Performance Evaluation of Social Network Analysis Algorithms Using Distributed and In-Memory Computing Environments

Elvin John V
PG Student
Department of Information Science & Engineering
M. S. Ramaiah Institute of Technology

Abstract

Analysis on Social Networking sites such as Facebook, Flickr and Twitter has long been a trending topic of fascination for data analysts, researchers and enthusiasts in the recent years to maximize the value of knowledge acquired from processing and analysis of the data. Apache Spark is an Open-source data-parallel computation engine that offers faster solutions compared to traditional Map-Reduce engines such as Apache Hadoop. This paper discusses the performance evaluation of Apache Spark for analyzing social network data. The performance of analysis varies significantly based on the algorithms being implemented. This is the reason to what makes this analysis worthwhile of evaluation with respect to their versatility and diverse nature in the dynamic field of Social Network Analysis. We compare performance of Apache Spark by evaluating the performance using various algorithms (PageRank, Connected Components, Counting Triangle, K-Means and Cosine Similarity) making efficient use of the Spark cluster.
Keywords: Social Network Analysis, Distributed Computing, in-Memory Computing, Apache Spark

I. INTRODUCTION

The idea of Social Network Analysis (SNA) has been around for many years now. Most organizations have realized the importance of it and understood that if they can collect all the data streaming in their business, they can apply analytics and acquire significant knowledge from it and to discover insights and trends.

Data Analytics experienced a paradigm shift from traditional data processing with the advent of Map-Reduce framework like Apache Hadoop. Though Hadoop has been extensively used for Data processing, performance wise a better solution will be Apache Spark, a data-parallel processing framework which integrates batch and stream processing and comprise of libraries which supports distributed machine learning, graph processing and SQL querying.

Apache Spark is a popular open-source Map-Reduce like data-parallel computation engine for large-scale data processing. Spark supports cyclic data flow and in-memory computing, that enables it to process data much faster compared to disk-based Map-Reduce engines such as Hadoop. It benefits from rich set of features such as in-memory storage abstraction called Resilient Distributed Datasets (RDDs), interactive operations, fault tolerance and transparent recovery. Apache Spark is implemented in Scala, a statically typed high-level programming language for Java Virtual Machine, and exposes an interactive programming interface. Spark is10x faster than Hadoop in iterative machine learning and can be used to scan a 39 GB dataset with sub-second latency.[1]

Mllib is a distributed machine learning library which enables efficient functionality for an extensive range of machine learning which consists of in-built statistical, optimization, and linear algebra primitives. Mllib provides us with a high-level API that makes use of Spark data-parallel framework and is designed to scale out on a cluster for general algorithms such as classification, clustering, collaborative filtering, and so on [2].

GraphX, a graph processing system which runs on top of Apache Spark data-parallel framework which provides various API to manipulate graphs. and also it provides a general execution model that can optimize arbitrary graph. It consists of a library of common graph algorithms such as PageRank, Connected Components, Triangle Counting etc. and they can also be implemented using only a few basic data-flow operators such as join, map, group-by etc. These Libraries efficiently formulates graph-parallel computation within the data-parallel framework of Apache Spark by making use of the graph data structures such as vertex cuts and graph-parallel operators [3][4].

The algorithms we have used for analysis are taken from the Naive Spark, Mllib and GraphX libraries. We evaluate Apache Spark, Mllib and GraphX on different sets of workloads and evaluate their performance over the base framework and the performance of graph processing systems while facilitating a wide range of SNA techniques.

All rights reserved by www.journalforresearch.org

A. Data Sets

The Datasets used in this work contains relation data from different networks, where the relationships are represented by unique IDs. A relation exists between two members of a network if they communicate with each other. In a graph, members are represented by vertices and the relations between them are represented by edges. The source data is taken from Stanford Large Network Dataset Collection [15].

Table - 1
Data different data set we have used for analysis and the respective size of data in MB.

Social Network Data Sets	*Data Size (in MB)*
Facebook	*1*
Amazon	*13*
YouTube	*40*
Flickr	*50*
Higgs-Twitter	*180*
Patents	*280*

B. Configuration

All the Spark application were run on a cluster with 1 master and 4 slaves with the configuration as specified in the table.

