International Journal of Database Theory and Application
Vol.9, No.10 (2016), pp.241-250
http://dx.doi.org/10.14257/ijdta.2016.9.10.20

Clustering Large Scale Data Set Based on Distributed Local Affinity Propagation on Spark

Wei Lu and Peng Cao

School of Software Engineering, Beijing Jiaotong University, Beijing 100044, China
luwei@bjtu.edu.cn; 604cp@163.com

Abstract

Affinity Propagation (AP) is a new clustering method to cluster data set efficiently. In this paper, a Distributed method of Local Affinity Propagation (DLAP) is proposed to solve hardware bottleneck and time-consuming problem. DLAP refines AP by reducing the calculating data scale in each iteration and keeps a high quality clustering result. The method is implemented on Apache Spark distributed computation framework. Depending on high iteration efficiency on Spark, the method has an impressive result in time complexity. Experiments are conducted on two-dimensional data to show that the time cost of LAP on single machine is better than the two methods, FSAP and FAP, meanwhile the result of DLAP on Spark is better than that on Hadoop.

Keywords*: BigData, Distributed Computing, Apache Spark, Affinity Propagation*

1. Introduction

Recently with the rapid increment of information in our network, how to deal with large scale data set efficiently is becoming a significant issue. Clustering is one of the most popular method to combine similar data together for data analysis, patterns detecting, graph partition and text summarizing. Usually we execute clustering for many iterations to maximize the sum of similarities between each point and its center, then globally optimal clustering result is established. Nowadays with the development of BigData, using clustering to analyze large scale data set in parallel is a very important data mining approach. As a result how to deal with these data to facilitate our analysis is becoming a big issue.

There are several classical algorithm for clustering such as K-means [1], Spectral clustering [2], Hierarchical clustering [3] and Affinity Propagation [4], but none of them is adapted to magnificent data. As a result, series of optimized algorithms are proposed to analyze large scale data in parallel. Parallel K-means [5] and Parallel Spectral clustering [6]. occurrence indicates that putting algorithm in parallel instead of a single machine effectively resolves large scale data issue. However, in K-means initially selected exemplars are important to final cluster result, and locally optimal solution is come out when initial exemplars choose a wrong set. Spectral clustering meets the same problem because it needs K-means to calculate final result.

Affinity Propagation is a novel clustering method which takes all the data as possible cluster centers and doesn't need to confirm cluster number. The standard AP algorithm is quite time-consuming, so in recent years plenty of methods are presented to reduce the time complexity such as parallel realization or dataset reduction. These methods improve the time efficiency, but they are still complicated and not efficient enough. In this paper, a Distributed method of Local Affinity Propagation (DLAP) is proposed to improve time efficiency. DLAP improves the standard AP algorithm by decreasing the data scale and using sparse graph structure in Apache Spark to facilitate computation without losing accuracy. The key contributions of our work are summarized as follows:

1. The new method is proposed to reduce the scale of data to improve the time efficiency of AP algorithm.
2. The Distributed method of Local Affinity Propagation is realized on Apache Spark.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 introduces the proposed Distributed Local Affinity Propagation method (DLAP). In Section 4, two experiments are conducted respectively on single machine and distributed system. Section 5 concludes the whole paper.

2. Related Work

2.1. Affinity Propagation

Affinity Propagation is a clustering method to find the globally optimal solution efficiently by means of trans information between data points. After series of iterations, the centers are generated for all points without setting initial number of clustering. AP has higher accuracy than other algorithms to deal with large scale data in images, n-dimensional data, gene expressions and semantic analysis.

In clustering algorithm, similarity indicates two points' close degree. There are a lot of ways to express similarity, and usually it is set to a negative squared Euclidean distance: For point x_i and x_j semantic analysis.

