ABSTRACT

DATA ANALYTICS WITH MAPREDUCE IN APACHE SPARK AND HADOOP SYSTEMS

by
Zongxuan Du

MapReduce comes from a traditional problem solving method: separating a big problem and solving each small parts. With the target of computing larger dataset in more efficient and cheaper way, this is implement into a programming mode to deal with massive quantity of data. The users get a map function and use it to abstract dataset into key / value logical pair and then use a reduce function to group all value with the same key. With this mode, task can be automatic spread the job into clusters grouped by lots of normal computers. MapReduce program can be easily implemented and gain much more efficiency than tradition computing programs. In this paper there are some sample programs and one GRN detection algorithm program to study about it.

Detecting gene regulatory networks (GRN), the regulatory molecules connection among various genes, is one of the main subjects in understanding gene biology. Although there are algorithms developed for this target, the increase of gene size and their complexity make the processing time more and more hard and slow. MapReduce mode with parallelize computing can be one way to overcome these problems. In this paper, a well-defined framework to parallelize mutual information algorithm is presented. The experiments and result performances shows the improvement of using parallelizing MapReduce model.

DATA ANALYTICS WITH MAPREDUCE IN APACHE SPARK AND HADOOP SYSTEMS

by
Zongxuan Du

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer Science

May 2016

APPROVAL PAGE

DATA ANALYTICS WITH MAPREDUCE IN APACHE SPARK AND HADOOP SYSTEMS

Zongxuan Du

Dr. Jason T. Wang, Thesis Advisor — Date
Professor of Bioinformatics and Computer Science, NJIT

Dr. Xiaoning Ding, Committee Member — Date
Assistant Professor of Computer Science, NJIT

Dr. Chase Qishi Wu, Committee Member — Date
Associate Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Zongxuan Du

Degree: Master of Science

Date: May 2016

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2016

- Bachelor of Science in Software Engineering,
  Inner Mongolia University, Inner Mongolia, People's Republic of China, 2014

Major: Computer Science

ACKNOWLEDGMENT

I am grateful to my thesis advisor, Professor Jason T.L.Wang, for his guidance and support. His expertise in bioinformatics and computer sciences improved my research skills and prepared me for future challenges. I thank my other committee members, Professor Chengjun Liu and Chaes Wu, for their helpful suggestions and comments during my study.

I am also thankful for their help Byron Kevin from NJIT and Abduallah Yasser from IBM. They have given me precious support and inspire me while finishing this thesis.

While finishing my thesis, my family members also give me huge support. Thank my mother and father's help.

感谢我的父母和其他家人在论文完成的过程中给予我的巨大鼓励支持。

TABLE OF CONTENTS

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Objective

The objective of this thesis is to overcome the growing dataset scale and the growing amount and complexity in gene analyze using map-reduce. We use TD-ARACNE method to measure gene connection and using mutual information based algorithms to get the connection in genes. Using the same dataset get from GRN and several different implementations to make sure in same accuracy based level, we can find the performance improvement for map-reduce and distributed computing.

1.2 Background Information and Study Significance

The basic result of biologic and genic study is to understand all constitutes of bio-system and the operating mechanism. The conventional study approach is to study various gene through experiment, which focus on analyzing single genic function. However, it is far away from enough to indicate and understand organism. To study the connections and expressions between genes, systems biology research area appears.

Systems biology is a subject that study all components (for example, genes, mRNA, proteins, etc.) in one bio-system such as a cell. As it studies all the genes and the connection, and the connection between different constituents in genes which influence the express and then influence the production of tissues, so it is very important. Because of it, the study of Gene Regulatory Network (GRN) is valued high to all researchers. GRN is a kind of complicate biochemical system which is constitute by one or more groups of genes, protein and micro molecule and the mutual effect between them.

As study GRN, one important point is to compute the inner connection and network out through given time-series gene expression profile data. However, as the scale of computing and gene quantity, it becomes harder and harder to use normal method get result both in reasonable time and keep high accuracy as difficult in computing and storing data. There are several new methods developed to resolve problems about storing and computing big quantity data and in this paper we talked about using these method on GRN computing area to get an improvement in gene connection study.

1.3 Organization Structure

This paper is mainly base on the implementation about one GRN computing approach TD-ARANCE. It begins with basic knowledge about GRN and the reason to focus on it. Then analyze more about TD-ARACNE approach and its more efficient distribution computing implementation. And later compare performance between the normal implement methods and the distribution approaches. The whole paper organized by four chapters, the structure is in follow:

Chapter 1 is introduction. Firstly, introduce the objective of this paper. And then introduce the background information about the meaning of this research. Finally present the overall structure of this paper.

Chapter 2 is about the necessary theory of this paper and the implementation prepare. Theory includes ARACNE and TD-ARACNE, and preparing also include data handle and main used distribution approach: MapReduce.

Chapter 3 is the main implementation done chapter. Using MapReduce and other approach such as spark, TD-ARACNE theory is been implemented in several ways to make a better performance comparison.

Chapter 4 includes the performance analyze and conclusion about this research.

CHAPTER 2

RESEARCH PREPARATION

This paper's work builds on regulatory network expression calculation algorithms which are named ARACNE (Margolin et al, 2006) and TD-ARACNE (Zoppoli et al., 2010).

Between these two algorithms, ARACNE has already been a steady state algorithm and TD-ARACNE is considered to be a time-series algorithm, which is an upward extension of a steady state algorithm. So the work in this paper is mainly focus on TD-ARACNE.

With the distribution approach apply, using map-reduce technology to implement TD-ARANCE algorithm and testing with standard dataset, the work in paper can be accurate and clearness.

2.1 TD-ARACNE

Time-Delay-ARACNE (Algorithm for Reconstruction of Accurate Cellular Networks) algorithm is a detecting method for mutual information network of time-series gene expression. The basic idea is showed in Figure 2.1, the activation of a gene A is assumed can influence a gene B activation in successive time instants, this information is carried out in gene A and gene B connection.

