

國立東華大學資訊工程學系碩士班

碩士論文

指導教授：張瑞雄 博士

一個基於網頁之 MapReduce 資料運算之  
使用者圖形化界面

*A Web-based Graphic User Interface for MapReduce Data Processing*



研究生：史晉銘 撰

中華民國一百年七月

# **A Web-based Graphic User Interface for MapReduce Data Processing**



Adviser: Prof. Ruay-Shiung Chang  
[rschang@mail.ndhu.edu.tw](mailto:rschang@mail.ndhu.edu.tw)

Student: Jin-Ming Shih  
[m9821009@ems.ndhu.edu.tw](mailto:m9821009@ems.ndhu.edu.tw)

A thesis

Submitted in Partial Fulfillment for the Requirements of Master Degree  
in Department of Computer Science and Information Engineering

National Dong-Hwa University

Hualien, Taiwan, Republic of China

July 2011

國立東華大學  
學位論文授權書

※說明※

本授權書請撰寫並簽名後，裝訂於紙本論文書名頁之次頁。

本授權書所授權之論文為立書人在國立東華大學 資訊工程 系所 99 學年度第 2 學期取得 碩士 學位之論文。

論文名稱：(中文) 一個基於網頁之 MapReduce 資料運算之使用者圖形化界面

(English) A Web-based Graphic User Interface for MapReduce Data Processing

指導教授姓名：張瑞雄

學生姓名：史晉銘

學號：69821009

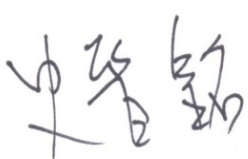
授權事項：

一、立書人具有著作財產權之上列論文全文資料，基於資源共享理念、回饋社會與學術研究之目的，非專屬、無償授權國立東華大學及國家圖書館，得不限地域、時間與次數，以微縮、光碟或數位化等各種方式重製散布、發行或上載網路，提供讀者非營利性質之線上檢索、閱覽、下載或列印。

二、上述數位化公開方式如下：(若未勾選下表，立書人同意視同授權校內、外立即公開。)

校 內	校 外	說 明
<input checked="" type="checkbox"/> 立即公開  <input type="checkbox"/> 於 1 年後公開 <input type="checkbox"/> 於 3 年後公開	<input checked="" type="checkbox"/> 立即公開  <input type="checkbox"/> 於 1 年後公開 <input type="checkbox"/> 於 3 年後公開	未立即公開原因： <input type="checkbox"/> 申請專利(案號： ) <input type="checkbox"/> 因隱私權需要(請指導教授附函說明特殊原因)

三、授權內容均無須訂立讓與及授權契約書，授權之發行權為非專屬性發行權利。依本授權所為之收錄、重製、發行及學術研發利用均為無償。

具有本論文 著作財產權 人共同簽名  (親筆正楷)		日期	中華民國 100 年 7 月 20 日
---------------------------------------	-------------------------------------------------------------------------------------	----	---------------------

## 學位考試委員會審定書

國立東華大學 資訊工程學系碩士班

研究生 史晉銘 君所提之論文

(題目)

一個基於網頁之MapReduce資料運算之使用者圖形  
化界面

A Web-based Graphic User Interface for MapReduce Data  
Processing

經本委員會審查並舉行口試，認為

符合碩士學位標準。

學位考試委員會召集人

之 有 礼

簽章

委 員

志 曙 紹

簽章

委 員

簽章

委 員

簽章

指導教授

張 瑞 雄

簽章

系主任  
(所長)

林 信 鋒

簽章

中華民國 100 年 7 月 20 日

## Acknowledgement

「一億八千九百二十一萬又六千秒，是我在東部那六年的稍縱即逝。」

六年前，我踏入美麗的台東後山求學，四年後，聳立在花蓮花東縱谷的國立東華大學是我下一個兩年的歸所。兩年後之如今，我又將踏離。

「向梅根冶後，幾番嘯傲；杏花村裡，幾度徜徉。」

即使世間萬物都似那飛鴻踏雪泥，而美好的事物也稍縱即逝，但那湛藍蒼穹尾閭滄浪、翠綠山谷長風溪流卻已經深深的刻印在我記憶裡，我很愛也很珍惜在東部的一切，因此我要先感謝這塊土地帶給我人生中這段永難忘懷的體驗。

接著我要感謝我的指導老師：張瑞雄教授，我們才能得有一個優良環境的實驗室、自由開放的研究風氣，老師總是能以那廣博的知識給予我們精準的提示與方向，同時也不忘殷殷地提醒我們要記得隨時吸收新知、增進英文能力。這兩年來非常感謝老師，學生將當以老師風範學習。

另外我也要感謝實驗室的學長們，尤其是日昇、翔升學長平時對我們這些學弟總能不厭其煩的為我們解答各種生活、學業、論文的問題。也感謝執善學長和系辦助理子綺平時對我們的諸多照顧，以及有勞家名學長的幫忙與景行學長在平時報告時給我的指點，才得以完成論文。另外還要感謝我那兩位從大學到現在都是同窗的好友阿誠(葉人誠)和峰偉(莊峰偉)，陪我度過這六年，若這段最後的學生成涯少了你們，將會遜色不少、也少了人可以互相吐槽。還有我另外兩位大學的強者我同學：目前在高應大受重用的阿福(李洪福)與在成功大學的菜頭大大(蔡承勳)，感謝你們平常可以忍受一直被我拉著討論程式問題的騷擾，沒辦法，誰叫你們兩位太厲害了。另外也要感謝實驗室的其他同學、學弟和已經畢業的學長們陪我度過這兩年的研究生生活。

壓軸當然是要感謝我最重要的父母還有家人，由衷感謝老爸老媽辛苦的工作與無私的犧牲奉獻讓我們幾個孩子可以安心的就讀；雖然你們應該會跟我說少三八了。還有老姐和老弟，有了你們我才能得以完成學業、心裡踏實地繼續朝下一個人生階段前進。

另外還有太多要感謝的人事物無法於此書盡謝、言盡謝，那我就謝天吧，感謝老天。

史晉銘 謹致

## 摘要

MapReduce 最早是由 Google 提出的一種用於運算大規模資料的程式模型，許多現實世界中的運算需求都可以使用 MapReduce 來達成。由於 MapReduce 的易於部署性、容錯性，使得它大幅降低了以往要在分散式環境中撰寫程式的困難與複雜度，並且可以輕易的部署在一群由普通電腦組成的叢集上，因此也成為了雲端計算中一個重要的角色。Hadoop 是一個開放原始碼的專案，它實作了 Google 的 MapReduce 以及其他相關的架構供開發者免費使用與開發，Hadoop 現今也已經廣泛地使用在各種應用上，例如：「FaceBook」、「Yahoo」、「Twitter」等知名網站也採用 Hadoop 來分析運算、儲存他們的大量資料。

但許多開發者在開發 MapReduce 專案時，發現要把一些應用單純的拆解成 Map 和 Reduce 是有難度的。另外想要使用 MapReduce 去運算大規模資料時，必須要先有安裝 Hadoop 的機器，並且要熟悉相關開發語言，這對於一般使用者來說並不容易達到。在我們的論文中，我們提出了一種可以簡化 MapReduce 中資料運算的方法，這個方法抽象化了 MapReduce 中資料運算並且提供了一個更高層面的使用方式，我們也設計了一個基於網頁的圖形化使用介面來操作這些我們提出的方法。使用者可以利用任何裝置連上網路，藉由這個網頁圖形化使用者介面達成使用 MapReduce 運算大規模資料的目的。

關鍵字：MapReduce、Hadoop、雲端計算、網頁圖形化使用者介面。

## **Abstract**

MapReduce is a programming model presented by Google for processing and generating large data sets in distributed environments. Many real-world tasks can be implemented by two functions; Mapping and Reducing. MapReduce plays a key role in Cloud Computing, since it decreases the complexity of the distributed programming and is easily developed on large clusters of common machines. Hadoop, an open-source project, is used to implement Google MapReduce architecture. Hadoop is widely used by many applications such as Facebook, Yahoo, Twitter and so on.

However, while using MapReduce to develop applications, it is difficult for developers to decouple an application into its constituent functions of Mapping and Reducing. The difficulties of using MapReduce to process large data sets are considerable for common users. In this thesis, we present methods to simplify MapReduce data processing. Our method can abstract the data processing while using MapReduce. We also create a Web-based Graphic User Interface in our architecture to implement the presented methods. Through it, users can access the Web-based GUI through the Internet on any device to process large data sets using MapReduce.

*Keywords: MapReduce, Hadoop, Cloud Computing, Web-based Graphic User Interface.*

## Table of Contents

Acknowledgement .....	I
摘要 .....	II
Abstract .....	III
Table of Contents .....	IV
List of Figures.....	VI
List of Tables .....	VII
Chapter 1 Introduction .....	1
1.1 Preface .....	1
1.2 Motivation .....	2
1.3 Main Contribution .....	4
1.4 Thesis Outline .....	4
Chapter 2 Background .....	5
2.1 Cloud Computing .....	5
2.2 Cloud Computing Technologies .....	6
2.2.1 GFS (Google File System) .....	7
2.2.2 Google MapReduce.....	8
2.2.3 Hadoop .....	9
2.2.4 HDFS (Hadoop Distributed File System) .....	9
2.2.5 HDFS Architecture and Design .....	10
2.2.6 Hadoop MapReduce .....	11
Chapter 3 Related Works .....	13
3.1 Cascading.....	13
3.1.1 Cascading Pipe assemblies .....	14
3.1.2 Cascading Operations .....	15
Chapter 4 The Proposed Web-based GUI for MapReduce Data Processing .....	19
4.1 Target-Value-Action .....	19



4.2 Operation Components .....	23
4.3 Layer .....	28
4.4 Container .....	29
4.5 System Architecture .....	30
Chapter 5 Case Study and Implementation .....	35
5.1 Distributed Grep .....	35
5.2 Count of URL Access Frequency .....	36
5.3 Reverse Web-Link Graph .....	36
5.4 Pairwise Document Similarity .....	37
5.4.1 Pairwise Similarity Algorithm .....	35
5.4.2 Pairwise Document Similarity in Our Proposed Methods .....	41
5.5 Track Statistic of Last.fm .....	43
Chapter 6 Conclusion and Future Work .....	51
Reference .....	53

## List of Figures

Figure 1. MapReduce data flow .....	2
Figure 2. The date flow in word counting .....	3
Figure 3. HDFS Architecture .....	10
Figure 4. Pipe flow in Cascading .....	14
Figure 5. Pipe assemblies .....	14
Figure 6. The operations in Cascading .....	15
Figure 7. Pipe flow example .....	16
Figure 8. “WordCount” in Cascading .....	16
Figure 9. WordCount dataflow in MapReduce .....	20
Figure 10. WordCount in Target-Value-Action .....	20
Figure 11. The Inverted Index in MapReduce .....	21
Figure 12. The Inverted Index dataflow in MapReduce .....	22
Figure 13. Inverted Index in Target-Value-Action .....	22
Figure 14. The operations of Target-Value-Action .....	23
Figure 15. Input and output path setting in Hadoop MapReduce .....	28
Figure 16. The dataflow of Layers .....	29
Figure 17. Container example .....	30
Figure 18. The proposed system architecture and workflow .....	31
Figure 19. Container and Layer information in a data serial .....	31
Figure 20. The processing flow in TVA Processor .....	33
Figure 21. Distributed Grep data flow in MapReduce .....	35
Figure 22. Computing the Pairwise Document Similarity of 3 documents .....	38
Figure 23. Formalize the columns for Container_PDS .....	42
Figure 24. The pairwise similarity implement screen of our Web-based GUI.....	43
Figure 25. Fitting the output to the Container_Last.fm .....	49
Figure 26. The Last.fm track statistic implement screen of our Web-based GUI .....	50

## **List of Tables**

Table 1. The Input types of operation components .....	24
Table 2. The Output types of operation components .....	25
Table 3. The Merge examples .....	26
Table 4. The Merge_Group_Sum examples .....	27
Table 5. The example details of Target-Value-Action. ....	37
Table 6. The pairwise similarity details in our presented methods. ....	41
Table 7. The processed data of track statistic .....	45
Table 8. The pairwise similarity details in our presented methods. ....	49

## **Chapter 1 Introduction**

### **1.1 Preface**

In recent years, Cloud Service and Cloud Computing have been developing rapidly. Cloud Service is based on Cloud Computing technology that allows users to enjoy traditional IT functions with lower-cost and more convenience. Traditionally, when a company decides to establish a traditional IT system, it can only predict how much its IT systems will cost. When an IT system starts to work, the resources of the IT system may actually be wasted. This resource utilization problem can be solved by using Cloud Computing technology or Cloud Service. The resources on the Cloud are on-demand and shared, so the resource utilization will be increased.