$$s(x_i, x_j) = -\|x_i - x_j\|^2 \tag{1}$$

The AP defines two parameters which are called *responsibility* and *availability*. *Responsibility* $r(x_i, x_k)$ is sent from point x_i to point x_k, reflecting how well x_k serves as the exemplar of x_i compared with other points $x_{k'}$. *Availability* $a(x_i, x_k)$ is sent from point x_k to point x_i, reflecting how appropriate x_i chooses x_k as its exemplar compared with other points $x_{i'}$. The two parameters are updated iteratively by:

$$r(x_i, x_k) = s(x_i, x_k) - \max_{x_{k'} \neq x_k} \{a(x_i, x_{k'}) + s(x_i, x_{k'})\} \tag{2}$$

$$a(x_i, x_k) = \min\{0, r(x_k, x_k) + \sum_{x_{i'} \notin \{x_i, x_k\}} \max\{0, r(x_{i'}, x_k)\}\} \quad (x_i \neq x_k) \tag{3}$$

$$a(x_k, x_k) = \sum_{x_{i'} \neq x_k} \max\{0, r(x_{i'}, x_k)\} \tag{4}$$

These parameters are calculated until they are convergent. The constraint that one point x_i chooses another point x_k as exemplar is:

$$k = \arg\max\{a(x_i, x_k) + r(x_i, x_k) | a(x_i, x_k) + r(x_i, x_k) > 0\} \tag{5}$$

All responsibilities and availabilities are set to 0 initially. For iterative algorithm, it is important to avoid numerical oscillations. We involve damp factor λ to ensure convergence. The values are updated iteratively as follows to compute convergence values:

$$r_L(i, j) = (1 - \lambda) r_L(i, j) + \lambda r_{L-1}(i, j)$$
$$a_L(i, j) = (1 - \lambda) a_L(i, j) + \lambda a_{L-1}(i, j) \tag{6}$$

The similarity of a point to itself $(s(x_i, x_i))$ is called *preference* P, which is very important to affect the number of clusters.

Due to iteration of computing *responsibility* and *availability*, the time complexity is $\mathrm{O}(N^2\mathrm{L})$, in which N means the number of data points and L means iteration time. So time consumption is out of expectation when clustering large scale data set. There are two ways to handle this issue: dataset scale reduction and parallel realization.

2.2. Dataset Scale Reduction

The method of Dataset Scale Reduction aims at decreasing the computing complexity, thus to ensure the efficiency of algorithm without losing accuracy.

A Fast Sparse Affinity Propagation (FSAP) is put forward to reduce the scale of data [7]. The method makes each point take K-nearest neighbor points into consideration as similarity and puts them into computation regardless of other similarity data. However, when a large group of points are similar, they should be put into one cluster instead of being divided by K-nearest neighbors. So when data are unbalanced, FSAP often does not have a good result.

Recently Fujiwara proposed An method called Fast Affinity Propagation (FAP) [8], which set a upper and lower bound to prune unnecessary message exchanges in the iterations. This method achieves a little better time efficiency compared with FSAP and has the same result as AP. However, some calculations in the methods are still complicated, which affect the time efficiency.

2.3. Parallel Realization

With the development of distributed computing, a lot of methods are realized on distributed platform in order to deal with Big Data issues [9].

MapReduce [10] is a popular programming model to decompose huge task into small pieces for distributed computing. The model composed of two steps: Map step receives text data and parallelly transforms them into key/value pairs, then sends to Reduce step. Reduce step groups pairs by key and executes parallelly in the same group, finally the result is written on disk. Most of programming method can adapt to uniform model. Among all the MapReduce models, Hadoop [11] is a classical framework and is widely used. A method of Affinity Propagation on Hadoop [12] is proposed to achieve good performance. However, Hadoop uses text files on disk to store data, which totally increases time consumption because of I/O delay. Besides, there are no data structure specialized for graph computation in Hadoop.

Apache Spark [13] is an emerging MapReduce framework aimed on memory computing. Compared with hadoop, Spark use memory to conduct computing instead of frequently reading or writing text file via disk. Spark supports iterative calculation as well as scalability and fault tolerance. To achieve these goals, Spark introduces an abstraction called resilient distributed datasets (RDD) [14], which is a highly efficient structure for data encapsulation and DAG computation. In every RDD computation Spark just records the transformations from old RDD to new RDD until the program need to output results (called linage). So optimizing can be conducted in linage to increase efficiency. Lineage can help program recovery when data is lost when one node of system is broken down. Spark provides a lot of Machine Learning lib and graph computing framework GraphX [15]. This algorithm takes advantage of GraphX to execute high performance.

