Clustering websites using a MapReduce programming model

Shafi Ahmed, Peiyuan Pan, Shanyu Tang*

London Metropolitan University, UK

Abstract

In this paper, we describe an effective method of using Self-Organizing Map (SOM) to group websites so as to eliminate or at least ease up slow speed, one of the fundamental problems, by using a MapReduce programming model. The proposed MapReduce SOM algorithm has been successfully applied to cluB, which is a typical SOM tool. Performance evaluation shows the proposed SOM algorithm took less time to complete computational processing (i.e. distributed computing) on large data sets in comparison with conventional algorithms, and performance improved by up to 20 percent with increasing nodes (computers).

Key words: website clustering, MapReduce, SOM

1 Introduction

With the boom of Web 2.0 technologies and a large rise in user-generated content, the World Wide Web (WWW) is expanding at an exploding rate. The recent information provided by

Google suggests that there are more than 10 billion web pages presented in their index [1]. Moreover, the Internet also contains images, videos and various types of files, e.g. documents, presentations, spreadsheets etc. With the ever-increasing information on the Internet, it will be even harder for users to find their required information. This is why categorisation comes into action – which essentially allows users to see more results but in a clustered manner.

One of the most popular algorithms for categorisation is called Self-Organizing Map, which adopts more specifically automatic and unsupervised categorisation techniques [2]. Self-Organizing Map (SOM) has been widely used both in the Data Mining and Artificial Intelligence community. It has also been rigorously used in various applications and its mathematical foundations are based on precise calculations.

In a SOM algorithm, entities (in this paper an entity refers to a web page) are closer when they are similar to each other; they are distant if their similarities are less significant [3]. Moreover, the SOM algorithm presents the similarities on a 2D plane where the entities are displayed as nodes, and a group of nodes form a cluster if they are highly concentrated at a certain point. The algorithm train itself to arrange in such an organised manner.

Categorisation has been studied in [4], and it was based solely on the user's navigational behaviour. See [5] for the most comprehensive coverage of Self-Organizing Map, which had been used to categorise documents like journals. However, there is little attention paid to websites clustering by means of SOM. Although both journals and websites are quite similar in content presentation, one of the fundamental differences is that additional HTML tags are present in web pages that constitute a website. Therefore, some special measures are needed in order to capture or understand the normal contents of websites.

In most of the mentioned cases associated with Self-Organizing Map, speed has been a major issue. The creation of SOM normally takes huge processing power and consumes time. Multiple computers have been used in [6], which used a Beowulf Cluster based on Linux boxes. The time required for processing the SOM reduced to a large extent. But the system is prone to hardware failures and so it is unreliable for SOM processing at enterprise level.

In this paper we present an effective method of using Self-Organizing Map to group websites by means of a MapReduce programming model so as to eliminate or at least ease up slow speed, one of the fundamental problems to be solved.

2 Problems with Self-Organizing Map

Search engines on the Internet provide results to users based on keywords. The search results are usually presented in a list of results, which does not show the relationships between the web pages in the results. Moreover, there are scenarios when users are willing to see how the results are grouped into various subjects. In an attempt to solve this particular problem, a new tool called cluB has been developed. The tool categorises a website into various subjects and thus allow the user to browse the website by viewing the relationships between the web pages of the site.

