VIRTUAL DISK BASED DATA CLUSTERING MAPREDUCE FRAMEWORK

M. Vijayalakshmi and T. Vinodh Kannan
Department of Information Technology, Mookambigai College of Engineering, India

Abstract

One of the important methods in data mining is segmentation. As each field is extended and digitized, large data sets are developed quickly. These wide clustering of data sets poses a problem for conventional sequential segmentation algorithms due to the enormous time consumed for development. Hence, distributed parallel architecture and algorithms are useful in meeting the efficiency and scalability requirements for clustering large data sets. In this analysis, we use MapReduce programming model to develop and experiment a parallel SVM algorithm, and compare the result with concurrent SVM for clustering the changing document database size. The result shows that the SVM proposed get better performance than existing methods.

Keywords:

Cloud Computing, Map Reduce, Clustering, Domain Clustering

1. INTRODUCTION

A Map Reduce system is built on an individual cluster which is designed explicitly for Map Reduce jobs. Nevertheless, increasing numbers of installations are now targeting development HPC nodes. Output clusters are constructed to handle higher-performance applications with a high computing power requirements. It is a typical scenario for small community jobs to run simultaneously on a similar HPC cluster, for example, Network jobs and MPI work. Which means the workers have to exchange the cluster configured hardware tools. On manufacture HPC clusters, the performing jobs do not appear to share the resources on a single computational node but still the actual clusters infrastructure [1]. The cluster property manager (RM), the so-called packet system, is charged with spreading the computational nodes across the user groups.

The Fig.1 shows a resource management example for a HPC output cluster. The core section of the example is cluster RM, which usually runs on a clusters head node. This splits the entire resources into partitions based on the user communities specifications and allocates a particular partition to a user community. Within the system, the resources are exchanged between users and frameworks under the supervision of a society-level scheduler or RM.

A problem with that plan is that the cluster resources are uniformly annexed without taking into account the dynamic activity in latency. This could result in load-unbalancing, where supplies are overwhelmed within some groups, while resources are underloaded to other neighborhoods. This reality is well known to the RM or scheduler at government level, that is, whether the computing nodes it manages are overloaded or undercharged. In both cases it will take action either demanding more resources from the RM cluster, or making scarce available.

Several methods in literature, based on conventional partial cumulative methods and acceleration techniques for the treatment of large-scale data, have been developed. These methods aim to increase speed by reducing computational complexity in the clustering process.

Parallelization is one of the most used acceleration techniques aimed at reducing the cost of computing conventional partial clustering. A process in which computation is divided into parallel tasks is defined as Parallelization. In the literature several parallel partial methods had been suggested. These methods are motivated by the assumption that the distance calculations between one data point and the cluster centers are independent from each other. Distance calculation can therefore be carried out in parallel between different data points and cluster Centers.

In the literature, various methods have been proposed for MapReduce clustering. This method first divides the data set into divisions in which each division is linked with map function given the input dataset that is stored in HDFS. The map function then assigns the associated split data point in computer distances to the closest cluster. The reduction function updates then new cluster centers by calculating the average data of each cluster. It is written to the HDFS to be used for the next iteration by the map function. Until convergence, the whole process is repeated.

Two MapReduce jobs are based on this method. The first job calculates the matrix for a cluster center and the second job calculates the distances to upgrade the member matrix. The first job calculates the distances. In addition, a number of data and a portion of the membership matrix are provided to the map function of the first MapReduce job and the cluster center is generated. Sub-matrices in the cluster center matrix are then combined with the reduction function of the first MapReduce task. Compared to the first task, the second MapReduce task includes additional computations. A selection of data are collected and distance sub-matrices and membership matrices are calculated during the map function. The reduction function then fuses the sub-matrices of the partition.

In order to optimize the calculation of new centers in reduction phase, an intermediate center is calculated for each cluster after each assignment. The data is a sum of the numeric values and the category values frequencies for each cluster that are then transmitted to the reduction function. The intermediate map centers then are merged with the reduction function to update new centers for the cluster. In HDFS, finally, value of new centers is stored until convergence in the next iteration.