3. Distributed Local Affinity Propagation Method

3.1. Local Dataset Reduction

The process of AP can be regarded as finding a set of exemplars to maximize the sum of similarities between each point and its exemplar, and a lot of similarity data are useless when two points have a very small value of similarity. So at the beginning these similarities can be deleted. According to the FASP, the K-nearest neighbors are selected to be the initial input. When data are unbalanced, there may be situation that some dense group of data lose a great deal of neighbors, but some isolated points keep lots of neighbors which are actually dissimilar.

So a method to reduce data scale efficiently is to set a threshold to limit the similarity which we put into consideration. When a similarity of x_i and x_j is under this threshold we delete it. Therefore we can ensure to maximize the sum of similarities.

In this situation, it is not sure how many neighbors a point has because of the unbalance of the data, but the data scale is easy to calculate. Example is presented by two-dimensional data set (Figure1):

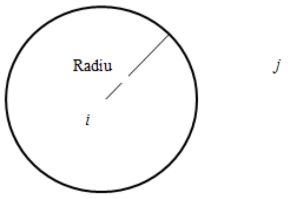

Figure 1. Relationship Between the Points within Neighborhood of Point *I*

In a two-dimensional space that has N points, the range of X coordinate and Y coordinate is R. The density of point in this space is N/R^2. For point x_i, in formula (1) we know the similarity between x_i and point x_j is $-\|x_i - x_j\|^2$ which means the negative square of two points' distance. Let T represent the square of two points' distance, and threshold represents a negative value of T. Then the neighbors of the point x_i with the similarity above the threshold are all in a circle, in which x_i is the center and $\sqrt{T}$ is the radius. In Figure1 the point x_k is in the circle, but the point x_j is not. So x_j will be deleted. The average number of these points x_k is as follows:

$$\frac{\pi T N}{R^2} = K \tag{7}$$

So threshold should be:

$$Threshold = -T = -\frac{KR^2}{\pi N} \tag{8}$$

When K is small enough compared with N, the data scale will decrease greatly, the remain data could be formed to a spare graph. The time complexity is O(*NL*) and all the large value of similarities are kept in the graph. After reduction, the data of AP can easily be calculated parallelly. In next section we detail a new realization of AP algorithm based on Apache Spark.

3.2. Parallel Realization on Spark

AP algorithm can be realized parallelly on MapReduce framework such as Hadoop. While compared with hadoop, Spark abstracts data into RAM for iterative computation, and the efficiency is higher than Hadoop. And Spark GraphX is a graph computing framework, which facilitates graph computation. So we use Spark as platform.

To optimize calculating on Spark, we use the following RDD data structure named TripleEdge:

$$(Row : i, Column : j, EdgeMessage : (Similarity, Availablity, Responsibility))$$

In TripleEdge the data can easily be separated according to row or column. In formula (2-4) we know that each update for a point only needs a row or a column to compute, and in our sparse graph there are only K points on average for one point to count. So it is necessary to put the graph data in parallel.

In Spark graph model facilitates the process of execution. In formula (2) the data are grouped by row number and computed in parallel, which means a vertex only needs all the edges from itself in graph structure (Figure2):

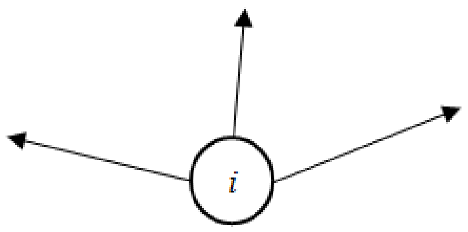

Figure 2. Edges Point from *I* which Stands for Row Vector

In formula (3-4) the data are grouped by column number and computed in parallel, which means a vertex only needs all the edges to itself in graph structure (Figure3):

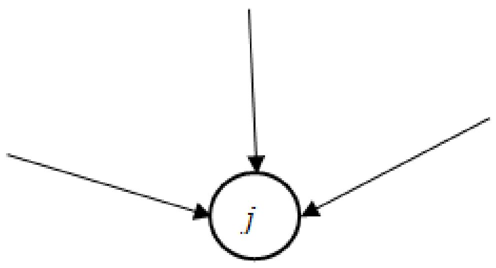

Figure 3. Edges Point to *J* which Stands for Column Vector

When responsibilities are counted, the data can be grouped in row. When availabilities are counted, the data can be grouped in column. As a result, all the calculation can be in parallel. The method step is as follow:

Step 1 Determine the threshold to reduce initial similarity data.