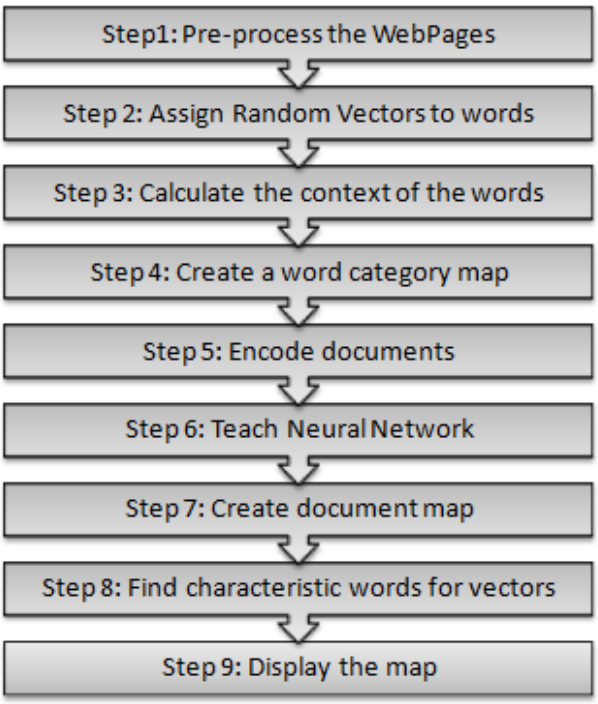

Fig. 1. Self-Organizing Map (SOM) algorithm

Figure 1 shows the Self-Organizing Map algorithm for the cluB system. Using cluB, web pages in a website are tagged automatically. As there is no control over the unsupervised SOM algorithm, the keywords displayed are not well structured when data are unstructured. Also, the resultant clusters differ significantly in consequent SOM trainings.

The same problem was observed in [7] and the solution was also proposed in the same paper. The work around was to build a structured SOM.

The SOM algorithm has a defined set of steps and formulas that an implementation can easily use. But, one of its fundamental problems is its speed. The computation of SOM takes quite a long time, and the time increases as the dataset gets bigger.

3 Comparisons of distributed computing algorithms

In this section we compare a number of distributed computing algorithms relevant to web sites clustering, such as MapReduce, Hadoop, and Beowulf Cluster.