Table - 2
The configuration of the Spark Cluster used for Analysis.

Processor	*Intel Core™ i5-4460 CPU @ 3.20GHz × 4*
Operating System	*Ubuntu 15.10*
RAM	*8 GB*
Spark Version	*1.6.0*

II. Literature review

Social network analysis is a study of social relationships between individuals in a society. SNA is a set of methods for analysis of social structures, that provides an investigation of relational aspects for analyzing social structures.[5]

From way back in the late 1970s till today SNA techniques have emerged as one of the most successful applications of the internet. There are several reasons to why we need a better understanding of the social structure, justify the need for analysis and also studying the impact of these on future internet. Various methods of analysis are applied from mathematical models to graph mining.[6]

Research has been done on finding the core members for centralized clusters of members communicating with one another by taking datasets from projects repositories and applying SNA techniques on them. Finding the core members of a network is necessary because most of the successful collaborations in projects likely involve core members differently than peripheral members [7].

Numerous analysis methods such as understanding relationship, methods of communication, and analysis of network structure, path and centrality analysis are taken into consideration to gain insights about influence to individual members [8]. SNA can also be used to analyze trust and reputation. Studies show that it is possible to understand about the behavior of individuals by analysis of their social network [9].

Some Research paper focuses on discovering communities in a social network data and analyzing the community evolution over time. These communities inherent characteristics of human interaction in online social networks [10]. And few on Link prediction based on measures of proximity or similarity of nodes in a network topology. Link prediction is a method of estimating frequencies for future connections that are most likely to happen [11].

Recent works have shown the growth of socially-enhanced applications that leverage relationships from social networks to improve security and performance of network applications, including spam mitigation, Internet search engine optimization, and defense system against Cyber threats [5].

Various works focus on the performance evaluation of data-processing engines used to perform network analysis. One of the most widely used processing engine to perform analysis is Map-Reduce. Some works focus on performance analysis of iterative graph algorithms on various frameworks. Pegsus, a pure message passing model to process graphs is 5x faster performance over the regular version. Distributed GraphLab can outperform Hadoop by 20-60x [12].

[13] Paper explores topics in Big Data Analysis and its tools and also introduces us to one of the emerging data-processing tool, Apache Spark along with justification of using it. [3] introduces us to GraphX, which combines the advantages of both data-parallel and graph-parallel systems and [2] present MLlib, a machine learning library; both of them provides a high-level API and makes use of Spark's rich distributed parallel framework.

A study compares the performance of Apache Spark and Map-Reduce using a standard machine learning algorithm for clustering (K-means) and shows that Spark turn out to be three times faster than Hadoop[14]. Other works focus on Graph parallel processing focus on static graphs on large-scale distributed graph processing, several other systems focus on stream processing. Some other works focus on various optimizations techniques developed for GraphX to improve the design of distributed graph-parallel processing.

All rights reserved by www.journalforresearch.org

III. FINDING INFLUENTIAL NODES

Problem Statement: To find the most influential individual among a large set of members in the social network.

An influential individual can be regarded as one who is well connected to most of the members in a social network and available in a denser location and acts as a channel for the spread of ideas, knowledge and information among the members of the network. Influence maximization is another problem of finding a subset of individuals in a social network who could maximize the spread of influence. PageRank is a mathematically model in analyzing influence.

A. PageRank

PageRank works by calculating the number and weight of links between members of a social network to estimate the importance of an individual, it is assumed that the most important individual receive more links from other members. The rank value indicates an importance of a particular member or the influence of the member in the network. A higher rank indicates the more influential an individual is. PageRank is a recursive algorithm and depends upon the total number of nodes and incoming links to it. It is a general algorithm and can be applied to any graph or network of any domain and is used in a wide area of analysis such as bibliometrics, social and information network analysis etc.