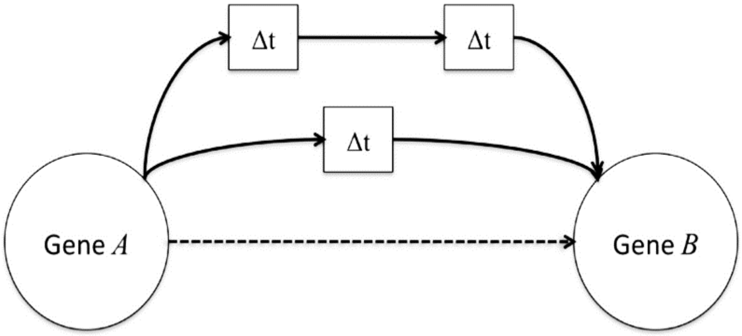


Figure 2.1 TimeDelay-ARACNE main idea.

This time-delay network inference algorithm is mainly separate into three steps with the handle of original dataset as showed in Table 2.1.

Table 2.1 Source Data Simple

Time Point	Variable 1 (g_1)	Variable 2 (g_2)	Variable 3 (g_3)	Variable 4 (g_4)
1	0.666511	0.127219	0.355065	0.774572
2	0.325775	0.121822	0.346412	0.722911
3	0.177501	0.044359	0.571289	0.586828
4	0.183885	0.061535	0.484977	0.633821
5	0.093069	0.139843	0.343555	0.535438
6	0.065302	0.088636	0.312431	0.525551
7	0.150214	0.094808	0.412641	0.459971
8	0.091332	0.090451	0.376457	0.421402

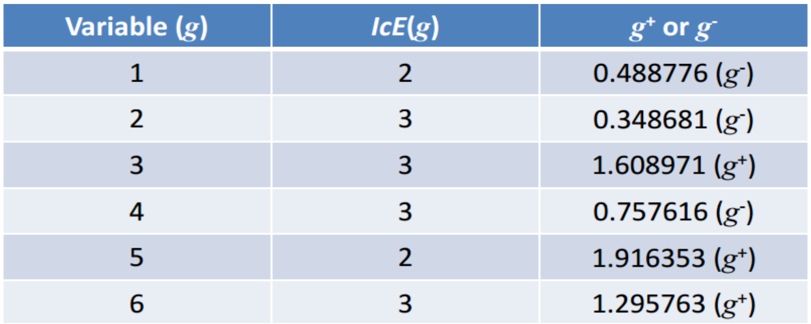
Generally, in step 1 we set a time point t ($t > 1$) which means the significant value changing. For each variable g, check the variable value $g(t)$ with a user selected variable threshold τ :

If $\frac{g(t)}{g(1)} > \tau$, then $g^{+}(t) = g^{+}(t) \cup \{t\}$, variable g induced at time t.

If $\frac{g(t)}{g(1)} < \frac{1}{\tau}$, then $g^{-}(t) = g^{-}(t) \cup \{t\}$, variable *g* repressed at time *t*.

After computing, the initial change of expression (*IcE*) comes from the minimum of *g(t)* : $IcE(g) = \min\{g^{+}(t) \cup g^{-}(t)\}$, then get the result just like Table 2.2 shows.

Table 2.2 Gene IcE value simple



Variable (*g*)	*IcE*(*g*)	g^{+} or g^{-}
1	2	0.488776 (g^{-})
2	3	0.348681 (g^{-})
3	3	1.608971 (g^{+})
4	3	0.757616 (g^{-})
5	2	1.916353 (g^{+})
6	3	1.295763 (g^{+})

For step 2, we calculate time-delayed mutual information among gene groups between each two gene pairs g_a g_b. This calculation is for measuring how much one gene affects another gene. That is for every pair of genes data, g_a and g_b with initial change of expression $IcE(g_a) \leq IcE(g_b)$, *I* is calculated as follows:

$$I(g_a, g_b) = \sum_{1<i<n-k} p(g_a^i, g_b^{i+k}) \log \frac{p(g_a^i, g_b^{i+k})}{p(g_a^i)\, p(g_b^{i+k})}$$

In here, n is time point numbers and *k* is the time shift used in the acquisition of the gene expression data; and $p(g_a^i, g_b^{i+k})$ is the calculated probability. The time delay calculate is like Figure 2.2 shows.

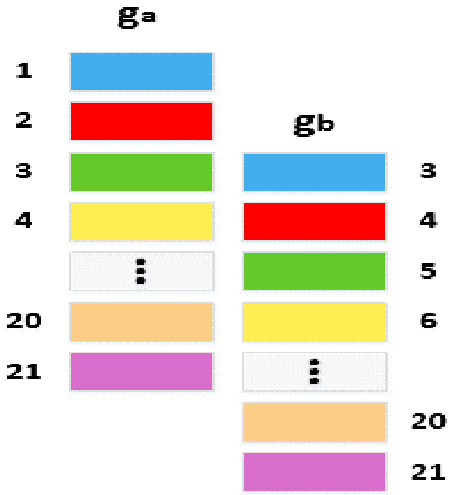


Figure 2.2 Time-Delay sample for two genes.

Then the maximum value (*infl* value) calculated for each time shift in k is chosen and compared with a variable threshold ε. Bigger than ε means g_a cause g_b or g_b depends on g_a.

The Influence (*infl*) as maximum time-time dependent I for g_a and g_b (Zoppoli et al., 2010) over k time delays is calculated as follows:

$$infl(g_a, g_b) = \max_{1 \le k \le h}\{I^k(g_a, g_b^k)\}$$

Where h is the total number of time shifts.

In step 2, with the connection between each g_a g_b discovered by comparing with ε, we can get the previous influence network between each genes.

