MapReduce

Group 18

Date:2020-06-08

CONTENT

Chapter 1 Introduction

1.1 Background

1.1.1 MapReduce

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.

A MapReduce program is composed of a **map procedure**, which performs filtering and sorting (such as sorting students by first name into queues, one queue for each name), and a **reduce method**, which performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "**MapReduce System**" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for **redundancy** and **fault tolerance**.

The model is a specialization of the split-apply-combine strategy for data analysis. It is inspired by the map and reduce functions commonly used in **functional programming**, although their purpose in the MapReduce framework is not the same as in their original forms. The key contributions of the MapReduce framework are not the actual map and reduce functions, but the scalability and fault-tolerance achieved for a variety of applications by optimizing the execution engine. As such, a **single-threaded** implementation of MapReduce is usually not faster than a traditional (non-MapReduce) implementation; any gains are usually only seen with **multi-threaded** implementations on multi-processor hardware. The use of this model is beneficial only when the optimized distributed shuffle operation (which reduces network communication cost) and fault tolerance features of the MapReduce framework come into play. Optimizing the communication cost is essential to a good MapReduce algorithm.

MapReduce libraries have been written in many programming languages, with different levels of optimization. A popular open-source implementation that has support for distributed shuffles is part of **Apache Hadoop**.

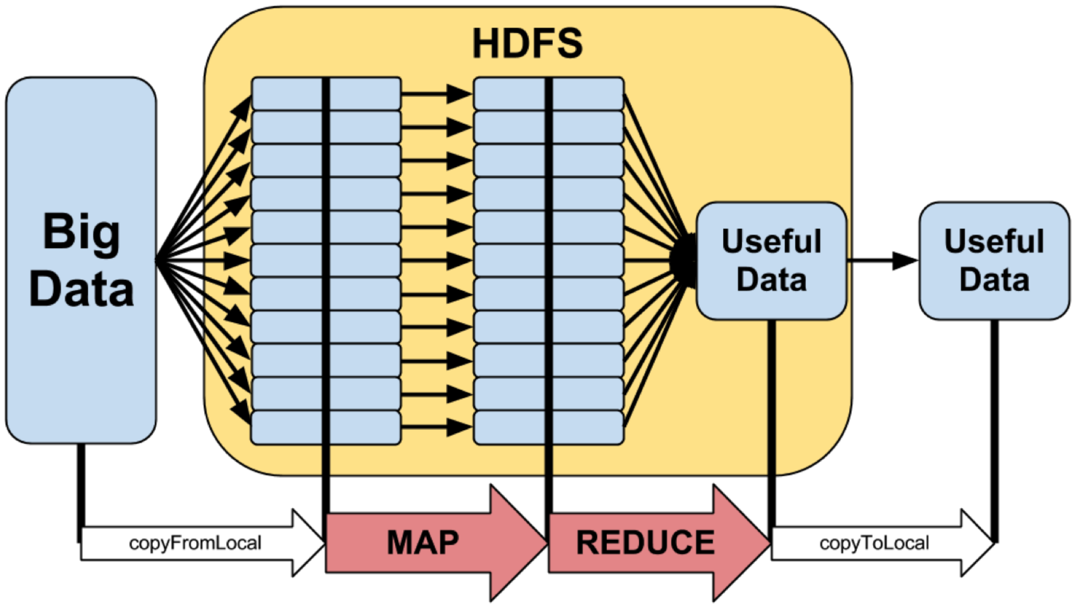


Figure 1 The overall MapReduce process

1.1.2 Apache Hadoop

Apache Hadoop is a collection of open-source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation. It provides a software framework for distributed storage and processing of big data using the MapReduce programming model. Originally designed for computer clusters built from commodity hardware—still the common use—it has also found use on clusters of higher-end hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurrences and should be automatically handled by the framework.

The core of Apache Hadoop consists of a storage part, known as **Hadoop Distributed File System (HDFS)**, and a processing part which is a MapReduce programming model. Hadoop splits files into large blocks and distributes them across nodes in a cluster. It then transfers packaged code into nodes to process the data in parallel. This approach takes advantage of data locality, where nodes manipulate the data they have access to. This allows the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking.