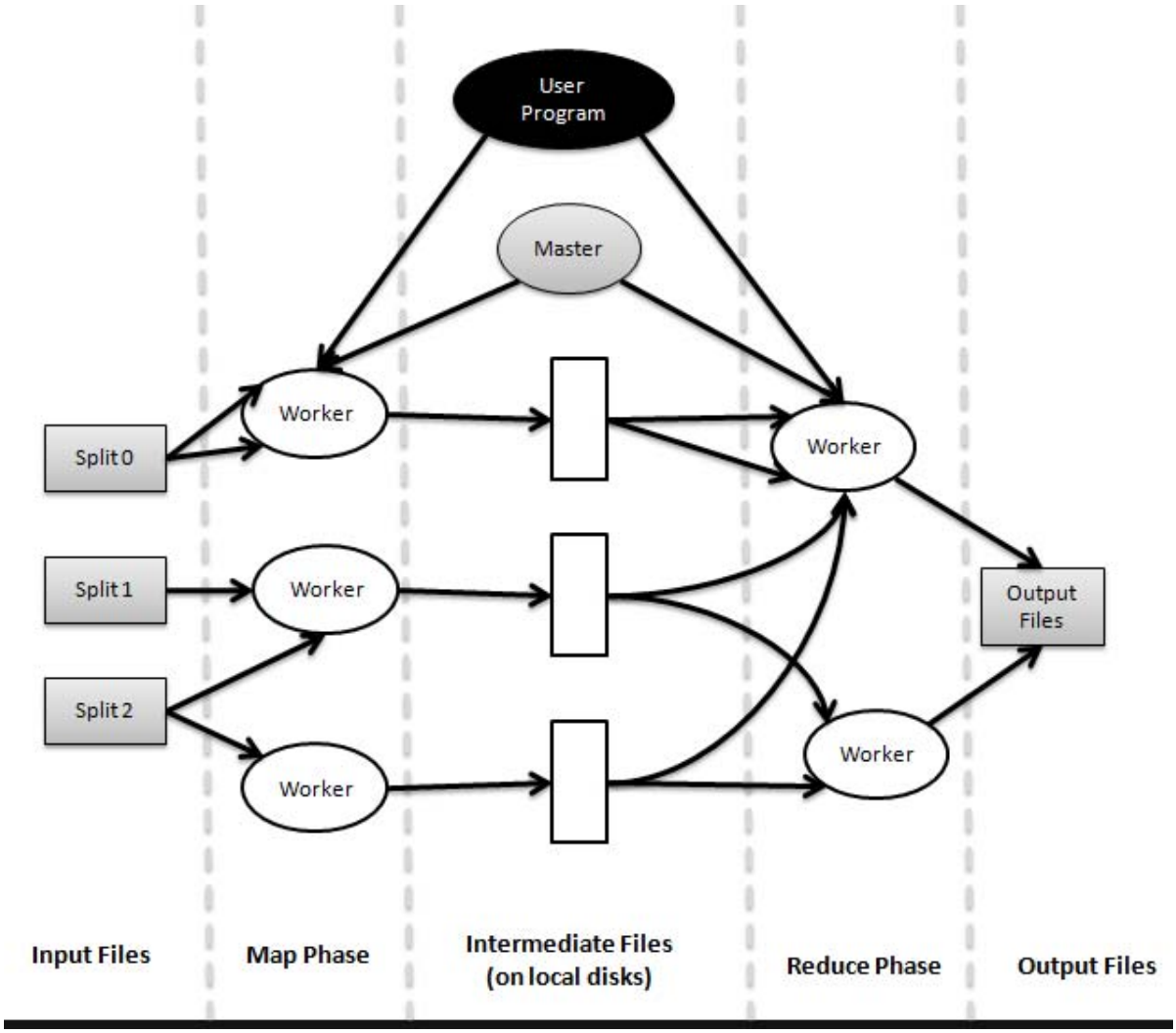

Fig. 2. MapReduce architecture

3.1 MapReduce

MapReduce is a programming model used by Google in many of its products [1]. Google has also developed an implementation for this model. The model chops a large amount of input into smaller sub-problems, and distributes those among a cluster of computers (processors). This allows large data sets to be processed within a short period of time compared with data processing running on a single processor. The implementation involves paralleling computations, distributing data and tackling hardware failures, which are quite complex, so an abstraction level has been created in MapReduce. This abstraction layer reduces the complexity for developers to use MapReduce.

The advantage of MapReduce is that it is resistant to hardware failures, which are normal for workstations (without RAID support) compared to Beowulf Cluster.

Figure 2 shows the overall flow of a MapReduce operation. When the user program calls the MapReduce function, the following sequence of actions occurs (the numbered labels in the illustration correspond to the numbers in the list below).

(1) The MapReduce library in the user program first divides the input files into M pieces of typically 16 megabytes to 64 megabytes (MB) per piece. It then starts up many copies of the program on a cluster of machines.

(2) One of the copies of the program is special: the master. The rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. The master picks idle workers and assigns each one a map task or a reduce task.

(3) A worker who is assigned a map task reads the contents of the corresponding input shard. It parses key/value pairs out of the input data and passes each pair to the user-defined Map function. The intermediate key/value pairs produced by the Map function are buffered in memory.

(4) Periodically, the buffered pairs are written to a local disk, partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.

(5) When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers. When the reduce worker has read all intermediate data, it sorts these data by the intermediate keys so that all occurrences of the same key are grouped together. If the amount of intermediate data is too large to fit in memory, an external sort is used.

(6) The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function. The output of the Reduce function is appended to a final output file for the reduce partition.

(7) When all map tasks and reduce tasks have been completed, the master wakes up the user program. At this point, the MapReduce call in the user program returns to the user code.

MapReduce achieves reliability by sharing out a number of operations on data sets among/between every node in the network; each node is expected to report back periodically

with completed work and status updates. If a node falls silent for longer than the interval, the master node (similar to the master server in the Google File System) records the node as dead, and sends out the node's assigned work to other nodes. Individual operations use automatic operations for naming file outputs as a double check to ensure that there are no parallel conflicting threads running; when a file is renamed, it is also possible to copy the file to another name in addition to the name of the task (allowing for side-effects).

The reduce operations operate much the same way. Because of their inferior properties with regard to parallel operations, the master node attempts to schedule reduce operations on the same node, or as close as possible to the node holding the data being operated on; this property is desirable for Google as it conserves bandwidth.

3.2 Hadoop

Hadoop, a similar implementation to MapReduce, is based on Hadoop File System (HDFS), and its implementation takes ideas from Google File System (GFS) [8], as shown in Figure 3. It is a distributed File System for applications that use computationally intensive applications and works with large amounts of data. This file system is extensively used by Google as the primary storage mechanism. Thousands of users make use of Google File System unknowingly as almost all systems are built on top of it. This is how it has been described [9].