MapReduce [1] was introduced by Google as a simplified data processing technology for use on large clusters. Google also developed BigTable [2], which is non-Relational Database, and GFS (Google File System) [3]. Subsequently, the ASF (Apache Software Foundation) presented an open-resource Cloud Computing project, “Hadoop” [10]. The Hadoop projects consist of Hadoop MapReduce [10], HDFS (Hadoop Distributed File System) [4], HBase [11] and so on, all of which make developing Cloud Computing and Cloud Service more convenient.

Cloud Service is designed for people to use, and should be convenient and Straightforward to use. Cloud Computing has not only powerful computation capabilities, but also offers reliable distributed storage. Users located anywhere can use any devices at anytime to demand the computing and storage power of a cloud through the Internet. A device used to access Cloud Computing only needs to be equipped with a

browser. For example, take the case where a staff member has just lost his laptop, and is going to a meeting. There is a lot of data he needs, and the work environment is in his laptop, but he can still get back the work environment and all the data he needs on any laptop, immediately, if he uses some kind of on-demand Cloud Service and has stored his data in the Cloud.

## 1.2 Motivation

MapReduce is a programming model for processing a large amount of data to generate useful information. In MapReduce, the complexity of distributed programming is decreased substantially by hiding details from the distributed computing system. The MapReduce process shown in Figure 1 splits input data into a lot of Map operation nodes, and produces a set of output key/value pairs. Reduce operation nodes group together all intermediate values with the same intermediate key.

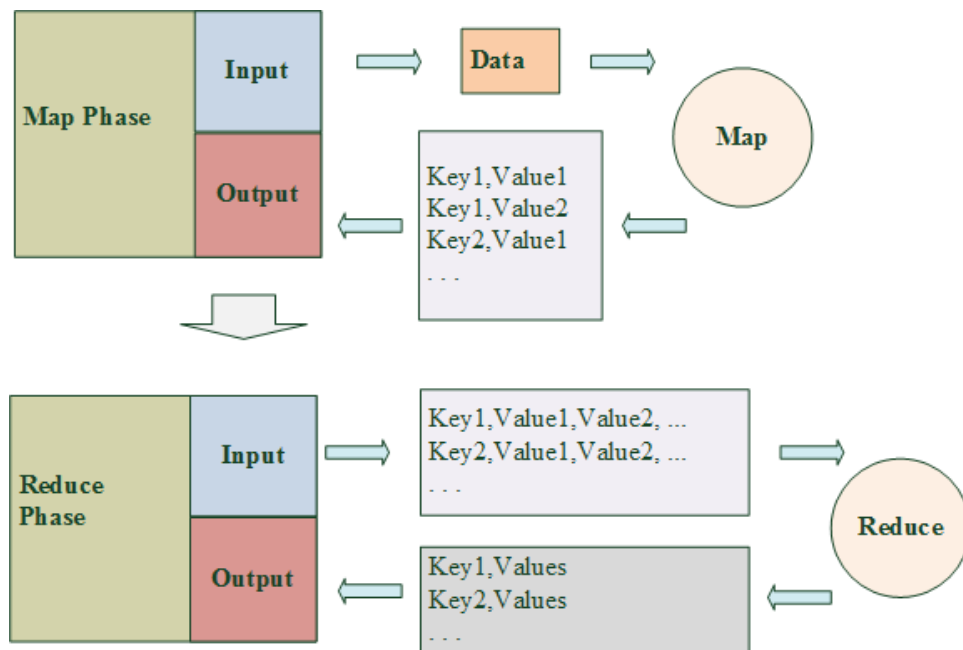


Figure 1. MapReduce data flow.

Figure 2 shows an example of simple word counting. The example counts the number of times a word appears in a sentence “Chase the water running from sky...” It maps the sentence into chunks and reduces the count results from all chunks.

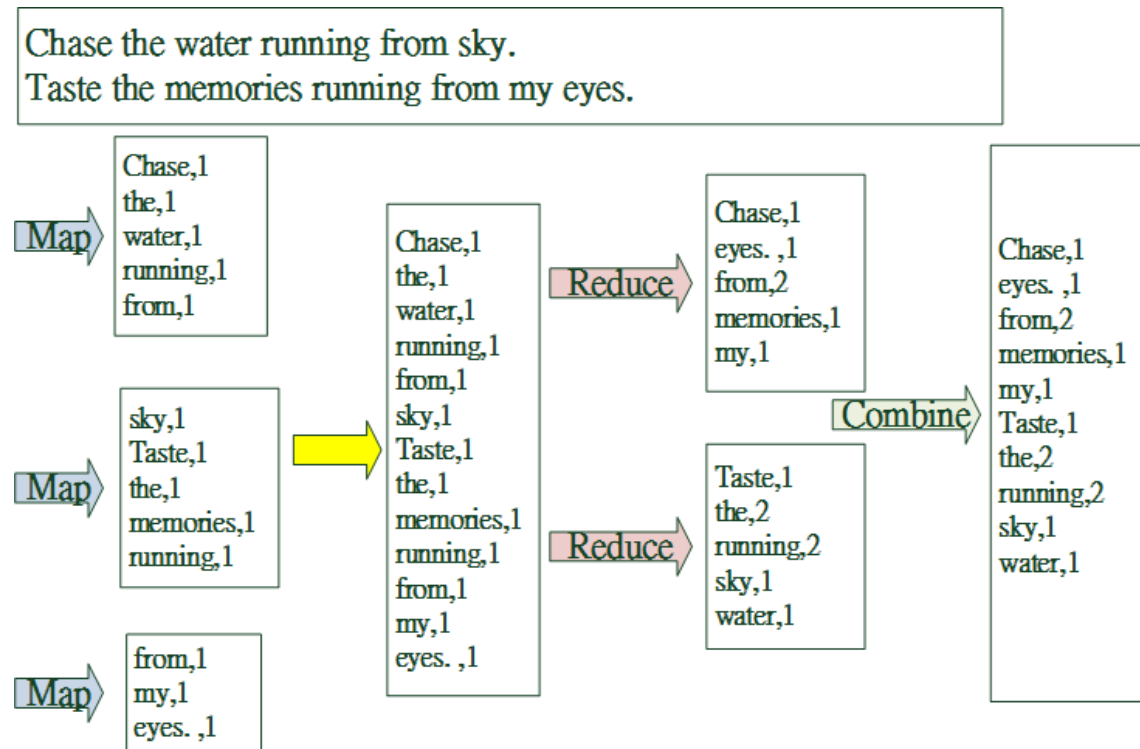


Figure 2. The data flow in word counting.

MapReduce computation is used in many applications [1,6,8,9,12] such as Distributed Grep, Count of URL Access Frequency, Reverse Web-Link Graph, Term-Vector per Host, Inverted Index, Distributed Sort, Security Enhanced DNS Group, Journey Dynamics and other real world applications. It is discussed in detail in Chapter 5. Even in Google’s new search engine technology, Percolator [7] in the new search engine and Caffeine [22], Google still use MapReduce to analyze and produce the initial search index.

Although the MapReduce model can simplify the complexity of programming for distributed computing, MapReduce is still not easy to implement. If users want to use MapReduce to compute large-scale data, they cannot do it on the computer without installing Hadoop first. Users can only use certain kinds of Cloud Services that supply limited MapReduce functions to process their large-scale data.

### **1.3 Main Contribution**

In this thesis, we present a Web-based GUI for MapReduce Data Processing. The GUI can let users design their MapReduce workflow intuitively and conveniently. Our proposed data processing method is suitable for users who cannot design programs. Therefore, the use of MapReduce data processing is simplified, so that users can easily process large data sets using MapReduce through the Web-based GUI, on any device. The development time will be decreased, and the MapReduce data processing using our method can thus be more efficient.

### **1.4 Thesis Outline**

The remainder of this thesis is organized as follows: Chapter 2 gives an overview of Cloud Computing. Chapter 3 refers to Cascading, which is a more abstract API to MapReduce. Chapter 4 introduces the data processing methods for MapReduce GUI which we propose. In Chapter 5, we study some real-world cases, implemented using our method. Chapter 6 concludes the thesis and summarizes future work.

## **Chapter 2 Background**

### **2.1 Cloud Computing**

According to the National Institute of Standards and Technology [18] definition, there are five essential characteristics in Cloud Computing: On-demand self-service, Broad network access, Resource pooling, Rapid elasticity, and Measured service. There are currently four deployment models: Public cloud, Private cloud, Hybrid cloud, Community cloud. Three service models are available: Soft as a Service (SAAS), Platform as a Service (PAAS) and Infrastructure as a Service (IAAS). Our methods are built on PaaS, and provide a SaaS.

The PaaS provides users with a specific platform with which to develop software projects. Users do not need to be concerned about what kind of hardware they use. One limitation of PaaS is the programming language provided by the PaaS provider. The Interoperability Portability between different PaaS providers is also an issue. Google App Engine [14], Microsoft Azure Platform [15], and Heroku [16] are all well-known PaaS providers. Microsoft Azure Platform is a Cloud-Based System on Windows Server 2008. The platform hides the hardware development and network architecture. There are some differences between traditional operating systems and Microsoft Azure. Microsoft Azure is a collection of the middleware in Datacenter, Virtual Machine Management and Virtual Machine Units. Microsoft Azure provides an IDE to use for developing and publishing their tested application package quickly. Microsoft Daytona [13] is an iterative MapReduce runtime for Windows Azure. It allows developers to develop the data analytics algorithms using MapReduce on the Windows Azure platform. Users can



configure the number of virtual machines for the deployment.

A Cloud Service provides specific software to clients. The client is not responsible for software maintenance. Many SaaS have a characteristic whereby users can share or edit common data, such as Dropbox, Ragic Builder (a trial balance online service) and Google Doc. Salesforce [17] is another well-known SaaS, and Customer Relationship Management (CRM) exists for Salesforce products. This CRM is a successful SaaS, with many international enterprises being Salesforce's CRM customers, such as DELL, Fujitsu Network Communications, and ACER. Traditional CRM or ERP systems must use specific versions of Java, Internet Explorer and OS to be stable. In contrast, some small businesses can have their own complete CRM systems by using Salesforce's CRM product.

Some of the characteristics of a good Cloud Service are summed up in the acronym 4A2S, which denotes the following: anytime, any authorized person, anywhere, any device, same network service and same result. For users, this means they can obtain the same service and results from anywhere, at anytime, and any authorized person can use any device. For service providers, this means they can use all resources fully. Combine Web2.0 is a nice idea for achieving 4A2S. Besides a cross-platform, there are many other mature technologies which can be used, such as PHP, MySQL, AJAX, .NET, JSP and so on. Combine Web2.0 It will create a variety of useful services by choosing the appropriate PaaS. Therefore, SaaS are suitable for developer design by virtue of their originality.

## **2.2 Cloud Computing Technologies**

Distributed computing, distributed file systems, and virtualization are the major

Cloud Computing technologies. We will now introduce the technologies of distributed computing and distributed file systems.

### **2.2.1 GFS (Google File System)**

GFS is a distributed storage technology which is designed for performance, scalability, reliability, and availability. Its design has been driven by key observations of Google's application workloads and technological environment. Because the GFS uses a lot of inexpensive machines, it is possible that there some errors may occur at anytime, such as application bugs, operating system bugs, human errors, and the failures of disks, memory, connectors, networking, and power supplies. The fault tolerance is a key consideration with GFS, too. The scale of data Google handles is large, and Multi-GB files are common. As a result, I/O operation and block sizes have to be revisited. GFS consists of a signal Master and some Chunk Servers. The data are divided into fixed-size chunks, and each chunk has a 64bit chunk handle. Each chunk is replicated on multiple Chunk Servers. The Master is responsible for managing all Chunk Servers, including functions such as chunk lease management, garbage collection of orphaned chunks, and chunk migration between chunk servers. The Master communicates periodically with each Chunk Server via Heart Beat messages, so that Master can collect system state.

The client translates the file name and byte offset specified into a chunk index by the application using the fixed chunk size within the file. Then, it sends the Master a request which contains the file name and chunk index. The Master replies with the chunk handle and locations of the replicas. The Client, using the chunk index and file name as the key, caches this information. The Client sends a request to the closest Chunk Server, and then gets the chunk handle and byte range within that chunk. No

further Client-Master interaction is required to read the same chunk until the cached information expires or the file is reopened. Finally, the Chunk Server sends the files to Client.