The base Apache Hadoop framework is composed of the following modules:

- *Hadoop Common* – contains libraries and utilities needed by other Hadoop modules;
- *Hadoop Distributed File System (HDFS)* – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster;
- *Hadoop YARN* – a platform responsible for managing computing resources in clusters and using them for scheduling users' applications;
- *Hadoop MapReduce* – an implementation of the MapReduce programming model for large-scale data processing.

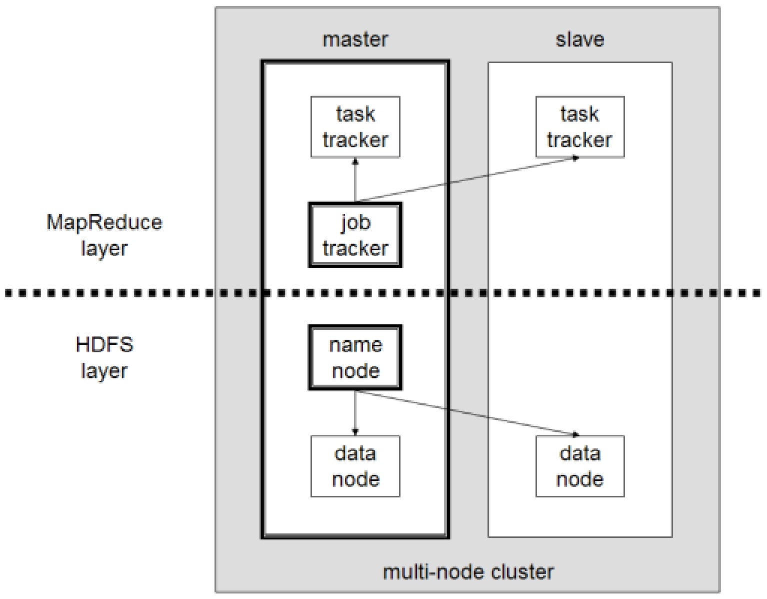


Figure 2 A multi-node Hadoop cluster

1.2 Problem Description

In this project, we are required to introduce the framework of MapReduce, explain how does it work, and implement a MapReduce program to count the number of occurrences of each word in a set of documents.

Our task includes the following steps:

1. Setup MapReduce libraries. The Hadoop framework itself is mostly written in the Java programming language. Therefore, we choose **Java** and the popular open-source implementation **Apache Hadoop** to program.

2. Implement a **parallel MapReduce** program and a **serial** program to print the number of occurrences of words in non-increasing order. If two or more words have the same number of occurrences, they must be printed in lexicographical order. Morever, each line contains one word, followed by its number of occurrences, separated by a space, and there must be no extra space at the end of each line.

   The MapReduce framework is composed of three operations:

   - **Map**: each worker node applies the map function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of the redundant input data is processed.
   - **Shuffle**: worker nodes redistribute data based on the output keys (produced by the map function), such that all data belonging to one key is located on the same worker node.
   - **Reduce**: worker nodes now process each group of output data, per key, in parallel.
3. Write a test of performance program to test the correctness and time of the program. We are to generate a set of documents which contains 1,000-1,000,000 words to test the time and split 1,000,000 words to 1-128 files to test the time preformance of each program.
4. Test the programs, then compare and plot the performances between parallel and serial algorithms. Analyze the influence of hardware, the scale of set of documents, and setting of Hadoop (the number of Map Task and Reduce Task) in the performance.

