: -0.2676141478940626
The calculated Pi value is: 3.2
The calculated diff in percentage is: -1.8591635788130243

The calculated mean Pi value is: 3.15

Mean difference in percentage: 0.2676141478940626
```

Achieve 1% accuracy!

I used 60 worker threads and each worker generate 80 darts (4800 darts in total). The mean difference is 0.26%. I tried some other combinations:

Random seed	Num of worker threads	Darts for each worker	Darts in total	Accuracy (%)
42	100	20	2000	0.68
42	200	10	2000	0.75
42	120	15	1800	0.97
1231314	120	15	1800	0.02
124	120	15	1800	1.11

So the accuracy depends on the random seed, number of worker threads and the darts for each worker. It needs at least around 2000 in total to get 1% accuracy.

Problem 2: Implement algorithm for counting words in a text described on page 44
with two functions:

a) map() that would take a file with a page of text in plain .txt format and produce a collection of name value pairs where the name is the word itself, and the value is number 1.
b) reduce() that would take any collection of (word, 1) pairs and aggregate it into an ordered word counts (word, count) collection.

Your code will need a master role as well. Namely, the master would fetch a file from the file system, pass it to the map() function, receive the results of map() calculation and pass them to the reduce() function.
Experiment with several pages of random text you copy from the internet.
Find an automated way to verify your results.

Total Points: 50

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.io.BufferedReader;

public class WordCounter {

	String delimiters = "[\\s\\t,;\\.\\?!-:@\\[\\]\\(\\)\\{\\}_\\*/=\\\"'\\d\\+\\^\\$]+";

	private List<HashMap<String, Integer>> mapper(String filename) {
		File file = new File(filename);

		String line = null;
		List<HashMap<String, Integer>> wordList = new LinkedList<HashMap<String, Integer>>();

		try {
			// Read the input file.
			BufferedReader br = new BufferedReader(new FileReader(file));
```

```
                // Divide each sentence into a list of words.
                while ((line = br.readLine()) != null) {
                        String[] words = line.split(delimiters);

                        for (String word : words) {
                                // Remove leading and trailing whitespace.
                                word = word.trim();
                                // Covert the word to lower case
                                word = word.toLowerCase();

                                if (word.isEmpty()) {
                                        continue;
                                }

                                // Produce the name value pairs, where the name is the word itself,
                                // and the value is 1. Like (word, 1)
                                HashMap<String, Integer> pair = new HashMap<String, Integer>();
                                pair.put(word, 1);

                                wordList.add(pair);
                        }
                }

                br.close();

        } catch (FileNotFoundException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }

        return wordList;
}

private List<Map.Entry<String, Integer>> reducer(List<HashMap<String, Integer>> wordList) {
        HashMap<String, Integer> reducedPairs = new HashMap<String, Integer>();

        // Take the (word, 1) pairs and aggregate it into the word counts (word, count).
        for (int i = 0; i < wordList.size(); i++) {
                HashMap<String, Integer> pair = wordList.get(i);
                // Each HashMap only have one entry here. Get the key value pair.
                Entry<String, Integer> entry = pair.entrySet().iterator().next();

                String word = entry.getKey();
                int val = entry.getValue();

                // Aggregate to (word, count) pairs.
                if (reducedPairs.containsKey(word)) {
                        reducedPairs.put(word, reducedPairs.get(word) + val);
                } else {
                        reducedPairs.put(word, val);
                }
        }

        // Sort all the entries based on its occurence. If the words have the same occurance,
        // sort them alphabetically.
        List<Map.Entry<String, Integer>> sortedEntries = sortHashMap(reducedPairs);

        return sortedEntries;
}

// Sort the name value pairs.
private List<Map.Entry<String, Integer>> sortHashMap(HashMap<String, Integer> hashmap) {
 List<Map.Entry<String, Integer>> sortedEntries = new LinkedList<Map.Entry<String,
```

```
Integer>>(hashmap.entrySet());

        Collections.sort(sortedEntries, new Comparator<Map.Entry<String, Integer>>() {
                public int compare(Map.Entry<String, Integer> left, Map.Entry<String, Integer> right) {
                        if (left.getValue() != right.getValue()) {
                                return right.getValue() - left.getValue();
                        }

                        return left.getKey().compareTo(right.getKey());
                }
        });

        return sortedEntries;
        }

        private void printHashMapList(List<HashMap<String, Integer>> list) {
                for (int i = 0; i < list.size(); i++) {
                        HashMap<String, Integer> hashmap = list.get(i);
                        Iterator<Entry<String, Integer>> iterator = hashmap.entrySet().iterator();
                        while (iterator.hasNext()) {
                                printHashMapEntry(iterator.next());
                        }
                }
        }

        private void printHashEntryList(List<Map.Entry<String, Integer>> list) {
                for (int i = 0; i < list.size(); i++) {
                        printHashMapEntry(list.get(i));
                }
        }

        private void printHashMapEntry(Map.Entry<String, Integer> entry) {
                System.out.println("(" + entry.getKey() + ", " + entry.getValue() + ") ");
        }

        public static void main (String[] args) {
                String filename = "/Users/hqiu/Documents/workspace/WordCounter/src/input2.txt";

                WordCounter wordCounter = new WordCounter();

                List<HashMap<String, Integer>> wordList = wordCounter.mapper(filename);
                System.out.println("The output pairs from the Mapper function are: \n");
                wordCounter.printHashMapList(wordList);

                List<Map.Entry<String, Integer>> sortedEntries = wordCounter.reducer(wordList);
                System.out.println("\n\nThe output pairs from the Reducer function are: \n");
                wordCounter.printHashEntryList(sortedEntries);
        }
}
```