At last step, in the network, between each two genes if their edge make a circle, compare the value of $infl(g_a, g_b)$ and $infl(g_b, g_a)$:

If $infl(g_a, g_b) > infl(g_b, g_a)$ then remove $infl(g_a, g_b)$.

Else if $infl(g_a, g_b) < infl(g_b, g_a)$, then remove $infl(g_b, g_a)$

For now from these three steps of TD-ARACN algorithm, we can detect expression network among genes using the large quantity of source gene dataset.

2.2 MapReduce

Generally, map-reduce is one kind of programming model which is used to handle large scale (over then 1 TB) of data. It’s a distribution computing approach which can fully utilize lots of computing resource to simplify and solve a big question.

This programming model include two main concepts, map and reduce. Map function is implemented to mapping a group of value to new key-value groups using different inner logic and then puss them to reduce function, in reduce function they get the key and values and regroup them to computing to a much smaller value group to be result. As the shortest describe words on internet say: We want to count all the books in the library, you count up shelf #1, and I count up shelf #2. That’s map. The more people we get, the fester it goes. Now we get together and add our individual counts. That’s reduces.

The general map-reduce big data handling process is like following:

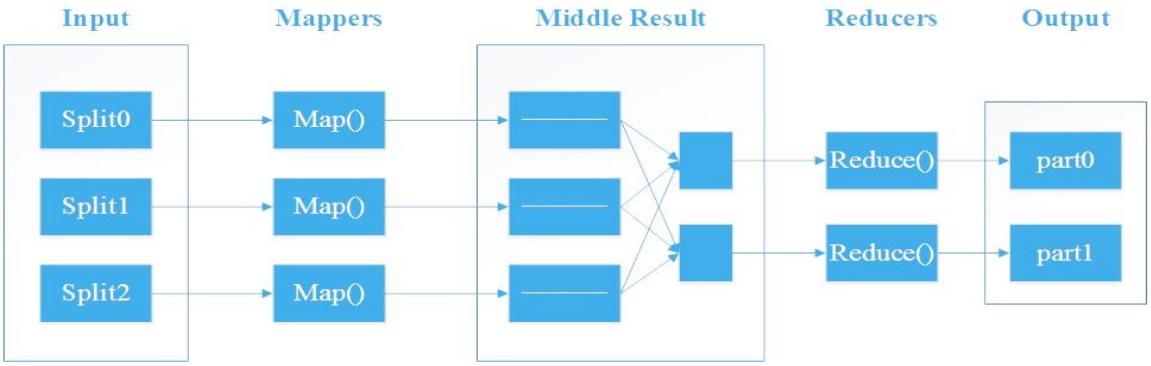


Figure 2.3 Work flue model for MapReduce.

Using pseudo-code to show, a map-reduce program is like:

- **Mapper**

```
map (String input_key, String input_value):
for each count target ct in input_value:
EmitIntermediate(ct, "1");
```

- **Reducer**

```
reduce (String output_key,Iterator intermediate_values):
 int result = 0;
 for each v in intermediate_values:
 result += ParseInt(v);
 Emit (AsString(result));
```

2.2.1 Hadoop Study

Hadoop is a distribution system architecture which is developed by Apache.

The cores of Hadoop are include HDFS (Hadoop Distribute File System) and MapReduce.

HDFS support the storage of big scale of data which normal computers. In file system connected and supported by normal computers, the big files are split into several small equally parts (64M for each part in normal) and store multiple times in different location in file systems and when reading, it can be read from different place in same time, which let the reading speed fast.

MapReduce support the big data computing. Hadoop create one task to making map computing for each input split, handling the records in the split of each task and then map will give out the result in key-value pair format. Hadoop responsible for range the map results with key pair and make it to be the input of reduce task. Reduce task do more computing for example add the value of pairs with same key together and to make the result to be the whole result of job store into HDFS.

Although Hadoop has the positive aspects for example have high computing ability, big date storage ability and highly fault tolerance ability, it still have some weakness. The job have only map and reduce two phase, so developers need to do lots of work to express their computing target and also need to take lots of energy to manage the relationship between jobs when there are lots of job in one mission. It also have a bad performance when handling iterative data.

2.2.2 Spark Study

Apache Spark in a new big data handling engine. It provide a crowded abstract of distribution system memory to support the application which need working set.

The abstract is called RDD (Resilient Distributed Dataset). It's an unchanged record set, which is also the programming model of spark. RDD is mainly group with three parts: partitions, functions and dependencies. We can imagine RDD to a very big list. Because it is so big, we need to separate it into several parts, which are partitions. So RDD is grouped by many partitions. Because we need to calculate each partitions, we use function for computing each split. RDD need have dependency for each partitions.

Spark provide two functions based on RDD, the transformation and the action. In these two functions, transformation is used to define a new RDD, include map, flatMap, filter, union, sample, join, groupByKey, ReduceByKey,cros, sortByKey, mapValues and so on. And action is to return a result, include collect, reduce, count, save, lookupKey. RDD must use transformation to make transform. Which are operators like map or reduce.

In spark, all the RDD transformations are lazy. Which means when RDD1 use transformation to RDD2, the real computing is not done, they just store this function. Until face the action, the computing functions then will be trigger. For example, when moving into trigger action, it is found that there is no RDD3, so trace back to RDD2 and then find it also not be triggered, so back to RDD1 to start the function.

2.3 Data Preparation

There are two parts of data for our study. The first part of dataset are web pages come from Wiki database. The form of these files are web page xml code, which consist all page context which can be detected by word-count program for several words we determine. All the xml files are downloaded from dumps.wikimedia.org. We select the wiki 20160305-2015025 pages for our test.The whole size of dataset is about 10GB

Another part of dataset is obtained from gene expression omnibus (GEO) public database. The dataset we select to use is identified as GSE10158. This dataset consists with 7312 gene transcripts, each line contains a gene name (id) and its expression grouped by 43 time points. While computing, TD-ARACNE map-reduce program will read each gene and also all other gene information from HDFS to process the gene network. For there are 7312 genes in total, and we need to compute (geneX, geneY) and (geneY, geneX) both, so the total lines are 7312 * 7312 = 53465344 lines. And the file size is 26.8 GB.