> *"We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients."*

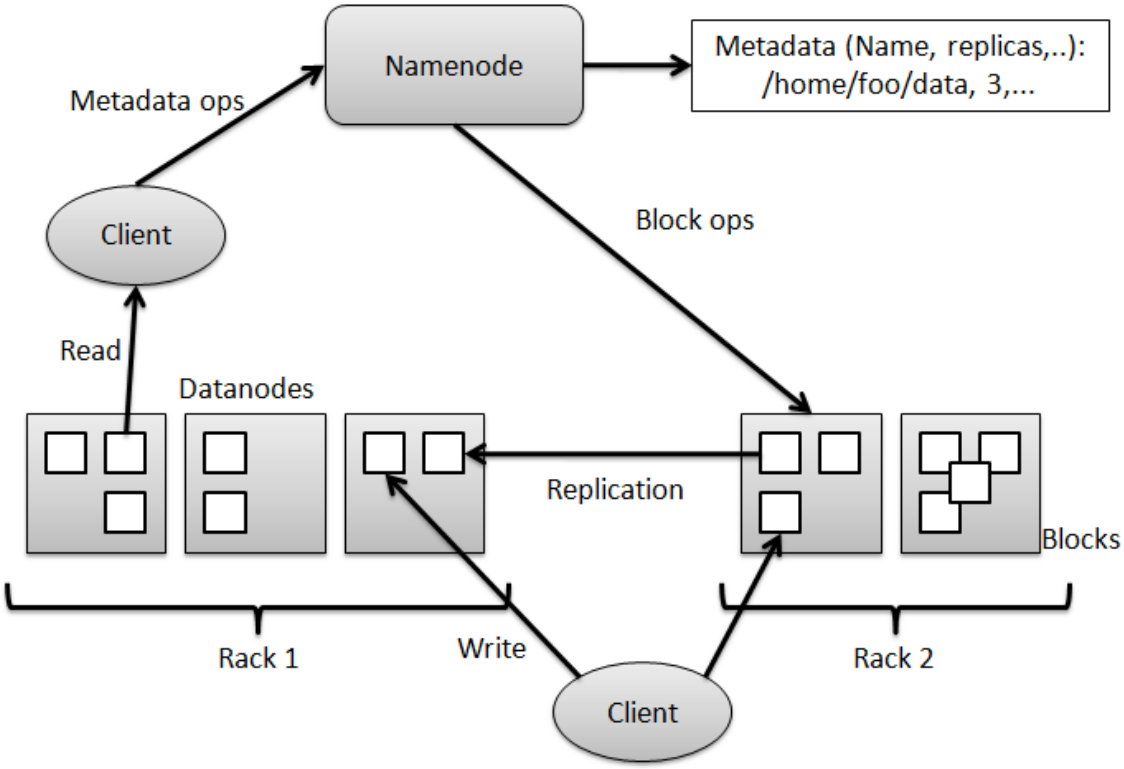

Fig. 3. Hadoop File System architecture

3.3 Beowulf cluster

The Beowulf cluster implementation is a parallel computation model used only on Linux machines. Any program written and then run on a cluster usually runs faster. But the program has to take care of hardware failures and make sure that the program itself deals with the parallelism involved in the cluster. There is no scope for fault-tolerance, error detection and work restart capabilities, and the Beowulf cluster is not a good solution for time-boxed applications that demand reliable and timely execution of a particular task e.g. finding the Euclidian distance between two points. Moreover, this cluster does not consider manageability and so the user or the programmer in this case has to manage each resource separately in the cluster rather than a single File System.

The following points summarise the problems of Beowulf cluster:

(1) Parallel applications under a Beowulf cluster use a message passing model rather than shared memory. While these implementations are available to emulate shared memory, more application tuning is required to make the application working than converting to message passing.

(2) Beowulf cluster focuses on developers and does not take into account the architectural model, testing and binary compatibility. This leads to a written application possibly being written again to take advantage of clustering in order to make any significant changes to the program.

(3) In most cases the developer is often responsible for system design and administration, which takes time and energy away from working on the actual application.

3.4 Proposed MapReduce SOM algorithm

MapReduce has an upper hand in terms of parallelism. One of the downsides of MapReduce is that it restricts to the programming model. But the opposite argument is that it provides a good model for managing problems dealing with large amounts of data. For our particular problem with large data sets, MapReduce provides fast execution without worrying about the underlying hardware infrastructure, and so we could focus on the application itself, i.e. solving the problem. Based on these facts we had decided to use Hadoop [10], which is a similar implementation to MapReduce [3]. Also, the approach is also supported by the following quotation [3].

> *"A key difference between these systems and MapReduce is that MapReduce exploits a restricted programming model to parallelize the user program automatically and to provide transparent fault-tolerance."*

cluB uses a SOM algorithm at first, but in order to take advantage of MapReduce, we had revised the SOM algorithm slightly. No changes have been made to the algorithm, but the way the calculations run has changed. An ‘Intermediate Step’ has been added by us to act as a buffer. The buffer allows the system to run computation in parallel.