1) Algorithm

```
//GraphX PageRank
val pr = PageRank.run(graph, numIter).vertices.cache()
println("GRAPHX: Total rank: " + pr.map(_._2).reduce(_ + _))
pr.map {case (id, r) => id + "\t" + r }).saveAsTextFile(outFname)
//For top 5 ranks
pr.takeOrdered(5)(Ordering[Double].reverse.on(x=>x._2)).foreach(println)
//Naive Spark PageRank
val data = sc.textFile(args(0), 1)
val links = data.map{ s =>
val parts = s.split("\\s+")
(parts(0), parts(1))      }.distinct().groupByKey().cache()
var pageranks = links.mapValues(v => 1.0)
for (i <- 1 to 10) {
val contribs = links.join(pageranks).values.flatMap{ case (nodes, rank) =>
val size = nodes.size
nodes.map(node => (node, rank / size))
}
pageranks = contribs.reduceByKey(_ + _).mapValues(0.15 + 0.85 * _)
}
pageranks.saveAsTextFile(outFname)
println("Top 5 ranks:")
//For top 5 ranks
pageranks.takeOrdered(5)(Ordering[Double].reverse.on(x=>x._2)).foreach(println)
```

2) Results

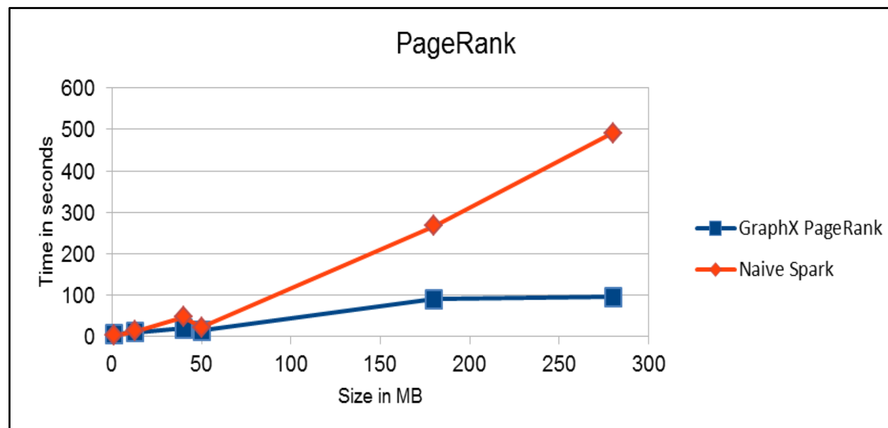

Fig. 1: Compares the execution time of GraphX PageRank and Naive PageRank algorithm over different sets of data.

The Graph clearly shows that Naive PageRank tends to grow linearly with the increase in the size of data where as the GraphX algorithm executes comparatively faster and is gradually levelled off towards the end.

All rights reserved by *www.journalforresearch.org*

IV. Clustering and Classification

A. Connected Components

Problem statement: To find all the connected nodes in a network who are connected with each other and if or not each node is reachable from other node through a path.

Determination of connected components in a network is a fundamental phase concerning various applications in analyzing the structure of networks. Connected components algorithm is based on depth-first search traversal (DFS).

Connected components are sub-graphs of the original graph of which any two nodes are connected to each other by one or more edges. The aim of finding the connected components in a graph is to find such clusters. In a social network, connected components can be considered as approximate clusters. Connected components is also an iterative algorithm. For each vertex, the connected components algorithm computes the lowest vertex id such that there is a path from the vertex to another. Every vertex is initialized to be in the value of its own self. Then, the value of each vertex is saved as the minimum component of its neighbors in each iteration. And finally once there is a no vertices change in the value then we declare convergence is reached. Once all the components are determined, each vertex is labeled according to its assigned component with a lowest vertex id.

A directed graph is called strongly connected if there is a path in each direction between each pair of vertices of the graph. In a directed graph G, a pair of vertices u and v are said to be strongly connected to each other if there is a path in each direction between them. Strongly connected components in a graph are identified by interconnected vertices. This is followed by merging all the vertices into a single vertex for a simplified view of the graph. Strongly connected component (SCC) of each vertex is computed and it returns a graph with the vertex value containing the lowest vertex id in the SCC containing that vertex.