Output from the console:

```
The output pairs from the Mapper function are:

(cloud, 1) (computing, 1) (also, 1) (known, 1) (as, 1) (on, 1) (demand, 1) (computing, 1) (is, 1) (a, 1) (kind,
1) (of, 1) (internet, 1) (based, 1) (computing, 1) (where, 1) (shared, 1) (resources, 1) (and, 1) (information,
1) (are, 1) (provided, 1) (to, 1) (computers, 1) (and, 1) (other, 1) (devices, 1) (on, 1) (demand, 1) (it, 1)
(is, 1) (a, 1) (model, 1) (for, 1) (enabling, 1) (ubiquitous, 1) (on, 1) (demand, 1) (access, 1) (to, 1) (a, 1)
(shared, 1) (pool, 1) (of, 1) (configurable, 1) (computing, 1) (resources, 1) (cloud, 1) (computing, 1) (and, 1)
(storage, 1) (solutions, 1) (provide, 1) (users, 1) (and, 1) (enterprises, 1) (with, 1) (various, 1)
(capabilities, 1) (to, 1) (store, 1) (and, 1) (process, 1) (their, 1) (data, 1) (in, 1) (third, 1) (party, 1)
(data, 1) (centers, 1) (it, 1) (relies, 1) (on, 1) (sharing, 1) (of, 1) (resources, 1) (to, 1) (achieve, 1)
(coherence, 1) (and, 1) (economies, 1) (of, 1) (scale, 1) (similar, 1) (to, 1) (a, 1) (utility, 1) (like, 1)
(the, 1) (electricity, 1) (grid, 1) (over, 1) (a, 1) (network, 1) (at, 1) (the, 1) (foundation, 1) (of, 1)
```

