Big Data Clustering using Parallel Differential Evolution Algorithm

Pyae Pyae Win Cho, Thi Thi Soe Nyunt
University of Computer Studies, Yangon
pyaepyaewincho@ucsy.edu.mm, thithi@ucsy.edu.mm

Abstract

Clustering is the task of discovering group of similar objects or items and there have been many applications for clustering such as image segmentation, document retrieval and data mining. The increasing volumes of information emerging by the development of technology makes clustering of very large scale of data a challenging task. Differential evolution (DE) algorithm is an innovative evolutionary algorithm (EA) for global optimization, where the mutation operator is based on the distribution of solutions in the population. Clustering can be viewed as optimization problem where the task is finding the optimal cluster solution. To deal with clustering of huge amount of data sets, the use of classical DE is time-consuming that it is infeasible. This paper proposes a parallel differential evolution algorithm for clustering enormous data based on Spark framework. The proposed approach will be efficient for large-scale data clustering.

1. Introduction

An exponential amount of data is being generated daily due to the rapid development of information technology. Big data is a buzz-word referred to data sets whose size or type is beyond the ability of traditional database management tools to capture, manage and analyze [3]. Processing of big data required specific tool and method to handle large amount of data efficiently. Big data frameworks like MapReduce [3] and Spark [4] help in processing and handling of big data efficiently and provide scalable implementation.

Clustering is the process of partitioning unlabeled data set into groups of similar data instances, called clusters such that data instances in each group are similar to each other and very different from data instances of other group. The goal of clustering is thus to minimize intra-cluster distance and maximize inter-cluster distance. Clustering needs similarity functions, in order to determine the difference of two data instances. Over the years, many clustering algorithms have been developed, each utilizing different distance/similarity measures and/or objective functions. Applying different methods, or the same method with different parameter choices to the same data, the algorithm can obtain varying clustering results [2].

A number of evolutionary algorithm has recently emerged to solve the optimization problem. In order to make clustering as an appropriate application of evolutionary algorithms (EAs), it can be regarded as a global optimization problem where the objective is to find the cluster solution according to some cluster validity measure. Differential evolution, a population-based search strategy similar to evolutionary algorithm, can effectively solve real point optimization problems. So, it is an effective way to solve clustering problem for small to moderate size of data sets. [1]. Although the DE approach is popular due to its good exploration and exploitation abilities of search space, a lot of time is required to process DE. Thus, the standard DE is not efficient when addressing large scale data sets.

The paralleling nature of DE is well suited for big data clustering where multiple cluster solutions are encoded in chromosome to search for appropriate clusters. Each chromosome search for optimal clustering solution in parallel. This paper proposes a parallel differential evolution algorithm for big data clustering on the top of Spark framework.

2. Related Work

There have been many reports on literature of clustering based on evolution and optimization algorithm.

Sandra Paterlini et al. [5] used Differential Evolution for partitional clustering problem. Medoid representation is used for the chromosome encoding where medoid represents the cluster medoid. For the fitness measure, TRW (Trace within Measure) and MC (Marriott's criterion) is used. Authors compared clustering with Genetic Algorithm, PSO (Particle Swarm Optimization), K-mean, and Differential Evolution algorithm. Experimental results stated that DE should be used for clustering rather than GA.

Swagatam Das and Ajith Abraham used DE for clustering large unlabeled data set without requiring the number of cluster to be specified [6]. The proposed algorithm selects the optimal number of clusters automatically on the run. Centroid based representation with activation thresholds was used as chromosome encoding scheme and Davies-Bouldin index (DB index) was used as the fitness function. The proposed algorithm was able to outperform two other state-of-the-art clustering algorithms, GA (Genetic Algorithm) and PSO (Particle Swarm

Optimization). However, computational burden is still remaining.