1) Algorithm

```
//Connected Components
val graph = GraphLoader.edgeListFile(sc, filename)
val cc = graph.connectedComponents()
println("Components: "+cc.vertices.map { case(vid, data) =>data }.distinct())
//Strongly Connected Components
val graph = GraphLoader.edgeListFile(sc,fname).partitionBy(PartitionStrategy.RandomVertexCut)
val sccGraph = graph.stronglyConnectedComponents(10)
for ((id, scc) <- sccGraph.vertices.collect()) {
println(id,scc)
}
```

2) Results

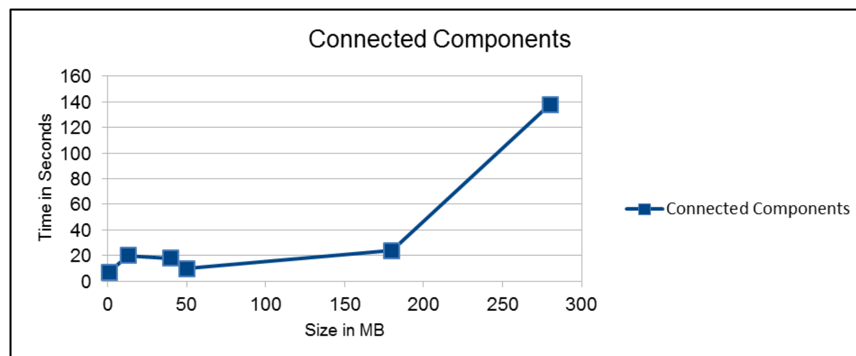

Fig. 2: Shows us the execution time of Connected Components algorithm over different sets of data

The Graph tends to grow super linearly at the beginning and then fluctuates with the increase in small range of the size of data later tends to steadily remain constant and then grows super linearly again towards the end.

The Graph of Strongly connected components is not plotted because it doesn't depend on the size of the dataset but rather on the connectivity of the members.

B. Counting Triangle

Problem statement: Finding the number of triangles (three node nontrivial sub-graph) formed among vertices in a social network. A triangle is the most basic component of a graph which is fully connected.

The number of triangles in a graph is a fundamental statistic which is used in complex network analysis to find the clustering coefficient and also the transitivity ratio of a social network. Counting Triangles algorithm calculates the number of triangles passing through every vertex, which provides a measure of clustering

Counting triangles is a basic graph mining technique and is widely used in community detection algorithms.. A community is have a high concentration of triangles among themselves and close more triangles with other vertices than with vertices outside of the community. Such communities are well isolated and can be classified from the rest of the graph.

All rights reserved by www.journalforresearch.org

The clustering coefficient is primarily based on three interconnected vertices known as triplets. It is mathematically defined as number of closed triplets divided by the number of total connected triplets. The clustering coefficient computes the degree of herding effect in a network graph; higher coefficient value indicates that the nodes are more dense and tightly interconnected among one another. This method of calculating the clustering coefficient in a network is often known as triangle counting.

1) Algorithm

```
Val graph=GraphLoader.edgeListFile(sc,fname,
canonicalOrientation=true,
.partitionBy(partitionStrategy.getOrElse(RandomVertexCut))
.cache()
val triangles = TriangleCount.run(graph)
println("Triangles: " + triangles.vertices.map {
case (vid, data) => data.toLong
}.reduce(_ + _) / 3)
```

2) Results

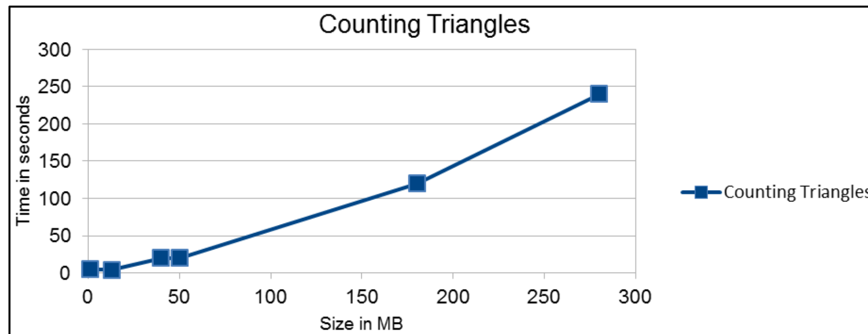

Fig. 3: Shows us the execution time of Counting Triangles algorithm over different sets of data

The Graph just tends to grow linearly with the increase in the size of data inferring that as the size grows the execution time also increases propotionally.