(cloud, 1) (computing, 1) (is, 1) (the, 1) (broader, 1) (concept, 1) (of, 1) (converged, 1) (infrastructure, 1) (and, 1) (shared, 1) (services, 1) (cloud, 1) (computing, 1) (or, 1) (in, 1) (simpler, 1) (shorthand, 1) (just, 1) (the, 1) (cloud, 1) (also, 1) (focuses, 1) (on, 1) (maximizing, 1) (the, 1) (effectiveness, 1) (of, 1) (the, 1) (shared, 1) (resources, 1) (cloud, 1) (resources, 1) (are, 1) (usually, 1) (not, 1) (only, 1) (shared, 1) (by, 1) (multiple, 1) (users, 1) (but, 1) (are, 1) (also, 1) (dynamically, 1) (reallocated, 1) (per, 1) (demand, 1) (this, 1) (can, 1) (work, 1) (for, 1) (allocating, 1) (resources, 1) (to, 1) (users, 1) (for, 1) (example, 1) (a, 1) (cloud, 1) (computer, 1) (facility, 1) (that, 1) (serves, 1) (european, 1) (users, 1) (during, 1) (european, 1) (business, 1) (hours, 1) (with, 1) (a, 1) (specific, 1) (application, 1) (e, 1) (g, 1) (email, 1) (may, 1) (reallocate, 1) (the, 1) (same, 1) (resources, 1) (to, 1) (serve, 1) (north, 1) (american, 1) (users, 1) (during, 1) (north, 1) (america, 1) (s, 1) (business, 1) (hours, 1) (with, 1) (a, 1) (different, 1) (application, 1) (e, 1) (g, 1) (a, 1) (web, 1) (server, 1) (this, 1) (approach, 1) (helps, 1) (maximize, 1) (the, 1) (use, 1) (of, 1) (computing, 1) (power, 1) (while, 1) (reducing, 1) (the, 1) (overall, 1) (cost, 1) (of, 1) (resources, 1) (by, 1) (using, 1) (less, 1) (power, 1) (air, 1) (conditioning, 1) (rack, 1) (space, 1) (etc, 1) (to, 1) (maintain, 1) (the, 1) (system, 1) (with, 1) (cloud, 1) (computing, 1) (multiple, 1) (users, 1) (can, 1) (access, 1) (a, 1) (single, 1) (server, 1) (to, 1) (retrieve, 1) (and, 1) (update, 1) (their, 1) (data, 1) (without, 1) (purchasing, 1) (licenses, 1) (for, 1) (different, 1) (applications, 1) (the, 1) (term, 1) (moving, 1) (to, 1) (cloud, 1) (also, 1) (refers, 1) (to, 1) (an, 1) (organization, 1) (moving, 1) (away, 1) (from, 1) (a, 1) (traditional, 1) (capex, 1) (model, 1) (buy, 1) (the, 1) (dedicated, 1) (hardware, 1) (and, 1) (depreciate, 1) (it, 1) (over, 1) (a, 1) (period, 1) (of, 1) (time, 1) (to, 1) (the, 1) (opex, 1) (model, 1) (use, 1) (a, 1) (shared, 1) (cloud, 1) (infrastructure, 1) (and, 1) (pay, 1) (as, 1) (one, 1) (uses, 1) (it, 1) (dubious, 1) (discuss, 1)

The output pairs from the Reducer function are:

(a, 13) (the, 13) (to, 12) (and, 10) (cloud, 10) (of, 10) (computing, 9) (resources, 8) (shared, 6) (users, 6) (on, 5) (also, 4) (demand, 4) (for, 4) (it, 4) (with, 4) (are, 3) (data, 3) (is, 3) (model, 3) (access, 2) (application, 2) (as, 2) (business, 2) (by, 2) (can, 2) (different, 2) (during, 2) (e, 2) (european, 2) (g, 2) (hours, 2) (in, 2) (infrastructure, 2) (moving, 2) (multiple, 2) (north, 2) (over, 2) (power, 2) (server, 2) (their, 2) (this, 2) (use, 2) (achieve, 1) (air, 1) (allocating, 1) (america, 1) (american, 1) (an, 1) (applications, 1) (approach, 1) (at, 1) (away, 1) (based, 1) (broader, 1) (but, 1) (buy, 1) (capabilities, 1) (capex, 1) (centers, 1) (coherence, 1) (computer, 1) (computers, 1) (concept, 1) (conditioning, 1) (configurable, 1) (converged, 1) (cost, 1) (dedicated, 1) (depreciate, 1) (devices, 1) (discuss, 1) (dubious, 1) (dynamically, 1) (economies, 1) (effectiveness, 1) (electricity, 1) (email, 1) (enabling, 1) (enterprises, 1) (etc, 1) (example, 1) (facility, 1) (focuses, 1) (foundation, 1) (from, 1) (grid, 1) (hardware, 1) (helps, 1) (information, 1) (internet, 1) (just, 1) (kind, 1) (known, 1) (less, 1) (licenses, 1) (like, 1) (maintain, 1) (maximize, 1) (maximizing, 1) (may, 1) (network, 1) (not, 1) (one, 1) (only, 1) (opex, 1) (or, 1) (organization, 1) (other, 1) (overall, 1) (party, 1) (pay, 1) (per, 1) (period, 1) (pool, 1) (process, 1) (provide, 1) (provided, 1) (purchasing, 1) (rack, 1) (reallocate, 1) (reallocated, 1) (reducing, 1) (refers, 1) (relies, 1) (retrieve, 1) (s, 1) (same, 1) (scale, 1) (serve, 1) (serves, 1) (services, 1) (sharing, 1) (shorthand, 1) (similar, 1) (simpler, 1) (single, 1) (solutions, 1) (space, 1) (specific, 1) (storage, 1) (store, 1) (system, 1) (term, 1) (that, 1) (third, 1) (time, 1) (traditional, 1) (ubiquitous, 1) (update, 1) (uses, 1) (using, 1) (usually, 1) (utility, 1) (various, 1) (web, 1) (where, 1) (while, 1) (without, 1) (work, 1)