### **2.2.2 Google MapReduce**

MapReduce is a programming model Google presented for processing and generating large data sets. MapReduce decreases the complexity of distributed programming substantially, and hide the messy details resulting from parallelization, fault-tolerance, data distribution and load balancing. MapReduce consists of two functions; namely, Map and Reduce. The MapReduce computation takes a set of input key/value pairs, and Map produces a set of output key/value pairs. Reduce groups together all intermediate values with the same intermediate key. Programmers do not need to understand the details of parallelization, fault-tolerance, data distribution and load balancing in MapReduce. Thus, programmers can be focused on designing Map and Reduce to solve problems.

MapReduce consists of a single Master and many Workers. The Master assigns tasks to Workers. MapReduce first splits the input files into M pieces (typically of 16 megabytes to 64 megabytes per piece), and starts up many copies of the program on a cluster of machines. Then Master assigns M Map tasks or R tasks to the Workers. If a Worker is assigned a Map task, the Worker reads the contents of the corresponding input split and parses key/value pairs out of input data. The Worker passes each pair of the input data to the user-defined Map function. The middle-results will be buffered in memory. The buffered pairs results are written to a local disk periodically and the locations are passed back to the Master. When the Master notifies the Reducer about these locations, the Reducer uses RPC to read the buffered data from the local disks of

the Map Workers. The Reducer then sorts the data so that all occurrences of the same key are grouped together. Here is a WordCount example in pseudo-code:

```
map(String key, String value):  
  
// key: document name    // value: document contents  
  
for each word w in value:  
  
EmitIntermediate(w, "1");  
  
reduce(String key, Iterator values):  
  
// key: a word    // values: a list of counts  
  
int result = 0;  
  
for each v in values:  
  
result += ParseInt(v);  
  
Emit(AsString(result));
```

In Map function, each word is given value and plus a count 1 of occurrences. The Reduce function sums all counts generated for a particular word. Following this logic, MapReduce computation can be used in many applications.

### **2.2.3 Hadoop**

Hadoop is an open-source project which implements Google MapReduce architecture. There are many sub-projects of Hadoop, such as: Hadoop MapReduce, HDFS (Hadoop Distributed File System), Pig [19], HBase and so on.

### **2.2.4 HDFS (Hadoop Distributed File System)**

HDFS is a distributed file system similar to GFS. HDFS can operate on a lot of inexpensive machines. Hadoop is also highly fault-tolerant and offers high throughput. The assumptions and goals of Hadoop are: Hardware Failure, Streaming Data Access,

Large Data Sets, Simple Coherency Model, “Moving Computation is Cheaper than Moving Data”, Portability Across Heterogeneous Hardware, and Software Platforms.

### 2.2.5 HDFS Architecture and Design

HDFS architecture is a master/slave design. Figure 3 shows the HDFS architecture. There is a single NameNode and a number of DataNodes. The NameNode plays a master role in an HDFS cluster. The NameNode manages the file system namespace and regulates client access to files. In HDFS, a file is split into one or more blocks, and stored in a set of DataNodes. The NameNode executes file operations such as open, close and rename. The NameNode is also responsible for determining the mapping of blocks to DataNodes. DataNodes are responsible for serving read and write requests from clients. The NameNode instructs DataNodes to create blocks, delete blocks and replicate blocks.

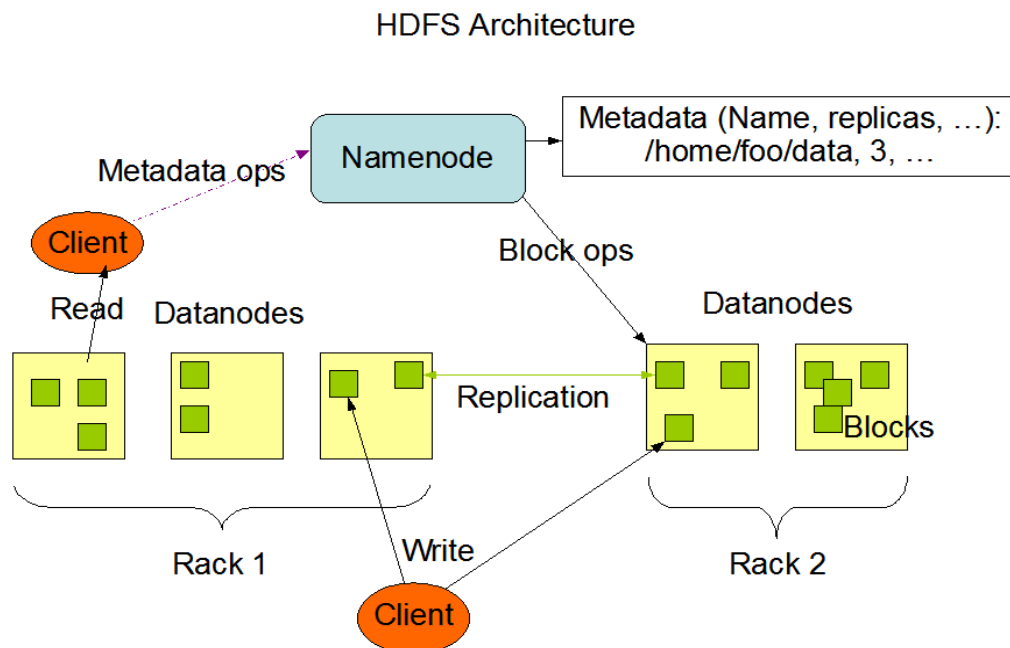


Figure 3 [4]. HDFS Architecture.

### **2.2.6 Hadoop MapReduce**

Hadoop MapReduce implements Google MapReduce. Hadoop MapReduce is a software framework for processing large-scale data in-parallel on large clusters of commodity hardware which is like Google MapReduce. There are a set of map tasks and reduce tasks in any given job. Map tasks split the input data-set into chunks in a completely parallel manner, and the outputs are sorted by framework. The framework is responsible for scheduling tasks, monitoring tasks and re-executing failed tasks. The Hadoop MapReduce framework consists of a single master (JobTracker) and many slaves (TaskTracker). Job Tracker is responsible for scheduling, monitoring and re-executing all job tasks on slave machines. TaskTracker executes the tasks assigned by the JobTracker. JobTracker corresponds to the Master, and TaskTracker corresponds to the Worker in Google MapReduce.



## Chapter 3 Related Works

Even though MapReduce provides two powerful simple functions, Map and Reduce for user building large-scale data solutions, it is difficult for developers to decouple an application into its component functions of mapping and reducing. We need more abstract functions to help us build MapReduce solutions for real-world cases. Cascading [20] is an open source Java library, and API offers an abstract layer for MapReduce. The Cascading processing model allows developers to rapidly develop complex data processing applications on Hadoop.

### 3.1 Cascading

In Cascading, the key-value pairs of MapReduce are replaced by “Field Name” and “Tuples”. In addition, there are no Map and Reduce functions in Cascading. More high-layer abstract functions are implemented, such as “Function”, “Filter”, “Aggregator” and “Buffer”.

Cascading simplifies the data flow in the MapReduce model with “Pipe” and by abstracting key-value pairs into Tuples with field names. Every Tuple has its own field name, and each Tuple operates user self-definition functions in each pipe as shown in Figure 4.

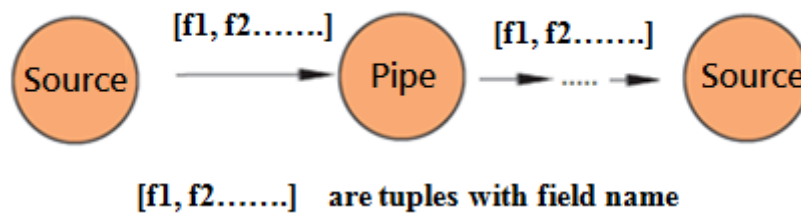


Figure 4. Pipe flow in Cascading.



### 3.1.1 Cascading Pipe assemblies

Figure 5 shows the Pipe assemblies definition of five pipe types: Each, GroupBy, CoGroup, Every, and SubAssembly.

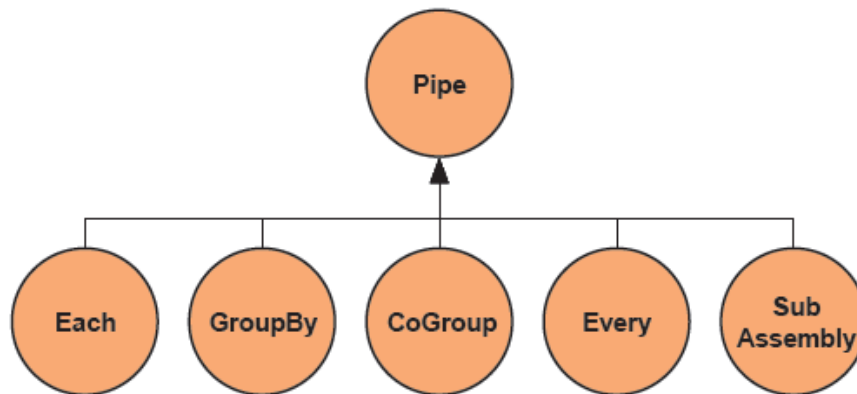


Figure 5. Pipe assemblies.

Each pipe applies operations such as “Function” or “Filter” to each Tuple which passes through it. The GroupBy pipe and CoGroup pipe are like SQL group and SQL join. Every pipe applies an operation, “Aggregator” or “Buffer”, to every Tuples group. The SubAssembly pipe allows for nesting of pipe assemblies into a larger pipe assembly.

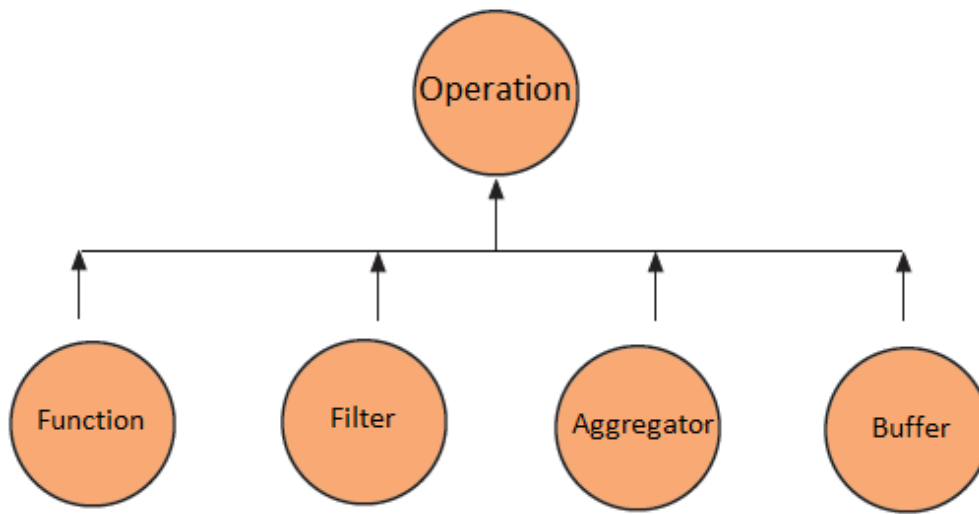


Figure 6. The operations in Cascading.

### 3.1.2 Cascading Operations

As mentioned above, Cascading replaces Map and Reduce functions with other high-layer operations such as “Function”, “Filter”, “Aggregator” and “Buffer”, as shown in Figure 6.

Function operates a single argument Tuple, and returns zero or more result Tuples. Filter determines whether the current Tuple in the Tuple stream should be discarded or not. The Even Function could achieve the same operation as Filter, but Filter is optimized for it. Aggregator and Buffer expect a set of argument Tuples in the same grouping and operate the same operations such as Sum, Count, Average, Max and Min. Figure 7 shows a Pipe flow example. Figure 8 shows the “WordCount” in Cascading code.