D.Zhaire *et al.* [7] used DE for document clustering problem and can automatically discover number of clusters. To reduce the computational time, they proposed a hybrid k-means with DE method and then, to evaluate fitness for high dimensional document clustering problems, a new validity index has also been proposed by modifying the CS measure. Bag-of-words representation is used for representing the document in a vector form. Nevertheless, computational time is a fact to consider when dealing with real world applications.

Kiyoharu Tagawa and Takashi Ishimizu [8] presented the concurrent differential evolution based on MapReduce framework. Concurrent Differential Evolution (CDE) divides the population into multiple chunks, and then CDE assigns the task for updating the individuals included in each chunk to a thread statically. The multi-core processor executes multiple threads in parallel by using multiple CPU. The authors concluded that computational time can be reduced by using the proposed CDE on the multi-core processor.

In [9] Diego Teijeiro et. al proposed and evaluated parallel differential evolution based on Spark framework. The authors experiment two schemes of DE namely master slave and island model. The experimental results showed that the proposed approach achieves more competitive speedup than the serial implementation. Moreover, they stated that their approach provides a good scalability when the number of node grows.

In our proposed approach, clustering is viewed as optimization problem and solved by Differential Evolution in order to search global optimal solution. Centroid based representation is used in which each individual chromosome encodes the cluster solution with centroids. Quantization error is used as the fitness measure for the DE algorithm. In this paper, master-slave parallel version of DE is proposed and the Spark framework is used for parallel DE implementation.

3. Background Theory

Clustering is one of the most hard and challenging tasks in machine learning. From an optimization point of view, clustering is regarded as a kind of NP-hard grouping problem and a great number of evolutionary algorithm have been proposed to solve clustering problem. In big data era, specific tools and methods have been proposed to handle massive data. This section describes clustering methods, and big data frameworks.

3.1. Clustering

Cluster analysis is an unsupervised learning method used for the exploration of inter-relationships among a collection of patterns, by organizing them into homogeneous clusters. Clustering is called unsupervised learning because unlike classification (known as supervised learning), no class label is required. Clustering work by grouping item of similar features. Each item in the group share the similar features and it is known as intra-connectivity that measures the density of connections between the instances of a single cluster. High similarity or intra connectivity among items in the same cluster represents the good cluster solution. A low degree of interconnectivity that measures connectivity between distinct clusters is desirable because it indicates that individual clusters are largely independent of each other. Clustering algorithms can be mostly categorized into partitional methods, and hierarchical methods.

Clustering can also be performed in two different modes:

Crip clustering: In crisp clustering, the clusters are disjoint and nonoverlapping in nature. Any pattern may belong to one and only one class.

Fuzzy clustering: In fuzzy clustering, a pattern may belong to all the classes with a certain fuzzy membership grade [10].

The proposed work in this paper considers crisp clustering algorithm only.

3.1.1. Partitional Clustering

Partitional clustering method partitions the data iteratively using some notion of similarity measure. One of the advantages of partitional clustering is handling of large data set unlike hierarchical clustering where construction of dendrogram is computationally expensive. There are two major approaches in partitional clustering, the centroid and the medoids algorithms. The centroid algorithms represent each cluster by using the gravity centre of the instances whereas medoid algorithms represent each cluster by means of the instances closest to the gravity centre. The most well-known centroid algorithm is the k-means [2]. The k-means method partitions the data set into k subsets such that all points in a given subset are closest to the same center. k-means algorithm sensitive to the selection of the initial cluster centroids and may converge to the local optima. In the DE based clustering approach multiple individual solution were used to initialized the algorithm and so DE is not sensitive to the selection of initial cluster centroid. DE is a global search algorithm unlike k-mean, so local optima solution can be avoided.