Validate the results through python script:

```
from collections import Counter
import re

def compareItems((w1,c1), (w2,c2)):
    if c1 != c2:
        return c2 - c1
    else:
        return cmp(w1, w2)

filename = '/Users/hqiu/Documents/workspace/WordCounter/src/input.txt'

words = re.findall(r'\w+', open(filename).read().lower())

digits = []
for word in words:
       if word.isdigit():
              digits.append(word)

for digit in digits:
```

```
        words.remove(digit)

counts = Counter(words)

items = counts.items()
items.sort(compareItems)

print(items)
```

Results from the python code:

```
hqiu@bos-mpdei>> python WordFreqCounter.py
[('a', 13), ('the', 13), ('to', 12), ('and', 10), ('cloud', 10), ('of', 10), ('computing', 9), ('resources', 8),
('shared', 6), ('users', 6), ('on', 5), ('also', 4), ('demand', 4), ('for', 4), ('it', 4), ('with', 4), ('are',
3), ('data', 3), ('is', 3), ('model', 3), ('access', 2), ('application', 2), ('as', 2), ('business', 2), ('by',
2), ('can', 2), ('different', 2), ('during', 2), ('e', 2), ('european', 2), ('g', 2), ('hours', 2), ('in', 2),
('infrastructure', 2), ('moving', 2), ('multiple', 2), ('north', 2), ('over', 2), ('power', 2), ('server', 2),
('their', 2), ('this', 2), ('use', 2), ('achieve', 1), ('air', 1), ('allocating', 1), ('america', 1),
('american', 1), ('an', 1), ('applications', 1), ('approach', 1), ('at', 1), ('away', 1), ('based', 1),
('broader', 1), ('but', 1), ('buy', 1), ('capabilities', 1), ('capex', 1), ('centers', 1), ('coherence', 1),
('computer', 1), ('computers', 1), ('concept', 1), ('conditioning', 1), ('configurable', 1), ('converged', 1),
('cost', 1), ('dedicated', 1), ('depreciate', 1), ('devices', 1), ('discuss', 1), ('dubious', 1),
('dynamically', 1), ('economies', 1), ('effectiveness', 1), ('electricity', 1), ('email', 1), ('enabling', 1),
('enterprises', 1), ('etc', 1), ('example', 1), ('facility', 1), ('focuses', 1), ('foundation', 1), ('from', 1),
('grid', 1), ('hardware', 1), ('helps', 1), ('information', 1), ('internet', 1), ('just', 1), ('kind', 1),
('known', 1), ('less', 1), ('licenses', 1), ('like', 1), ('maintain', 1), ('maximize', 1), ('maximizing', 1),
('may', 1), ('network', 1), ('not', 1), ('one', 1), ('only', 1), ('opex', 1), ('or', 1), ('organization', 1),
('other', 1), ('overall', 1), ('party', 1), ('pay', 1), ('per', 1), ('period', 1), ('pool', 1), ('process', 1),
('provide', 1), ('provided', 1), ('purchasing', 1), ('rack', 1), ('reallocate', 1), ('reallocated', 1),
('reducing', 1), ('refers', 1), ('relies', 1), ('retrieve', 1), ('s', 1), ('same', 1), ('scale', 1), ('serve',
1), ('serves', 1), ('services', 1), ('sharing', 1), ('shorthand', 1), ('similar', 1), ('simpler', 1), ('single',
1), ('solutions', 1), ('space', 1), ('specific', 1), ('storage', 1), ('store', 1), ('system', 1), ('term', 1),
('that', 1), ('third', 1), ('time', 1), ('traditional', 1), ('ubiquitous', 1), ('update', 1), ('uses', 1),
('using', 1), ('usually', 1), ('utility', 1), ('various', 1), ('web', 1), ('where', 1), ('while', 1),
('without', 1), ('work', 1)]
```

The results are the same!

We can also check from the online tool:

http://www.writewords.org.uk/word_count.asp

23 the
15 and
13 a
12 mapreduce
10 of
7 is
6 in
5 reduce
5 for
5 by
5 as
4 with
4 that
4 name
4 model
4 map

4 implementation
4 framework
4 distributed
3 tolerance
3 to
3 such
3 programming
3 processing
3 on
3 not
3 fault
3 data
3 been
2 various
2 usually
2 threaded
2 their
2 system
2 students
2 sorting
2 since
2 queue
2 performs
2 parallel
2 optimizing
2 operation
2 only
2 method
2 into
2 have
2 has
2 google
2 functions

…..

I also tried some other inputs. The results are also correct.