C. K-means Clustering

Problem statement: finding a number of desired clusters in a network and the center of the clusters.

K-means is a widely used clustering algorithm that is used to group data points into a desired number of clusters. Where k denotes the number of desired clusters and epsilon denotes the threshold distance for which convergence is considered to have been achieved or it converges when no change occur in the assignment of clusters. The algorithm takes the data points, the number of desired clusters and the number of iterations as input and gives us the centers of the clusters as output.

K-Means is an unsupervised algorithm in other words a training example is not required for it to work and hence is more faster compared to other clustering methods. Bisecting K-Means is a kind of hierarchical clustering. Hierarchical clustering is another cluster analysis method which builds a hierarchy of clusters.

We have tested both Naive K-means and Machine Learning K-Means implementation on Spark. Mllib Bisecting K-Means is the implementation of Bisecting K-Means algorithm and Mllib Dense K-Means is specialized to work on denser graph data.

1) Algorithm

```
//Naive K-Means Clustering
while(tempDist > convergeDist) {
val closest = data.map (p => (closestPoint(p, kPoints), (p, 1)))
val pointStats = closest.reduceByKey{case ((p1, c1), (p2, c2)) => (p1 + p2, c1 + c2)}
val newPoints = pointStats.map {pair =>
(pair._1, pair._2._1 * (1.0 / pair._2._2))}.collectAsMap()
tempDist = 0.0
for (i <- 0 until K) {
tempDist += squaredDistance(kPoints(i), newPoints(i))
}
for (newP <- newPoints) {
kPoints(newP._1) = newP._2
}
//Mllib Bisecting K-Means
def parse(line: String): Vector = Vectors.dense(line.split("").map(_.toDouble))
val data = sc.textFile(fname).map(parse).cache()
val bkm = new BisectingKMeans().setK(5)
val model = bkm.run(data)
```

All rights reserved by *www.journalforresearch.org*

```
model.clusterCenters.zipWithIndex.foreach { case (center, idx) =>
println(s"Cluster Center ${idx}: ${center}")
}
//Mllib Dense K-Means
val lines = sc.textFile(fname).map { line =>
Vectors.dense(line.split('').map(_.toDouble))
}.cache()
val initMode = KMeans.K_MEANS_PARALLEL}
val model = new KMeans()
.setInitializationMode(initMode).setK(k).setMaxIterations(numIterations).run(lines)
val cost = model.computeCost(lines)
println(s"Total cost = $cost.")
model.clusterCenters.zipWithIndex.foreach {case (center, idx) => println(s"Cluster Center ${idx}: ${center}")
}
```

2) Results

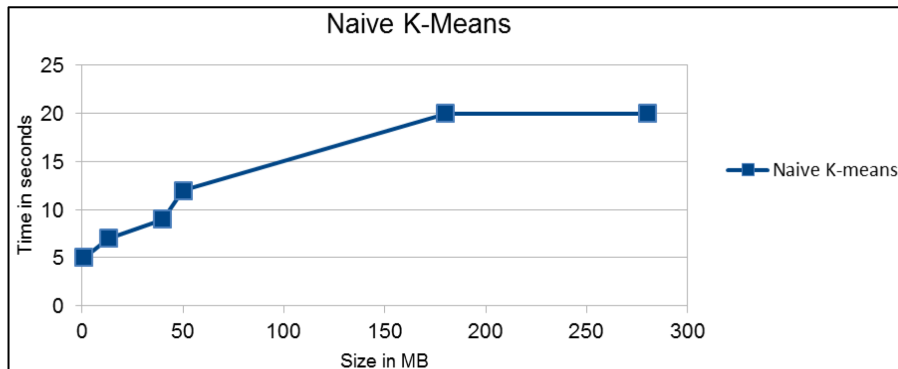

Fig. 4: Graph 4 shows us the execution time of naïve K-Means algorithm over different sets of data