3.1.2. Hierarchical Clustering

The hierarchical methods group data instances into a tree of clusters or dendrograms. There are two hierarchical clustering approaches, agglomerative method, which constructs the clusters in a bottom-up

fashion until all data instances belong to the same cluster, and another approach is the divisive method, that splits up the data set into smaller clusters in a top-down fashion until each cluster contains only one instance [2]. Both agglomerative and divisive methods are known for their quick termination, but they suffer from their inability to perform adjustments once the splitting or merging decision is made. Other advantages are: 1) they can automatically discover the number of cluster, 2) computes a complete hierarchy of clusters, 3) visualization method can be applied, 4) a "flat" partition can be derived afterwards (e.g. via a cut through the dendrogram). Major drawback of the hierarchical clustering algorithm is the expensive computation needed for large dataset in forming dendrograms.

3.2. Differential Evolution

Algorithm 1: Differential evolution algorithm
Set the generation counter, *t=0*;
Initialize the control parameters, β and p_r;
Create and initialize the population, *C*(0), of *n* individuals;
while *stopping condition(s) not true* **do**
  for *each individual,* $x_i(t) \in C(t)$ **do**
  Evaluate the fitness, *f(xi(t))*;
    Create the trial vector, $u_i(t)$ by applying the mutation operator;
    Create an offspring, $x'_i(t)$, by applying the crossover operator;
    if $f(x'_i(t))$ *is better than* $f(x_i(t))$ **then**
      Add $x'_i(t)$ to $C(t+1)$;
    end
    else
      Add $x_i(t)$ to $C(t+1)$;
    end
  end
end
Return the individual with the best fitness as the solution;

Figure 1. Differential evolution algorithm

The DE algorithm, as proposed by Storn and Price [2], is a new floating-point encoded evolutionary algorithm for global optimization. The DE algorithm has demonstrated good convergence properties, is fundamentally easy to understand, and has been used in many different fields. The aim of DE is to find an individual which minimizes the objective function by mutation, crossover and selection operators, as shown in figure 1.

3.2.1. Mutation

The DE mutation operator produces a trial vector for each individual of the current population by mutating a target vector with a weighted differential. This trial vector will then be used by the crossover operator to produce offspring. For each parent, $x_i(t)$, generates the trial vector, $u_i(t)$. Select a target vector, x_{i1} (t), from the population, such that $i \neq i_1$. Then, randomly select two individuals, x_{i2} and x_{i3}, from the population such that $i \neq i_1 \neq i_2 \neq i_3$.vUsing these individuals, the trial vector is calculated by perturbing the target vector as follows:

$$u_i(t) = x_{i1}(t) + \beta(x_{i2}(t) - x_{i3}(t)) \quad (1)$$

where $\beta \in (0, \infty)$ is the scale factor, controlling the amplication of the differential variation.

3.2.2. Crossover

The DE crossover operator implements a discrete recombination of the trial vector, $u_i(t)$, and the parent vector, xi(t), to produce offspring, $x'_i(t)$. Crossover is implemented as follows:

$$x'_i(t) = \begin{cases} u_{ij}(t) & if\ j \in J \\ x_{ij}(t) & otherwise \end{cases} \quad (2)$$

where $x_{ij}(t)$ refers to the j-th element of the vector $x_i(t)$, and J is the set of element indices that will undergo perturbation (or in other words, the set of crossover points).

3.2.3. Selection

Selection is applied to determine which individuals will take part in the mutation operation to produce a trial vector, and to determine which of the parent or the offspring will survive to the next generation. To construct the population for the next generation, deterministic selection is used: The offspring replaces the parent if the fitness of the offspring is better than its parent; otherwise the parent survives to the next generation.

3.3. Big data framework

The MapReduce [3] programming model has been one of the most prevalent approaches used to manage data intensive computational pipelines that need to be processed on multiple compute nodes in a scalable cluster. This model divides the computation into two phases: map and reduce. During the map phase, input data is first formatted as key-value <K, V> pairs followed by performing a specific mapping function on all of these pairs, resulting in a mapping of the input to an output set of <K, V > pairs. The mapping function is executed in a distributed fashion using various mapping tasks that run locally on the data present in the nodes of the cluster. The output is then taken by the reduce tasks which first shuffle the data by grouping all the values that belong to the same key together. Afterwards, the reduce tasks compute a single output from the grouped <K, V> pairs. Apache Hadoop is an open source implementation of the MapReduce programming

model, which consists of three components: 1) the Hadoop Distributed File System (HDFS) that allows storing large data les on multiple nodes, 2) a resource manager called YARN that distributes the tasks to be executed to the various nodes in the cluster, and 3) the Hadoop MapReduce framework itself.