Step 2 Put similarity data into Spark as sparse graph structure.

Step 3 Divide graph into parts according to source vertex, then update responsibility in parallel based on formula (2).

Step 4 Divide graph into parts according to destination vertex, then update availability in parallel based on formula (3-4).

Step 5 Calculate clusters and decide whether the result is convergent. If the result is not convergent, go to Step 3.

In each calculation, the clusters should be checked to make sure that the algorithm can converge. The program should stop to avoid time waste when the result does not change during several times.

Here we show the algorithm pseudo-code:

Algorithm 1 DLAP algorithm

```
Require: $IterNum$,$Convergence$,$Graph(V,E)$
Ensure: $APmodel$
1: function DLAP($Preference$,$IterNum$)
2:     Set the $threshold$ according to the cluster number
3:     Eliminate the similarity data below the $threshold$, then put the remain data into Graph(V,E)
4:     for $n = 1 \rightarrow IterNum$ do
5:         $Exemplar \leftarrow \{\}$
6:         for $x_i = 1 \rightarrow |V|$ do
7:             Aggregate $edge$ from vertex $x_i \rightarrow edgeList(x_i)$
8:             Based on $formula(2)$, calculate $r(x_i, x_j)$ for every adjacent vertex j, and write it back to Edge($x_i, x_j$)
9:         end for
10:        for $x_i = 1 \rightarrow |V|$ do
11:            Aggregate $edge$ to vertex $x_i \rightarrow edgeList(x_i)$
12:            Based on $formula$(3-4), calculate $r(j, i)$ for every adjacent vertex j, and write it back to Edge($x_j, x_i$)
13:            Based on $formula(5)$, calculate exemplar for every vertex $x_j$, if exemplar does exist, put it into $Exemplar$
14:        end for
15:        Put $(e, cluster[e]) \rightarrow APmodel$
16:    end for
17:    Return $APmodel$
18: end function
```

In algorithm, if the *Exemplar* don't change for *Convergence* times then the iteration is stopped. In Graph(V,E) *V* stand for all the Vertex, *E* stand for all the Edges. The final result is stored in *APmodel* .

From the pseudo-code, it is clear that algorithms can be easily paralleled. If the program is based on M worker nodes, every cluster has K points, the time complexity in algorithm is O (*NKL/M*).

4. Experiments

The experiments are conducted in two groups: 1) Compare the LAP with FSAP and FAP on single machine. 2) Compare the DLAP efficiency on Spark platform with Hadoop. The experiments are conducted on a small group of clusters which are composed of three computers. One is used for NameNode and others are used for DataNodes. All of the machines have the same configure: CPU Intel Xeon X5550 4 cores, Memory 8G, Hard Disk 300G with CentOs 64 bits system, Jdk Version is 1.6.0.47.

The performances of those clustering method are based on the clustering accuracy and time efficiency. For accuracy, the sum-similarities of all the points with their exemplars are considered. For efficiency, running time and time per iteration are considered. The first experiment is conducted on single machine. We use two-dimensional data with numbers ranging from 1000 to 10000, λ is set to 0.5. The value ranges from 0 to$\sqrt{pointnum}$, and Threshold is set to$\sqrt{pointnum}/\pi$. We manually set preference to ensure fair condition in both methods. Then we set preference to -10, *MaxIter* to 300 and ConvergeIter to 10. The result is as follows:

Table 1. Comparison among FSAP, FAP and LAP on Single Machine with Preference -10

Num	$Cluster$	$Iter$	$Time$	$\frac{Time}{Iter}$	Sim	$\frac{Sim}{Num}$
FSAP-1000	95	72	6m	5s	-2210	-2.21
FAP-1000	82	65	4.5m	4.15s	-2035	-2.035
LAP-1000	82	70	4m	3.42s	-2035	-2.035
FSAP-2000	167	84	21m	15s	-4690	-2.345
FAP-2000	157	69	17m	14.78s	-4090	-2.045
LAP-2000	157	76	15m	11.84s	-4090	-2.045
FSAP-5000	415	91	1.2h	47.4s	-13075	-2.615
FAP-5000	388	80	1h	45s	-10234	-2.047
LAP-5000	388	87	1h	41.37s	-10234	-2.047
FSAP-10000	900	227	6.5h	103.08	-28010	-2.801
FAP-10000	764	200	5.5h	99s	-20427	-2.042
LAP-10000	764	206	5h	87.37s	-20427	-2.042

In Table 1, the result is confirmed that FAP and LAP have the same clusters and total similarity, which means LAP doesn't affect the accuracy of the result. But LAP is more efficient than FAP according to the time per iteration. FSAP is slower and can't keep the same result as standard AP. Figure4 shows that LAP is faster than FSAP and FAP.