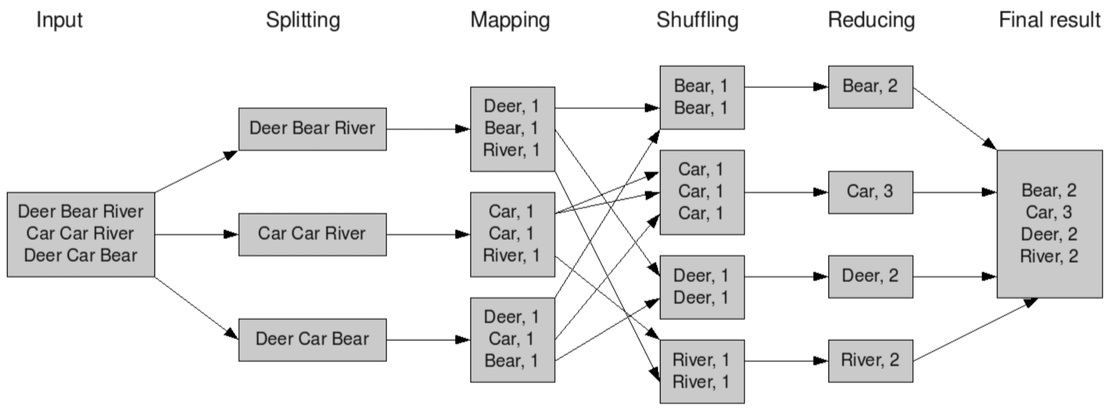


Figure 3 The overall MapReduce word count process

Chapter 2 Algorithm Specification

2.1 Parallel MapReduce Program

The MapReduce framework operates exclusively on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

The key and value classes have to be serializable by the framework and hence need to implement the Writable interface. Additionally, the key classes have to implement the WritableComparable interface to facilitate sorting by the framework.

Input and Output types of a MapReduce job:

> (input) <k1, v1> -> **map** -> <k2, v2> -> **combine** -> <k2, v2> -> **reduce** -> <k3, v3> (output)

The data structure of the class **WordCountMapper** and **WordCountReducer** is:

```
      /**
       * Object        : the content of input file
       * Text           : every single line of input data
       * Text           : output the type of key
       * IntWritable : output the type of value
       */
      private static class WordCountMapper extends Mapper<Object, Text, Text, IntWritable> {
            @Override
            protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {
                  StringTokenizer itr = new StringTokenizer(value.toString());
                  while (itr.hasMoreTokens()) {
                        context.write(new Text(itr.nextToken()), new IntWritable(1));
                  }
            }
      }

      /**
       * Text             :  Mapper the input key
       * IntWritable  :  Mapper the input value
       * Text             :  Reducer the output key
       * IntWritable  :  Reducer the output value
       */
      private static class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
            @Override
            protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
                  int count = 0;
                  for (IntWritable item : values) {
                        count += item.get();
                  }
                  context.write(key, new IntWritable(count));
            }
      }
```

The pseudocode of the parallel program is:

```
void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {
            // create configuration
            new Configuration();
            // set job of hadoop jobName is WordCount
            new job();
            // set jar
            setJarByClass();
            // set Mapper's class
            setMapperClass();
            // set Reducer's class
            setReducerClass();
            // set the type of key and value outputted
            setOutputKeyClass();
            setOutputValueClass();

            // set input & output path
            Input(new Path(args[0]));
            OutputPath(new Path(args[1]));
            // exit after executing job
            if job is completed
            exit(0);
            else
            exit(1);
    }
```

2.2 Serial Program

The serial program reads word from data separated by space or newline until the end of file, count the words' occurrence and sort it in occurrence and lexicographical, and then output the result.

The data structure of the class **WordCount** is:

```
//WordCount contains the main class
class WordCount{
public:
      explicit WordCount(string new_word) { word = std::move(new_word); num = 1; };
      string get_word() const { return word; }                //return the string of wht word
      int get_number() const { return num; }          //return the frequency of the word
      void add_number() { num++; }                        //add one more word
private:
      string word;
      int num;
};
```

The pseudocode of the serial program is:

```
int main (int argc, char* argv[]){
      //start traverse the input files
      for i := 1 to argc-1
            //get the input stream
            input_file();
            while getline(input_file, temp_line) is true
                  //deliver one word each time
                  while temp_line -> temp_word
                        //check duplication
                        if word_count.empty() is true
                              Add WordCount;
                              continue;
                        end if
                        for j := 0 to word_count.size()-1
                              if get word is true
                              //if find the word, then add one more time
                                    add_number();
                                    break;
                              end if
                              else
                              //otherwise, create a new item for the vector
                                    new_word(temp_word)
                                    break;
                              end else
                              j:=j+1;
                        end for
                end while
            end while
            close output_file;
            i:=i+1;
      end for

      //sort the vector
      sort(word_count.begin(), word_count.end(), compare);

      //finally traverse all the words in the vector and put them into the result files
      output_file("");
      write in output_file;
      close output_file;
}
```