One disadvantage of the MapReduce framework is that it stores all data generated between the two phases (map and reduce) on disk, and if multiple map/reduce stages are chained together, it stores the output of each stage on the HDFS. This implies significant overhead due to the intensive disk access. Another disadvantage is that the only transformations allowed are map and reduce. If a different transformation has to be applied, it has to be done by modifying the map and reduce functions, thus making development with this framework very cumbersome.

Apache Spark [4] is a more recent big data framework that addresses the disadvantages of MapReduce listed above. First, it allows more transformations than mere map and reduce. It provides transformations such as join, cogroup, intersection, distinct and many others as predefined operations. This makes development of programs much easier as compared to the MapReduce framework. Moreover, the output of each transformation is saved in memory, but still allowing disk to be used to save data that does not fit within the available memory size. In this way, Spark avoids the overhead of disk access that is so prevalent in the MapReduce framework. In addition, Spark provides a programming interface to implement algorithms in various programming languages such as Scala, Python and Java, which allows writing programs in the language that is best suitable for the problem.

3.4. Clustering with Differential Evolution Algorithm

In order to apply the DE as clustering algorithm, clustering must be viewed as optimization problem to find optimal cluster solution. Two important design factors are need to solve optimization problem using DE; chromosomes representation and fitness calculation. In the context of data clustering, each chromosome represents the K cluster centroids. A population represents a number of candidate clustering solutions. That is, each individual cluster solution is encoded as chromosomes in the DE algorithm and then, multiple cluster solutions are find simultaneously. Several encoding schemes for EAs have been proposed in [10]. The proposed approach uses centroid based representation that is associated with real encoding because DE was developed for continuous-valued landscape. Fitness function describes how suitable the candidate solutions in the population are to the given problem. When dealing with clustering problems, cluster validity measures are used as fitness function. As the fitness measure, quantization error is used, which simply calculates the sum of distance of data points to their corresponding cluster centroid.

$$J_e = \frac{\sum_{j=1}^{Ni}\left[\sum_{\forall \boldsymbol{z_p} \in \boldsymbol{C_{ij}}} d(z_p, \boldsymbol{m_j})/|C_{ij}|\right]}{N_c} \qquad (4)$$

Where N_c is the number of cluster, $d(z_p,m_j)$ is the Euclidean distance between data point mj and center z_p. $|C_{ij}|$ is the number of member in cluster j.

4. Parallel Differential Evolution Algorithm on Spark

In order to parallelize DE, Spark, an open-source clustering-computing framework is adopted. This paper is based on the master slave model of computation of Spark. There is a driver that talks to a single coordinator called master that manages workers in which executors run. The driver and the executors run in their own Java processes. The proposed parallel differential evolution algorithm for Clustering is shown in figure 2.

```
Algorithm 2: Parallel Differential Evolution
             Algorithm
Initialize populations populationRDD
for i=1 to maxIteration do
    for each p in populationRDD Do in parallel
        fitnessRDD ← evaluateFintess(p)
    End for
    fitnessResultRDD ← fitnessRDD.collect()
    for each p in fitnessResultRDD Do in
    parallel
        donarVector← mutation(p)
        trialVector← corssover(p)
        selectionRDD ← selection(p)
    End for
    populationRDD ← selectionRDD.collect()
End for
```

Figure 2. Parallel differential evolution algorithm

In the first step of the algorithm Spark driver initialize the population with corresponding RDD. There are two parallel stages in the algorithm, the first parallel stage evaluates the fitness of each individual chromosome in the population of DE algorithm by workers. DE mutation, crossover require other chromosomes' fitness, so this information is send back to spark driver by workers. Another parallel stage performs mutation, crossover and selection by workers. After all workers completed the process, best individuals are collected by the driver. The algorithm will run until a pre-defined number of maximum iterations is reached. The flowchart of the parallel differential evolution algorithm is shown in figure 3.