The proposed MapReduce programming model is illustrated in Figure 4. The programmer expresses the whole computation as Map and Reduce functions. The Map function takes keys and corresponding values as inputs and produces Intermediate keys which are forwarded to the reduce function. The reduce function merges these values for each Intermediate key via an iterator.

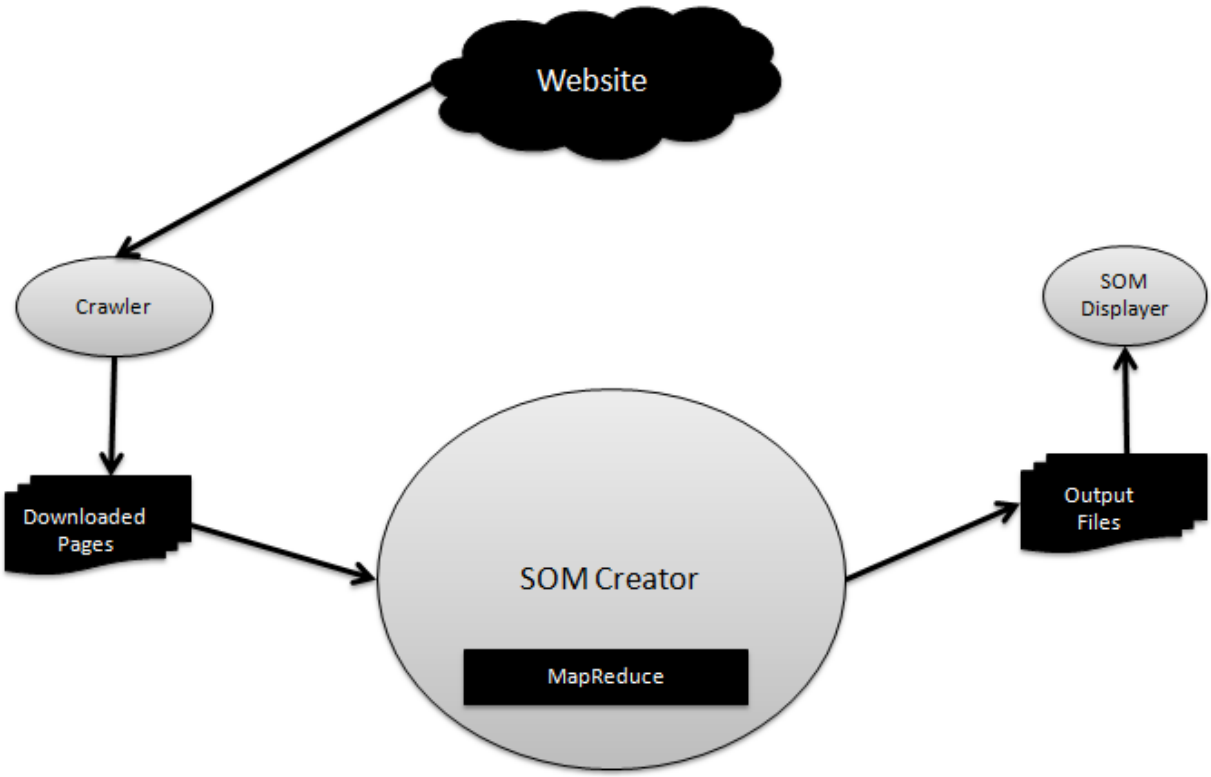

Fig. 4. cluB implementation incorporating Map and Reduce

The revised MapReduce SOM algorithm has been developed using Java. In order to understand the system the Java classes used to build the system are described in Section 4.

The main strength of the modified cluB system is that no matter how many web pages are present in a website, each page is assigned a tag. Tags are nowadays used in blogs, forums, pictures and videos on the Internet for identifying and grouping similar content. Also, the distribution content inside the web pages is vividly clear to the user as the topographic map represents ‘concentration’ and ‘hollowness’.

4 Implementation

cluB or ‘clustered WEB’ consists of various tools which have been built using Java programming to solve the above problem using Self-Organizing Map but with improved speed. These tools have been integrated so that the product produces the output as a SOM. The reason for cluB being designed as various tools is that it is easier to test the system and find bugs in the system separately. None of the tools are linked directly in the same source code repository. Rather the tools use the outputs of other tools as inputs.

