A SURVEY ON CLUSTERING ALGORITHMS IN MAPREDUCE AND THEIR EFFICIENCY

B.S.Sangeetha[1], Dr. N.Chandrakala[2]

B.S.Sangeetha[1], Research Scholar, Assistant Professor, Department of Computer Science Shri Sakthikailassh Women's College Salem. meetsangee@gmail.com

Dr. N.Chandrakala[2], Head of the Department of Computer Science, SSM College of Arts and Science, Kumarapalayam. nchandrakala15@gmail.com

ABSTRACT

Hierarchical agglomerative clustering (HAC) is a clustering method widely used in various disciplines from astronomy to zoology. HAC is useful for discovering hierarchical structure embedded in input data. The cost of executing HAC on large data is typically high, due to the need for maintaining global inter-cluster distance information throughout the execution. On implementation on HAC in MapReduce can be a most efficient idea to improve the efficiency of data storage. In this work, we have surveyed many clustering algorithms and their implementation in MapReduce and their pros and cons in them.

INTRODUCTION

Hierarchical clustering algorithms are either top-down or bottom-up. Bottom-up algorithms treat each document as a singleton cluster at the outset and then successively merge (or *agglomerate*) pairs of clusters until all clusters have been merged into a single cluster that contains all documents. Bottom-up hierarchical clustering is therefore called *hierarchical agglomerative clustering* or *HAC*. Top-down clustering requires a method for splitting a cluster. It proceeds by splitting clusters recursively until individual documents are reached.

The HAC (Hierarchical Agglomerative Clustering) method is one of the most widely used and most simple clustering algorithm. The base steps of HAC are the followings:

- compute the proximity matrix containing the distance values between the elements
- find the pair of elements with smallest distance value. These two elements are merged into a single cluster.
- if the termination criterion is met, terminate the algorithm, otherwise go back step 1

In the praxis, the termination condition is either an upper threshold on the minimum distance or an upper threshold on the count of generated clusters. The partitioning generated by the proposed HAC method should meet the requirements (R1), (R2). It can be seen that many different partitioning can be generated fulfilling (R1) and (R2). In order to minimize the cost of the resulting partitioning, an objective function is defined to determine the optimal solution. The goal function to be minimized is the size of the partition.

Hierarchical agglomerative clustering (HAC) is a clustering method widely used in various disciplines from astronomy to zoology. HAC is useful for discovering hierarchical structure embedded in input data. The cost of executing HAC on large data is typically high, due to the need for maintaining global inter-cluster distance information throughout the execution

Hierarchical Agglomerative Clustering (HAC) is one of the most popular clustering methods that can find hierarchical structure hidden in input data in many disciplines from astronomy to zoology, researchers frequently rely on HAC for data analysis.

MapReduce

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster. A MapReduce program is composed of a Map() procedure (method) that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a Reduce() method that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance. As such, a single-threaded implementation of MapReduce will usually not be faster than a traditional (non-MapReduce) implementation; any gains are usually only seen with multi-threaded implementations. MapReduce is a framework for processing parallelizable problems across large datasets using a large number of computers (nodes), collectively referred to as a cluster (if all nodes are on the same local network and use similar hardware) or a grid (if the nodes are shared across geographically and administratively distributed systems, and use more heterogenous hardware). Processing can occur on data stored either in a file system (unstructured) or in a database (structured). MapReduce can take advantage of the locality of data, processing it near the place it is stored in order to minimize communication overhead.

- "Map" step: Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of redundant input data is processed.
- "Shuffle" step: Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.
- "Reduce" step: Worker nodes now process each group of output data, per key, in parallel.

MapReduce allows for distributed processing of the map and reduction operations. Provided that each mapping operation is independent of the others, all maps can be performed in parallel – though in practice this is limited by the number of independent data sources and/or the number of CPUs near each source. Similarly, a set of 'reducers' can perform the reduction phase, provided that all outputs of the map operation that share the same key are presented to the same reducer at the same time, or that the reduction function is associative. While this process can often appear inefficient compared to algorithms that are more sequential (because multiple rather than one instance of the reduction process must be run), MapReduce can be applied to significantly larger datasets than "commodity" servers can handle – a large server farm can use MapReduce to sort a petabyte of data in only a few hours. The parallelism also offers some possibility of recovering from partial failure of servers or storage during the operation: if one mapper or reducer fails, the work can be rescheduled – assuming the input data is still available.

Another way to look at MapReduce is as a 5-step parallel and distributed computation:

1. Prepare the Map() input – the "MapReduce system" designates Map processors, assigns the input key value *K1* that each processor would work on, and provides that processor with all the input data associated with that key value.
2. Run the user-provided Map() code – Map() is run exactly once for each *K1* key value, generating output organized by key values *K2*.
3. "Shuffle" the Map output to the Reduce processors – the MapReduce system designates Reduce processors, assigns the *K2* key value each processor should work on, and provides that processor with all the Map-generated data associated with that key value.
4. Run the user-provided Reduce() code – Reduce() is run exactly once for each *K2* key value produced by the Map step.
5. Produce the final output – the MapReduce system collects all the Reduce output, and sorts it by *K2* to produce the final outcome.