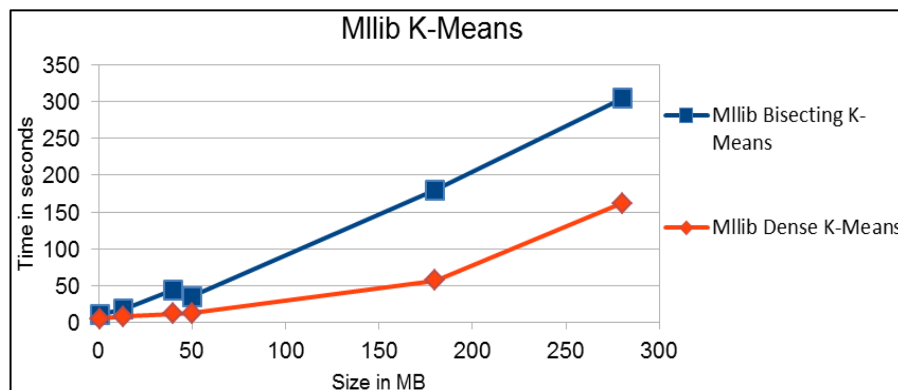

Fig. 5: Graph 5 shows us the comparison of the execution time of Mllib Bisecting K-Means and Dense K-means algorithms over different sets of data.

Graph 4 tends to grow quite linearly in the beginning and with more rise in the size of data it tends to remain almost constant where as in Graph 5 it clearly shows that Dense K-Means executes faster than Bisecting K-Means but since it's a machine learning algorithm it takes more time compared to naïve K-Means.

K-Means provides different results every time because we randomly chose our initial clusters. Machine learning K-Means algorithm therefore finds it very difficult to infer on the results obtained effectively and so takes a lot more time to calculate than naive K-means.

V. Link prediction

Problem statement: Identifying future interactions between individuals who have never interacted before.

Link prediction in networks is found a strong hold in social communities. It is widely used to find missing information, identify bogus interactions and also used to evaluate evolving network mechanisms etc. Link prediction can be done using various methods, proximity measure algorithm, maximum likelihood algorithms and also probabilistic models etc [16].Link Prediction algorithm identifies the interactions that are most likely to happen in the future taking into consideration of their interests.

All rights reserved by www.journalforresearch.org

A. *Computing Similarity*

A proximity measure computes the distance measure between nodes with respect to their location. The proximity measures is nothing but a similarity measure. They ranges between values of 0 and 1, if the value is closer to 1 then it indicates that the graph has high similarity. Similarity measure is first finds the number of mutual neighbors and thereby predicting the probability of future links. It is assumed that if the mutual neighbors are high, then it is more likely that the communication will happen in the future.

Link Prediction algorithms are being applied in recommendation systems. Collaborative filtering algorithms computes similarity measure between two users and recommends items purchased by similar users. A distance measures between the users is used to compute the similarity of the items. Then the similarity values are used to produce a ranked list of recommended items. Among many methods to compute similarity, Cosine Similarity are the most popular and widely used. Cosine similarity is computed by representing each user as a vector of their ratings on items and then measuring the cosine of the angle formed between them.

Here we compute all cosine similarities between columns of the matrix using Brute-Force approach(computing normalized dot products) and also using DIMSUM a sampling approach(threshold).

1) Algorithm

```
val rows = sc.textFile(Fname).map { line =>
val datavalues = line.split('').map(_.toDouble)
Vectors.dense(datavalues)
}.cache()
val matrix = new RowMatrix(rows)
// Compute similarities using brute force approach
val exact = matrix.columnSimilarities()
// Compute similarities with estimation using DIMSUM approach using a threshold value
val approx = matrix.columnSimilarities(threshold)
```

2) Results

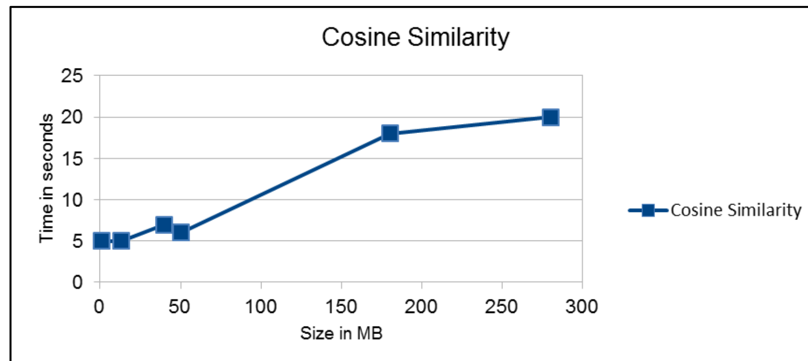

Fig. 6: Graph 6 shows us the execution time of Cosine Similarity algorithm over different sets of data

The Graph tends to grow a quite linearly in the beginning and with increase in the size of data it tends to converge for higher datasizes.