The cluB system used in our experiments is composed of four components as follows:

(1) Crawler: This tool downloads all the web pages from a targeted website and stores the files in the local file system. This is done by using a crawler that visits and downloads the home page of the website, retrieves all the links, and then repeats the same process, i.e. visiting and downloading all the web pages referred by the links and so on. A list of ‘crawled’ web pages is maintained so that the same web page is not downloaded twice.

(2) HTML Parser: HTML pages have tags around each element which are not necessarily part of the original content, and the tags are not meant for users but for web browsers. The browsers use these tags to understand the layout and the formatting of the content. After HTML pages have been downloaded from the Internet, the web pages are parsed using this tool. Parsing involves removing tags from the HTML pages and retrieving text that is understandable and readable to a normal user.

(3) SOM Creator: It is the heart of the cluB system and involves the creation of SOM based on the techniques mentioned in [5] apart from the use of search. This creates a SOM which the user can browse where tags are attached to the map at place of higher concentration of documents or commonly known as clusters.

(4) SOM Displayer: This tool displays a SOM on a grid that allows users to click on nodes and the documents assigned to the nodes. By clicking one of the links, a web browser opens showing the web page connected by the hyperlink.

4.1 Java classes

The design of the implementation is composed of three classes below, Mapper, Reducer and Driver, as shown in Figure 5.

Class	**Usage**
Mapper	This class takes an input pair of values and produces an intermediate key/value pairs. The programming model groups the values and forwards them to the reducer class.

Reducer	This class receives the intermediate values with the corresponding intermediate keys. Generally, one or none output is produced per reduce method call. Sometimes the computer's main memory is not enough for large data sets, and an iterator is used instead for easy handling.
Driver	This class is the main program that contains the main method of the program. It sets the input and output folders and the configurations needed for jobs to run.

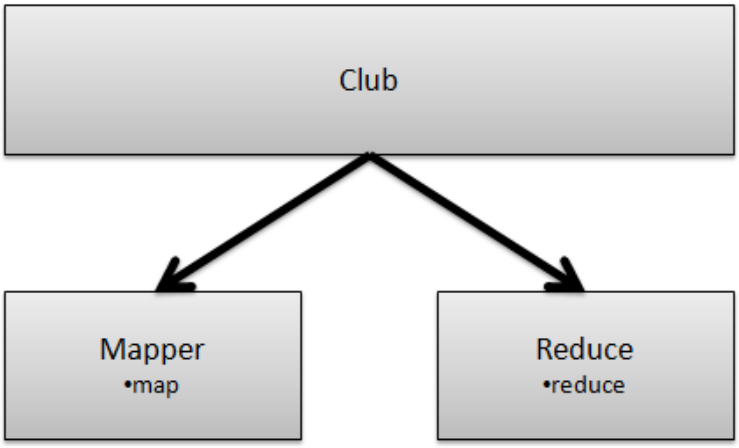

Fig. 5. Class diagram showing the two methods in Map and Reduce classes

At the end of the job run, the results are aggregated by the reducer class and are used to display the final SOM. The same program can be theoretically run on many machines without any modification to the program and the test results demonstrate it clearly.

4.2 Software

The whole system has been implemented using the Java Programming Language. Data sets have been downloaded from the Internet by the crawler (part of cluB) and the web pages have

been stored in the File System. SOM visualisation has made the best use of Graphics 2D Application Programming Interface. The creation of SOM has been accelerated by using MapReduce.

4.3 Hardware configuration

The algorithm was used in three machines for experimental purposes. Each machine consisted of 4 GHz processor and 2GB RAM. The machines in the cluster were networked using 100 Mbps Ethernet links. All the machines had Windows Operating System with Java Virtual Machine installed.

5 Performance evaluation

Performance tests were carried out on different data sets, one with loads of images and the others with fewer images. The test results showed that the performance is dependant upon the amount of text of the web site. The new system using the MapReduce model worked better in some ways and was essentially faster. Table 1 and Figure 6 show how the implementations compare with and without the MapReduce model.

Table 1. Performance results with and without MapReduce

Number of nodes	**Number of web pages**	**Mean time to complete tasks (seconds)**		**Performance increase**
		cluB running on a single core computer	**cluB running on a cluster with MapReduce**	

1	979	292	N/A	N/A
2	979	292	284	2.7 %
3	979	292	261	10.6 %
4	979	292	244	16.4 %

The system with MapReduce is faster than the earlier implementation on a single processor and is now more scalable. Compared to Beowulf cluster, the system is more easily manageable and usable. Beowulf has the disadvantage of providing wrong results if one of the machines breaks down. On the other hand cluB also has the advantage of being interoperable in various operating systems as the system itself is written with Java. Therefore, as long as the machine has a Java Virtual Machine, the cluB tool can deal with computing on large data sets on any operating system.