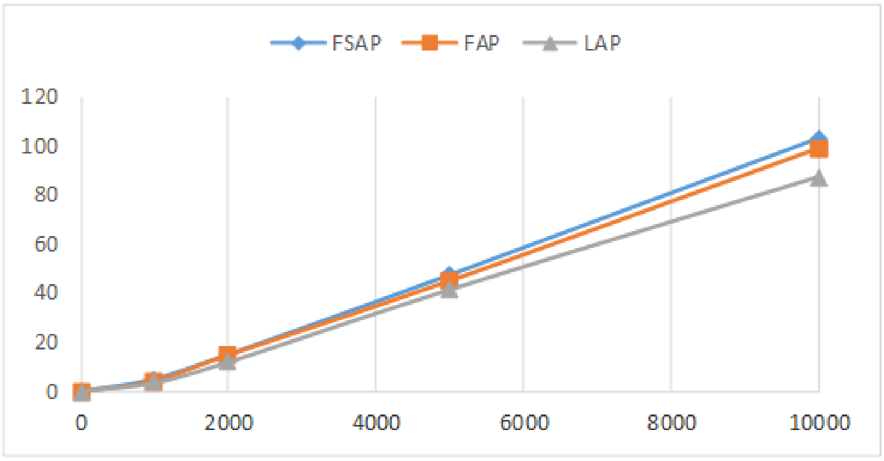

Figure 4. Comparison among FSAP, FAP and LAP on Single Machine with Preference -10

In the second group, the experiments are executed on Hadoop and Spark to compare the performance of DLAP on two platforms. The data ranges from 1000 to 50000, which can obviously show the differences for large scale data. Here is the experiment result:

Table 2. Comparison between LAP on Hadoop and Spark with Preference -10

Num	$Cluster$	$Iter$	$Time$	$\frac{Time}{Iter}$	Sim	$\frac{Sim}{Num}$
H-5000	388	80	42m	32s	-10234	-2.047
S-5000	388	87	5.5m	3.79s	-10234	-2.047
H-10000	764	195	4h	1.26m	-20427	-2.042
S-10000	764	206	22.5m	6.6s	-20427	-2.042
H-20000	1552	200	2.5h	2.88m	-40816	-2.041
S-20000	1552	220	9.6h	27s	-40816	-2.041
H-50000	3876	240	45h	11.7m	-102190	-2.0438
S-50000	3876	235	12.6h	3.22m	-102190	-2.0438

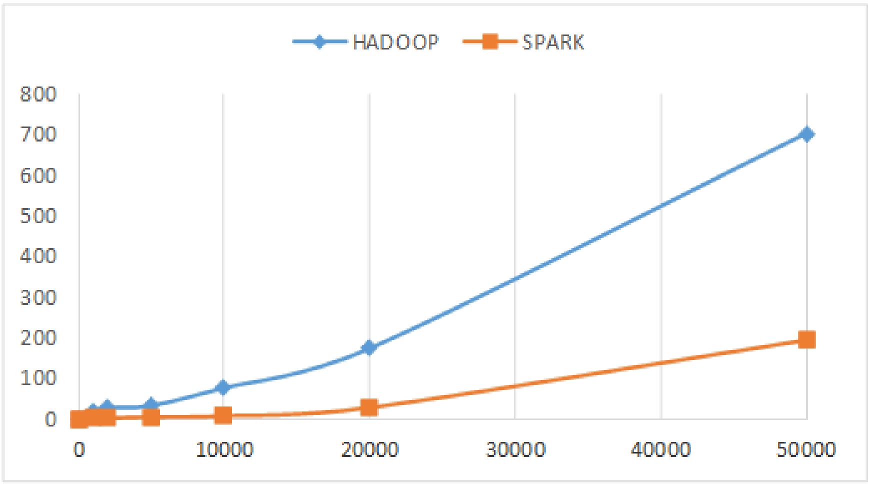

Figure 5. Comparison Between LAP on Hadoop and Spark with Preference -10

Table 2 shows that the result on Spark is superior to that on Hadoop in time efficiency. In Figure5 LAP on Hadoop does not execute an ideal result at the beginning, because the data scale is small, while HDFS data block is bigger than the data file which wastes more time.