Chapter 3 Testing Samples and Results

3.1 Correctness Test

3.1.1 Case 1

> a simple input case with one word appreciating a lot of times.

```
cat cat dog cat bear cat cat cat bear
[Result]
cat     6
bear    2
dog     1
```

3.1.2 Case 2

> totally empty

```
NULL
[Result]
NULL
```

3.1.3 Case 3

> some words include numbers and some begin as numbers which is not a word.

```
11cats cats123 cats123
aacats
[Result]
cats123     2
aacats      1
```

3.1.4 Case 4

> same word printed in lower case as well as capital letters randomly.

```
GOOD Morning
GooD mOrNing
good MORNING
gOOd morNINg
[Result]
good        4
morning     4
```

3.1.5 Case 5

> Mix with symbols

```
&&Friday(Monday)**CCAt#@Tuesdat$$
[Result]
friday      1
monday      1
ccat        1
tuesdat     1
```

3.1.6 Case 6

> a normal article with about 300 words.

```
refer to the file test.assets
```

3.1.7 Case 7

> a large set of input case

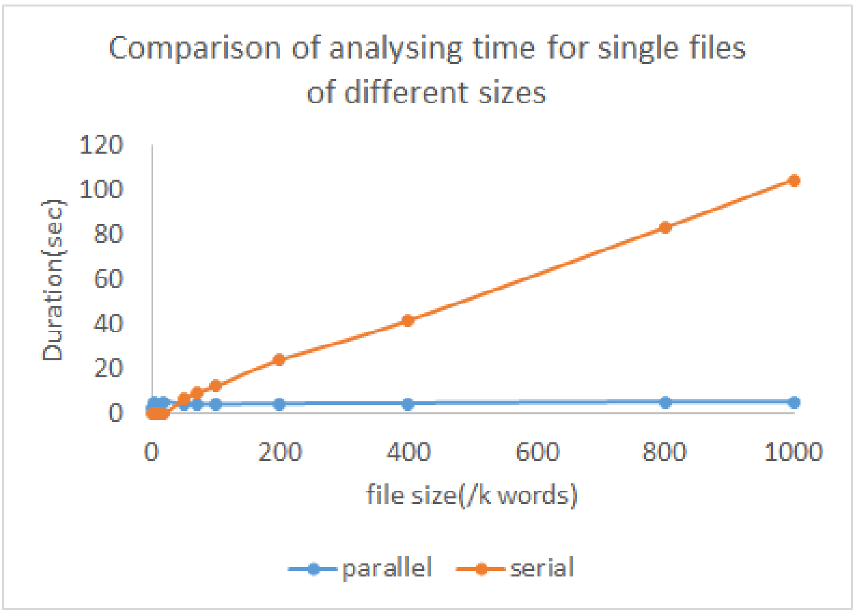
```
refer to the file test.assets
```

3.2 Time Test

3.2.1 Methods 1

Enter the same text for both serial and parallel programs, and test the time required for them to run individual texts of different sizes.