These five steps can be logically thought of as running in sequence – each step starts only after the previous step is completed – although in practice they can be interleaved as long as the final result is not affected. In many situations, the input data might already be distributed ("sharded") among many different servers, in which case step 1 could sometimes be greatly simplified by assigning Map servers that would process the locally present input data. Similarly, step 3 could sometimes be sped up by assigning Reduce processors that are as close as possible to the

Map-generated data they need to process. There are number of algorithms used to cluster data with MapReduce. Here we have discussed a very few algorithms here.

2. LITERATURE REVIEW

1. Wei "et all" [1] propose a parallel *k*-means clustering algorithm based on MapReduce which is a simple yet powerful parallel programming technique. The experimental results demonstrate that the proposed algorithm can scale well and efficiently process large datasets on commodity hardware. we adapt *k*-means algorithm in MapReduce framework which is implemented by Hadoop to make the clustering method applicable to large scale data. By applying proper <key, value> pairs, the proposed algorithm can be parallel executed effectively. But this algorithm have the following drawbacks. a) They assume that all objects can reside in main memory at the same time; b) Their parallel systems have provided restricted programming models and used the restrictions to parallelize the computation automatically.

Hui et al [2] designed a model Text clustering has been adopted as an effective tool for sorting a large amount of documents to assist readers with different interests. It mainly based on the famous cluster assumptions: the elements in the same cluster would be more similar than the elements outside the cluster. As an unsupervised machine learning methods, there is no need training process, with a higher degree of flexibility and the automation of handling capacity, text clustering has become one of important contents in text mining. Text clustering tasks with high dimension and large datasets will extremely compromise the performance of clustering algorithms. Therefore, the common technique is feature selection from reduced feature dimensions and the utilization of distributed processing. These methods have the same disadvantage that need make sure the number of clustering in advance, the time and space complex rate of them make it almost impossible for them to process the large dataset, so they can not apply in text clustering satisfactory.

To solve these problems, many improved algorithms are proposed aimed to improve performance, most of them choose to reduce the distance calculation process. Like the method based on the k-d tree structure and pruning function proposed by Alsabti , P-CLUSTER, the parallel clustering algorithm utilizes three kinds of pruning methods proposed by Dan Judd and the parallel algorithm based on the k-d tree structure proposed by Attila Gursoy and ilker Cengiz. Obviously, by reducing distance or similarity calculation, the algorithm doesn't guarantee accuracy. It [2] use parallel computing, assign the distance computing to different nodes in a distributed environment, which improved the efficiency and ensure the effectiveness, also the algorithm is relatively simpler. But this work supports only text based clustering of data.

Chen Jin et al [3] proposed an algorithm called DiSC: A Distributed Single-Linkage Hierarchical Clustering Algorithm has been widely used in numerous applications due to its informative representation of clustering results. But its higher computation cost and inherent data dependency prohibits it from performing on large datasets efficiently. As mentioned earlier, hierarchical clustering provides a rich representation about the structure of the data points without predetermining the number of clusters. However the complexity is at least quadratic in the number of data points. Due to the advance of modern computer architectures and largescale system, there are various implementations using different kinds of platform. including multi-core GPU, MPI as well as recently popularized MapReduce framework. It is a parallel single-linkage hierarchical clustering algorithm based on SLINK. SHRINK exhibits good scaling and communication behavior, and only keeps space complexity in O(*n*) with *n* being the number of data points. The algorithm trades duplicated computation for the independence of the subproblem, and leads to good speedup. However, the authors only evaluate SHRINK on up to 36 shared memory cores, achieving a speedup of roughly 19. In this work, they presented a distributed singlelinkage hierarchical clustering algorithm (DiSC) based on MapReduce, one of the most popular programming models used for scalable data analysis. The main idea is to divide the original problem into a set of overlapped subproblems, solve each subproblem and then merge the sub-solutions into an overall solution. Further, our algorithm has sufficient flexibility to be used in practice since it runs in a fairly small number of MapReduce rounds through configurable parameters for data merge phase. When the dataset is large and high-dimensional, the time complexity would be an obstacle to apply the algorithm. Moreover, the algorithms usually require storing the entire distance matrix in memory, which makes it more challenging for a single machine to compute. Several efforts were taken to parallelize hierarchical clustering algorithms. In their experiments, they evaluate the DiSC algorithm using synthetic datasets with varied size and dimensionality, and find that DiSC provides a scalable speedup of up to 160 on 190 computer cores. Though it supports very good space complexity, it left time complexity factor unaddressed.

Koji Kumanami et al [4] focused on two particular types of predicate-argument relations as matrix attributes (columns), which however makes the resulting matrix larger and sparser than the one encoding mere word occurrences. This property is not desirable for some applications including phrase clustering since we do not know in advance how many synonymous phrases would exist in the data at hand. To alleviate the problem, a large-scale corpus should be used as a knowledge source from which a predicate-argument relations are extracted. Since handling a larger data requires a scalable framework, it propose an efficient approximation for hierarchical co-clustering using a parallel distributed programming model MapReduce. It doesn't bother about the time complexity of data storage.