In this method, MapReduce programming using Support Vector Machine (SVM) model to develop and experiment a parallel SVM algorithm, and compare the result with concurrent SVM for clustering the changing document database size. The result shows that the proposed SVM MapReduce get better performance than existing methods

2. VIRTUAL DISK CLUSTERING

MapReduce is a numerical computation model specifically designed for data analysis by Google on recognizing that its handlers generate large amounts of data, such as logs and documentation for web requests. Unlike traditional parallel programming models such as MPI and OpenMP, MapReduce emphasizes on social data and not on empirical evidence. Hence, key-value pairs are its basic data sort. The delivery model for Map Reduce contains two stages, namely the Map-phase and the Reduce-phase [2].

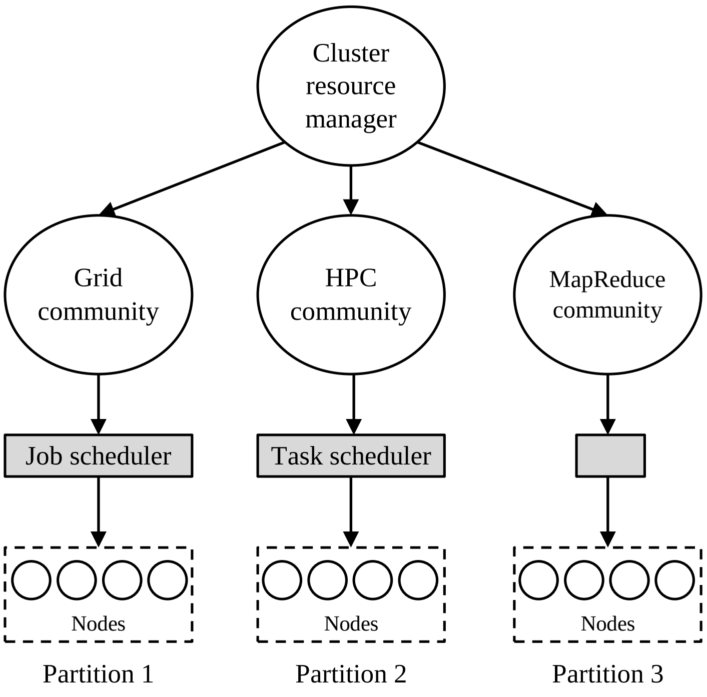

Fig.1. Source management scenario on HPC clusters

In the Map-phase, the input data is subdivided to produce intermediate vital-value pair with each divider processed through a single Map task. It is a matter that the model has not been suggested for the analysis of scientific data, but has been the JobTracker, typically running on a clusters heading node, which is the master and handles all of the Map Reduce tasks. Hadoop Generated File System handles the input/output data for a MapReduce execution.

Fault tolerance is one important characteristic of the Hadoop MapReduce system. The JobTracker assigns the job to other Task Trackers in case a TaskTracker fails to perform a task. The TaskTracker frequently report the status of their jobs to the JobTracker.

There are many clustering concerns applications in various fields including image analysis, social science, web development, problem solving, telecommunications, etc. Information network topologies is used in various requirements such as organizing of documents, perusing of documents, automated ordered representation of files, filtering of knowledge, generation of search engine results, extraction of keywords, retrieval of information. ICDM Convention classified it second of the top 10 algorithms for clustering. SVM algorithm groups *N* items in K clusters continuing a high correlation in the intra unit [3].

Clustering of SVM is sensitive to the random range of initial cluster centres. The clustering result depends largely on early centroid traditions but there are no formal rules to choose from a good set of initial centroids. Instead of testing SVM with accidental centroids, preliminary centroids are selected based on SVM provisional run. Multi-threading platforms with a view to comparing their performance. In editors SVM were implemented in various parallel paradigms such as OpenMP, MPI, and Cuda-C and their performance was compared. It is observed that OpenMP comes out on top for small datasets while cuda works well with large datasets [4]. Likewise, on OpenMP, MPI and Cuda, parallel version of SVM is implemented and the article shows that the performance of parallel SVM is much better than sequential access, and that the performance differs with different computer and hardware combinations.