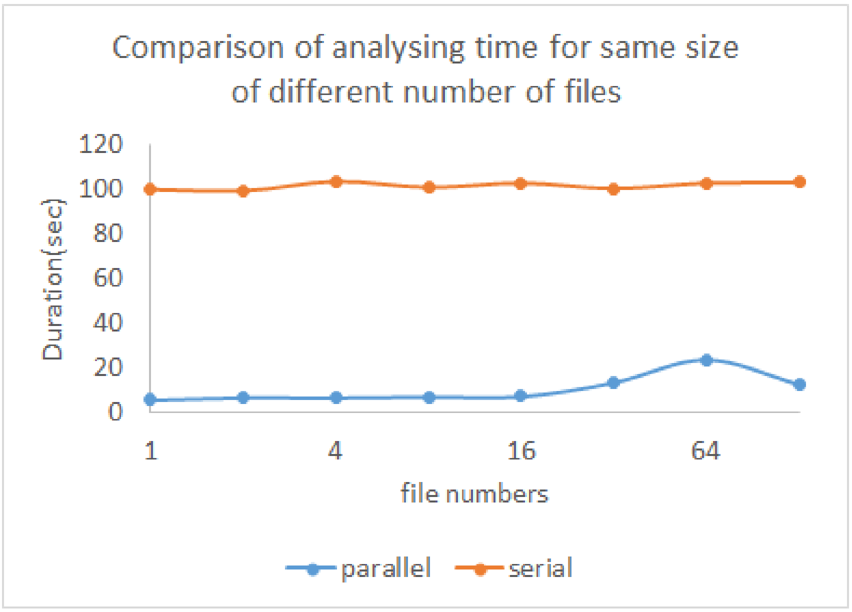
Size	**1K**	**2K**	**3K**	**4K**	**8K**	**10K**	**20K**
Parallel	3.137	2.864	2.941	4.792	3.112	3.179	4.982
Serial	--	0.019	0.027	0.085	0.216	0.187	0.292
Size	**50K**	**70K**	**100K**	**200K**	**400K**	**800K**	**1000K**
Parallel	4.129	4.002	4.058	4.315	4.521	5.503	5.094
Serial	6.516	9.269	12.289	24.004	41.749	83.304	104.668



3.2.2 Methods 2

Enter the same text for both serial and parallel programs, and test the time required for 1, 2, 4, 8, 16, 32, 64, 128 files at the same time, and the total number of words in each input file is 1,000,000.

num	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**
Parallel	5.697	6.736	6.682	7.151	7.276	13.508	23.844	12.785
Serial	100.055	99.433	103.915	101.060	103.169	100.531	103.147	103.307



Chapter 4 Analysis and Comments

4.1 Complexity for Serial Program

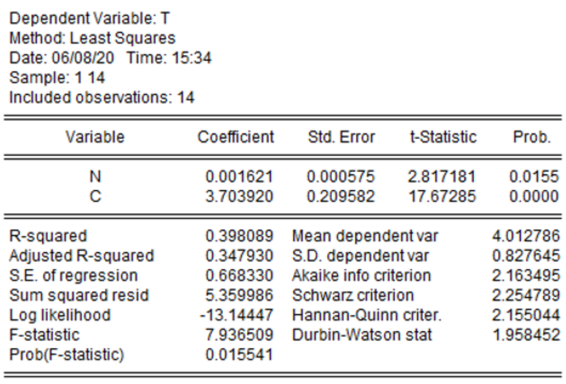
Serial Program needs to read all the input file once and count the words in a C++ map container. Then it sorts the output according to the rules and outputs. Time complexity is $O(N + N\log N) = O(N\log N)$. In the figure shown in Chapter 3, the actual result fits the time complexity.

Serial program needs a C++ map container to store the numbers of words, so the space complexity is $O(N)$.

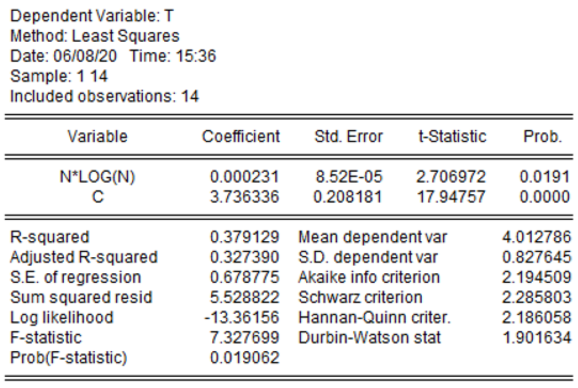
4.2 Complexity for Map Reduce

Map	**shuffle**	**reduce**	**merge**	**total**
$O(N/m)$	$O(N)$	$O(N\log(\frac{N}{r})/r)$	$O(N*r)$	$O(N\log N)$

