ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 8, Issue 7, July 2019

Optimised Parallel K-Means Clustering using YARN in Hadoop

Shesh Narayan Mishra[1], Shalini Vashisht[2]

M.Tech, Student, Department of CSE, SRCEM College, Palwal, Haryana, India[1]

Assistant Professor, Department of CSE, SRCEM College, Palwal, Haryana, India[2]

ABSTRACT: Parallel K-Means clustering using YARN in Hadoop is one of the most commonly used data processing algorithms. Over half a century, K-means remains the most popular clustering algorithm because of its simplicity. Recently, as data volume continues to rise, some researchers turn to Yet Another Resource Negotiator (YARN) to get high performance. However, MapReduce is unsuitable for iterated algorithms owing to repeated times of restarting jobs, big data reading and shuffling. In this scheme, we address the problems of processing large-scale data using K-means clustering algorithm and propose a novel processing model in with Euclidean distance function and YARN to eliminate the iteration dependence and obtain high performance. We analyze and implement our idea. Extensive experiments on our cluster demonstrate that our proposed methods are efficient, robust and scalable.

KEYWORDS: Machine Learning, K-Means, Big Data, Apache Hadoop, Yet Another Resource Negotiator (YARN), Hadoop Distributed File System (HDFS), MapReduce.

I. INTRODUCTION

Since when it started to store information practically, this information altogether expands each year. One reason is the expanding itemizing in the age of information, other than existing a huge assortment in these informational collections. At the point when there are numerous bytes of date, there are a ton of proficiency issues to execute them customarily. This affliction was settled by Dean and Ghemawat when they proposed the programming model MapReduce. It works essentially by utilizing the guide and lessen, which ones are intelligent capacities, and it changes over information in key-esteem sets to move them over the system. Your fundamental bit of leeway is preparing assignments in parallel. This programming model is executed in a circulated record framework, where huge informational collections are divided in pieces and they are put away in various hubs. A disseminated document framework, which one can execute this model, is Apache Hadoop. The principle explanation behind its utilization is the offices allowed to the parallel programming, which one expels from the designer the need to stress over burden adjusting, information dispersion, task assignments and different challenges in MapReduce. Such challenges are the reason for some individuals selecting by successive programming, whose calculations are worked to emphasize assignments in a specific request. This model is simple, straightforward and great to work with little informational indexes, however it relies upon information and finished undertakings. These perspectives are the explanation behind successive writing computer programs being moderate to process enormous informational indexes. As a rule, the consecutive writing computer programs is unfeasible. Not at all like what happens in parallel programming, whose productivity is better for permitting a few centers than execute similar undertakings with various bits of informational index. It is now that there is an improvement in execution since information are prepared in parallel. By showing these points of interest in a domain including Big Data, this paper plans to change over consecutive calculations in parallel calculations, utilizing the programming model MapReduce, so as to have the option to be executed in a group with the structure Apache Hadoop, looking at and assessing their presentation. This assessment depends on acknowledging tests with various situations and gathering calculations times of execution. These situations fluctuate the quantity of gatherings in three and ten, the quantity of centers in two, four and seven and they utilize a dataset with numerous articles and few qualities for item and another with less articles and more characteristics for the article. It reasons that the best situation accomplished in

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 8, Issue 7, July 2019

the dataset with more articles was the parallel K-Means working in four centers, both isolating in three gatherings or in ten. In any case, in the dataset with less articles, the parallel K-Means working with four centers was the best partitioning in three gatherings and the parallel K-Means working with seven centers was the best isolating in ten gatherings. The parallel calculations working with two centers accomplished occasions of execution fundamentally the same as when they are contrasted and the successive calculations. What's more, the parallel calculations with seven centers were comparative when they are contrasted and the parallel calculations with four centers.

The K-Means algorithms is a "greedy algorithms", that is, they are algorithms that use brute force to find an optimal or near optimal solution. Therefore, clustering techniques process small amounts of data in an acceptable time. The same does not occur for large amounts of data, whose execution time increases exponentially. For this reason, Linden (2009, p.19) defines clustering as a problem of complexity of an exponential order.