VI. Conclusion

Social Network Analysis is gaining a lot of prominence due to the growth of online social network services and thereby has led into exploring the information involving such networks. The main concern of this paper is analyzing the social networks data, extract and understand hidden information and perform a knowledge discovery on them. In this work we have introduced Apache Spark, Mllib and GraphX, an efficient data processing system for social network analysis that enables distributed-parallel computing frameworks for efficiently executing iterative network analysis algorithms. We believe that our work on Apache Spark, Mllib and GraphX benefits an extensive research agenda in data processing systems. Spark being open source and highly scalable computing framework, it can effectively serve the needs of the ever growing data size as well.

Acknowledgment

I would like to express my deepest appreciation to The Department of Information Science, M.S. Ramaiah Institute of Technology to have provided me the possibility to complete this research paper. I would like to thank Krishnaraj P M for providing his insights and expertise that greatly assisted the research.

All rights reserved by www.journalforresearch.org

REFERENCES

[1] Zaharia, Matei, et al. "Spark: Cluster Computing with Working Sets."HotCloud 10 (2010): 10-10.
[2] Meng, Xiangrui, et al. "Mllib: Machine learning in apache spark." arXiv preprint arXiv:1505.06807 (2015).
[3] Xin, Reynold S., et al. "Graphx: A resilient distributed graph system on spark."First International Workshop on Graph Data Management Experiences and Systems. ACM, 2013.
[4] Gonzalez, Joseph E., et al. "Graphx: Graph processing in a distributed dataflow framework."11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). 2014.
[5] John Scott. Social network analysis. Sage, 2012.
[6] Anurag Srivastava, Dimple Juneja Gupta, et al. Social network analysis: Hardly easy. In Optimization, Reliabilty, and Information Technology (ICROIT), 2014 International Conference on, pages 128–135. IEEE, 2014.
[7] Kevin Crowston, Kangning Wei, Qing Li, and James Howison. Core and periphery in free/libre and open source software team communications. In System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on, volume 6, pages 118a–118a. IEEE, 2006.
[8] Stanley Wasserman and Katherine Faust. Social network analysis: Methods and applications, volume 8. Cambridge university press, 1994.
[9] Jordi Sabater and Carles Sierra. Reputation and social network analysis in multi-agent systems. In Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1, pages 475–482. ACM, 2002.
[10] Yu-Ru Lin, Yun Chi, Shenghuo Zhu, Hari Sundaram, and Belle L Tseng. Analyzing communities and their evolutions in dynamic social networks. ACM Transactions on Knowledge Discovery from Data (TKDD), 3(2):8, 2009.
[11] Liben-Nowell, David, and Jon Kleinberg. "The link-prediction problem for social networks." Journal of the American society for information science and technology 58.7 (2007): 1019-1031.
[12] Gu, Lei, and Huan Li. "Memory or time: Performance evaluation for iterative operation on hadoop and spark."High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on. IEEE, 2013.
[13] Shoro, Abdul Ghaffar, and Tariq Rahim Soomro. "Big Data Analysis: Apache Spark Perspective." Global Journal of Computer Science and Technology15.1 (2015).
[14] Gopalani, Satish, and Rohan Arora. "Comparing Apache Spark and Map Reduce with Performance Analysis using K-Means."International Journal of Computer Applications 113.1 (2015).
[15] Jure Leskovec and Andrej Krevl. "Stanford Large Network Dataset Collection." SNAP, June 2014. Web. Available at http://snap.stanford.edu/data
[16] Lü, Linyuan, and Tao Zhou. "Link prediction in complex networks: A survey."Physica A: Statistical Mechanics and its Applications 390.6 (2011): 1150-1170

All rights reserved by *www.journalforresearch.org*