CHAPTER 3

MAPREDUCE IMPLEMENTATION

The implementation is separated into two parts. In the first, we talking about the basic MapReduce programming logic and use basic map-reduce principle to implement map-reduce structure into Hadoop and Spark programming. The map-reduce mode is extend to read web pages and count several words appearance times. Then, with the knowledge of TD-ARACNE, we implement this mutual information computing algorithm into map-reduce format and testing with our GEO dataset for performance study.

3.1 Map-Reduce Simple Application Implementation

The basic map-reduce thought is like the pseudo-code showed in second section, the map method get source input data, separate it into treatable pieces and then abstract the keys out, which are the assigned words like ‘country’ and ‘fish’ in here. With those abstracted keys, in map stage we always give them a value pair ‘1’ to group the key-value pair and then export them out. Those key-value pairs becomes the input of reduce method. In the reduce stage we sum the value of words with same key and give the final result out.

3.1.1 Programming Implementation with Hadoop

For Hadoop programming, we extend map and reduce methods and select the key-value pairs for each stages. In program the map and reduce data flow is like following:

Input => Map => Mapper Output => Combiner => Sort and Shuffle => Reduce =>

Final Output.

Among the whole process, there is one stage called combiner, which is between map and reduce sections. The combiner is for preparing work to merge the same keys, its do the work just like reducer. The different between combiner and reduce method is this combiner section begins just follow each map section and only merge the same keys for the output of each map to relieve reduce method's stress, while reduce stage handle all the output. In our program, we do not need it because the job is not complex, so we set the combiner to null. Another stages, sort and shuffle stage is for transporting correct result into right reduce method, which is controlled by system.

The whole program include three elements: mapper, reducer and invoker.

First we create map function with dataset file as its input and context variable. In the map function we appoint several English words: 'country, education, and fish' to be the target words we want to count from web pages then input them into a hash set. After that the source dataset is split into lines and then into words. While there is a word equal to target words in hash group, we generate one key-value pair (word, '1') and use context variable to pass it out.

Second we create reduce function with word and its value counts to be input. In the reduce function we get all the pair and sum its counts value based on shuffled same words key.

After the implementation of map and reduce functions, we write an invoker function to set the context variable and set map and reduce class to call our methods and run whole process.

3.1.2 Programming Implementation with Spark

Although Spark comes from MapReduce and appear to solve traditional Hadoop MapReduce slow disc reading problem, it have little different programming logic. In Hadoop the typical points are:

1. Usually using key-value pair to be map and reduce function input and output

2. One key is for one reduce in Reducer class process

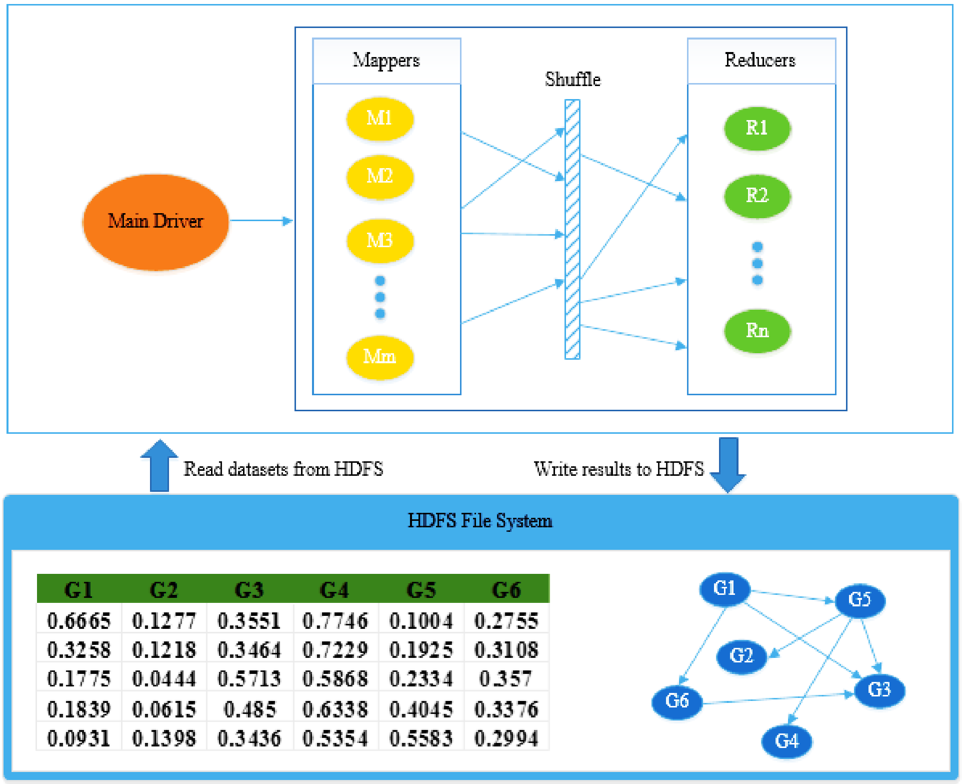
3. Every mapper or reducer may process every kind of key or value instead of using a standard data format.

While in Spark, the input is a resilient distributed dataset. And also there is no obvious key-value pair, instead, we use Scala’s tuple. This kind of tuple is made by (a, b) language logic such as (line.length, 1). One of a traditional map function in Hadoop is expressed into and RDD, (line.length, 1) tuple. If one RDD owns tuples, it relay other function like reduceByKey() to handle it.