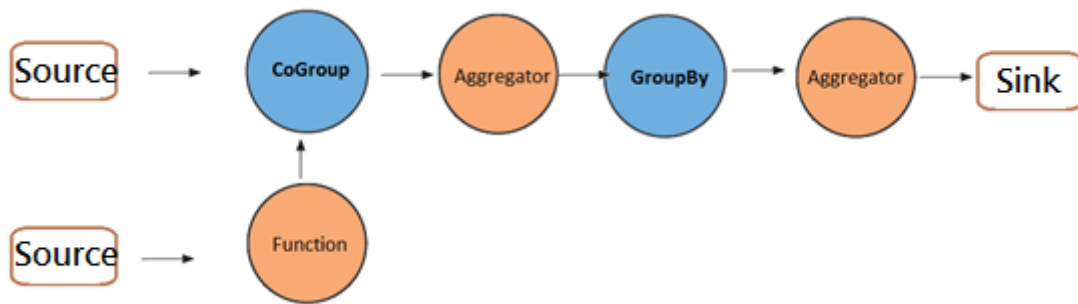


Figure 7. Pipe flow example.

```

// define source and sink Taps.
Scheme sourceScheme = new TextLine( new Fields( "line" ) );
Tap source = new Hfs( sourceScheme, inputPath );

Scheme sinkScheme = new TextLine( new Fields( "word", "count" ) );
Tap sink = new Hfs( sinkScheme, outputPath, SinkMode.REPLACE );

// the 'head' of the pipe assembly
Pipe assembly = new Pipe( "wordcount" );

// For each input Tuple
// parse out each word into a new Tuple with the field name "word"
// regular expressions are optional in Cascading
String regex = "(?<!\pL) (?=\pL) [^ ]*(?<=\pL) (?!\pL)";
Function function = new RegexGenerator( new Fields( "word" ), regex );
assembly = new Each( assembly, new Fields( "line" ), function );

// group the Tuple stream by the "word" value
assembly = new GroupBy( assembly, new Fields( "word" ) );

// For every Tuple group
// count the number of occurrences of "word" and store result in
// a field named "count"
Aggregator count = new Count( new Fields( "count" ) );
assembly = new Every( assembly, count );

// initialize app properties, tell Hadoop which jar file to use
Properties properties = new Properties();
FlowConnector.setApplicationJarClass( properties, Main.class );

// plan a new Flow from the assembly using the source and sink Taps
// with the above properties
FlowConnector flowConnector = new FlowConnector( properties );
Flow flow = flowConnector.connect( "word-count", source, sink, assembly );

// execute the flow, block until complete
flow.complete();

```

Figure 8[20]. “WordCount” in Cascading.

On the surface, Cascading is more complex than MapReduce. There are many pipe types and operations, but Cascading is actually easier in terms of thinking about how to solve real-world problems in programming, compared with MapReduce. However, Cascading is not convenient enough because there are simply too many concepts of data processing involved. Our approach focuses on developing more high-layer, simpler, and more abstract methods, that hide MapReduce programming details. In our model, users control the Web-based GUI, as a substitute for programming, to process MapReduce. Therefore, users can solve real-world data processing problems directly and conveniently.



## **Chapter 4 The Web-based GUI for MapReduce Data Processing**

The proposed Web-based GUI for MapReduce Data Processing presents an intuitive data processing method. In MapReduce, the dataflow of the Map phase, where input data is split by Map operation nodes, allows for user's definition. The Map output key/value pairs will be grouped in the Reduce Phase or the Combine Phase. Developers have to schematize the programming details of key/value pairs operations in the Map phase and the Reduce phase.

In the Web-based GUI for MapReduce Data Processing which we propose, our methods decrease the complexity of MapReduce data processing because the proposed methods can abstract the data processing when using MapReduce. The Map phase and Reduce phase are hidden by "Target-Value-Action" in each "Layer". There is a "Container" which integrates the different formats of different outputs. In our system, the proposed method's input will be transformed into MapReduce. Even users who cannot program can directly use our proposed methods to process large data sets using MapReduce.

After users uploaded data to HDFS and the initialized columns of Container, they can start using the Web-based GUI to process data in MapReduce. Therefore, users have a simpler and more direct way to process data when using MapReduce.

### **4.1 Target-Value-Action**

Target-Value-Action is a direct function, and represents an original idea in data processing. It still consists of Map and Reduce functions but is rather more abstract. The

Target-Value-Action is a convenient method of integrating the processing between Map and Reduce functions. Most of MapReduce's job can be directly expressed to the Target-Value-Action concept we propose: "users just choose some objects as targets; give each object a value, and take actions". In the choosing of targets, users can choose a part of the data they want to process and set it as the target. In the choosing of value, if the results which users want involve integer operations (such as counting or sum), users can give the integer value to the target. If the result users want involves text operations, such as merge, users can give the text a value such as a file name and assign others column to the target. Users can use the operation components to filter or process the values with the same target in the choosing of action. For example, Figure 9 shows the WordCount dataflow.

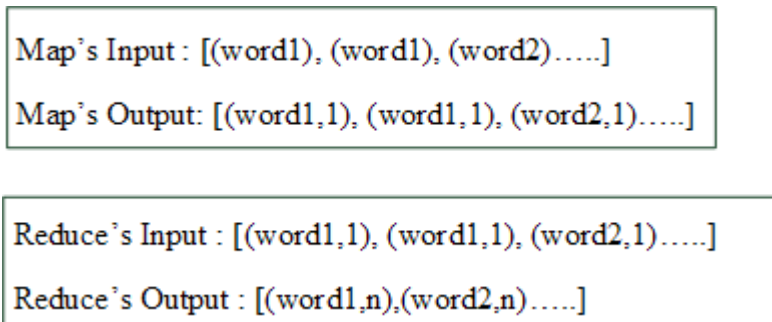


Figure 9. WordCount dataflow in MapReduce.

In Target-Value-Action, the operation of WordCount will be expressed as shown in Figure 10.

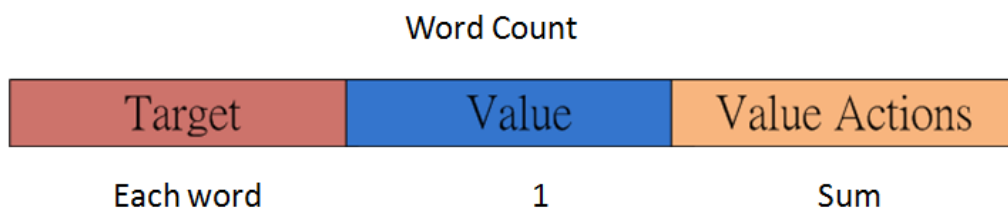


Figure 10. WordCount in Target-Value-Action.

The logic involves choosing each word as the Target, Value is “1” and Action is “Sum”. Another example can be seen in an inverted index, which is an index data structure storing the mapping list from content to its location, where the locations are in a database or in a set of documents. In Figure 11, there are two documents Doc1 and Doc2.

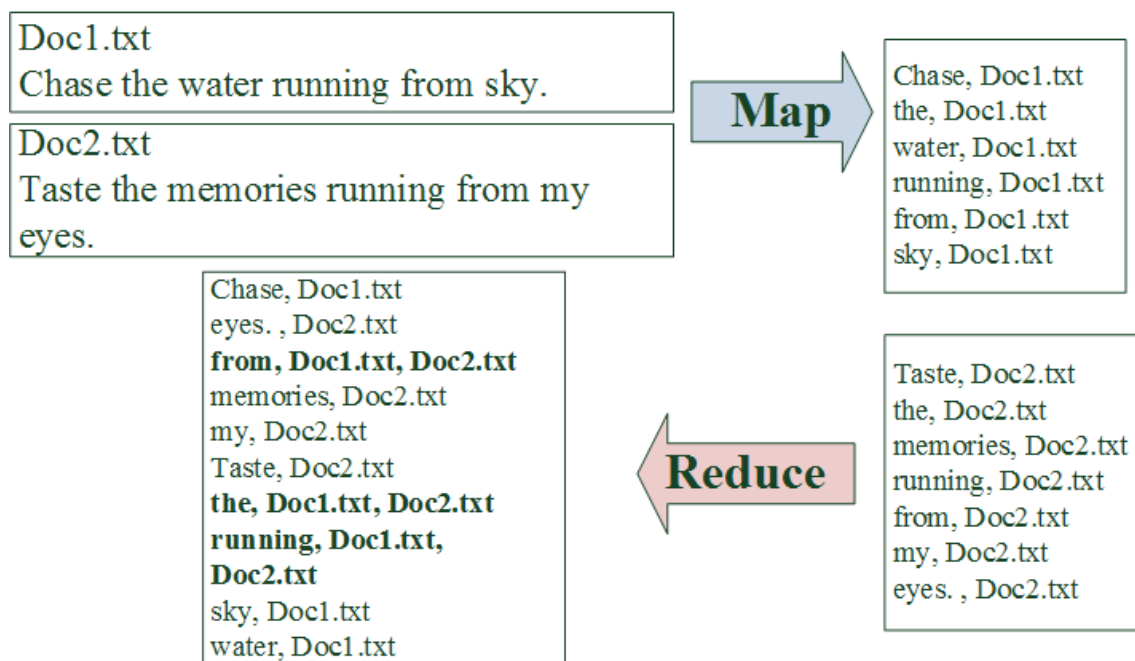


Figure 11. The Inverted Index in MapReduce.

The first step is mapping each word to its locations for each document, then outputting the results. The next step involves reducing and combining the map output, which then gives an inverted index result. Figure 12 shows the dataflow.



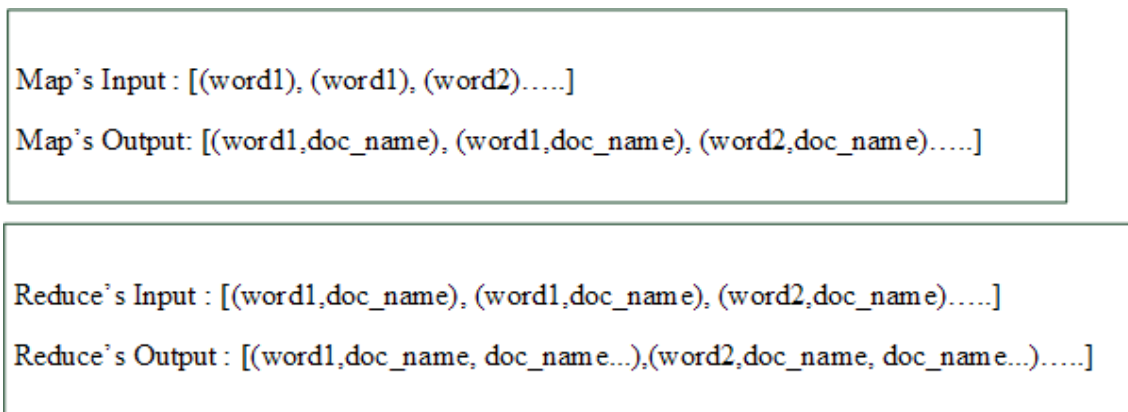


Figure 12. The Inverted Index dataflow in MapReduce.

As Figure 13 illustrates, the inverted index in Target-Value-Action involves choosing each word as the Target and giving its document name as the Value; finally, we can get the same result after the Merge Action.

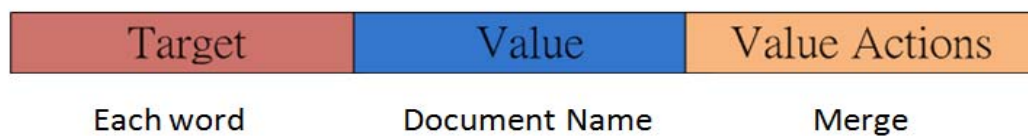


Figure 13. Inverted Index in Target-Value-Action.

Target and Value expect a single column or a set of columns. Users can assign any columns as the Target and Value to process. A set of Target-Value-Action can thereby combine a complex Job. Therefore, Target-Value-Action decreases the complexity of MapReduce's data processing logic.

4.2 Operation Components

As discussed above, most Map and Reduce operations can be integrated using many operation components with different data types. There are two main data types in MapReduce: Integer and Text. “Target” and “Value” are extracted from the input data. The operation components we propose are shown in Figure 14.