Based on the theoretical frameworks of the MapReduce programming model, Apache Hadoop and the clustering technique, this work has the hypothesis that it is possible to find a better performance when processing large datasets for the K-Means and K-Medoids algorithms. This can be possible by implementing them so that they can be run in parallel on MapReduce on HDFS. Therefore, this paper proposes: 1: Understand and master the operation of the MapReduce programming model, Apache Hadoop, and the Parallel K-Means algorithms. 2: Investigate and implement K-Means sequential algorithms so that they can be converted to a parallel algorithm using the MapReduce programming model.3: Install and configure a cluster of nodes using the Apache Hadoop implementation.4: Implement the parallel algorithms that correspond to the sequential algorithms K-Means and maintaining the same functionality and same results.5: Collect and tabulate the results of sequential and parallel algorithm runs.6: Compare the performance of the sequential algorithm results in relation to the parallel algorithms.7:Evaluate the performance of parallel algorithms by varying the node number in the cluster.

II. RELATED WORK

"*By Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, Christos Kozyrakis*" in the paper "*Evaluating MapReduce for Multi-core and Multiprocessor Systems*" depicted that, the suitability of the MapReduce model for multi-core and multi-processor systems. MapReduce was created by Google for application development on data-centers with thousands of servers. It allows programmers to write functional-style code that is automatically parallelized and scheduled in a distributed system. We describe Phoenix, an implementation of MapReduce for shared-memory systems that includes a programming API and an efficient runtime system. The Phoenix runtime automatically manages thread creation, dynamic task scheduling, data partitioning, and fault tolerance across processor nodes. We study Phoenix with multi-core and symmetric multiprocessor systems and evaluate its performance potential and error recovery features. We also compare MapReduce code to code written in lower-level APIs such as P-threads. Overall, we establish that, given a careful implementation, MapReduce is a promising model for scalable performance on shared-memory systems with simple parallel code As multi-core chips become ubiquitous, we need parallel programs that can exploit more than one processor. Traditional parallel programming techniques, such as message passing and shared-memory threads, are too cumbersome for most developers. They require that the programmer manages concurrency explicitly by creating threads and synchronizing them through messages or locks. They also require manual management of data locality. Hence, it is very difficult to write correct and scalable parallel code for non-trivial algorithms. Moreover, the programmer must often re-tune the code when the application is ported to a different or larger-scale system.

"*By Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung"* in the paper "*The Google File System*" depicted that, they 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. While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points. The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 8, Issue 7, July 2019

generation and processing of data used by our service as well as research and development efforts that require large data sets. The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients. In this paper, we present file system interface extensions designed to support distributed applications, discuss many aspects of our design, and report measurements from both micro-benchmarks and real world use. We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space. First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines. The quantity and quality of the components virtually guarantee that some are not functional at any given time and some will not recover from their current failures. We have seen problems caused by application bugs, operating system bugs, human errors, and the failures of disks, memory, connectors, networking, and power supplies. Therefore, constant monitoring, error detection, fault tolerance, and automatic recovery must be integral to the system. Second, files are huge by traditional standards. Multi-GB files are common. Each file typically contains many application objects such as web documents. When we are regularly working with fast growing data sets of many TBs comprising billions of objects, it is unwieldy to manage billions of approximately KB-sized files even when the file system could support it. As a result, design assumptions and parameters such as I/O operation and blocksizes have to be revisited

"*By J. Macqueen*" in the paper "*Some Methods For Classification And Analysis Of Multivariate Observations*" depicted that, the main purpose of this paper is to describe a process for partitioning an N-dimensional population into k sets on the basis of a sample. The process, which is called 'k-means,' appears to give partitions which are reasonably efficient in the sense of within-class variance. That is, if p is the probability mass function for the population, S = {S1, S2, - * *, Sk} is a partition of EN, and ui, i = 1, 2, * - , k, is the conditional mean of p over the set Si, then W2(S) = ff= ISif z - u42 dp(z) tends to be low for the partitions S generated by the method. We say 'tends to be low,' primarily because of intuitive considerations, corroborated to some extent by mathematical analysis and practical computational experience. Also, the k-means procedure is easily programmed and is computationally economical, so that it is feasible to process very large samples on a digital computer. Possible applications include methods for similarity grouping, nonlinear prediction, approximating multivariate distributions, and nonparametric tests for independence among several variables. In addition to suggesting practical classification methods, the study of k-means has proved to be theoretically interesting. The k-means concept represents a generalization of the ordinary sample mean, and one is naturally led to study the pertinent asymptotic behavior, the object being to establish some sort of law of large numbers for the k-means. The k-means process was originally devised in an attempt to find a feasible method of computing such an optimal partition. In general, the k-means procedure will not converge to an optimal partition, although there are special cases where it will. So far as the author knows, there is no feasible, general method which always yields an optimal partition. Cox has solved the problem explicitly for the normal distribution in one dimension, with k = 2, 3, - - *, 6, and a computational method for finite samples in one dimension has been proposed by Fisher. A closely related method for obtaining reasonably efficient 'similarity groups' has been described by Ward. Also, a simple and elegant method which would appear to yield partitions with low within-class variance, was noticed by Edward Forgy and Robert Jennrich, independently of one another, and communicated to the writer sometime in 1963. This procedure does not appear to be known to workers in taxonomy and grouping, and is therefore described in section 3. For a thorough consideration of the biological taxonomy problem and a discussion of a variety of related classification methods, the reader is referred to the interesting book by Sokal and Sneathm therefore described a procedure called "adaptive sample set construction," which involves the use of what amounts to the k-means process. This is the earliest explicit use of the process with which the author is familiar. Although arrived at in ignorance of Sebestyen's work and monograph.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 8, Issue 7, July 2019