5. Conclusions

In this paper, Distributed Local Affinity Propagation method is proposed to effectively clustering data on higher performance and less time complexity. The method refines AP on data reduction using threshold and optimization on new platform Apache Spark. The experiments show that when cluster number is not assigned, DLAP processes well both on accuracy and efficiency. The method has improved time efficiency compared with FSAP, FAP and DLAP on Hadoop.

Further research will be conducted on several issues. First, adaptive method of Affinity Propagation is necessary to control number of clusters by automatically setting *preference* instead of testing step by step. Second, Apache Spark is a newborn platform that is still need update, so there are a lot of issues on Spark that can cause huge disk overflow, such as long lineage in iterative computation and output file of shuffle step. The last one is how to set parameter threshold, which can get the perfect clustering result.

Acknowledgements

I would like to express my gratitude to all those who have helped me during the writing of this thesis. I gratefully acknowledge the help of my supervisor Professor Lu Wei. I do appreciate his patience, encouragement, and professional instructions during my thesis writing. Also, I would like to thank Miss Xing WeiWei, who kindly gave me a hand when I was making the questionnaire among the college English learners.

Last but not the least, my gratitude also extends to my family who have been assisting, supporting and caring for me all of my life.

References

[1] A. K Jain, M. N. Murty and P. J. Flynn, “Data clustering: a review”, ACM computing surveys (CSUR), vol. 31, no. 3, **(1999)**, pp. 264-323.
[2] U. V. Luxburg, “A tutorial on spectral clustering. Statistics and computing”, vol. 17, no. 4, **(2007)**, pp. 395-416.
[3] J. Ran, H. Lvshi, W. Cui, W. Lifei and S. Wangchun, “A Hierarchical clustering community algorithm which missed the signal in the process of transmission”, Review Of Computer Engineering Studies, vol. 2, no. 3, **(2015)**, pp. 27-34.
[4] B. J. Frey and D. Dueck, “Clustering by passing messages between data points”, science, vol. 315, no. 5814, **(2007)**, pp. 972-976.
[5] W. Zhao, H. Ma, and Q. He, “Parallel k-means clustering based on MapReduce”, In Cloud Computing, **(2009)**, pp. 674-679.
[6] W. Y. Chen, Y. Song, H. Bai, C. J. Lin and E. Y. Chang, “Parallel spectral clustering in distributed systems”, Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 33, no. 3, **(2011)**, pp. 568-586.
[7] Y. Jia, J. Wang, C. Zhang and X. S. Hua, “Finding image exemplars using fast sparse affinity propagation”, In Proceedings of the 16th ACM international conference on Multimedia, ACM, **(2008)**, pp. 639-642.
[8] F. Yasuhiro, I. Go and K. Tomoe, “Fast algorithm for affinity propagation”, In Proceedings of International Joint Conference on Artificial Intelligence, **(2011)**.
[9] W. Rengan, J. Zhen and L. Bao, “Study on Mining Big Users Data in the Development of HuBei Auto-Parts Enterprise”, In Mathematical Modelling of Engineering Problems, vol. 2, no. 4, **(2015)**, pp. 1-6.
[10] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters”, Communications of the ACM, vol. 51, no. 1, **(2008)**, pp. 107-113.
[11] D. Borthakur, “The hadoop distributed file system: Architecture and design”, Hadoop Project Website, vol. 11, **(2007)**.
[12] W. C. Hung, C. Y. Chu, Y. L. Wu and C. Y. Tang, “Map/reduce affinity propagation clustering algorithm”, In International Conference on Control, Robotics and Cybernetics, **(2015)**.
[13] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica, “Spark: cluster computing with working sets”, In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, vol. 10, **(2010)**, pp. 10.
[14] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker and I. Stoica, “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing”, In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, USENIX Association **(2012)**, pp. 2-2.
[15] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin and I. Stoica, “Graphx: Graph processing in a distributed dataflow framework”, In Proceedings of OSDI, **(2014)**, pp. 599-613.