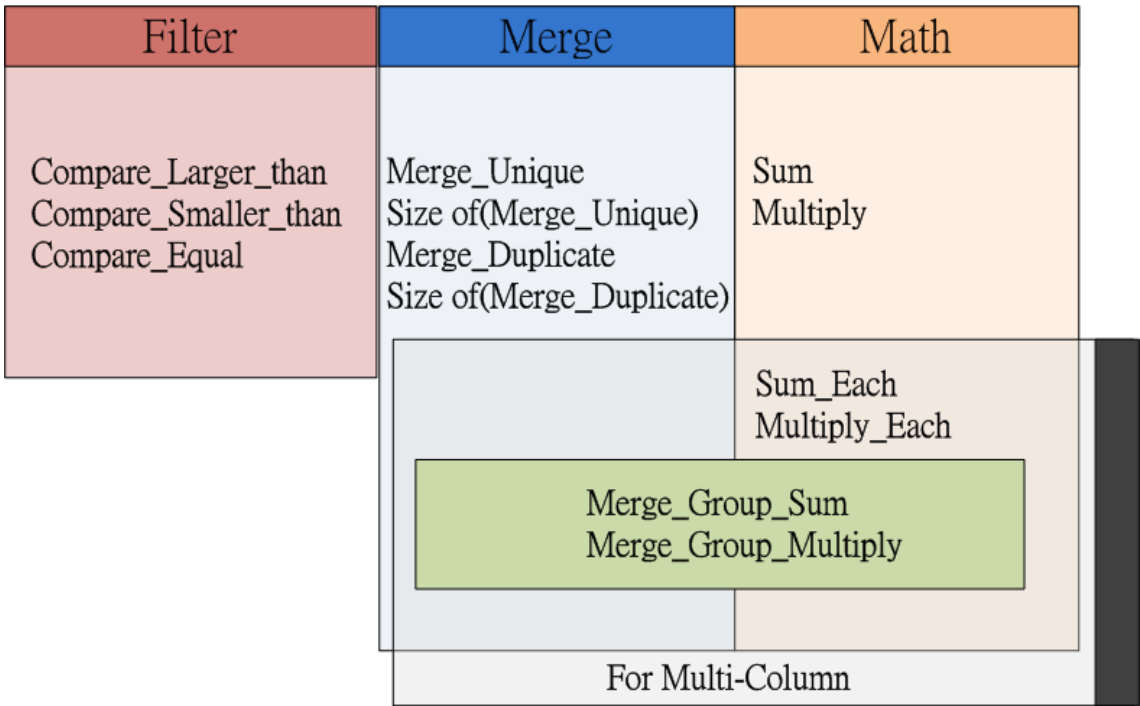


Figure 14. The operations of Target-Value-Action.

Each operation component we propose expects a specific input type and a specific return type, as shown in Table 1 and Table 2.

<i>Input Type</i>	<b>Integer</b>	<b>Text</b>	<b>Regex</b>	<b>User's Input</b>
Compare_Larger_than	•			•
Compare_Smaller_than	•			•
Compare_Equal	•		•	•
Merge_Unique		•		
Size of (Merge_Unique)		•		
Merge_Duplicate		•		
Size of (Merge_Duplicate)		•		
Sum	•			
Multiply	•			
Sum_Each	•			
Multiply_Each	•			
Merge_Group_Sum	•	•		
Merge_Group_Multiply	•	•		

Table 1. The Input types of operation components.

<i><b>Return Type</b></i>	<b>Integer</b>	<b>Text</b>
Merge_Unique		•
Size of (Merge_Unique)	•	
Merge_Duplicate		•
Size of (Merge_Duplicate)	•	
Sum	•	
Multiply	•	
Sum_Each	•	
Multiply_Each	•	
Merge_Group_Sum	•	•
Merge_Group_Multiply	•	•

Table 2. The Output types of operation components.

The Compare operations expect the integer or regular expression from the user's input. The Compare operations can filter the unnecessary rows from the input data. Users can assign a part of a column as the Value. If the assigned columns of the row are not conforming to the filter rules of users in the Action, this row will not be output to the Reduce phase. The Compare operations are only responsible for filtering data in the dataflow, so there is not any return value.

The Merge operation components expect the text input, and Merge operations will combine the text input. There are four different Merge operations: Merge\_Unique, Merge\_Duplicate, Size of (Merge\_Unique) and Size of (Merge\_Duplicate). Size of (Merge\_Unique) and Size of (Merge\_Duplicate) will return the amount of the combined

collection. Table 3 contains Merge examples:

<b>Input:</b>	<b>Operations</b>	<b>Return</b>
A	Merge_Unique	(A,B,C,D)
A	Merge_Duplicate	(A,A,B,C,C,D)
B	Size of (Merge_Unique)	4
C	Size of (Merge_Duplicate)	6
C		
D		

Table 3. The Merge examples.

The Math operation components expect an integer input. Sum\_Each and Multiply\_Each expect multi-column input, then operate on each column and output all columns. Merge\_Group\_Sum and Merge\_Group\_Multiply expect a column's pair input in the Value, where one is the text and the other one is the integer. The text one will be merged, and the value one will be operated on.

Therefore, Merge\_Group\_Sum and Merge\_Group\_Multiply output the groups which consist of text results in group (1) and integer results in group (2). Next, Target-Value-Action's Target can only accept group (1), and Value accepts only group (2). If the number of a column's pairs with same Target of Merge\_Group\_Sum and Merge\_Group\_Multiply are more than two, the column pairs will be operated on accordance with the following rule:

***If there are N columns pairs with the same Target:***

***(Target, key 1, value 1), (Target, key 2, value 2), ..... (Target, key N, value N)***

***For (int i=1; i<N; i++) {***

***For (int j=i+1; j<N; j++) {***

***Merge (key i, key j);***

***Sum (value i, value j); // Multiply (value i, value j);***

***}***

***}***

If there is only one column's pair with same Target, the single pair will be ignored.

Table 4 is an example of Merge\_Group\_Sum:

<b>Word (Target)</b>	<b>User (Value)</b>	<b>Score (Value)</b>
A	User1	2
A	User3	3
B	User1	4
B	User2	0
B	User3	1
<b>After Merge_Group_Sum , the Outputs [Group(1),Group(2)] are:</b>  [(User1,User3) , 5]  [(User1,User2) , 4], [(User1,User3) , 5], [(User2,User3) , 1]		

Table 4. The Merge\_Group\_Sum examples.

### 4.3 Layer

In MapReduce, Input and Output is a Folder Path, as shown in Figure 15.

```
01  user_input="/user/hadoop/input/";
02  user_output="/user/hadoop/output/";
03  Path inPath = New Path(user_input);
04  Path outputPath = New Path(user_output);
05  FileInputFormat.setInputPaths(job,user_input);
06  FileOutputFormat.setOutputPath(job,user_output);
```

Figure 15. Input and output path setting in Hadoop MapReduce.

In Hadoop, the developer defines each Job's input and output path in FileInputFormat and FileOnputFormat class. Developers can set another Job's output as input. If there is a Job which depended on other Jobs, for example, Job-A's and Job-B's output is job-C's input. In this way, we consider them as chaining MapReduce Jobs. In our Web-based GUI, Chaining MapReduce Jobs is controlled by "Layer" which we propose. Each Layer consists of single instance or a set of Target-Value-Action and Container. Layers can accept other Layers' output or original data as the input:

***$i = 1$ , Layer  $i$ 's input is origin data.***

***When  $i > 1$ , Layer  $i$ 's input is origin data or Layer  $j$ 's Output, where  $j < i$***

Figure 16 shows the dataflow in Layers, the input of Layer1 and Layer2 is origin data. Layer 3 can assign the output from Layer1 and Layer2 as the input. In our GUI, users directly assign the input by choosing the output from other Layers.

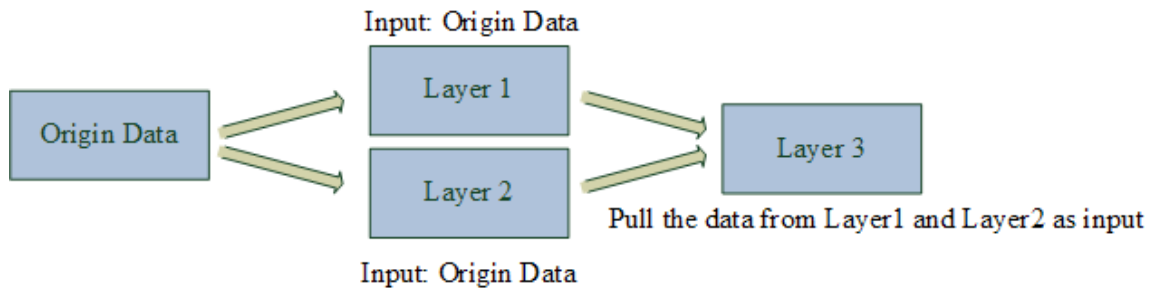


Figure 16. The dataflow of Layers.

Therefore, the Layer concept in Web-based GUI makes chaining MapReduce Jobs viewable and easy to control.

#### 4.4 Container

We propose the “Container” which integrates output in different formats into a uniform format with multi-columns. It can make the output dataflow viewable, which is convenient when operating the inputs from different Layers with different formats. The maximum amount of Containers in each Layer is two; the default Container and the output Container. The origin data is split into the user defined columns in the default Container. Figure 17 shows the dataflow of the Container. Even though there are two different output formats, the two outputs can be fitted for the same output format in the same output Container. Users can assign any column from the operated result to the output Container in each Layer.



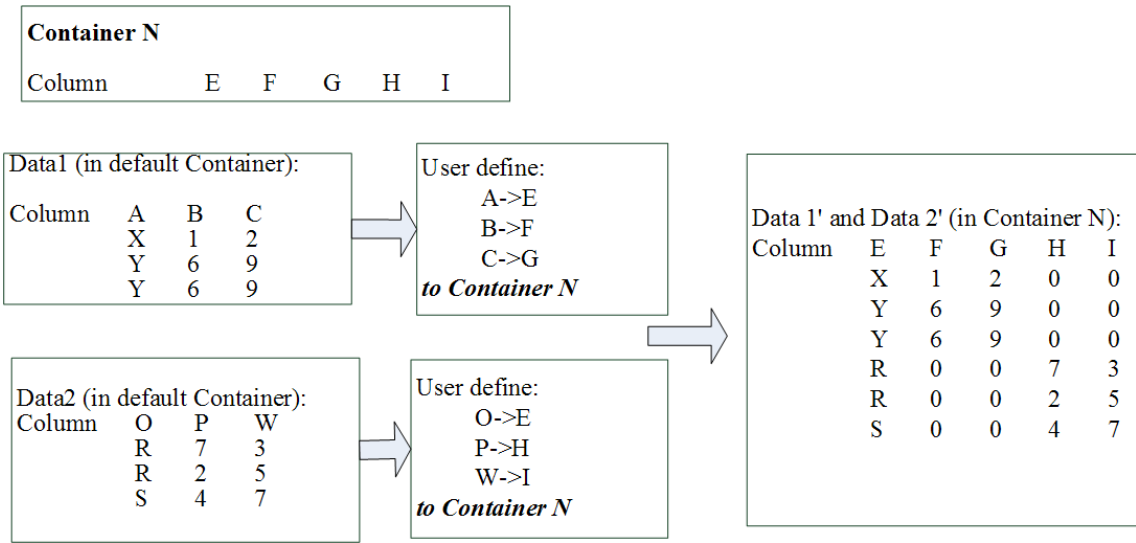


Figure 17. Container example.

## 4.5 System Architecture

The Web-based GUI for MapReduce Data Processing is based on a Web-based GUI and a Hadoop cluster. Figure 18 shows our proposed system architecture and the associated workflow. Our proposed GUI sends the control information from the client to the Hadoop Cluster. The information will be analyzed and split to the processors. The processed result is stored in HDFS, and the Manager notifies the GUI. Users can download the result from HDFS through the GUI.

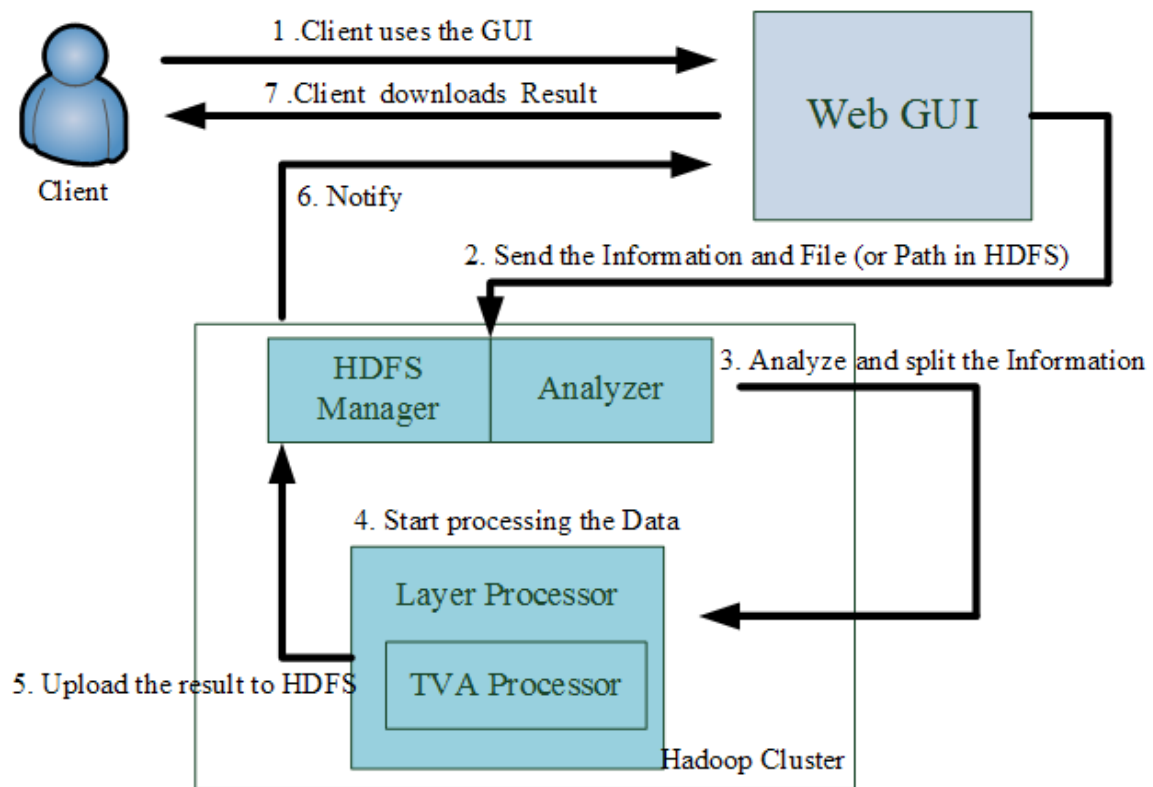


Figure 18. The proposed system architecture and workflow.

The Web-based GUI expects the user's controls and then serializes all of the user's control information into a data serial. In the GUI, the user's control information will be collected to a data serial: Container information and a set of Layers information. Figure 19 shows the Container and Layers information.

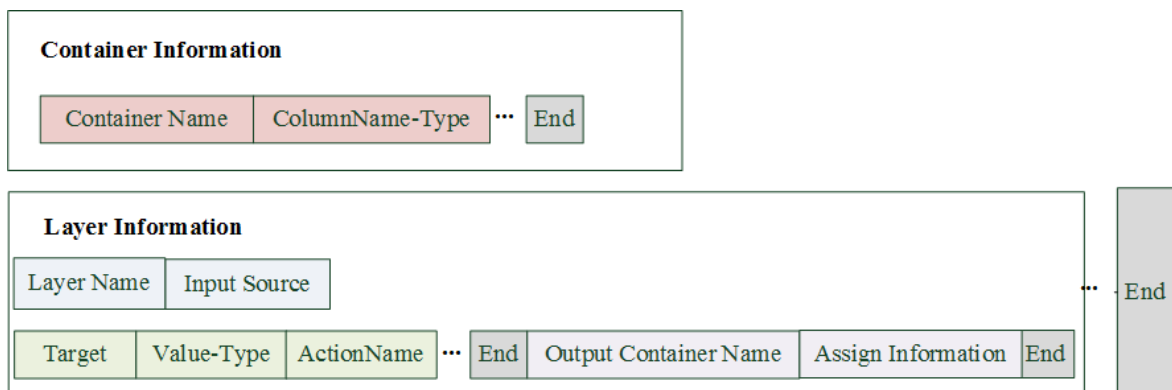


Figure 19. Container and Layer information in a data serial

The Web-based GUI sends the information serial to Hadoop Cluster. In the Hadoop Cluster, the analyzer splits and analyzes the details from the data serial. The HDFS Manager is responsible for managing each file or path from input and output. The Layer Processor will process all Layers in order, and trigger the TVA Processor for each Layer. Figure 20 shows the processing flow in the TVA Processor. In the TVA Processor, all functions have their own signals, and the signaled functions execute when the Job starts. Although the GUI is convenient, it is not flexible, like coding. Too many complex operations in a Job are out of the range of GUI capability. Our method is suitable for a modular collection.

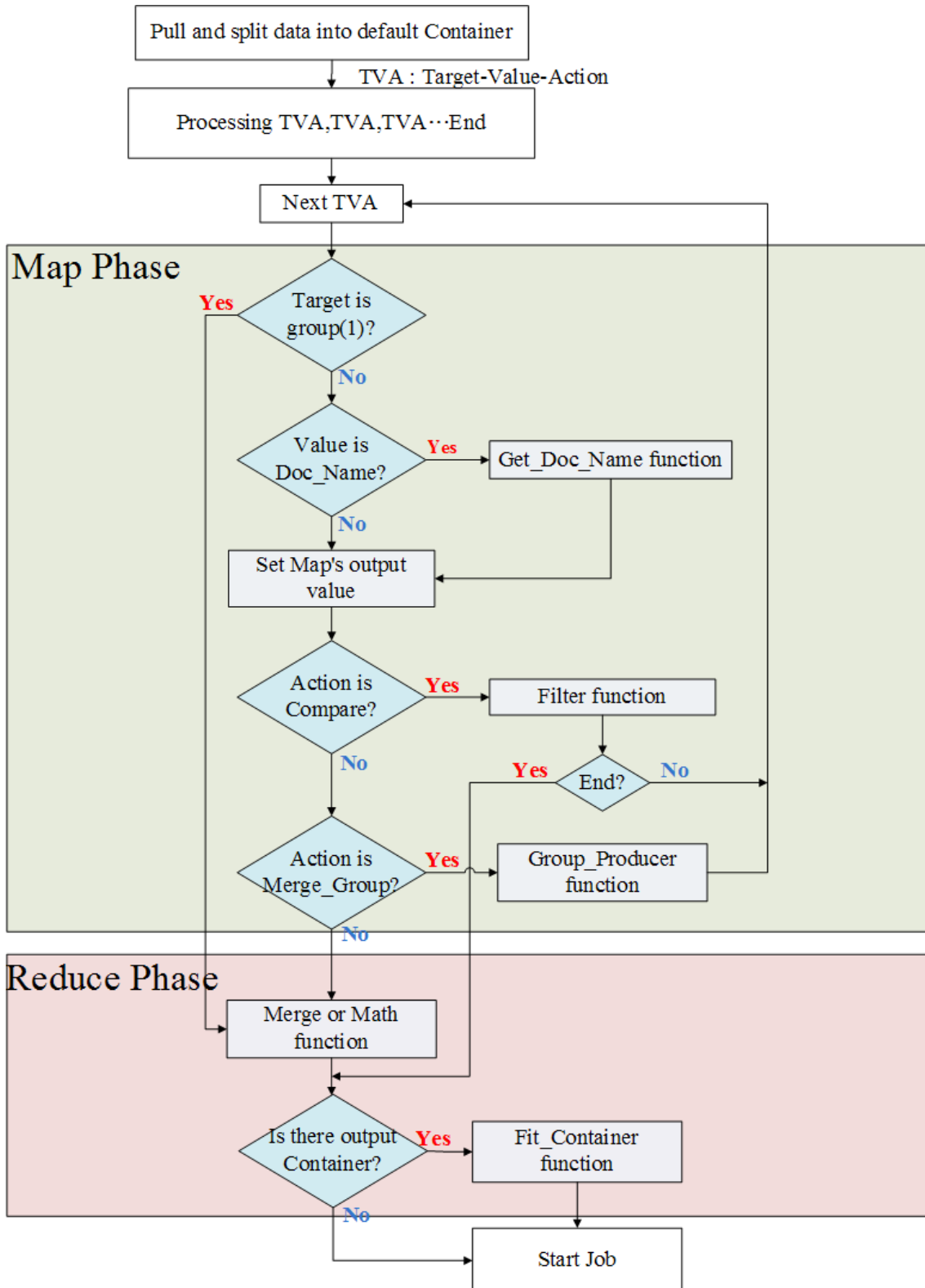


Figure 20. The processing flow in the TVA Processor.



## Chapter 5 Case Study and Implementation

In our Web-based GUI, users can drag the Target-Value-Action, input path form, and container to put them in the drop area on the Web page. Drag and Drop offers a convenient control method. Our Web-based GUI is implemented based on JQuery and PHP. The Hadoop cluster uses Hadoop 0.20.2. There are some real-world examples that can be processed in MapReduce. We also achieved the same results by using our Web-based GUI for MapReduce Data Processing.

### 5.1 Distributed Grep

*Distributed Grep* processes the words if they match the supplied pattern. In Figure 21, we want to calculate the number of the words which contain of “A or C” and “0 to 5”.

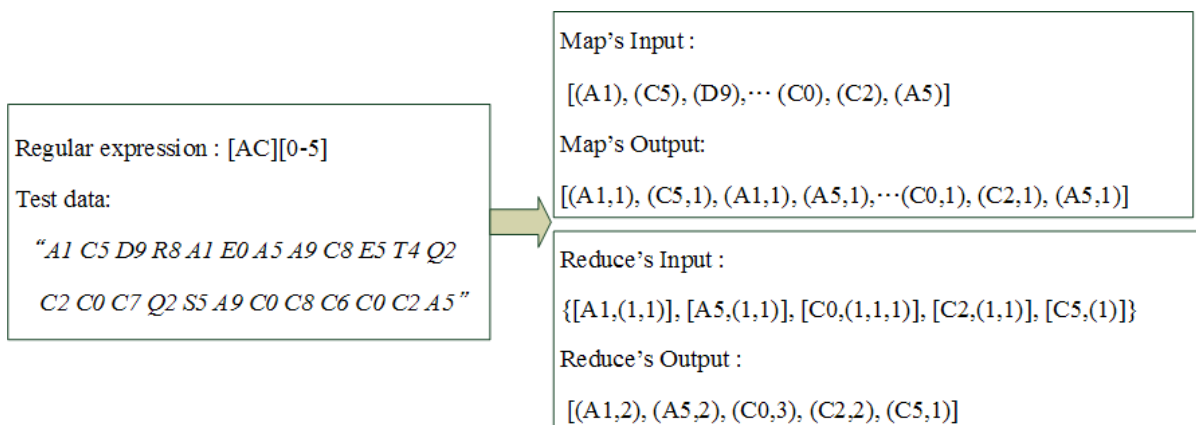


Figure 21. Distributed Grep data flow in MapReduce.

In the Map phase, the map function emits the word which matches the regular

expression. Each matching word emitted is given a value “1”. In the Reduce phase, the values will be added from the same word. Users need to design the input/output in each phase. However, in our Web-based GUI, if we want to find the number of matching words; we just set the columns of the default Container and produce two Target-Value-Actions. In the first Target-Value-Action, we used the Compare\_Equal action to filter words. Then we give each matching word a value “1”. We will get the number of matching words by summing the value.

## 5.2 Count of URL Access Frequency

*Count of URL Access Frequency* is for calculating the total requests of each URL. In MapReduce, the map function only outputs requested “URL” with a value “1”. The values are added together from the same URL in the Reduce function. In our Web-based GUI, we choose the URL as the Target, and give it a value “1”. This means we give a value “1” to each requested URL. So If we add the values from the same URL together, we will get the number of the URL requests.

## 5.3 Reverse Web-Link Graph

The *Reverse Web-Link Graph* records the target URL with its source page. It shows the associations of source URLs with a given target URL. In MapReduce, the Map function outputs the target/source pairs for each link to a target URL with its source page. The Reduce function concatenates the list of source URLs from the same target. The result is like  $[target, list(source)]$ . In our Web-based GUI, we choose the target URL as the Target, giving a “source URL” as the Value. We merge the values from the

same target to get the result  $[target, list(source)]$ . Table 5 shows the details of the above examples.

<i>Distributed Grep</i> <i>column: [word]</i>	<b>Target</b>	<b>Value</b>	<b>Action</b>
	word	[AC][0-5]	Compare_Equal
	word	1	Sum
<i>Count of URL Access Frequency</i> <i>column: [URL][others]</i>	<b>Target</b>	<b>Value</b>	<b>Action</b>
	URL	1	Sum
<i>Reverse Web-Link Graph</i> <i>column: [Target][Source]</i>	<b>Target</b>	<b>Value</b>	<b>Action</b>
	Target	Source	Merge

Table 5. The example details of Target-Value-Action..

## 5.4 Pairwise Document Similarity

The authors in [5] present a MapReduce algorithm for computing pairwise document similarity in large document collections.

### 5.4.1 Pairwise Similarity Algorithm

In Pairwise Document Similarity in Large Collections with MapReduce, the authors calculate the similarity of pairwise documents in MapReduce. They consider symmetric similarity measures which are defined as follows:

$$sim(d_i, d_j) = \sum_{t \in V} (w_t, d_i) \cdot (w_t, d_j)$$



Where  $d$  is a document,  $sim(d_i, d_j)$  is the similarity between documents  $d_i$  and  $d_j$ ,  $V$  is the vocabulary set and  $w_{t,d}$  is the weight which is the number of times term  $t$  appears in  $d$ .

```

1:  $\forall i, j : sim[i, j] \leftarrow 0$ 
2: for all  $t \in V$  do
3:    $Pt \leftarrow postings(t)$ 
4:   for all  $d_i, d_j \in Pt$  do
5:      $sim[i, j] \leftarrow sim[i, j] + w_{t, d_i} \cdot w_{t, d_j}$ 

```

The algorithm is computing the similarity between all pairs of documents. The authors proposed an efficient solution to the pairwise document similarity problem. The presented solution can be expressed as two MapReduce jobs: “Indexing” and “Pairwise Similarity”. It is illustrated in Figure 22.

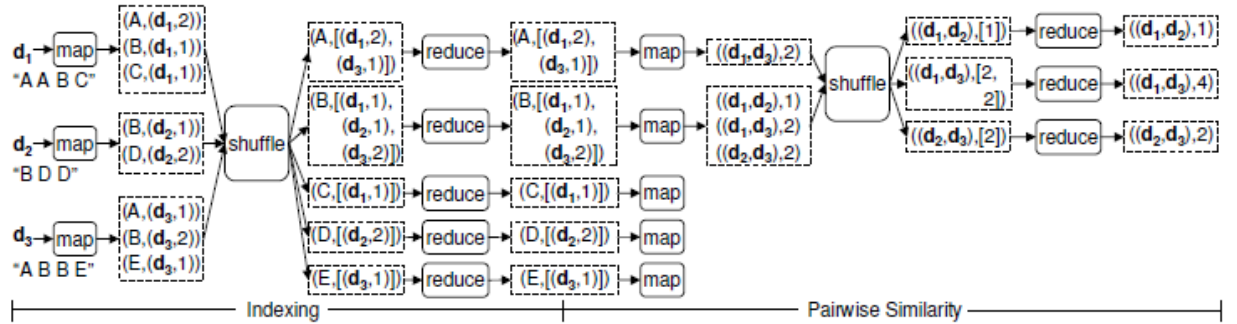


Figure 22 [5]. Computing the Pairwise Document Similarity of 3 documents.

The reduce function calculates the pairwise similarity. The pseudo code in Hadoop 0.20.0 is as follows:

### ***Indexing Mapper***

```
Map < Object, Text, Text, Pair > {  
    map ( Object key, Text value, Context context ){ // setting the I/O type  
        fileName = getName ();  
        valueInLine = value.split ();  
        mapOutput.set ( valueInLine );  
        context.write ( valueInLine, Pair ( filename, 1) );  
    }  
}
```

### ***Indexing Reducer***

```
Reduce < Text, Pair, Text, ArrayList >{ // setting the I/O type  
    reduce ( Text key, Iterable < Pair > values, Context context ){  
        for ( Pair value: values )  
            newValue += value;  
        newPair = ( key, newValue );  
        context.write ( key, newPair );  
    }  
}
```

### ***Pairwise Similarity Mapper***

```
Map < Object, Text, Text, Integer > { // setting the I/O type  
    void map ( Object key, Text value, Context context ){  
        eachPair [ ] = value.getEachPair (); // get each Pair from same Target  
        for (i=0; i<eachPair.length; i++)
```

```

        for (j=i++; j< eachPair.length; j++){
            namePair = eachPair[ i ].name + eachPair[ j ].name;
            pairValue = eachPair[ i ].value + eachPair[ j ].value;
            context.write = ( namePair, pairValue );
        }
    }
}

```

### ***Pairwise Similarity Reducer***

```

Reduce < Text, Integer, Text, Integer >{ // setting the I/O type
    reduce(Text key, Iterable < Integer > values, Context context){
        sum=0;
        for ( Integer value: values)
            sum += value;
        context.write ( key, sum);
    }
}

```

### ***Job***

```

Main(){
    Job1.setMapperClass ("Indexing Mapper");
    Job1.setReducerClass ("Indexing Reducer");
    Job2.setMapperClass ("Pairwise Similarity Mapper");
    Job2.setReducerClass ("Pairwise Similarity Reducer");
    All jobs start
}

```

### 5.4.2 Pairwise Document Similarity in Our Proposed Methods

In indexing, the Map function emits Tuples consisting of terms and document names/ number of times term appears pairs. The reduce function copies and outputs the Tuples where  $sim(di, dj)$  is not zero. In pairwise similarity, the Map function combines the similarity between all pairs of documents.

Table 6 shows the details of each Layer in our Web-based GUI. In Layer 1, we want to calculate the weight of each word in each document. This is like the Inverted Index and WordCount. We choose the word and document name as the Target, giving “1” as the Value, and take the *Sum* as the Action so that we can get the weight. Because the default output of Target-Value-Action is the content of Target and the result, the word and document name in Target is considered a single word.

<i>Pairwise Document Similarity</i>			
<i>Layer1, input is origin data</i> <i>column: [word]</i>	<b>Target</b>	<b>Value</b>	<b>Action</b>
	word, doc name	1	Sum
<i>Layer 2, input is Layer 1's output</i> <i>column: [word], [doc_name],</i> <i>[weight]</i>	<b>Target</b>	<b>Value</b>	<b>Action</b>
	word	doc name, weight	Merge_Group _Mutiply
	group(1)	group(2)	Sum

Table 6. The pairwise similarity details in our presented methods.

Therefore, we need to define the new Container “Container\_PDS” to formalize the columns from Layer 1's output data. Figure 23 shows how Layer 2 formalizes the

columns. There are three columns in Container\_PDS.

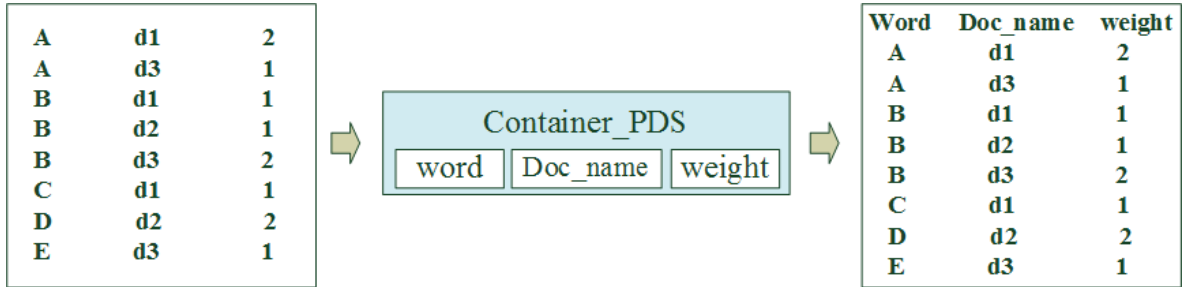


Figure 23. Formalization of the columns for Container\_PDS.

In Layer 2, we choose the word as the Target, giving a document name and weight pair as the Value, and take *Merge\_Group\_Multiply* as the Action. *Merge\_Group\_Multiply* will output a set of groups where group (1) is the merged document name, and group (2) is the multiplied value. For example, the first and second lines in Container\_PDS will be grouped as “[(d1, d3), 2]”. *Merge\_Group\_Multiply* will also filter the *sim* ( $d_i, d_j$ ) which is zero due to there only being a single  $wt, d$  with the same Target. The Merge\_Group actions ignore the single pair. We choose the group (1) as the Target and give group (2) as the Value. Group (1) is the documents name pair. Group (2) is the weight of the pair. Finally, we sum the weight and get the Pairwise Document Similarity result. Figure 24 shows the implementation of the screening process.

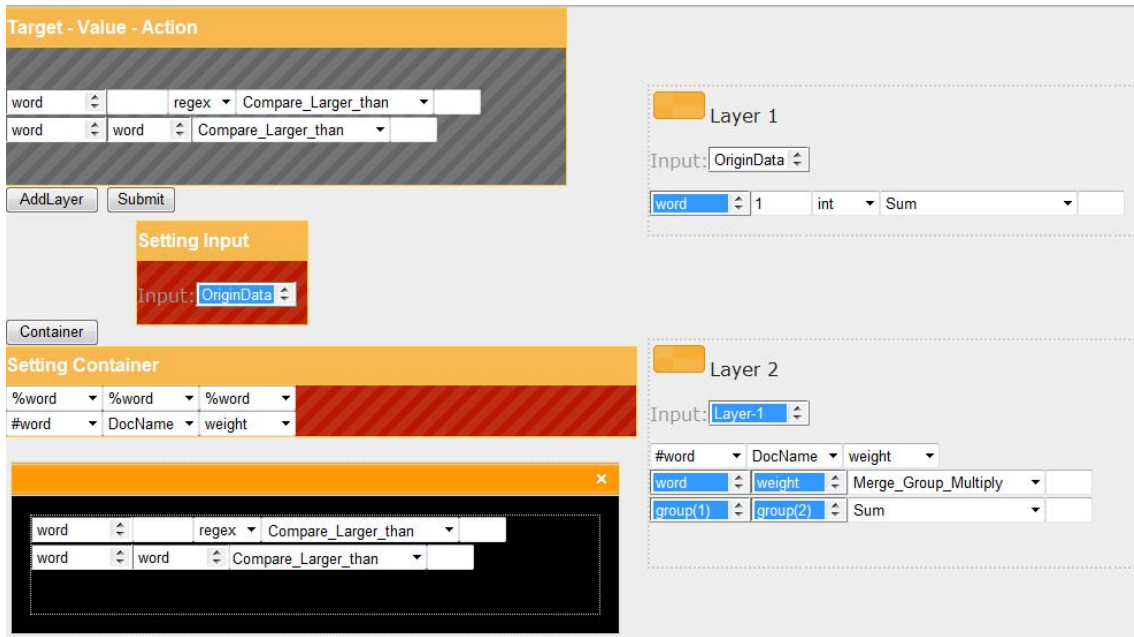


Figure 24. The pairwise similarity implemented screening process of our Web-based GUI.

Using our methods, processing Pairwise Document Similarity is made simpler, more convenient and more readily viewable than direct programming in MapReduce. There is no need for users to concern themselves with the details of MapReduce programming.

## 5.5 Track Statistics from Last.fm

Last.fm [21] is a famous music website which provides internet radio and community-driven music service. Last.fm collects users listening information to generate music rankings and billboard positions, and is processed on the Hadoop platform. Last.fm started using Hadoop in 2006 because the number of users of its services grew rapidly. At present, Last.fm has two Hadoop clusters; there are over 50 machines, 300 CPU-cores and 100TB hard disks. Hundreds of jobs involve working on

the Hadoop Clusters, such as Site-log analysis, indexing for searches, formatting data for recommendations, track statistics, and so on. Managing track statistics is one real-world case using MapReduce [10]. We will show how to obtain the track statistics by using our GUI. Table 7 shows the origin data and the processed data. The “scrobbled” is transferred to the database when a user plays a track of the user’s own and sends the information to Laset.fm via a client application or plug-in.

The processed track statistics include statistical data for each single track. Moreover, there are two new columns, “listeners” and “plays”, in the processed track statistics. The “Listeners” column is the number of different listeners for each single track, and the “plays” column is the sum of scrobbled and radio play. There are three MapReduce jobs in the track statistic: Unique Listeners Job, Sum Job and Merge Job.

The Unique Listeners Job is responsible for calculating the number of different listeners for each single track, and the Sum Job is responsible for calculating the sum of scrobbled and radio play.

Finally, the Merge Job merges the outputs from the Unique Listeners Job and the Sum Job. In the Unique Listeners Job, the Map function emits the TrackId and UserId in the same line where its scrobbled or radio play is not zero. The Reduce function calculates the number of UserIds with the same TrackId, and the result is emitted as “listeners”. For example, the Map function’s output with TrackId “225” is “[ (225, 11113), (225, 11115) ]” and the Reduce function processes it like “[ 225, (11113, 11115) ]  $\rightarrow$  (225, 2) ”.

In Sum Job, the map function emits lines in a new format which contains “listeners” and “plays” columns. The Reduce function emits the calculated results with same TrackId and the “listeners” is set to zero in the new format.

Origin Data					
UserId	TrackId	Scrobbled	Radio play	Skips	
11115	222	0	1	0	
11113	225	1	0	0	
11117	223	0	1	1	
11115	225	1	0	0	
Sum Job Output					
TrackId	Listeners	Plays	Scrobbled	Radio play	Skips
222	0	1	0	1	0
225	0	2	2	0	0
223	0	1	0	1	1
Unique Listeners Job Output					
TrackId	Listeners	Plays	Scrobbled	Radio play	Skips
222	1	0	0	0	0
225	2	0	0	0	0
223	1	0	0	0	0
Result					
TrackId	Listeners	Plays	Scrobbled	Radio play	Skips
222	1	1	0	1	0
225	2	2	2	0	0
223	1	1	0	1	1

Table 7. The processed data of track statistics.



The pseudo code is as follows:

### ***Unique Listeners Job***

*Map < Object, Text, Integer, Integer > {*

*map ( Object key, Text value, Context context ){ // setting the I/O type*

*parts [ ] = value.toString().split();*

*set the value of scrobbles and radio from columns.*

*if (scrobbles\*radio <= 0) return //Ignore track if has not been played*

*set the value of trackId and userId from columns.*

*context.write ( trackId, userId );*

*}*

*}*

*Reduce < Integer, Integer, Integer, Integer >{ // setting the I/O type*

*reduce (Integer key, Iterable < Integer > values, Context context ){*

*Set < Integer > userIdset = new HashSet< Integer > ();*

*for ( Integer value: values ) userId = value;*

*add value to userIdset*

*context.write ( key, userIdset.size );*

*}*

*}*

### **Sum Job**

```
Map < Object, Text, Integer, Integer > { // setting the I/O type

    map ( Object key, Text value, Context context ){ // setting the I/O type

        parts [ ] = value.toString().split();

        set the value of trackId, scrobbles, radio and skip from columns.

        TrackStats track= new TrackStats(0, trackId, scrobbles, radio, skip)

        // TrackStats is a collector user defined

        context.write ( trackId, track );

    }

}

Reduce < Integer, TrackStats, Integer, TrackStats >{ // setting the I/O type

    reduce (Integer key, Iterable < TrackStats > values, Context context ){

        TrackStats sum_result = new TrackStats ();

        for ( Integer value: values )

            Get and sum each part to sum_result from the same key.

            context.write ( key, sum );

    }

}
```

The Merge Job involves processing the different inputs with different mappers by the “MultipleInputs” in Hadoop API:

```
MultipleInputs.addInputPath (conf, sumInputPath,
    SequenceFileInputFormat.class, IdentityMapper.class);

MultipleInputs.addInputPath (conf, listenersInputPath,
    SequenceFileInputFormat.class, MergeListenersMapper.class);
```

There are two mappers in Merge Job: MergeListenersMapper and IdentityMapper. The MergeListenersMapper is responsible for transferring the output from Unique Listeners Job to the new format like Sum Job. IdentityMapper expects the output from Sum Job without any special requirement, so IdentityMapper just copies and outputs. The Reduce function adds each of the columns with the same TrackId.

In our GUI, we use three Layers and one new Container to process the track statistics job. Layer 1's output and Layer 2's output are fitted for the same Container: "Container\_Last.fm". Table 8 shows the details of each Layer. In Layer 1, we are choosing all columns as the Target, giving the content of Scrobbled and Radio play columns as the Value. Scrobbled is multiplied by Radio play. Each row is filtered where the value is larger than zero. The first Target-Value-Action emits the rows. Second Target-Value-Action emits the number of UserIds with the same TrackId. The amount is the number of listeners. Moreover, Figure 25 shows how we fit the output to the Container\_Last.fm. In Layer 2, we want to calculate the sum of Scrobbled, Radio play and Skips columns with the same TrackId. We therefore choose the TrackId as the Target. Scrobbled, Radio play and Skips columns are the Value. The action sums each of the Scrobbled, Radio play and Skips columns from the same Target. We also sum content of Scrobbled, Radio play to Plays. We can thus get the number of Scrobbled, Radio play and Skips for each Track. Figure shows how we fit Layer 2's output to the Container\_Last.fm. Because the Sum\_Each returns and re-writes the contents from the columns, there is no result which will be output. The Result is therefore the result from the second Target-Value-Action.

<i>Layer 1, input is origin data</i>  <i>column:</i> <i>[UserId],[TrackId],[Scrobbled],</i> <i>[ Radio play],[Skips]</i>	<b>Target</b>	<b>Value</b>	<b>Action</b>
	all	<i>Scrobbled,</i> <i>Radio</i>	Compare_Larger_than : 0
	TrackId	UserId	Size of (Merge_Unique)
<i>Layer 2, input is origin data</i>  <i>column:</i> <i>[UserId],[TrackId],[Scrobbled],</i> <i>[ Radio play],[Skips]</i>	<b>Target</b>	<b>Value</b>	<b>Action</b>
	TrackId	<i>Scrobbled,</i> <i>Radio play,</i> <i>Skips</i>	Sum_Each
	TrackId	<i>Scrobbled,</i> <i>Radio play</i>	Sum
<i>Layer 3, input is Layer 1's and</i> <i>Layer 2's output</i>  <i>column:</i> <i>[TrackId],[Listener],[Scrobbled</i> <i>],[ Radio],[ play],[Skips]</i>	<b>Target</b>	<b>Value</b>	<b>Action</b>
	TrackId	others	Sum_Each

Table 8. The pairwise similarity details in our presented methods.

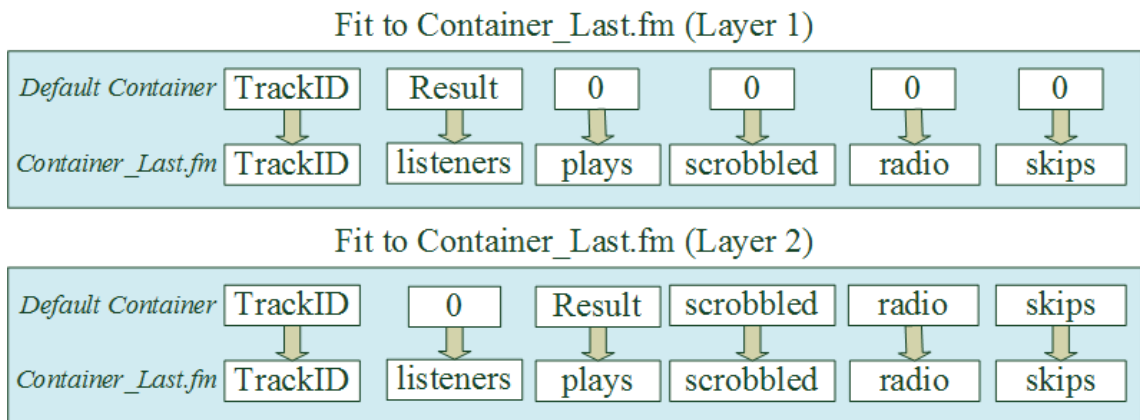


Figure 25. Fitting the output to Container\_Last.fm.

In Layer 3, the inputs are in the same Container, so we can process the Target-Value-Action directly. We choose the TrackId as the Target, the other columns as the Value, and sum each of the columns. Then we can get the Scrobbled, Radio play and Skips for each Track. The track statistic function is thus implemented. The implementation screen is shown in Figure 26

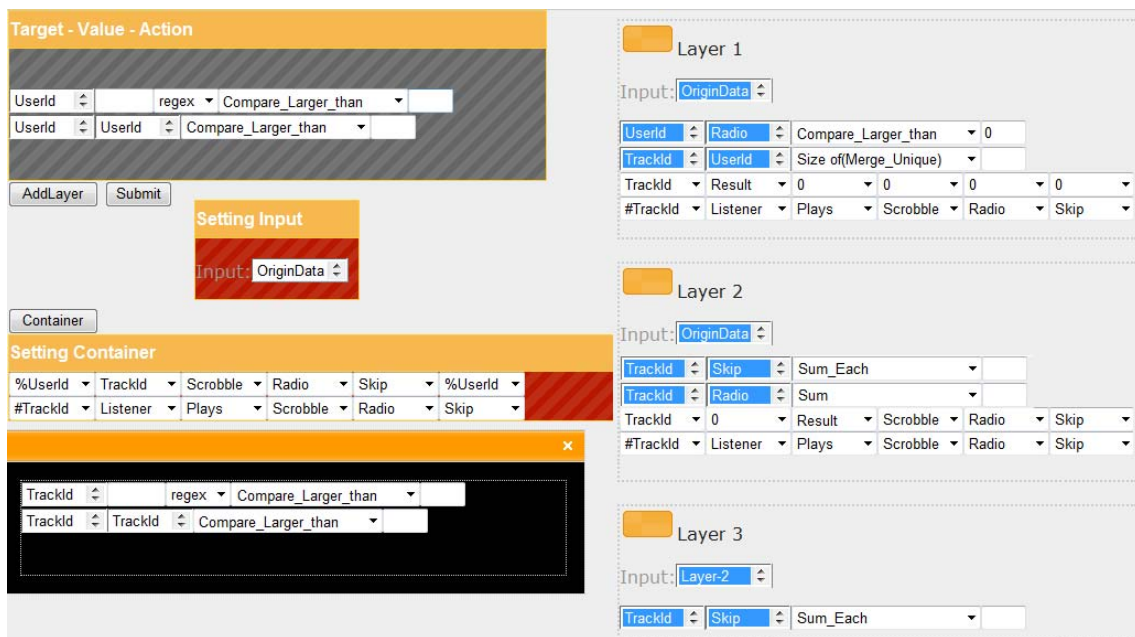


Figure 26. The Last.fm track statistic implementation screen of our Web-based GUI.

In our Web-based GUI, we do not need to consider what to do in terms of the programming details of I/O in the Map phase and Reduce phase in each job. Therefore, we can get the same result for the track statistic simply and in a clearly viewable manner by using our Web-based GUI.

## Chapter 6 Conclusions and Future Work

In this thesis, we present a Web-based GUI for MapReduce data processing. We also present three abstract data processing methods, including Target-Value-Action, Layers, and Container. Target-Value-Action is a direct and original way of thinking about data processing. It hides the programming details of Map and Reduce from users. Users just choose objects as targets; give each object a value, and then take actions. The Layer and Container make the input/output viewable and easy to control. The presented methods are suitable for a Web-based GUI since the complexity of the MapReduce data processing has been significantly decreased. Therefore, users do not need to install Hadoop in their computers, and they can use any device, anywhere, to process their data through the Web-based GUI. Users can also directly drag and drop the operational components to process large-scale data using MapReduce.

In the future, we will improve the operational components and management, because we hope the operation components can be applied in more complex cases. If users can store, reuse, and share the operation components structures they use, data processing using our method can be made more efficient. We will also make the GUI more user-friendly, and do more research into MapReduce to improve the data processing methods and optimize the program construction.



## Reference

- [1] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", *Symposium on Operating Systems Design and Implementation*, 2004
- [2] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, "Bigtable: A Distributed Storage System for Structured Data", *Symposium on Operating Systems Design and Implementation*, 2006
- [3] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google File System", *Symposium on Operating Systems Principles*, 2003
- [4] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, "The Hadoop Distributed File System", *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium*, May. 2010
- [5] Tamer Elsayed, Jimmy Lin, and Douglas W. Oard, "Pairwise Document Similarity in Large Collections with MapReduce", *46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies*, 2008
- [6] Johnson, J.L., "SQL in the Cloud", *Computing in Science & Engineering*, Vol 11, no 4, July 2009
- [7] Daniel Peng and Frank Dabek, "Large-scale Incremental Processing Using Distributed Transactions and Notifications", *Operating Systems Design and Implementation*, Oct. 2010
- [8] *Hadoop: The Definitive Guide*, Tom White, 2010
- [9] *Hadoop in Action*, Chuck Lam, 2010
- [10] The Apache Hadoop project website, <http://hadoop.apache.org/>



- [11] The Apache HBase website, <http://hbase.apache.org/>
- [12] NCHC Cloud Computing Research Group website, <http://trac.nchc.org.tw/cloud>
- [13] Daytona, <http://research.microsoft.com/en-us/projects/azure/daytona.aspx>
- [14] Google App Engine website, <http://code.google.com/intl/zh-TW/appengine/>
- [15] Microsoft Azure Platform Taiwan Official website,  
<http://www.microsoft.com/taiwan/windowsazure/>
- [16] Heroku Cloud Application Platform website, <http://www.heroku.com/>
- [17] CRM & Cloud Computing Salesforce.com website, <http://www.salesforce.com/>
- [18] National Institute of Standards and Technology website,  
<http://www.nist.gov/itl/cloud/>
- [19] Apache Pig Platform website, <http://pig.apache.org/>
- [20] Cascading website, <http://www.cascading.org/>
- [21] Last.fm website, <http://www.last.fm/>
- [22] The Official Google Blog website: Our New Search Index Caffeine,  
<http://googleblog.blogspot.com/2010/06/our-new-search-index-caffeine.html>