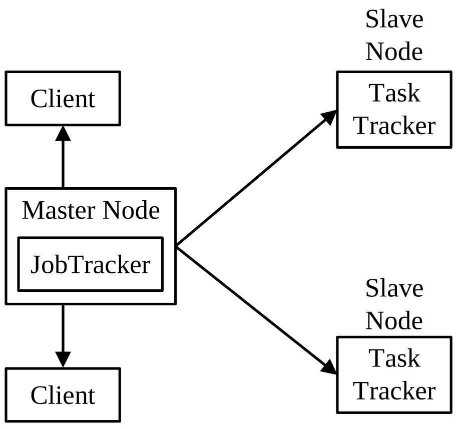

Fig.2. Hadoop Master-Slave design in type of hierarchical model

Hadoop+ is a large and diverse MapReduce system that enables GPUs and CPUs to process big data and exploit the heterogeneity setup model to help users pick cloud infrastructure for various purposes. The PMap and PReduce in Hadoop+ allows software developers to write obvious parallel CUDA/OpenCL processes working on GPUs as plug-ins, as can be seen in User-Provided PMap/PReduce Mechanism box [5].

The word count operation in the Hadoop tool happens in three stages: Mapper, Shuffle and a Reducer. Within Mapper, the source file is first divided into words and then altered with these terms to key and value pairs - the key is the phrase itself and the value 1. Consider, for example, the sentence NoteBook Pencil Banana Pencil Pen Ball Book in Mapper where the sentence would be split as words and from the initial key value pair.

The amount of intermediate data is moved from Mapper to Reducer during the shuffle process after the Map mission is complete. Shuffle data transfer occurs from the Mapper discs rather than their main associations and the in-between outcome will be sorted by the keys to bring the pairs together for the same data. The keys are clustered together in Reducer, and values are applied for identical keys [6]. Thus, there is only one pair of similar Book keys that would add the values for these keys, so that the output key value pairs and these would give the number of appearances of each term in the data, and Reducer forms a process of key initialization.

Wordcount Reducer

```
public static class decrease extends MapReduceBase
implements decrease<Text, IntWrite, Text,
IntWrite>
{
```

```
public void decrease(Text key, Iterator<IntWrite> values,
OutputCollector<Text, IntWrite> output,
Reporter reporter) throws IOException
{int s = 0;
while (values.hasNext()) {
s += values.next().get();}
output.collect(key, new IntWrite(s));}}
```

There are also several extremely valuable properties in Cloud MapReduce, something these highly-scalable systems seem to share.

Incremental Scalability: Cloud MapReduce will slowly scale out the percentage of big data nodes. Not only can a user initially run a number of servers, but if the user thinks the improvement is too slow, multiple servers can also be started in the beginning of just a computation [7].

Symmetry and Decentralization: Each conceptual node at Cloud Map Reduce now has the same set of requirements as its counterparts. Masters or slaves do not get nodes. Uniformity eliminates system failure access control, allowing, and recovery. As demonstrated by the symmetry, there is no single political agent (master) which makes the system more available.

Heterogeneity: The computational capacities of the technical nodes could be different. It would be the fast nodes doing more work than the faster nodes. Alternatively the software nodes could be spread economically. To the maximum, even idle bandwidth can be derived from network-distributed servers/computers and laptops [8].

3. CLUSTERING USING MAPREDUCE

Long latency solves cloud issues and our ultimate solution: The gap could be significant as Amazon services are conveyed through the network. In our study the SQS frequency ranges from 20ms to 100ms, even within EC2. Consequently, if we view it sequentially, a significant portion of the time is spent looking for SQS to answer. We get somewhere this provision by two methods: message filtering and multi-threading. One downside of the current configuration of CMR is that it employs no specificity optimization. It makes exclusive use of the network for I/O, trying to circumvent all local storage. Such an implementation will inevitably experience network congestion in today cloud services, when the network connections between the device nodes and cloud services become exhausted [12] - [14]. The lack of optimization of the localities [9]. The internal cloud operating system is based on a consumer product that enables queues and storage facility to co-locate on the same nodes as the computing nodes, and introduces locality hint so that we can maximize data placing. Modern data centers architecture may no longer be needed in the future [10] [11].

The main concept of the SVM algorithm implemented by using MapReduce will be described in this section. Firstly, the SVM algorithm will be explained and what parts of the process can be dispersed on a cluster computer analyzed. Then, we will discuss how we developed MapReduce SVM algorithm.