III. PROPOSED WORK

The proposed scheme is responsible to generate the voluminous corpus data from various sources and then migrating to HDFS thereinafter the pre-processing will remove the noise from the voluminous dataset and create the segregated data partitions to be passed to YARN for clusters creating and Parallel processing. Consequently, K-Means along-with Euclidean Distance function will calculate the data similarity distance and error rate among the cluster using a parallel processing model by YARN and at last the resulted clustered will be promoted as a result. Below diagram elaborate the proposed scheme.

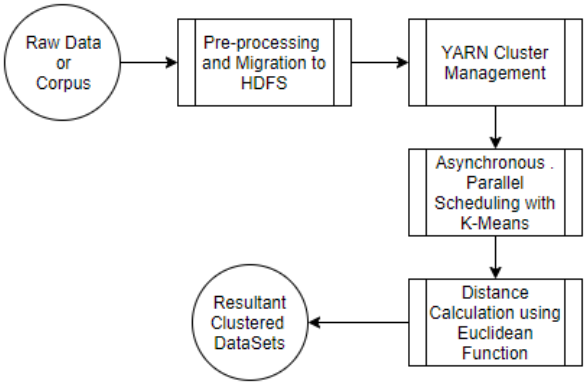

Figure 1: Proposed Scheme Comprising of Workflow Model Yielding Raw Data or Corpus Collection, Pre-Processing using Hadoop Parallel File System, YARN Cluster Management, Asynchronous Parallel Scheduling with K-Means and Cluster Distance Calculations using Euclidean Function.

IV. PSEUDO CODE

Step 1

1. Select initial cluster points.
2. omp_set_num_threads(20); //YARN
3. /* Start of parallel code */

```
#pragma omp parallel sections
{
    #pragma omp section
      do_clustering(0);
    #pragma omp section
    do_clustering(1);
}
```

4. Implicit barrier /* End of parallel code */
5. Sum up the cluster wise totals and cluster wise count.
6. Calculate the new cluster points. Compare the new cluster points with previous cluster points. If both are same go to step 7, other wise Set new cluster points as current cluster points and go to step 3.
7. Display final cluster points.

Step 2

Calculating Euclidean Distance Where as the initial distance is :-

```
dist=0
for d=1 to N   // d = dimension
dist=dist+(x1[d]-x2[d])^2
```

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 8, Issue 7, July 2019

next

return sqrt(dist)

end function

V. RESULTS AND DISCUSSIONS

The proposed scheme concludes that Parallel K-Means is best for the Document clustering. We have discussed about Parallel K-means with pre-processing techniques. In the next chapter we provide the Experimental setup with result analysis on different datasets. The data set contains the textual data. We have divided data into four inputs.

S. No	Number of Doc in Each Sub-directory	Total Number of Documents	Unique Words Identified	Number of Cluster
1	250	5000	69,996	20
2	500	10,000	1,11,203	20
3	750	15,000	1,35,162	20
4	1000	20,000	1,53,832	20

Table 1: Detailed Information about Data inputs of 20_Newsgroups

The above table contains the data which we identify while pre-processing of the input data of Newsgroups data set. This data set contains 20 sub-directories and each sub-directory contains 1000 documents, so total documents are 20,000 which are in the categorical form. After completion of pre-processing , we get two vector form files, one contains the term frequencies of the input with the category they belongs to; and another file is centroid file which contains the initial centroids on random basis. Comparing Execution time of Newsgroups dataset Provide by K-Means algorithm on stand-alone computer and Multi Node Computer , based on YARN with Parallel K-Means with Distance Function vide Euclidean Technique.