The first task to make this programming is to create a sparkContext object to tell Spark how to access the clusters. Then use RDD function API which is called JavaRDD and JavaPairRDD to transform the input file and call mapToPair function to write our word detect and count method to process new Tuples with words (key) and numbers (value). After that use reduceByKey to runn total and count the numbers and give result.

3.2 TD-ARACNE Program Implementation

Based on the algorithm we talked in chapter 2, we implement it into MapReduce program. The main logical is showed in Figure 3.2.



G1	G2	G3	G4	G5	G6
0.6665	0.1277	0.3551	0.7746	0.1004	0.2755
0.3258	0.1218	0.3464	0.7229	0.1925	0.3108
0.1775	0.0444	0.5713	0.5868	0.2334	0.357
0.1839	0.0615	0.485	0.6338	0.4045	0.3376
0.0931	0.1398	0.3436	0.5354	0.5583	0.2994

Figure 3.1 TD-ARACNE MapReduce implementation model

There is a main driver function reads data from HDFS, set some environment variables and invoke MapReduce program. The mapper response to do the calculations as we see from chapter 2. First we abstract the data of two genes as each key-value pairs. So the key is set such as <geneA_Data> and value can be set like <geneB_Data>. As in our dataset each gene has 43 different time point expressions, every keys and values are included with gene identifier like ‘at_bosr_Data_’ and its expression data like ‘7.243512342’. Firstly we computes two genes’ IcE(initial change of expression) values. If IcE from geneA is no more than geneB, we can compute to detect time-delayed mutual information between to genes. With the compare of *infl* and threshold, we can get whether the max *infl* between these two genes is bigger than threshold, if so, we know geneA cause geneB and make it to the output with the key-value pair format of the geneA identifier and geneB indentifer as key, their max *infl* as value. The reducer function get this key-value pair from mapper and check if there is same key while we reverse the identifier of geneA geneB into geneB geneA. If there are same ones, compare their *infl* value and delete the smaller one. So the result can be like: ats_b231_Data_ -> ats_b567_Data_. Which is the connection network among those genes.

The logic of this implementation can be write like:

Input: Gene Dataset :{Gene(1) Gene(2)},{Gene(2) Gene(1)} …. , {Gene(n-1) Gene(n)},{Gene(n) Gene(n-1)}

```
    Mapper
    IcE_A = calculate ICE(geneA);
    IcE_B = calculate IcE(geneB);
    If (IcE_A <= IcE_B):
InfoMax = max (calculate Mutual Information (geneA, geneB) with time delay 1,2,3);
    If (InfoMax >= threshold):
        key= geneA-geneB,    value = InfoMax;
```

```
Output(key, value);


Reducer(key, value)
Get key(geneA-geneB), reverse it to keyTemp(geneB-geneA)
For : all keys :
        If( there is other keys same as keyTemp) :
                compare their value pairs, delete small one;
    end For;
key = geneX from key(geneX,geneY)
value = geneY from key(geneX,geneY)
Outpu(key, value);
```

Output: Network graph: {Gene(a) -> Gene(b},   ….. Gene(x) -> Gene(y)}

CHAPTER 4

PERFORMANC MEASURE AND ANALYSIS

4.1 Performance Contrast

For easy to compare and get the accuracy result, for non- MapReduce and MapReduce performance compare, we use TD-ARACNE program to measure. For the compare of Hadoop and Spark computing, the simple wordcount program can be more useful because the inner logic can be easy to handle and unify.

For how MapReduce paralleling can improve the computing efficiency, we use standalone server to run a TD-ARACNE java sequential program compare with TD-ARACNE Hadoop MapReduce program. For easy to compare, the source data is been separated into from 1000 genes to 7000 genes. As it showed in Table 4.1. The difference appears after computing more than 4000 genes and as the Figure 4.1 shows, the curve for sequential program test rise faster and faster as the gene number become larger while MapReduce program do not have much rise in time using.

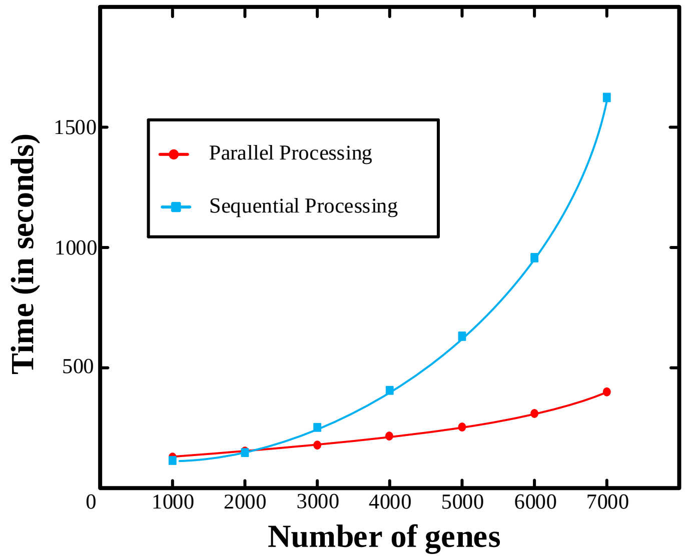


Figure 4.1 Parallel sequential program comparison result.

Table 4.1 Result for parallel sequential program comparison

Gene Number Result In Seconds	1000	2000	3000	4000	5000	6000	7000
Sequential Program	25	76	233	470	675	979	1694
Parallel Program	48	89	142	187	222	257	329

In the comparison between Hadoop and Spark program, we use wordcount programs with same logic and same wiki page dataset. With all other testing parameter keep the same, we can get the experiment result just like Table 4.2 shows.