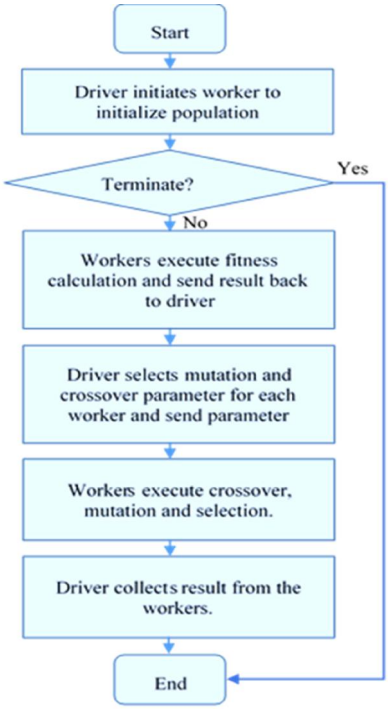

Figure 3. Flowchart of the parallel differential evolution algorithm

5. Conclusion

In big data era, the abundance of data can bring benefits for various organizations by uncovering the secreted relationships amongst these data. However, most of the traditional algorithms are not suitable to handle huge data. The challenging issue is becoming to design new methods and algorithm that are mainly based on parallel and distributed computing models and make use of dedicated platforms. This paper proposes a parallel differential evolution algorithm for clustering enormous data based on Spark framework. Especially, using the proposed algorithm will reduce the computation time. As a future work, many experimental evaluations have to be conducted in order to get the efficiency of the proposed algorithm. Moreover, many experimental evaluations have to be made in order to get better cluster quality. As well as, the speed of convergence and the robustness will be considered for the performance of the proposed approach.

References

[1] A. P. Engelbrecht, *Computational Intelligence_An Introduction,* Second Edition, John Wiley & Sons Ltd, England, 2007.

[2] J. Han, M. Kamber, *Data Mining: Concepts and Techniques*, Second Edition, Morgan Kaufmann Publishing, 2006.

[3] J. Dean, S. Ghemawat, "MapReduce: simplified data processing on large clusters". *in Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation*, 2004.

[4] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M.J. Franklin, S. Shenker, I. Stoica, "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing", *in Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation*, 2012.

[5] S. Paterlini, T. Krink, "High Performance Clustering with Differential Evolution", *Proceedings of IEEE Congress on Evolutionary Computation* (CEC), vol. 2, 2004.

[6] S. Das, A. Abraham, "Automatic Clustering Usingan Improved Differential Evolution Algorithm, *IEEE Transcations On Systems, Man, And Cybernetics*, vol.38, no.1, January 2008.

[7] D.Zaharie, F.Zamfirache, V.Negru, D.Pop ,and H.Popa, "A Comparison Of Quality Criteria For Unsupervised Clustering of Document Based On Differential Evolution", *Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques* (KEPT), Romania, June6-8, 2007, pp.25-32.

[8] K. Tagawa, T. Ishimizu, "Concurrent Differential Evolution Based on MapReduce", *International Journal of Computers*, vol 4, Issue 4, 2010.

[9] D. Teijeiro, X. C. Pardo, P. González, J. R. Banga, and R. Doallo, "Implementing *Parallel Differential Evolution on Spark", Applications of Evolutionary Computation* (EvoApplications), Lecture Notes in Computer Science, vol 9598, Springer, 2016.

[10] E. R. Hruschka, R. J. G. B. Campello, A. A. Freitas, and A. C. P. L. F. de Carvalho, "A Survey of Evolutionary Algorithms for Clustering", *IEEE Transactions on Systems, Man, and Cybernetics, vol. 39, no. 2, March* 2009.