S. No.	Corpus Size (Number of Documents)	Computation Time		
		Single Node With K-Means	YARN Multiple Node with K-Means	YARN Multiple Node with K-Means and Euclidean Distance Calculation
1	5000 (18.26 MB)	09:26	02:01	01:23
2	10,000 (39.96 MB)	19:14	05:39	03:59
3	15,000 (55.51 MB)	20:02	06:23	04:12
4	20,000 (80.26 MB)	27:36	11:28	8:02

Table 2: Comparison of Computation time (Newsgroups) on Stand-alone and Multi Node System

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 8, Issue 7, July 2019

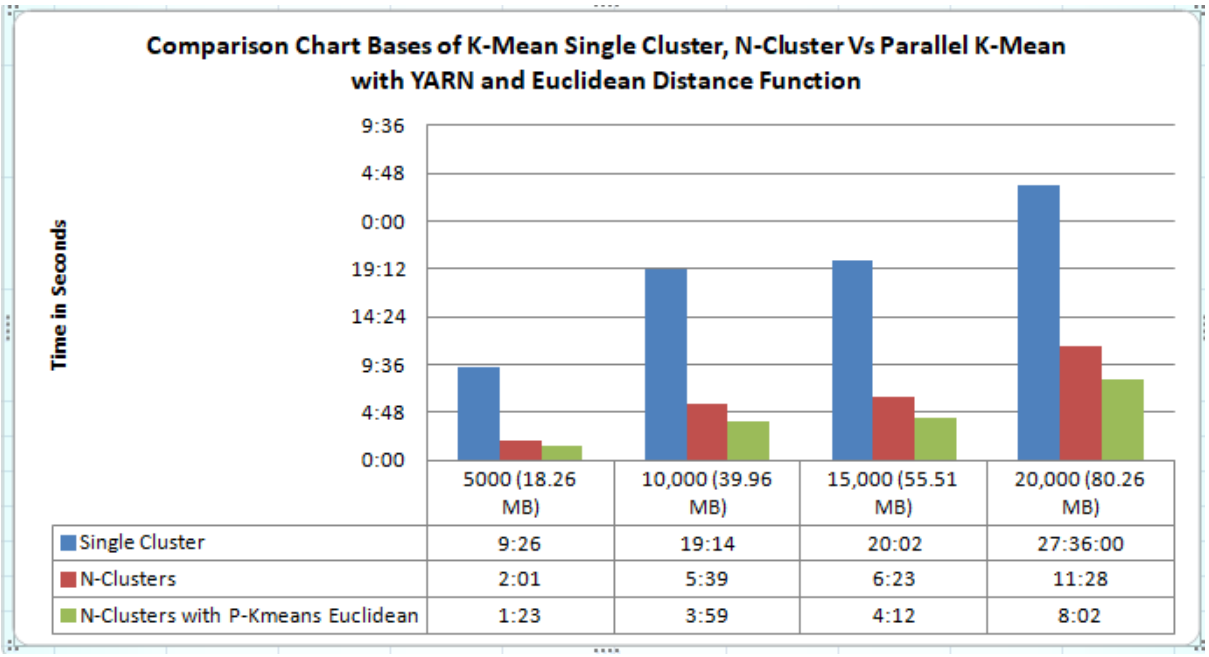

Figure2: Comparison of Computation time (Newsgroups) on Stand-alone and Multi Node System and Proposed Scheme

VI. CONCLUSION AND FUTURE SCOPE

In this scheme, we have applied Parallel K-Means using YARN and Euclidean Distance Function technique to k-means clustering algorithm and clustered over 10 million data points up to 109.8MB in size. We compared the results with Parallel k-means clustering with distance function. We have shown that k-means algorithm can be successfully parallelized and clustered on commodity hardware. MapReduce can be used for k-means clustering. The results also show that the clusters formed using MapReduce is identical to the clusters formed using K-Mean algorithm. Our experience also shows that the overheads required for MapReduce algorithm and the intermediate read and write between the mapper and reducer jobs makes it unsuitable for smaller datasets. However, adding a combiner between Map and Reduce with Clusters using Data Nodes vide YARN job performance increases by huge rift and the amount of intermediate read/write. We also noted that the number of nodes available for map tasks affect performance and more the number of nodes, the better is the performance. Therefore, using the Euclidean Distance Function the distance between the nodes and clusters are calculated on short time span. Therefore, we believe, that proposed scheme will be a valuable tool for clustering larger datasets that are distributed and cannot be stored on a single node.