Table 4.2 Data for Spark Hadoop against experiment

	Input Data Size	Time Using	Rate
Hadoop	10GB	132 s	78M/s
Spark	10GB	52 s	197M/s

4.2 Performance Analyzing

From the sequential parallel mode comparison test result, we can find that at first time when data size is not much, standalone sequential program can have better efficient that parallel MapReduce program. This is because the system will spend lots of time to start and destroy map and reduce tasks from the cluster pool. And also data transform will spend large quantity of time. In this experiment and more other experiments, we can find that if we control system to run too much mappers in cloud, the performance will decline by spending much time on task dispatch. But as the size of source data raise, MapReduce program shows its stable performance in dealing with large quantity of data, while sequence program uses much more time to compute. By the size raise, sequential program uses about 5 times processing time to compute same kind of program compare with MapReduce program.

However, MapReduce program based on Hadoop also have its problem. The basic idea of computing based on HDFS, which means mapper and reducer get data from HDFS grouped with hard drivers and put result back into HDFS, keep the security of tasks out of the one normal computer break problem but also spend much time in data transform, copy and move from and into hard driver. To overcome this, Spark put all the computing into memory which can reduce the data settle time. From the Table 4.2 test wordcount result we can see that with 10GB data use to run same logic wordcount program, Spark use less time than Hadoop. With the raise of dataset there are more separated pieces of task and more mappers, the performance of Spark will be better and better than Hadoop.

CHAPTER 5

CONCLUSION AND DISCUSSION

From the experiment we get that MapReduce computing has its own limitation, a fairly large number of data can let us gain the benefit from using the cloud computing resource. But how large the data can give us benefit by using MapReduce computing? From our experiment, we find that we MapReduce computing has benefit when the dataset size is larger than about 5GB and obvious different can be seen only after the data size raises to 15GB. But this result can be more accuracy only after testing enough mapdreduce programs with different algorithm and different purpose, which cannot be done in this paper but can be a good topic in my further study.

Also in the performance comparison for Hadoop against Spark, we can find that RDD in-memory computing gives Spark obvious efficient improve. My Spark wordcount program is programmed by JAVA, it's a good experiment to compare the efficient to run same logical Spark program with different language with huge size of data. As I believe, as scala the language to build Spark, the program under scala will performances better than other languages, but its need experiments to confirm. Also in the experiment of this paper, I implement the Spark TD-ARACNE program with JAVA, however, with the unstable of running performance, I cannot get a useful experiment result for it. I will continuous work on it and get the final performance comparison among standalone program, Hadoop program and Spark program for TD-ARACNE. In the predictable result, Spark TD-ARACNE program will also have the best performance.

Overall, the MapReduce model parallelizing can become more and more valuable and powerful as it provide an easy and cheap approach to analysis large scale of data. For example, in bioinformatics area, more and more algorithms will be implemented into MapReduce format to meet the fast growing of data size.

APPENDIX A

SOUCE CODE

A.1 Hadoop WordCount

```
import java.io.IOException;
import java.util.Arrays;
import java.util.HashSet;
import java.util.regex.Pattern;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

import org.apache.log4j.Logger;

public class WordCount extends Configured implements Tool {

    private static final Logger LOG = Logger.getLogger(WordCount.class);

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new WordCount(), args);
        System.exit(res);
    }

    public int run(String[] args) throws Exception {
        Job job = Job.getInstance(getConf(), "wordcount");
        job.setJarByClass(this.getClass());
        // Use TextInputFormat, the default unless job.setInputFormatClass is used
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {

        private static final Pattern UNDESIRABLES =
Pattern.compile("[(){},.;!+\"?<>%]");
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        private long numRecords = 0;
        private static final Pattern WORD_BOUNDARY =
Pattern.compile("\\s*\\b\\s*");
        private String elements[] = { "education", "politics", "sports", "agriculture" };
        private HashSet<String> dict = new
HashSet<String>(Arrays.asList(elements));

        public void map(LongWritable offset, Text lineText, Context context)
                throws IOException, InterruptedException {
            String line = lineText.toString();
            Text currentWord = new Text();
            for (String word : WORD_BOUNDARY.split(line)) {
                if (word.isEmpty()) {
                    continue;
                }
                String cleanWord =
UNDESIRABLES.matcher(word.toString()).replaceAll("");
                if(dict.contains(cleanWord)) {
                  context.write(new Text(cleanWord), one);
                }
            }
        }
    }

    public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

        @Override
        public void reduce(Text word, Iterable<IntWritable> counts, Context context)
                throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable count : counts) {
                sum += count.get();
            }
            context.write(word, new IntWritable(sum));
        }
    }
```

```
}
```

A.2 Spark WordCount

```
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.FlatMapFunction;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.api.java.function.Function2;
import org.apache.spark.api.java.function.PairFunction;
import scala.Tuple2;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class WordCount2 {

    public static void main(String[] argv){
        if (argv.length != 2) {
            System.err.printf("Usage: %s [generic options] <input> <output>\n",
                    WordCount.class.getSimpleName());
            return;
        }
        String inputPath = argv[0];
        String outputPath = argv[1];

        System.out.printf("Starting WordCount program with %s as input %s as 
output\n", inputPath,outputPath);

        SparkConf conf = new 
SparkConf().setAppName("WordCount").setMaster("local");
        JavaSparkContext sc = new JavaSparkContext(conf);

        JavaRDD<String> file = sc.textFile(inputPath);

        JavaRDD<String> words = file.flatMap(new FlatMapFunction<String, 
String>() {
            @Override
            public Iterable<String> call(String s) throws Exception {
                return Arrays.asList(s.split(" "));
            }
        });
        words = words.filter(new Function<String, Boolean>() {
            @Override
            public Boolean call(String s) throws Exception {
                System.out.println("Inside filter words ->" +s);
                if( s.trim().length() == 0)
                    return false;
                return true;
            }
        });

        JavaPairRDD<String, Integer> wordToCountMap = words.mapToPair(new
PairFunction<String, String, Integer>() {
            @Override
            public Tuple2<String, Integer> call(String s) throws Exception {
                return new Tuple2<String, Integer>(s,1);
            }
        });

        JavaPairRDD<String, Integer> wordCounts =
wordToCountMap.reduceByKey(new Function2<Integer, Integer, Integer>() {
            @Override
            public Integer call(Integer first, Integer second) throws Exception {
                return first + second;
            }
        });

        wordCounts.saveAsTextFile(outputPath);

    }
}
```