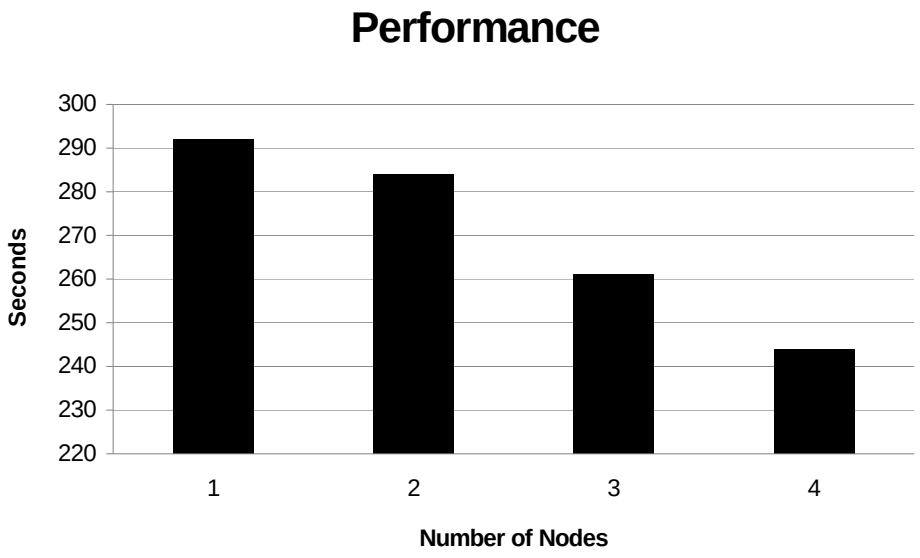

Fig. 6. Execution time to complete tasks vs. number of nodes

6 Conclusions

MapReduce has been used with great success at Google and the success has been rediscovered in cluB with the revised ‘MapReduced’ SOM algorithm proposed in our study. We have successfully applied the proposed algorithm to the cluB tool, which now takes less processing time for categorisation.

Moving the implementation from Java to C++ is a subject of future work. As compiled languages are faster, it is a natural move from an interpreted language to a faster language. It would improve the speed of calculating SOM, and allow downloading web pages at a fast rate. Another possible future work is to create a more efficient hadoop implementation. The current implementation is not accessible to all the programmers who would be interested to leap into parallel programming.

References

1. Google. *Google.* [Online] www.google.com.

2. Kohonen, Tuevo. *Self-Organizing Maps.* Germany : Springer, 2001.

3. Jeffrey Dean, Sanjay Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters.* San Francisco, CA : OSDI’04: Sixth Symposium on Operating System Design and Implementation, 2004.

4. Kate A. Smith. *Web page clustering using a self-organizing map of user navigation patterns.* Amsterdam, May 2003, Decision Support Systems, Vol. 35, pp. 245-256. 0167-9236.

5. Teuvo Kohonen, Samual Kaski, Krista Lagu, Jarkko Salojarvi, Jukka Hinkela, Vesa Paatro and Antti Sareela. *Self-Organization of a Massive Document Collection.*

6. Gustavo Arroyave, Oscar Ortega Lobo, Andrés Marín. *A Parallel Implementation of the SOM Algorithm for Visualizing Textual Documents in a 2D Plane.* 2002.

7. Ivan Perelomov, Arnulfo P. Azcarraga, Jonathan Tan, Tat Seng Chua. *Using Structured Self-Organizing Maps in News Integration Websites.* Singapore : National University of Singapore, 2002.

8. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. *The Google File System.* Lake George, NY : ACM Symposium on Operating Systems Principles, 2003.

9. Apache. Hadoop. *Hadoop.* [Online] Apache. [Cited: 9 July 2007.] http://lucene.apache.org/hadoop/.

10. Ralf L¨ammel. *Google's MapReduce Programming Model - Revisited.* Redmond, WA, USA : Microsoft Corp.

11. Google. Introduction *to Parallel Programming and MapReduce. Google Code for Educators.* [Online] Google. [Cited: 09 July 2007.] http://code.google.com/edu/parallel/mapreduce-tutorial.html.