SVM is an algorithm that can separate a group object into a subset that does not overlap. It means that each object is mapped to a cluster precisely. SVM uses centroid in this paper to determine how many clusters are identified in data. It is considered to be rapid, simple and efficient in addressing the problem of data clusters.

In order to calculate the distance between the centroid and data object SVM algorithm often performs calculation. At any iteration, the number of objects requires to measure the distance, and the number of clusters created by n is k. Therefore, the distance between an irrelevant centroid object and a relevant object is always calculated. Distance calculation generally takes longer to distribute data on a cluster computer.

SVM also has another side down on the initialization of the centroid. Initially, a center, so that the clusters become unstable, is chosen randomly. The second problem is to carefully examine the number of clusters to ensure that the optimal cluster is obtained.

In MapReduce algorithm, two steps exist: the choice of the initial centroid and the following candidate centroid. These two processes have a single MapReduce task that includes and reduces one map function. In the initial step of the center map, the distance between each object and the data centroid is calculated. In the meantime, the function sort objects on the basis of the data centroid distance.

The data used in the cartographic function is extracted from a pair of HDFS files (*key*, *value*) and each of them is a data row. Key as a data object identity value. Key as a value

Map Function: Data can be divided and distributed into several pieces to each computer. The computer process can then be performed on every computer in parallel. The map function output is a pair of identification data and the medium value of the data object distance.

Reduce Function: The map output is used to reduce the function, which is the identity and distance pair of objects. In reducing function, the distance has been sorted and then the value of the lowering of the object identity and distance has been lost.

Three major functions such as Map, Combine and Reduce SVM MapReduce algorithm. The Map function played a role for every data object close to the center as a function which carried out the computing procedure. Meanwhile, a process to update new centroids has been implemented to reduce function. Combine temporarily grouped function with a similar map function to minimize network communications.

4. EXPERIMENTAL EVALUATION

Instead we are showing a few sample requests in this section, contrasting it through Hadoop to underscore their similarities and show that CMR is a technical system. To assess the efficiency of Cloud Map Reduce, we have introduced three different mutual Map Reduce programs: Word Count, Reverse Index, and String Tracking. All tests mentioned in this section will use avoidance parameters in both Cloud Map Reduce and Hadoop, unless it is mentioned. Rather than using the Amazon Elastic Map Reduce we position Hadoop ourselves in our EC2 cluster.

Random points are generated and data package consists of randomly generated clusters around the centers. A Zipf distribution samples the number of points generated inside the cluster. Note that when $\alpha = 0$, each cluster is almost equal in size and, when α is growing, it is not uniform to size the clusters. The distance between a dot and center is sampled with a fixed global standard deviation from the normal distribution. Each experiment was replicated 3 times with the same parameter set and calculated the average.

The proposed method is compared with existing MapReduce algorithm. The results of simulation shows that the proposed method achieves reduced computational time (Fig.3) and latency (Fig.4) than the existing model.

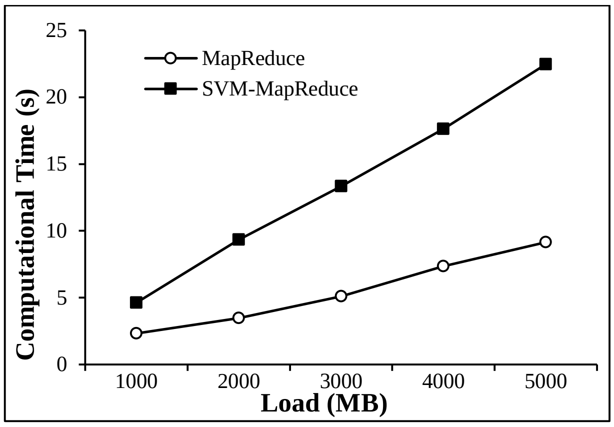

Fig.3. Computational Time

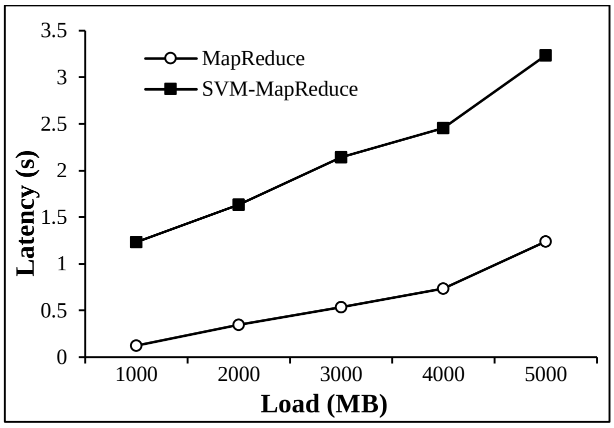

Fig.4. Latency

5. CONCLUSION

It is very clear that we can improve the design and development of large-scale systems if we create them on a cloud operating system. But, the tradeoffs made by the cloud platform for the purpose of enhancing its scalability has made it difficult for working on top structures. For example, compromising on time for the sake of achieving reliability is an unsophisticated aspect that counts as an underperformance by the system. So, using MapReduce as an example, we have revealed that such limitations of the cloud platform can be overcome without degrading its efficiency. We have deployed generic methods in this study that could be applied across a multiple range of systems.

REFERENCES

[1] Shettar Rajashree and P.V. Bhimasen, "A Review on Clustering Algorithms Applicable for Map Reduce", *Proceedings of International Conference Computational Systems for Health and Sustainability*, pp. 1-8, 2015.

[2] Olman Victor, Mao Fenglou, Wu Hongwei and Xu Ying, "Parallel Clustering Algorithm for Large Data Sets with Applications in Bioinformatics", *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, Vol. 6, No. 2, pp. 344-352, 2009.

[3] Neepa Shah and Mahajan Sunita, "Document Clustering: A Detailed Review", *International Journal of Applied Information Systems*, Vol. 4, No. 5, pp. 30-38, 2012.

[4] Sunita Bisht and Amit Paul, "Document Clustering: A Review", *International Journal of Computer Applications*, Vol. 73, No. 11, pp. 26-33, 2013.

[5] Michael Steinbach, Karypis George and Kumar Vipin, "A Comparison of Document Clustering Techniques", *Proceedings of International Workshop on Text Mining*, pp. 23-29, 2000.

[6] Jing Zhang, Gongqing Wu and Shuilong Hao, "A Parallel Clustering Algorithm with Mpi", *Proceedings of International Symposium on Parallel Architectures, Algorithms and Programming*, pp. 1-12, 2013.

[7] X. Wu, V. Kumar and Q. Yang, "Top 10 Algorithms in Data Mining", *Knowledge and Information Systems*, Vol. 14, No. 1, pp. 1-37, 2008.

[8] V.S. Bawane and M. Sandesha Kale, "Clustering Algorithms in MapReduce: A Review", *Proceedings of National Conference on Recent Trends in Computer Science and Engineering*, pp. 31-33, 2015.

[9] L. Yang, C. Chiu Steve and W.K. Liao, "High Performance Data Clustering: A Comparative Analysis of Performance for GPU, RASC, MPI, and OpenMP Implementations", *Supercomputing*, Vol. 70, No. 71, pp. 284-303, 2014.

[10] J.S. Kang, S. Yeon Lee and K.M. Lee, "Performance Comparison of OpenMP, MPI, and MapReduce in Practical Problems", *Advances in Multimedia*, Vol. 2015, pp. 1-9, 2015.

[11] M Arvindhan and Abhineet Anand, "Scheming an Proficient Auto Scaling Technique for Minimizing Response Time in Load Balancing on Amazon AWS Cloud", *Proceedings of International Conference on Advances in Engineering Science Management and Technology*, pp. 1-8, 2019.

[12] S. Dhillon Inderjit and S. Modha Dharmendra, "A Data-Clustering Algorithm on Distributed Memory Multiprocessors", *Proceedings of International Workshop on Large Scale Parallel KDD Systems*, pp. 245-260, 2002.

[13] Bhimani Janki, Leeser Miriam and Mi Ningfang, "Accelerating K-Means Clustering with Parallel Implementations and GPU Computing", *Proceedings of IEEE International Conference on High Performance Extreme Computing*, pp. 1-12, 2015.

[14] Aristidis Likas, Nikos Vlassis and J. Verbeek Jakob, "The Global k-Means Clustering Algorithm", *Pattern Recognition*, Vol. 36, No. 2, pp. 451-463, 2003.