A.3 TD-ARACNE

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URI;
import java.util.LinkedHashMap;
import alorgithm.AracneMain;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

public class ARACNEDriver {

    public static boolean timeDelay = true;
    public static boolean setNumReducers = false;
    public static int numberOfReducers = 1;
    public static boolean setNumGenes = false;
    public static int numberOfGenes = 1000;
    public static boolean setNumMappers = false;
    public static int numberOfMappers = 10;
//public static double threshold = 1.2;
    //public static double epsilon = 0.37;
    public static double threshold = 1.2;
    public static double epsilon = 0.97;
    public static int[] kArray = new int[]{1, 2, 3};
    public static int distributionArraySize = 200;

    public static double getThreshold() {
        return threshold;
    }

    public static double getEpsilon() {
        return epsilon;
    }

    public static int[] getKArray() {
        return kArray;
    }

    public static int getDistributionSize() {
        return distributionArraySize;
    }

    public static void main(String[] args) throws Exception {
        String inputFile = args[0];
        String outputFile = args[1];

        long start = System.currentTimeMillis();
        System.out.println("input file: " + inputFile);
        Configuration conf = new Configuration();
        for (String a : args) {

            if (a != null && (a.trim().toLowerCase().indexOf("threshold") > -
1)) {
                threshold = Double.parseDouble(a.trim().split("=")[1].trim());
            }
            if (a != null && (a.trim().toLowerCase().indexOf("epsilon") > -1)) {
                epsilon = Double.parseDouble(a.trim().split("=")[1].trim());
                System.out.println("Setting epsilon: " + a + " " + epsilon);
            }
            if (a != null &&
(a.trim().toLowerCase().indexOf("numberofshits") > -1)) {
                int shifts = Integer.parseInt(a.trim().split("=")[1].trim());
                kArray = new int[shifts];
                for (int i = 0; i < shifts; i++) {
                    kArray[i] = (i + 1);
                }
            }
            if (a.trim().indexOf("countonly") > -1) {
                boolean countOnly = new Boolean("" +
a.trim().split("=")[1]).booleanValue();
                conf.setBoolean("countOnly", countOnly);
                System.out.println("Count only process: " + a + " " +
countOnly);
            }
            if (a.trim().indexOf("timedelay") > -1) {
                timeDelay = new Boolean("" +
a.trim().split("=")[1]).booleanValue();
                System.out.println("timedelay: " + a + " " + timeDelay);
            }
            if (a.trim().indexOf("numreducers") > -1) {
                setNumReducers = true;
                numberOfReducers = Integer.parseInt(a.trim().split("=")[1]);
                System.out.println("Setting number of reducers to: " +
numberOfReducers);
            }
            if (a.trim().indexOf("nummappers") > -1) {
                setNumMappers = true;
                numberOfMappers = Integer.parseInt(a.trim().split("=")[1]);
                System.out.println("Setting number of mappers to: " +
numberOfMappers);
            }
            if (a.trim().indexOf("numgenes") > -1) {
                setNumGenes = true;
                numberOfGenes = Integer.parseInt(a.trim().split("=")[1]);
                System.out.println("Using number of genes : " +
numberOfGenes);
            }
            if (a.trim().indexOf("distributionarraysize") > -1) {
                distributionArraySize = Integer.parseInt(a.trim().split("=")[1]);
                System.out.println("Setting number of distributionArraySize
to: " + distributionArraySize);
            }
        }
        if (timeDelay) {
```

```
                        System.out.println("Processing Time Delay Mutual Information
Algorithm");
                } else {
                        System.out.println("Processing Steady State Mutual Information
Algorithm");
                }

                AracneMain.main(args);

        }
}


import access.ARACNEDriver;
import support.ARACNETDCalculator;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

import java.io.IOException;
import java.util.Iterator;



public class AracneMain    {
 Log log = LogFactory.getLog(AracneMain.class);

 public static class FPRMapper extends MapReduceBase implements
Mapper<LongWritable, Text, Text, Text>    {
        public Log log = LogFactory.getLog(FPRMapper.class);
         ARACNETDCalculator atdc = null;

         @Override
         public void map(LongWritable key, Text value, OutputCollector<Text, Text>
output, Reporter reporter) throws IOException {
               try {
                      String[] data = (""+value).split("\t");

                             String[] tokensA = (""+data[0]).split("__DATA__");
                             String[] tokensB = (""+data[1]).split("__DATA__");
                             String geneA = tokensA[0];
                             String geneB = tokensB[0];
                             String[] gA = tokensA[1].split(",");
                             String[] gB = tokensB[1].split(",");
                             double ice = atdc.calculateIce(gA);
                             double iceB = atdc.calculateIce(gB);
                             if(ice == iceB){
                                 String result =
atdc.getMutualInformationResultReverse( geneA,    geneB,    gA,    gB);
                                 if(result.indexOf("-->") > -1) {
                                     //String[] tokens = result.trim().split("-->");
                                     output.collect(new Text(result), new Text(""));
                                     //output.collect(new Text(tokens[0].trim()), new
Text(tokens[1].split("\t")[0].trim()));

                                 }
                             } else if (ice < iceB) {
                                 String result = atdc.getMutualInformationResult( geneA,
geneB,    gA,    gB);
                                 if(result.indexOf("-->") > -1) {
                                     output.collect(new Text(result) , new Text(""));
//                      output.collect(new Text(geneA.trim()), new
Text(geneB.trim()));