In theory, if m and r are fixed, N affects the performance the most. When N grows, T grows as $N*logN$. In order to judge whether the time complexity of the test is closer to N or $N\log N$, we perform mathematical analysis on the running time of MapReduce in Chapter 3.1.



```
Dependent Variable: T
Method: Least Squares
Date: 06/08/20   Time: 15:34
Sample: 1 14
Included observations: 14

        Variable     Coefficient    Std. Error    t-Statistic    Prob.

           N          0.001621      0.000575      2.817181      0.0155
           C          3.703920      0.209582      17.67285      0.0000

R-squared              0.398089   Mean dependent var      4.012786
Adjusted R-squared     0.347930   S.D. dependent var      0.827645
S.E. of regression     0.668330   Akaike info criterion   2.163495
Sum squared resid      5.359986   Schwarz criterion       2.254789
Log likelihood        -13.14447   Hannan-Quinn criter.    2.155044
F-statistic            7.936509   Durbin-Watson stat      1.958452
Prob(F-statistic)      0.015541
```



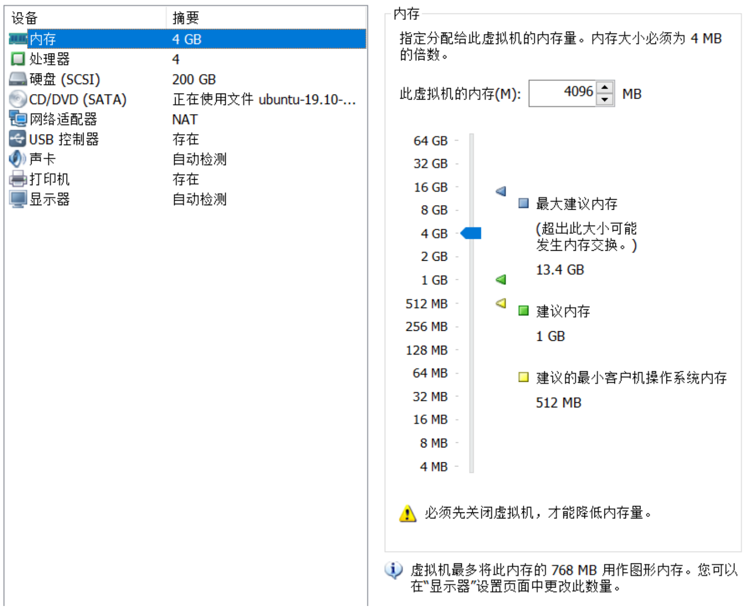
```
Dependent Variable: T
Method: Least Squares
Date: 06/08/20   Time: 15:36
Sample: 1 14
Included observations: 14

        Variable     Coefficient    Std. Error    t-Statistic    Prob.

        N*LOG(N)      0.000231      8.52E-05      2.706972      0.0191
           C          3.736336      0.208181      17.94757      0.0000

R-squared              0.379129   Mean dependent var      4.012786
Adjusted R-squared     0.327390   S.D. dependent var      0.827645
S.E. of regression     0.678775   Akaike info criterion   2.194509
Sum squared resid      5.528822   Schwarz criterion       2.285803
Log likelihood        -13.36156   Hannan-Quinn criter.    2.186058
F-statistic            7.327699   Durbin-Watson stat      1.901634
Prob(F-statistic)      0.019062
```

It can be seen that the time complexity of Map Reduce is actually closer to $n\log N$, which is consistent with our theoretical analysis.

Easily we can know the space complexity of Map Reduce is $O(N)$.

4.3 Hardware

Linux version 5.3.0-55-generic (buildd@lcy01-amd64-009) (gcc version 9.2.1 20191008 (Ubuntu 9.2.1-9ubuntu2)) #49-Ubuntu SMP Thu May 21 12:47:19 UTC 2020



设备	摘要
内存	4 GB
处理器	4
硬盘 (SCSI)	200 GB
CD/DVD (SATA)	正在使用文件 ubuntu-19.10-...
网络适配器	NAT
USB 控制器	存在
声卡	自动检测
打印机	存在
显示器	自动检测

内存

指定分配给此虚拟机的内存量。内存大小必须为 4 MB 的倍数。

此虚拟机的内存(M): 4096 MB

最大建议内存
(超出此大小可能发生内存交换。)
13.4 GB

建议内存
1 GB

建议的最小客户机操作系统内存
512 MB

必须先关闭虚拟机，才能降低内存量。

虚拟机最多将此内存的 768 MB 用作图形内存。您可以在"显示器"设置页面中更改此数量。

4.4 Conclusion

Above all, parallel programs are much faster than serial programs, especially when we have many files to test.