Haimonti at al [5] proposed An algorithm called The PArallel RAndom-partition Based HierarchicaL clustEring algorithm (PARABLE) randomly splits the large data set into several smaller partitions on the mapper. Each partition is distributed to a reducer on which the sequential hierarchical clustering algorithm is executed. After this *local clustering* step, the dendrograms obtained at each reducer are aligned together.

In this paper, they presented a novel parallel random-partition based hierarchical clustering algorithm that solves the problem using a divideand-conquer approach – first, the algorithm randomly chops a large data set into smaller partitions and distributes it amongst available nodes; then a sequential hierarchical clustering algorithm is applied to each partition. A global clustering phase ensures that results of local clusterings are further analyzed and put into appropriate clusters. The algorithm is implemented on an Apache Hadoop framework using the MapReduce programming paradigm. But it is more efficient only when data are of same variety.

Zeehasham et al [6] modelled an approach called MrMC-MinH, which is an extension of our previously developed, greedy clustering algorithm MC-MinH [7]. The key contributions of this work include: (i) development of the distributed map-reduce based implementation of clustering algorithm and (ii) ability to perform hierarchical agglomerative clustering instead of a greedy clustering approach. One of the main challenges with current methods for metagenome clustering is the ability to handle large volumes of data while maintaining the quality of clustering. This is particularly an issue for terabyte-scale metagenomics projects.

Yongli Wang et al [7] create an algorithm called NBC algorithm based on quad-tree and mutual nearest neighbor technology that focuses on the partition of large data sets and proposes an efficient hierarchical clustering, NBC. NBC can greatly reduce the time complexity of hierarchical using NNB. The performance of NBC is remarkable in two-dimensional space. Based on NNB, each group of data is independent with each other, so the multi-process and MapReduce models can be used to accelerate computing of NBCP. Nearest neighbor search is an important method in many fields. The concept of NNB provides us efficient method only with nearest neighbor search in large datasets

REFERENCES

[1] Weizhong Zhao1,2, Huifang Ma1,2, and Qing He1,, "Parallel K-Means Clustering Based on MapReduce", The Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Graduate University of Chinese Academy of Sciences {zhaowz,mahf,heq}@ics.ict.ac.cn.

[2] Hui Gao, Jun Jiang, Li She, Yan Fu, "A New Agglomerative Hierarchical Clustering Algorithm Implementation based on the Map Reduce Framework ",International Journal of Digital Content Technology and its Applications Volume 4, Number 3, June 2010

[3] Chen Jin, Md. Mostofa Ali Patwary, Ankit Agrawal, William Hendrix, Wei-keng Liao, Alok Choudhary Northwestern University Evanston, IL 60208, "DiSC: A Distributed Single-Linkage Hierarchical Clustering Algorithm using MapReduce".

[4] Koji Kumanami, Kazuhiro Seki, Kuniaki Uehara," Agglomerative Co-Clustering for Synonymous Phrases Based on Common Effects and Influences", 2013 IEEE International Conference on Big Data.

[5] Haimonti Dutta," PARABLE:A Parallel RAndom-partition Based HierarchicaL ClustEring Algorithm for the MapReduce Framework"

[6] Zeehasham Rasheed, Huzefa Rangwala," A Map-Reduce Framework for Clustering Metagenomes",2013 IEEE 27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum.

[7] Wei Zhang*t, Gongxuan Zhang*, Yongli Wang*, Zhaomeng Zhu*, Tao Li," NNB: An Efficient Nearest Neighbor Search Method for Hierarchical Clustering on Large Datasets "Proceedings of the 20 IS IEEE 9th International Conference on Semantic Computing (IEEE ICSC 20 IS)

[8] W-Y. Chen, Y. Song, H. Bai, c.-J. Lin, and E. Chang, "Parallel spectral clustering in distributed systems," Pattern Analysis and Machine intelligence, iEEE Transactions on, vol. 33, pp. 568-586, March 20 II.

[9] T. Sun, C. Shu, F. Li, H. Yu, L. Ma, and Y. Fang, "An efficient hierarchical clustering method for large datasets with map-reduce," in Parallel and Distributed Computing, Applications and Te chnologies, 2009 International Corifer- ence on, pp. 494-499, Dec 2009.

[l0] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," Acta iriformatica, vol. 4, no. I, pp. 1-9, 1974.

[11] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, "Computational geometry," in Computational Geometry, pp. 291-306, Springer Berlin Heidelberg, 2000.

[12] H. Samet and R. E. Webber, "Storing a coUection of polygons using quadtrees," ACM Transactions on Graphics (TOG), vol. 4, no. 3, pp. 182-222, 1985.

[13] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 107-113, 2008.

[l4] B. Mirkin, Mathematical classification and clustering: From how to what and why, pp. 139-158. Springer, 1998.

[15] C. Ding and X. He, "Cluster aggregate inequality and multi-level hierarchical clustering," in Knowledge Discov- ery in Databases: PKDD 2005, vol. 3721 of Lecture Notes in Computer Science, pp. 71-83, Springer Berlin Heidelberg, 2005.