                                 }
                             }
               }catch(Exception e) {

               }
         }

         /**
          * configures the result directory from the current job
          * (non-Javadoc)
          * @see
org.apache.hadoop.mapred.MapReduceBase#configure(org.apache.hadoop.mapred
.JobConf)
          */
          @Override
          public void configure(JobConf job) {

              atdc = new ARACNETDCalculator();
          }
 }
 public static class FPRReducer    extends MapReduceBase implements
Reducer<Text, Text, Text, Text>    {
        private Text result = new Text();

        @Override
```

```
      public void reduce(Text key, Iterator<Text> values, OutputCollector<Text,
Text> output, Reporter reporter) throws IOException {
          StringBuffer genes = new StringBuffer();
          while (values.hasNext()) {
               Text val = values.next();
               genes.append(val.toString());
          //    output.collect(new Text(key    +    " --> " + val),new Text(""));

          }
          result.set(genes.toString());
          output.collect(key, new Text( result));
      }
 }
   public static void main(String[] args) throws Exception {

          String inputFile = args[0];
          String outputFile = args[1];

          String algorithmName = "AracneMain";
          long start = System.currentTimeMillis();
          System.out.println("input file: " + inputFile);
            Configuration conf = new Configuration();
            JobConf jobConf = new JobConf(conf);
            String optionalName = "";
            for(String a : args){
               if(a.trim().indexOf("timedelay") > -1) {
                    ARACNEDriver.timeDelay = new
Boolean(""+a.trim().split("=")[1]).booleanValue();
                    System.out.println("timedelay: " + a + " "    +
ARACNEDriver.timeDelay);
               }
               if(a.trim().indexOf("optionalname") > -1) {
                    optionalName = a.trim().split("=")[1];
                    System.out.println("a : " + a + " "    + optionalName);
               }
          }
            algorithmName = algorithmName + "" + optionalName;
            if(ARACNEDriver.setNumGenes) {
               algorithmName = algorithmName + "" + optionalName + "_" +
ARACNEDriver.numberOfGenes + " genes" ;
            }
            if(ARACNEDriver.timeDelay) {
               System.out.println("Processing Time Delay Mutual Information
Algorithm");
            } else {
                System.out.println("Processing Steady State Mutual Information
Algorithm");
            }
            System.out.println("Processing Algorithm : " + algorithmName);
            jobConf.setJobName(algorithmName);

            if(ARACNEDriver.setNumReducers) {
                System.out.println("Setting number of reducers to: " +
ARACNEDriver.numberOfReducers);
                jobConf.setNumReduceTasks(ARACNEDriver.numberOfReducers);
            }
            if(ARACNEDriver.setNumMappers) {
                System.out.println("Setting number of mappers    to: " +
ARACNEDriver.numberOfMappers);
                jobConf.setNumMapTasks(ARACNEDriver.numberOfMappers);
            }
            jobConf.setJarByClass(ARACNEDriver.class);
            jobConf.setMapperClass(FPRMapper.class);
            jobConf.setReducerClass(FPRReducer.class);
            jobConf.setCombinerClass(FPRReducer.class);
            jobConf.setOutputKeyClass(Text.class);
            jobConf.setOutputValueClass(Text.class);

         //  jobConf.setMapOutputKeyClass(Text.class);
         //  jobConf.setMapOutputValueClass(Text.class);

            jobConf.setInputFormat(TextInputFormat.class);
            jobConf.setOutputFormat(TextOutputFormat.class);

            FileInputFormat.setInputPaths(jobConf, new Path(inputFile));
            FileOutputFormat.setOutputPath(jobConf, new Path(outputFile));

            System.out.println("Num Reduce Tasks: " +
jobConf.getNumReduceTasks());
            JobClient.runJob(jobConf);
           System.out.println("Finished all in using " + algorithmName+ " : " +
(System.currentTimeMillis() - start )/1000 + "    second(s) ") ;


    }
}
```

REFERENCES

Margolin AA, Nemenman I, Basso K, Wiggins C, Stolovitzky G, Dalla Favera R, Califano A, ARACNE. *BMC Bioinformatics*. 2006;1:S7.

Zoppoli P, Morganella S, Ceccarelli M. TimeDelay-ARACNE: Reverse engineering of gene networks from time-course data by an information theoretic apprach. *BMC Bioinformatics.* 2010; 11:154.

Emad A Mohammed**,** Behrouz H Far**,** Christopher Naugler**,** Applications of the MapReduce programming framework to clinical big data analysis: current landscape and future trends. *BioData Min.* 2014; 7:22.

Quan Zou, Xu-Bin Li,Wen-Rui Jiang, Zi-Yu Lin,Gui-Lin Li and Ke Chen. Survey of MapReduce frame operation in bioinformatics, *BRIEFINGS IN BIOINFORMATICS.* 2012, 1:11.

Aisling O'Driscoll, Vladislav Belogrudov, John Carroll, Kai Kropp, Paul Walsh , Peter Ghazal, Roy D. Sleator. HBLAST: Parallelised sequence similarity – A Hadoop MapReducable basic local alignment search tool. *Journal of Biomedical Informatics*. 2015; 1:58-62.

Schadt EE, Turner S, Kasarskis A. A window into third-generation sequencing. *Human Molecular Genetics.* 2010; 19:R227-R240

Werner T, Dombrowski S, Zgheib C, Zouein FA, Keen HL, Jurdi M, Booz GW. Elucidating functional context within microarray data by integrated transcription factor focused gene-interaction and regulatory network analysis. *European Cytokine Network*. 2013; 24:75-90.

Filkov V. Identifying Gene Regulatory Networks from Gene Expression Data. 2005 1:27.1-27.8

Filkov V, Skiena S, Zhi J (2002) Analysis techniques for microarray time-series data. J Comput Biol 9: 317-330.

Brazhnik, P, de la Fuente A, Mendes P. Gene networks: how to put the function in genomics. *Trends in Biotechnology.* 2002; 20:467-472.