Appendix

I Source Code for WordCount.java

```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;
import java.util.StringTokenizer;

public class WordCount {

      /**
       * Object        : the content of input file
       * Text           : every single line of input data
       * Text           : output the type of key
       * IntWritable : output the type of value
       */
      private static class WordCountMapper extends Mapper<Object, Text, Text, IntWritable> {
            @Override
            protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {
                  StringTokenizer itr = new StringTokenizer(value.toString());
                  while (itr.hasMoreTokens()) {
                        context.write(new Text(itr.nextToken()), new IntWritable(1));
                  }
            }
      }

      /**
       * Text              :  Mapper the input key
       * IntWritable  :  Mapper the input value
       * Text              :  Reducer the output key
       * IntWritable  :  Reducer the output value
       */
      private static class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
            @Override
            protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
                  int count = 0;
                  for (IntWritable item : values) {
                        count += item.get();
                  }
                  context.write(key, new IntWritable(count));
            }
      }

      public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {
            // create configuration
            Configuration configuration = new Configuration();
            // set job of hadoop jobName is WordCount
            Job job = Job.getInstance(configuration, "WordCount");
            // set jar
            job.setJarByClass(WordCount.class);
            // set Mapper's class
            job.setMapperClass(WordCountMapper.class);
            // set Reducer's class
            job.setReducerClass(WordCountReducer.class);
            // set the type of key and value outputted
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(IntWritable.class);

            // set input & output path
            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));
            // exit after executing job
            System.exit(job.waitForCompletion(true) ? 0 : 1);
      }
}
```

II Source Code for serial.cpp

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <sstream>
#include <ctime>
using namespace std;

//WordCount contains the main class
class WordCount{
public:
    explicit WordCount(string new_word) { word = std::move(new_word); num = 1; };
    string get_word() const { return word; }                //return the string of wht word
    int get_number() const { return num; }          //return the frequency of the word
    void add_number() { num++; }                        //add one more word
private:
    string word;
    int num;
};

//this function is used when sorting
//if a appears earlier than b, then a.num > b.num
//or, a.num == b.num and a.word < b.word
bool compare(const WordCount& a, const WordCount& b) {
    if (a.get_number() > b.get_number())   return true;
    string A(a.get_word()),B(b.get_word());
    //If two or more words have the same number of occurrences, they must be printed in lexicographical order
    transform(A.begin(),A.end(),A.begin(),::tolower);
    transform(B.begin(),B.end(),B.begin(),::tolower);
    return a.get_number() == b.get_number() && A <= B;
}

int main (int argc, char* argv[]){
    //start timing
    clock_t start, total;
    start = clock();

    //word_count is used to save the information of a word
    vector<WordCount> word_count;
    string temp_line, temp_word;

    //start traverse the input files
    for (int i = 1; i < argc; i++) {
        //get the input stream
        ifstream input_file(argv[i]);
        while (getline(input_file, temp_line)) {
            //deliver one word each time
            istringstream is_word(temp_line);
            while (is_word >> temp_word){
                //check duplication
                if (word_count.empty()) {
                    WordCount new_word(temp_word);
                    word_count.push_back(new_word);
                    continue;
                }
                for (int j = 0; j < word_count.size(); j++) {
                    if (temp_word == word_count[j].get_word()) { //if find the word, then add one more time
                        word_count[j].add_number();
                        break;
                    } else if (j == word_count.size() - 1) { //otherwise, create a new item for the vector
                        WordCount new_word(temp_word);
                        word_count.push_back(new_word);
                        break;
                    }
                }
            }
        }
        input_file.close();
    }

    //sort the vector
    sort(word_count.begin(), word_count.end(), compare);

    //finally traverse all the words in the vector and put them into the result files
    ofstream output_file("result.txt");

    for (auto & i : word_count) {
        output_file << i.get_word() << "             " << i.get_number() << endl;
    }
    //end timing
    total = clock();
    output_file << endl << "Total time : " << (double)(total - start)/(double)CLOCKS_PER_SEC <<" s"<< endl;
    output_file.close();
}
```

III Source Code for splitting.cpp