For future scope we can incorporate the incremental Parallel K-Means using YARN and Euclidean Function. Consequently, the existing clusters will be compared with new incorporated or inculcated clusters. Therefore there is no need to form the clusters from scratch again and again.

REFERENCES

[1] Rasmussen, E.M., Willett, P.: Efficiency of Hierarchical Agglomerative Clustering
[2] Using the ICL Distributed Array Processor. Journal of Documentation 45(1), 1–24 (1989)
[3] Li, X., Fang, Z.: Parallel Clustering Algorithms. Parallel Computing 11, 275–290 (1989)
[4] Rasmussen, E.M., Willett, P.: Efficiency of Hierarchical Agglomerative Clustering Using the ICL Distributed Array Processor. Journal of Documentation 45(1), 1–24 (1989)
[5] Li, X., Fang, Z.: Parallel Clustering Algorithms. Parallel Computing 11, 275–290 (1989)
[6] Olson, C.F.: Parallel Algorithms for Hierarchical Clustering. Parallel Computing 21(8), 1313–1325 (1995)
[7] Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: Proc. of Operating Systems Design and Implementation, San Francisco, CA, pp. 137–150 (2004)

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 8, Issue 7, July 2019

[8] Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Communications of The ACM 51(1), 107–113 (2008)
[9] Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., Kozyrakis, C.: Evaluating MapReduce for Multi-core and Multiprocessor Systems. In: Proc. of 13th Int.
[10] Symposium on High-Performance Computer Architecture (HPCA), Phoenix, A (2007)
[11] Lammel, R.: Google's MapReduce Programming Model - Revisited. Science of Computer Programming 70, 1–30 (2008)
[12] Hadoop: Open source implementation of MapReduce, http://lucene.apache.org/hadoop/
[13] Ghemawat, S., Gobioff, H., Leung, S.: The Google File System. In: Symposium on Operating Systems Principles, pp. 29–43 (2003)
[14] 14. MacQueen, J.: Some Methods for Classification and Analysis of Multivariate Observations. In: Proc. 5th Berkeley Symp. Math. Statist, Prob., vol. 1, pp. 281–297 (1967)
[15] Borthakur, D.: The Hadoop Distributed File System: Architecture and Design (2007)
[16] Xu, X., Jager, J., Kriegel, H.P.: A Fast Parallel Clustering Algorithm for Large Spatial Databases. Data Mining and Knowledge Discovery 3, 263–290 (1999)
[17] Aditya B. Patel, Manashvi Birla, Ushma Nair, Addressing Big Data Problem Using Hadoop and Map Reduce,6-8 Dec. 2012.
[18] Ashish A. Golghate, 2 Shailendra W. Shende, Parallel K-Means Clustering Based on Hadoop and Hama. International Journal of Computing and Technology, Volume 1, Issue 3, April 2014 ISSN : 2348 - 6090.
[19] An Introduction to apache hadoop [online], available: http://opensource.com/life/14/8/intro-apache-hadoop-big-data
[20] Brandeis University, Networks and Distributed Computing, Introduction to Hadoop, [online], available http://www.cs.brandeis.edu/~cs147a/lab/hadoop-intro/
[21] C.W Tsai, B.C Huang, M. Chiang, A novel spiral optimization for clustering, Mobile, Ubiquitous, and Intelligent Computing, pp. 621-628. Springer, 2014.
[22] Diana MacLean, A Very Brief Introduction to MapReduce, [online], available at http://hci.stanford.edu/courses/cs448g/a2/files/map_reduce_tutorial.pdf, 2011
[23] Garlasu, D., Sandulescu, V. ;Halcu, I. , Neculoiu, G, A Big Data implementation based on Grid Computing, 17-19 Jan. 2013.
[24] Getting Started On KMeans Clustering, [online], available at http:// cmj4.web.rice.edu/Kmeans.html
[25] Hadoop [online]. Available: http://hadoop.apache.org/. [6 October 2013].