```cpp
#include <iostream>
#include <cstring>
#include <fstream>

using namespace std;

//待处理样本的路径
#define  TXT_PATH_NAME     "1.txt"
void getTxtPartName(char *partName, string txtPathName);

int main()
{
    //统计txt中的总行数
    ifstream countSamplesNum(TXT_PATH_NAME);
    string samplesPathName;

    int samplesNum = 0;             //txt中的总行数
    int splitNum = 0;                //计划拆分的份数
    int averageSamplesNum = 0;   //平均每份的行数
    int remainder = 0;               //余数

    //循环读取txt的各行
    while (1)
    {
        if (!getline(countSamplesNum, samplesPathName))
        {
            break;
        }
        if (samplesPathName.empty())
        {
            continue;
        }
        samplesNum++;
    }
    cout << "输入txt的总行数：" << samplesNum << endl;
    cout << "请输入拆分数量：";
    cin >> splitNum;

    averageSamplesNum = samplesNum / splitNum;
    remainder = samplesNum % splitNum;
    char   dispParams[1024];    //读入原文件的名字
    char saveNameExt[1024];    //保存新文件的名字，源文件加数字
    getTxtPartName(dispParams, TXT_PATH_NAME);    //从输入txt的路径获取txt文件名

    if (remainder) //不能整除
    {
        cout << endl << endl << "拆分结果如下：" << endl << "前" << (splitNum - 1) << "份的行数为：" << averageSamplesNum
            << ",    最后1份的行数为：" << (samplesNum - averageSamplesNum * (splitNum - 1)) << endl << endl;
    }
    else    //可以整除
    {
        cout << endl << endl << "拆分结果如下：" << endl << splitNum << "份的行数均为：" << averageSamplesNum << endl << endl;
    }

    //实现拆分，文件保存
    ifstream totalSamplesPath(TXT_PATH_NAME);
    string singleSamplesPath;

    for (int i = 0; i < splitNum - 1; i++)
    {
        sprintf(saveNameExt, "%s_%d.txt", dispParams, i);
        FILE *labelfilename = fopen(saveNameExt, "w+t");

        for (int j = 0; j < averageSamplesNum; j++)
        {
            getline(totalSamplesPath, singleSamplesPath);
            const char* ch = singleSamplesPath.c_str();

            if (j == 0) {
                fprintf(labelfilename, "%s", ch);
            }
            else {
                fprintf(labelfilename, "\n%s", ch);
            }
        }
        fclose(labelfilename);
    }

    //保存最后1份
    sprintf(saveNameExt, "%s_%d.txt", dispParams, (splitNum - 1));
    FILE *labelfilename = fopen(saveNameExt, "w+t");
    int flg = 0;

    while (getline(totalSamplesPath, singleSamplesPath))
    {
        const char* ch = singleSamplesPath.c_str();
        if (0 == flg)
        {
            fprintf(labelfilename, "%s", ch);
        }
        else
        {
            fprintf(labelfilename, "\n%s", ch);
        }
        flg++;
    }

    fclose(labelfilename);

    system("pause");
    return 0;
}

//从txtPathName截取最后一个“\\”后与“.”之前的部分名称
void getTxtPartName(char *partName, string txtPathName)
{
    char tmpChar1[1024] = { '0', };
    char tmpChar2[1024] = { '0', };

    strcpy(tmpChar1, txtPathName.c_str());

    int pathNameLen = strlen(tmpChar1);
    int pos = pathNameLen;

    while (pos > 0)
    {
        pos--;
        if (tmpChar1[pos] == '\\')
        {
            pos++;
            break;
        }
    }

    memcpy(tmpChar2, tmpChar1 + pos, pathNameLen - pos);
    pos = strlen(tmpChar2);

    while (pos > 0)
    {
        if (tmpChar2[pos] == '.')
            break;
        pos--;
    }

    tmpChar2[pos] = 0;
    sprintf(partName, "%s", tmpChar2);

}
```

IV Declaration

We hereby declare that all the work done in this project titled "*MapReduce*" is of our independent effort as a group.

V Duty Assignments

Programmer:
Tester:
Report Writer: