https://doi.org/10.51153/kjcis.v5i1.83

Automatic Taxonomy Generation and Incremental Evolution on Apache Spark Parallelization Framework

Kanwal Aalijah[1] Rabia Irfan[1] Umara Umar[3] Sanam Nayab[4]

Abstract

The term "Big Data" refers to a large volume of information usually in terabytes and petabytes. It includes both structured and unstructured data. Unstructured data is conventionally text-heavy, but may also contain data such as facts, dates, and numbers. To use this unstructured information effectively, it needs to be processed and organized. Taxonomy is considered a powerful way of organizing information. For automatic taxonomy generation, various techniques have been proposed in the past. However, the substantial nature of big data presently crosses the processing abilities of traditional techniques. Thus, to meet this challenge an extensible and scalable technique is required to potentially accelerate the process of taxonomy generation and its evolution upon arrival of new data, hence catering to a large amount of unstructured big data. This paper proposes a technique for both the taxonomy generation and evolution of Apache Spark infrastructure. The proposed technique is evaluated on a text dataset from a computing domain. The evaluation results show that the technique presented in this paper outperformed the existing techniques in terms of time and quality metrics. The time and quality-based evaluation showed that the use of the MapReduce environment has resolved the scalability issues of the current taxonomy generation and evolution process.

Keywords: Big Data, Apache Spark, Unstructured Data, Taxonomy, Map-Reduce, Hadoop, Scalable

1. Introduction

During the past two decades, communication using electronic media has acquired extreme popularity and has gained a significant role in developed societies. Electronic media provides several services such as the World Wide Web (WWW), mobile devices, Internet of Things (IoT)-based devices, social networks, etc. This era is marked by the circulation of the intense amount of data (in petabytes and zettabytes) across the globe. This large volume of data produced from various sources [1] can be both structured and unstructured. This bulk of data is called Big Data-A Technology Giant [2], [4], [5]. The

[1]*Kanwal Aalijah l ksair.mscs8seecs@seecs.edu.pk*
[2]*Rabia Irfan l rabia.irfan@seecs.edu.pk*
[3]*Umara Umar l uumar.msit19seecs@seecs.edu.pk*
[4]*Sanam Nayab l snayab.msit19seecs@seecs.edu.pk*

5V's of big data – volume, velocity, variety, veracity, and value make data management and analytics challenging for the conventional data warehouses. Big data, which is unstructured, is the data with no standard formatting [3] and no definite structure as the name shows. In order to draw useful information from this data, it should be managed, processed, and effectively transformed. In other words, this data needs to be organized into a structured form, like taxonomy.

Taxonomy is a hierarchical structure that organizes the given data in parent-child relationships, based on the inherent concepts present in the data [6]. Taxonomy is an efficient and effective way of organizing and classifying data [7] that also provides standardization in case of the exchange of information. Taxonomy also provides an infrastructure for knowledge management [8]. Taxonomy arranges information in a hierarchical structure that makes navigation and searching for information easier [9] [10].

Automatic taxonomy generation has two types i.e. (1) Incremental (2) Non-incremental. Non-incremental taxonomy generation rebuilds the taxonomy from very scratch on the entry of new documents into the current system. Kashyap et al [11] proposed an innovative method for taxonomy generation that uses the Principal Direction Divisive Partitioning (PDDP) approach [12] to generate taxonomy. Anke et al. [13] suggested a conditional random field classifier for taxonomy generation. Velerdi et al. [14] presented a graph-based method for taxonomy generation. All these techniques successfully resulted in taxonomy generation, but upon the intervention of the new documents into the system, these techniques regenerate the taxonomy from the very basis to get the taxonomy updated, consequently producing non-incremental taxonomy architecture. This approach is very time-consuming when a large dataset is involved. So, there was an extreme need for a technique that generates taxonomy on top of the current taxonomy on the arrival of the new document into the system. This process is known as "Incremental/ Progressive Taxonomy Generation" or may be named as "Taxonomy Evolution".

There are rare techniques which have focused on incremental taxonomy generation like [15], AdaptTaxa [16], IHTCTaxa [17], TIE [18]. The methodology EvoTaxa [15] is especially developed for tagged data. AdaptTaxa [16] focuses on incremental taxonomy generation technique for unstructured textual data. It adopts a supervised approach that requires training data. The technique IHTCTaxa [17] uses an unsupervised hierarchical clustering-based approach by adjusting the newly introduced documents. TIE [18] is an incremental taxonomy generation algorithm that updates taxonomy upon the entry of new documents. All these techniques, be it non-incremental or incremental, provide a more or less good quality taxonomy, however, lacks the focus on rapidly increasing, voluminous big data. With the emerging trend of big data and cloud computing, the data is being produced from varying sources as well as being stored and processed electronically and automatically [19][20].

In the realm of big data, we are always in search of certain techniques and algorithms which prove to be dependable and scalable to negotiate with the varying kind of data. Some progress has already been achieved in the field of hierarchical clustering for huge datasets, such as [21] and [22]. The scope of these studies was limited to hierarchical clustering and they did not adequately concentrate on the idea of taxonomy generation and evolution. Besides, none of these techniques addressed the concept of parallelization for developing a scalable and efficient algorithm for generating and evolving taxonomy.

A new technique has been devised in our work [37] for the taxonomy generation and incremental evolution comprised of the MapReduce paradigm incorporating Apache Spark. MapReduce is capable of minimizing time by parallel data processing. Fault tolerance is also being provided by capitalizing on a distributed file system [19]. MapReduce environment can improve the scalability issues of present taxonomy generation and evolution methodologies. However, our previous work didn't focus on the evaluation of the proposed technique with respect to the parallelization framework, thus, we were not be able to figure out the essence of achieving scalability previously. This paper particularly focuses on this aspect.

The major problem with existing taxonomy generation algorithms was the amount of data it can process. Our algorithm processes the data in a parallel fashion in small chunks applying HAC on each chunk of data. That is where map-reduce comes in. The principle behind map-reduce is you divide the tasks into smaller tasks and then combine them. Exactly in the same fashion, we are making small taxonomies on each chunk of data and once all those taxonomies are made, they are combined. We use HAC on the spark engine which at the backend uses map-reduce to perform HAC.

In our research, we have made the following contributions:

1. The proposed technique provides us a solution for taxonomy generation and evolution in a considerably limited span of time in comparison with the existing techniques, thereby making taxonomy utilization more effective.
2. As clustering is the base of the adopted taxonomy generation algorithm, the clustering quality of taxonomy generated from the proposed methodology is compared and evaluated with the clustering quality of taxonomy generated using the existing taxonomy generation techniques. According to Silhouette's score and Davies Bouldin's score, the clustering quality of the proposed methodology is higher than the present techniques.
3. Zero or no similarity of a document with the current clusters case is being addressed.
4. For the case of evolution, the application of Newick tree graph facilitates the technique to incorporate even a graph-based taxonomy instead of just clustering-based taxonomy.

The salient features of the remaining part of this article are as follows:
Succeeding the Introduction in Section I, Section II discusses the Literature Review. Literature review elaborates the existing techniques for non-incremental and incremental taxonomy generation in detail. This section also throws light on the basic taxonomy generation process that has been used by the existing techniques. Section III presents the background and discusses the preface of big data techniques and tools used in this research work. Section IV explains the proposed technique devised for the processes of taxonomy generation and taxonomy evolution. Section V compares the proposed methodology with the current non-incremental and incremental taxonomy generation techniques and tests the scalability of the proposed technique. Finally, Section VI summarizes the Conclusion and Future Work.

2. Literature Review

This section describes the automatic taxonomy generation in detail. Automatic taxonomy generation process consists of two types: incremental and non-incremental. Be it as a non-incremental taxonomy or an incremental taxonomy generation process, a basic taxonomy generation algorithm is used in order to build the initial taxonomy in both cases. The commonly used steps of taxonomy generation are: data preprocessing, data modeling, hierarchy formation and node labeling. Different works have used different approaches in order to perform these steps. In general, taxonomy generation algorithms first cleanse the data using preprocessing that includes the removal of unnecessary details from the data. Once the data is preprocessed, it is then modeled to bring into a computational form. Using the modeled data, hierarchical relationships are produced, organized, and then labeled to obtain a structure in a hierarchical form of taxonomy.

Non-incremental type of taxonomy generation procedure utilizes the basic process of a taxonomy generation to generate taxonomy and the process runs every time when the newly arriving documents are presented into the system. The work TaxGen [23] presented an automatic taxonomy generation algorithm for unstructured data. The algorithm uses hierarchical clustering algorithm (HCA) for building the underlying structure for taxonomy generation. TaxaMiner [11] was also an addition in the pool of existing non-incremental taxonomy generation techniques. The cluster cohesion is used to extract the taxonomy among the successive levels of the hierarchical clustering tree. TaxoLearn [24] is also a non-incremental taxonomy generation algorithm. In this work, taxonomy hierarchy is built using an unsupervised hierarchical clustering algorithm [25]. On the other hand, an incremental taxonomy generation or taxonomy evolution technique works in a fashion that in-occurrence of the new documents in the system the process does not re-build the entire taxonomy from scratch; instead of that, the new documents are presented in the current taxonomy based upon the similarities with the existing dataset.

AdaptTaxa [16] generates taxonomy incrementally for group profiling problem. EvoTaxa [15] generated taxonomy incrementally for particularly large collection of tags. In this technique, a graph called association rules graph is produced. In an association rules graph, the vertices are tags and based on support and confidence values these tags are connected. Manipulation on the association rules graph is done by taxonomy extraction step. Only those associations are kept which don't add to noisy associations. The technique successfully generated and evolved taxonomy but it does so for tag data only. IHTCTaxa [17] uses unsupervised incremental hierarchical clustering approach to generate taxonomy for unstructured textual data. IHTC (Incremental Hierarchical Term Clustering) algorithm considers the problem of hierarchical clustering as online in contrary to the batch mode non-incremental hierarchical clustering, like HAC [26] and Bisect K-means [27].

TIE [18] algorithm was as advancement in the domain of incremental taxonomy generation. The TIE algorithm takes as an input the following: 1) existing taxonomy 2) respective hierarchical structure (i.e., clusters hierarchical structure) 3) new documents. The nearest cluster of new arriving document is recognized based upon the similarity score. The similarity score range may well identify the level of impact that a new arriving document has on its closest or nearby cluster. For the level of impact, to accommodate the new documents in a current hierarchical structure most of the reorganization operators came into practice. Hence, the current taxonomy develops to identify the change take place in the data [18]. In short, it was observed that the majority of the available non-incremental or incremental taxonomy generation approaches produce the good worth taxonomy. But, these approaches may lack attention on speedily expanding, voluminous and varying natured big data.

Furthermore, it was observed from the analysis of the literature that underlying technique for building a hierarchical structure in a taxonomy generation or evolution technique is mostly clustering-based [28]. Clustering techniques are very useful tools in case an unstructured data needs to be organized in a hierarchy [29]. In our work, we, particularly focus on clustering-based incremental taxonomy generation techniques. However, new challenges of big data make it difficult to apply conventional clustering techniques. Large data volume and time complexity of clustering algorithms lead to the problem of efficient deployment of clustering algorithms for big data to get an outcome in a reasonable amount of time.

Clustering algorithms dealing with big data are generally classified into categories as [30]: partitioning-based clustering approaches, hierarchical clustering approaches, grid-based clustering approaches and model-based clustering approaches. All these techniques have their own advantages and disadvantages. Partitioning-based clustering technique has a disadvantage that it requires a pre-defined value of K parameter to be given by a user.

For a clustering solution the value of K is often non-deterministic [31]. In a hierarchical clustering technique once a stage is completed it cannot be un-done. All the hierarchical clustering algorithms have the limitation stated above [32].

Density-based clustering algorithms contain noisy objects because they work in such a way in which clusters are described as dense areas separated by low density regions [32], therefore, not considered appropriate for very huge size datasets. Clustering algorithms that are based on a model are slow and unsuitable for very large dataset for a classification problem as they utilize the multivariate probability distribution. The grid size is usually far smaller than the database size. In case of highly irregular data distributions, using a single uniform grid might not be a good idea as a single uniform grid will fail to provide the required clustering quality and also is not able to fulfill the required time requirement [31].

Moreover, clustering techniques for big data mentioned here are specifically designed for dealing with big data but to be run on a single machine. New challenges of big data can be solved using multiple machines clustering techniques that can be able to achieve results in a much smaller time. Such parallel algorithms divide the data into various smaller data partitions and distribute them on different machines. This makes the overall running time of the algorithm smaller and increases its scalability. MapReduce algorithm is a task partitioning algorithm designed for distributed execution of a task on many servers which gives a good base for the implementation of such parallel forms of algorithms for data clustering. To understand its working, the next section discusses the MapReduce environment and tools used for big data processing.

3. Background and Preliminaries

This section discusses prominent tools in the world of big data processing: Apache Hadoop and Apache Spark which are based on MapReduce paradigm.

A. MapReduce

Researchers at Google presented a new programming model called MapReduce [33], which was able to solve the challenges of efficient processing of massive datasets using large clusters. MapReduce solves the problems faced in parallelizing the data across the individual machine's clusters [33]. MapReduce gives an easy and simple model for distributed computing by solving the problems of data partition, scheduling of machine failure and decreasing inter-machine communications. MapReduce is a programming paradigm that works by decomposing the problem into multiple map and reduce tasks. An Input is inserted in the form of key or value pairs to the mapper function. This key value pair input is then passed to reducer which then gives it as an input to the reduce

function. Associated with the intermediate key, the reducer merges the intermediate values and finally produced a combined output.
In real world scenarios, several map reduce functions can be applied on various machines individually in order to achieve parallelization. Apache Hadoop and Apache Spark are prominent big data processing environments that uses MapReduce algorithm for processing and analyzing the data [33], which are discussed in the succeeding subsections.

B. Apache Hadoop

Hadoop is a software framework based on MapReduce algorithm. The framework can write applications that can handle huge size of data in-parallel over the large size of clusters. The size of the data to be processed is in multi-terabyte. The size of the cluster is of thousands of nodes. Hadoop provides efficient, reliable and fault tolerance system for processing of big data. There are many different tools and products in Hadoop ecosystem.

The two important components of Hadoop ecosystem are HDFS and YARN. Hadoop Distributed File System [34], commonly referred as HDFS, is a single reliable file system. HDFS is reliable file system as it offers the monitoring of failures of data blocks. Each data block has its replica stored on another block and incase of failure data can be retrieved from other block. This feature of HDFS makes it easier to use commodity hardware for processing of big data. YARN stands for Yet Another Resource Negotiator. It separates MapReduce from resource manager, workflow manager and fault-tolerance. It allows other frameworks to be built on top of it. The original Hadoop framework was modified to use YARN. The initial version of Hadoop had technical deficiencies [34] that the current system is dealing by introducing a structure called linear data flow on the distributed computing programs in the Hadoop cluster. Hadoop gets an input data from the disk, perform mapping function on the data, reduce results of map function, and then finally, stores reduce results on the disk. Everything was to be read and written to disk. This made the implementation of the iterative algorithms difficult [35]. An Iterative algorithm works on dataset multiple times in a loop and then applying data analysis on side. These training algorithms used in systems having machine learning standards. Current version of Hadoop does not have the capability for processing of iterative machine learning algorithms and if processed it will take a lot of time to finish a job. This provided a need of a technology that would solve these issues. This leads to the development of Apache Spark [36].

C. Apache Spark

Apache Spark was developed to facilitate the iterative and machine learning algorithms. Spark was born along with its important component the Resilient Distributed Datasets (RDDs) [36]. The RDDs perform in-memory computations on big data. It runs on large

clusters and nodes. It runs to provide fault-tolerance. Apache Spark also has many discretized streams. The discretized streams were developed in order to provide high-level programming API. An efficient fault tolerance and consistency achieves by high level programming APIs. Spark is one of most primitive high-level systems that not only supported the distributed batch and stream computation but also the iterative querying.

The key feature of Spark is an RDD [36]. RDDs are basically "immutable objects". These objects are usually stored into different partitions. When one RDD is modified, a new RDD is created. A new RDD generation leaves the previous RDD unconverted. It provides fault tolerance due to intelligence that decides when to regenerate and when to re-compute a dataset. The groundwork of a complete project Spark Core may deliver the scheduling, distribute the task, and fulfill essential input output functionalities. They are revealed by an Application Programming Interface (API). This API is centered on RDD abstraction. RDDs consist of two different kinds of operations: transformations and action transformations. Transformations always return pointers to the latest new RDDs. Another transformation called an Action transformation may return results to a driver program. Different transformations and actions can work together in a Spark job. MLlib is a distributed machine learning library framework that operates on the top of Spark core. The spark job operates nine times efficiently and faster than the disk-based implementation due to distributed memory-centered Spark architecture. Various machine learning algorithms has been proposed and transported to MLlib that enables ML large scale pipelines.

D. Comparison of Tools for Implementation

In order to support our choice of Apache Spark, we have demonstrated the suitability of Apache Spark (MLlib) for machine learning applications by comparing the performance of the two parallelization frameworks, i.e. Spark and Hadoop.
The comparison table of Apache Hadoop and Apache Spark with major differences is given in Table 1.

Table 1: Comparison Table

Attributes	Apache Hadoop	Apache Spark
Unified APIs	No	Yes
I/O Operations	Disk-based	Memory-based
Processing Speed	Slow	Fast
Execution	Slow	Fast
ML Support	Limited (for newer machines)	Full

It is also worth mentioning here that Apache Spark fully supports agglomerative clustering being used in the proposed technique whereas Apache Mahout does not

support agglomerative clustering. It supports two algorithms for clustering i.e. 1) Canopy clustering 2) K-means clustering.

The next section contains the well demonstrated discussion of the proposed technique in detail.

4. Proposed Technique

This section is further divided into two subsections. Taxonomy generation algorithm is discussed in the first subsection and the second subsection discusses taxonomy evolution process for updating taxonomy on arrival of new documents in a dataset. Both the taxonomy generation and evolution algorithms are based on a parallelization framework.

A. Taxonomy Generation

The proposed technique performs taxonomy generation on Apache Spark framework and has been divided into six general steps: loading the data, data pre-processing, data modeling, hierarchy formation, node labeling and conversion into tree graph. The process has been explained in detail in our work [37], highlight of which is below:

1. Loading the Data: Resilient distributed dataset (RDD) is used to effectively load text documents as input.
2. Data Pre-processing: In the pre-processing step, stop-word removal using NLTK and stemming using Porter stemmer [38] have been performed.
3. Data Modeling: A feature vectorization method called term frequency-inverse document frequency (TF-IDF) is used in this step.
   In Apache Spark, TF-IDF is performed in MapReduce paradigm whereas, TF-IDF is not calculated in a simple fashion, rather several number of map and reduce tasks are carried out for the implementation of TF-IDF. Apache Spark implements it using hashing trick or kernel trick. Hashing trick is a quick and compact way of vectorizing features. The hash function used here is MurmurHash3$_2$. Figure 1 demonstrates this process of vectorization and hashing.
4. Hierarchy Formation: To form a hierarchy, hierarchical agglomerative clustering approach is used. For implementation of this phase, Parallel prims algorithm is used. This algorithm is available in open-source Spark's library. First the algorithm divides the dataset into multiple sub-datasets. A serial minimum spanning tree algorithm is applied locally on each of the sub problems on it. Spark's programming model supports iterative algorithms because of RDD. In RDD the computation is carried out only in the RDDs that are required at the moment. In an iterative program, RDDs are consumed in a loop. This phase is

called Map phase. Each MST is

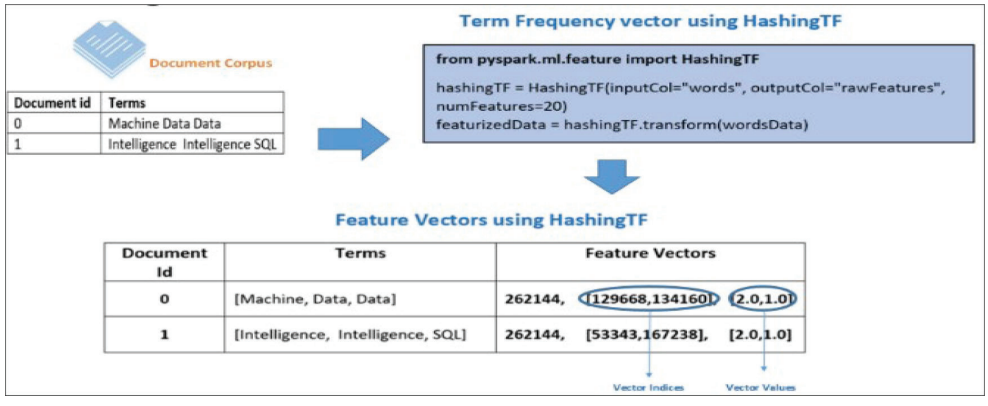

Figure 1: Implementation of TFIDF as HashingTF on Apache Spark

a cluster itself. Now multiple clusters that were created in the map steps are re-arranged in reduce steps based on the distance between two clusters iteratively and arranged in the form one bigger cluster. This is the Reduce phase. The information of distance between the trees clusters are maintained in the similarity matrix S_{gen}.

5. Node Labeling: The hierarchical composition built in the preceding step is unlabeled. This phase acquires labels for these unlabeled clusters. For labeling purpose, titles of documents in a cluster were chosen as labels. The titles were selected as labels because titles are easier to read as compared with the list of top terms in a cluster [18]. The technique basically labels a cluster with the title of the document that is attached to the edge having minimum weight. By the end of this step, the taxonomy T_{gen} has been created.
6. Conversion into a Tree Graph: To use this taxonomy for evolution subsequently, T_{gen} is then transformed into a Newick Tree Graph[3]. Newick is a standard for representing trees in a computer readable form by making use of nested parentheses as shown in Figure 2. The bottom-most node in the tree is an interior node. Matched parentheses represent interior nodes. In between them, there are the images of nodes that are instantly descended from a node which is comma separated. Real numbers are used to incorporate branch lengths. This represents the length of a branch immediately below a node.

```
((((((datamining.9:1.28,3dtech.11:1.28):0.05,(datamining.11:0.92,datamining.10:0.92):0.41):0.42,(computational
geometry.2:0.76,computationalgeometry.1:0.76):0.99):0.33,((((:1.09,3dtech.13:1.09):0.17,architectureeducation.
7:1.25):0.15,(microarchitecture.2:0.97,microarchitecture.1:0.97):0.44):0.22,(communicationsystems.2:0.70,commu
nicationsystems.1:0.70):0.93):0.45):0.40,((databasesystems.3:0.57,databasesystems.2:0.57):0.31,databasesystem
s.1:0.88):1.59):0.21,(((mobilemultimedia.12:0.74,mobilemultimedia.11:0.74):0.12,mobilemultimedia.10:0.85):0.5
2,(adhoc.10:0.81,adhoc.9:0.81):0.56):1.31);
```

Figure 2: Snippet of a generated Newick Tree

B. Taxonomy Evolution

New documents when added in a dataset for which taxonomy is being maintained, they followed the same process of taxonomy generation as mentioned in the previous subsection. Once taxonomy is generated, a new tree structure T_{evo} is constructed that represents the newly introduced documents. A similarity matrix S_{evo} is also produced. Finally for the taxonomy evolve step, Tree Merge [40] technique is used. The Tree Merge practice takes following as input: the existing T_{gen} , new taxonomy T_evo, the existing similarity matrix S_{gen} and the new similarity matrix S_{evoas} input. After the input has been taken, the next step is the building of a compatibility super tree T_S.

NTMerge algorithm is used for building compatibility super tree [41]. The super tree method constructs trees from smaller trees for overlapping subsets of taxonomies. NJMerge basically runs on an input pair of T_{gen} and $T_{evo,}$ and it also takes similarity matrices S_{gen} and S_{evo} as auxiliary information. As mentioned earlier, in Newick trees numbers are used to represent branch length. NJMerge results the correct neighbors of the tree T_{evo} by comparing and analyzing the branch length of tree structure T_{evo} withT_{gen}. Branch lengths sum achieves for all the branches of both tree structures. The pair having smallest length is called a true neighbor. Once the true neighbors have been identified, the next step is the merging of the two trees. Strict Consensus Merger is used for merging pair of trees in which a merged tree. The proposed technique successfully generates a taxonomy for text documents from a given corpora. The technique also successfully evolves the previously created taxonomy in a very short time. The foundation of the algorithm is a MapReduce in which the capability to minimize the time by parallel data processing and facilitates the fault tolerance feature by using distributed file system. MapReduce environment aids for improving the scalability challenges of an existing taxonomy generation and taxonomy evolution techniques. The algorithm runs on Apache Spark environment. The comparison between our proposed technique and existing taxonomy generation and evaluation with respect to running time and clustering quality has been done. The next section discusses the evaluation of the proposed technique.

5. Evaluation

The technique presented in this research work was evaluated on a textual dataset based upon quality and time parameters. Various experiments were executed using the following experimental configurations:

1. Processor: Intel Core i5
2. RAM: 32 GB
3. Apache Spark version: 2.3
4. Apache Hadoop version: 2.10.0

In the first set of experiments, the generation part of the proposed technique was assessed

by comparing it with a current non-incremental taxonomy generation method TaxGen. In the second set of experiments, the entire algorithm (generation as well as evolution) was evaluated by comparing it with the current incremental taxonomy generation methodology TIE. A textual dataset of ACM scholarly articles, taken from [18] was used for performing these experiments. In the third set of experiments, the scalability of the methodology was evaluated using a separate cluster of Apache Spark, on an individual machine. Due to the limited size of the ACM dataset for testing the scalability PubMed dataset comprising of 17785 documents was used. Last but not least, the focus has been made on the comparison of the two parallelization frameworks namely, Apache Hadoop and Apache Spark. The running time of the parallelization part of the proposed technique was compared on both Apache Hadoop and Apache Spark. The rest of this section will discuss evaluation metrics, experiments, and test results.

A. Evaluation Metrics for Clustering Quality

To evaluate the quality of hierarchical clustering, Silhouette's score and Davies-Bouldin's score are being used as quality metrics [cite our previous work].

1. Silhouette's Score: We can compute the Silhouette's score [42] by using the distance called intra cluster distance and mean closest cluster distance for every data point. The range of Silhouette's score is between [-1, +1]. The values near zero may represent an overlapping cluster. The values that are negative may signify that the data point or document has been assigned to wrong cluster. The higher silhouette value shows that the document matched or assigned to its own cluster and inadequately matched to the other nearby clusters.

2. Davies-Bouldin's Score: Davies-Bouldin's [43] score can be computed as by finding the ratio of sum of within-cluster scatter to the between-cluster separation. In a Davies-Bouldin's score, better clustering quality can be achieved by getting lower score. Zero is the minimum score. If two algorithms are being compared the algorithm with lower score will have well-defined and well-separated clusters.

B. Experiments and Results

Our obtained taxonomy consists of two different kinds of evaluation. The first type is called time-based, whereas the second type is quality-based. We obtained efficiency of time by running time of algorithms for taxonomy generation and evolution. To evaluate hierarchical clustering quality, Silhouette's score and Davies-Bouldin's score are being used. Our evaluation results are given below:

1. Experiments for Generation Process: We compared the generation part of our

technique with an existing taxonomy generation technique, TaxGen [23], by comparing the running time and clustering quality. Initially, 220 documents were involved for taxonomy generation process. Newer documents were then added to evaluate the generation process of taxonomy, which shows better results for the proposed technique [37]. Figure 3(a) & 3(b) shows the hierarchical clustering quality of generated taxonomies, whereas the running time is shown in Figure 4.

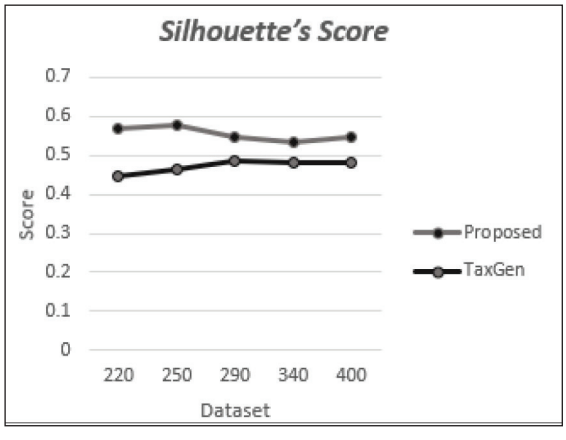

Figure 3(a): Results for Quality-Based Evaluation - Taxonomy Generation Process

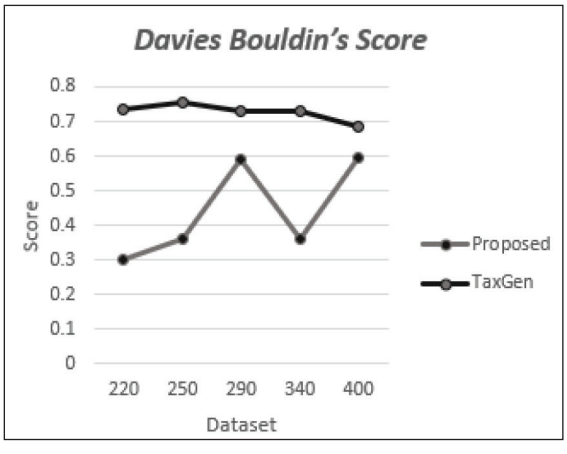

Figure 3(b): Results for Quality-Based Evaluation - Taxonomy Generation Process

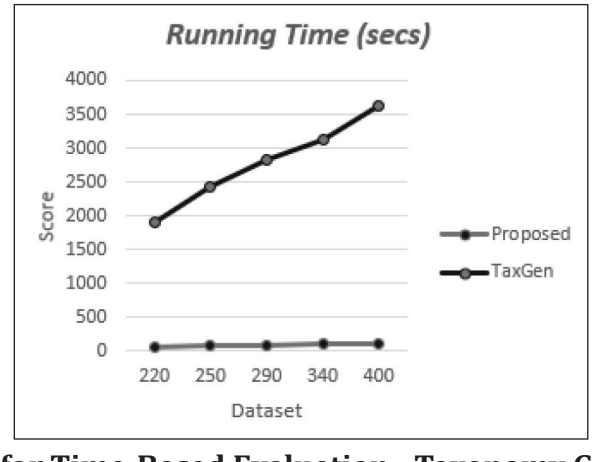

Figure 4: Results for Time-Based Evaluation - Taxonomy Generation Process

2. Experiments for Evolution Process: We compared the evolution part of our proposed method with the evolution of existing method, TIE [18] by performing the result comparisons of hierarchical clustering quality and running rime. To generate the taxonomy, 200 documents were initially used for the taxonomy evolution process using proposed technique and TIE. Then there was a gradually increase in dataset and taxonomy evolution was done using both the methods. The

hierarchical clustering quality scores are shown in Figure 5(a) & 5(b), whereas Figure 6 indicates running time results for both the techniques. The results obtained in both the cases favors the proposed technique [37].

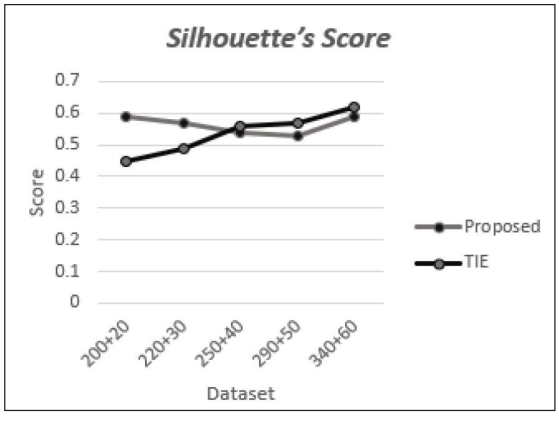

Figure 5(a): Quality Based Evaluation Results-Taxonomy Evolution Process

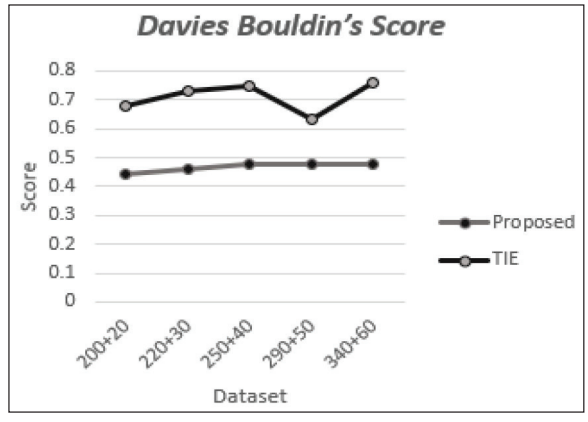

Figure 5(b): Quality Based Evaluation Results-Taxonomy Evolution Process

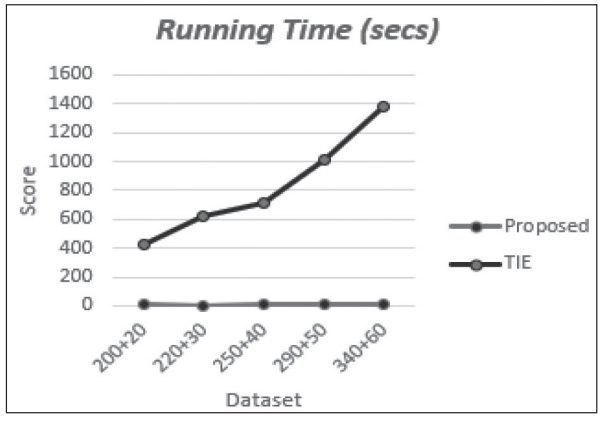

Figure 6: Time-Based Evaluation Results- Taxonomy Evolution Process

3. Experiments for Testing the Scalability: The scalability of the proposed model was tested using a separate cluster of Apache Spark on an individual machine. Due to the limited size of the ACM scholarly articles dataset obtained from [18], for testing the scalability, a dataset namely PubMed was used. This dataset is comprised of 17785 documents. Using 5000 text documents initial taxonomy was generated and after that by adding 5000 documents dataset was gradually increased for the procedure of evolution. Clustering quality for generation and evolution through the proposed methodology was evaluated and the running time was assessed as well. The results of the experiment are shown in Figures 7(a), 7(b) & 7(c), which again show the better cluster.

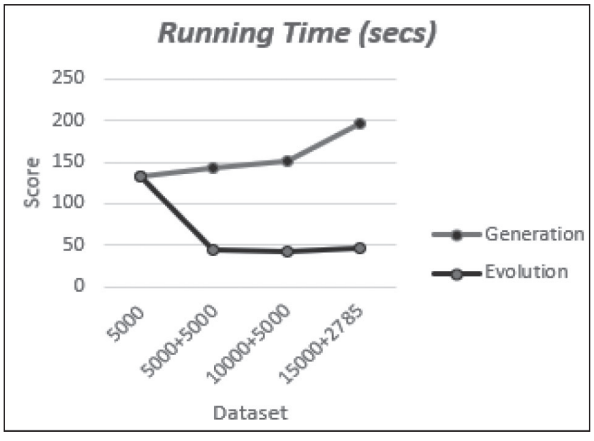

Figure 7(a): Scores showing the Scalability of the Proposed Technique

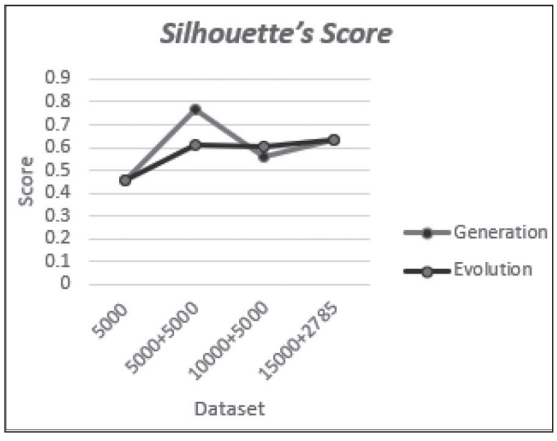

Figure 7(b): Scores showing the Scalability of the Proposed Technique

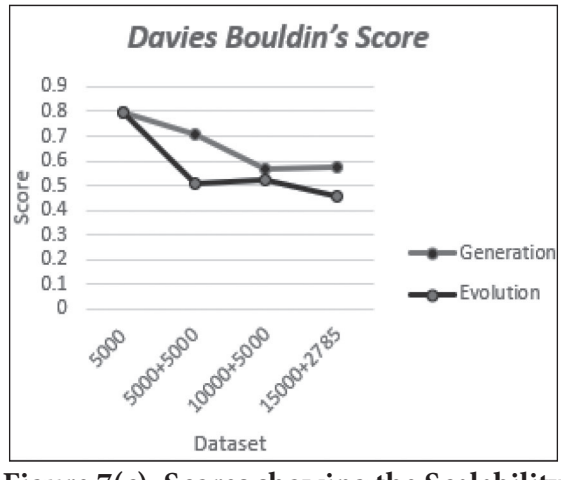

Figure 7(c): Scores showing the Scalability of the Proposed Technique

4. Experiments on Parallelization Framework: In this part of experiment, focus has been made on the comparison of the two parallelization frameworks namely, Apache Hadoop and Apache Spark. Running time of the parallelization part of the proposed technique was compared on both; Apache Hadoop as well as Apache Spark. For the sake of this experiment, we only generated taxonomy, Newick trees were not generated for this experiment. The algorithm was run on 3 cores of Apache Spark and compared it to the processing capability of Apache Hadoop. 3 cores were chosen because that is the minimum number of cores that can be chosen for running of any Spark job. Table 2 shows the running time of the proposed technique on both the environments. It can be observed that the running time of Apache Spark is much smaller as compared with Apache Hadoop.

Table 2: Apache Spark vs. Apache Hadoop Based on the Running Time of the Proposed Technique

Dataset	Running Time (secs)	
	Apache Spark	Apache Hadoop
200	7.97	241.8
300	6.29	369.6
400	8.16	571.8
500	10.11	616.2
600	12.28	673.8
700	14.23	842.4
800	16.68	963
900	18.57	1083.6
1000	19.48	1228.8
1500	42.96	1830.6
2000	45.55	2310

On the basis of running time of the technique on both the environments, their running time ratio, i.e. ⟦RT⟧_r was calculated. The sum of the running time of the algorithm was considered for Apache Spark and Apache Hadoop both and their ratio is calculated as given in (11). According to this time ratio, Spark is 53.05 times faster than that of Hadoop.

$$RT_r = \frac{\sum Running\ Time\ of\ Algorithm\ on\ Hadoop}{\sum Running\ Time\ of\ Algorithm\ on\ Spark} = 53.05 \quad (11)$$

Spark performance, has found out to be optimal over Hadoop as evaluated by processing speed due to following reasons:

1. Spark performs computation using in-memory calculation. It runs a selected part of a MapReduce task and is not bound by input-output concerns every time.

2. Spark's directed acyclic graphs support optimization between steps, whereas any cyclic interconnection between MapReduce steps and levels is not possessed by Hadoop. This means performance tuning cannot be done at that level.

Hence both the theoretical aspects as discussed above and in Section 3.4, as well as the experimental results favor Apache Spark for the case of the proposed methodology.

Apache Spark basically comes with various units and sub-units that aid in the process

of running of Spark jobs. Tuning the resources, parallelism, and using different data representation affect Spark job performance. Schema of data (the way data is arranged) and number of cores for running a job are important factors. The *--executor-cores* specifies the number of cores when submitting a Spark job. A Spark job is submitted by invoking *spark-submit*. In pyspark *--executor-cores* flag are set from the command line. This can also be achieved by using the *spark-defaults.conf* file or a *SparkConf* object and setting the *spark.executor.cores* property. Further experiments were performed to evaluate running time of the evolution process on different number of Apache Spark cores. In the proposed technique running time of taxonomy evolution process against different number of Apache Spark cores is shown in Table 3.

Table 3: Running Time for Taxonomy Evolution on Different No. of Apache Spark Cores

Size of data	Running Time (secs)		
	3 cores	5 cores	8 cores
100+100	7.97	6.79	6.29
200+100	6.29	7.12	6.25
300+100	8.15	8.29	8.08
400+100	10.11	9.72	9.83
500+100	12.28	12.09	11.99
600+100	14.23	14.06	13.68
700+100	16.68	15.98	15.44
800+100	18.57	18.07	17.69
900+100	19.48	19.87	18.46
1100+100	29.73	24.56	23.34
1200+100	34.97	26.78	27.12
1300+100	39.14	30.89	29.87
1400+100	42.96	32.02	30.38
1500+100	44.67	35.44	33.43
1600+100	45.23	37.89	36.32
1700+100	45.76	39.34	38.56
1800+100	46.26	41.56	40.53
1900+100	47.55	43.05	42.67

In Table 3, it can be seen that when taxonomy evolution process was run on different number of Spark cores, the running time of algorithm for 8 cores gives the minimum time. When we specify the number of cores to be 8 that means each executor runs 8 tasks at a given time. It should be noted here that initially when dataset is small the time taken by all three cores to evolve taxonomy is comparable but as the dataset increases the significant difference can be seen in the running time. The impact of using different cores

can be better visualized when dataset is even larger.

C. Discussion

In this section, the proposed methodology is assessed and evaluated by comparing it with an existing algorithm of taxonomy generation i.e., TaxGen. Evolution part of the technique has also been compared with another algorithm of incremental taxonomy generation i.e., TIE. The technique has been assessed and evaluated on the basis of the running time and quality of clustering. Clustering quality was evaluated using two different techniques i.e., 1) Silhouette's score 2) Davies Bouldin's score. It was observed that the proposed technique shows better clustering quality when compared with TaxGen and TIE.
It is evident that the running time of the proposed methodology is significantly smaller as compared to its counter parts. Due to the usage of map reduce framework, the asymptotic complexity of the proposed technique is also reduced from O(n^3) for hierarchical clustering to $O\left(\frac{n^3}{k}\right)$, where k is the number of nodes in which task is divided in map reduce setup.

The technique was also evaluated by running it on Apache Spark and Apache Hadoop both and their running time was compared. It was found that Apache Spark generated taxonomy in much smaller time as compared with Apache Hadoop. So, it can be said that for a clustering problem like taxonomy generation Apache Spark is a better choice. It was also evaluated by experiment that by using how many cores of Apache Spark the proposed technique can evolve taxonomy faster. It was found that when data size is small number of cores do not matter. As the size of data grows using 8 cores can bring significant time improvement. The execution time taken for an analysis to perform is critical in big data applications. The execution time is measured to evaluate the performance. Smaller execution times indicate that the program runs fast and gives good performance. It should also be noted that the proper resource utilization is also crucial in case of large datasets. A good application should give high performance with minimal resource utilization. Since the technique utilizes MapReduce algorithm as its core technique while running on Apache Spark, this makes the technique scalable. The next chapter concludes this research work.

5. Conclusion and Future Work

This research work has reviewed the existing techniques of taxonomy generation and evolution from the perspective of today's data which is particularly fast-evolving and voluminous. It was identified that in the modern era of big data, it is required that there must be some efficient and scalable taxonomy generation and evolution techniques to handle this type of data. Although some work has been done in the field of hierarchical

clustering for substantial datasets, little focus has been made on generating and evolving taxonomy. As per the available information, none of the existing techniques have focused on the idea of parallelization to develop an effective and scalable algorithm for the process of taxonomy generation and evolution. In this research work, a novel and unique technique has been developed for the process of taxonomy generation and evolution which is based upon the MapReduce paradigm using the framework of Apache Spark has the ability to minimize the time by parallel data processing. It also provides the feature of fault tolerance by using the distributed file system (DFS).

The proposed technique is evaluated on the basis of the clustering quality and the time takes to generate and evolve taxonomy in contrast to the present taxonomy generation (TaxGen) and evolution (TIE) methodologies. It is quite clear from the results obtained so far, that the proposed methodology consumes less time for taxonomy generation and evolution. Evaluation based on quality metrics has been done by applying Silhouette's and Davies-Bouldin's scores. When compared with the existing techniques, both the indices verify improved hierarchical clustering for the proposed methodology. Some experiments are also performed for comparing the two parallelization frameworks, namely Apache Hadoop and Apache Spark using 3 cores setting. The running time of the parallelization part of the proposed technique has been compared on both, Apache Hadoop and Apache Spark. Spark's performance is observed to be optimum over Hadoop as measured by processing speed. Furthermore, some specific experiments have been also performed to test the scalability of the suggested technique by using a specifically large dataset. The time and quality-based evaluation have made it clear that the use of the MapReduce environment has improved the scalability issues of current techniques of taxonomy generation and evolution.

There were certain challenges faced during the implementation of algorithms. Initially, we had decided to use Hadoop to perform taxonomy generation and evolution. We faced no issue in performing taxonomy generation on Hadoop but for evolution, we ran into a problem as we are using Newick tree graph technique for the evolution of taxonomy, and Hadoop's scope is limited when it comes to Newick graphs. Hence, we selected Apache spark as it supports map-reduce as well as Newick graph techniques.

This work too is bound to observe some limits. The proposed model is capable of evolving a taxonomy that has been converted into a Tree graph only. Prospectively, we are in the view of working on proposing a more generalized algorithm that can upgrade/evolve any taxonomy being given as an input. The labeling technique and the hierarchical clustering quality of the taxonomy can be further improved. In the future, we also strategize to evaluate our proposed technique using cloud computing to acquire better results in terms of scalability and performance in the spirit of big data.

References

[1] M. Zwolenski, and L. Weatherill, "The Digital Universe: Rich Data and the Increasing Value of the Internet of Things,"Journal of Telecommunications and the Digital Economy, vol. 2, no. 3, pp. 1—47, 2014.

[2] Rajesh Math. "Big Data Analytics: Recent and Emerging Application in Services Industry. Part of the Advances in Intelligent Systems and Computing book series (AISC, volume 654)" SpringerDoi: 978-981- 10-6620-7_21

[3] Coronel, C., Morris, S., Rob, P. (2013). Database Systems: Design, Implementation, and Management, (10th. Ed.). Boston: Cengage Learning.

[4] ImenChebbi, WadiiBoulila, ImedRiadh Farah."Big Data: Concepts, Challenges and Applications"Springer Doi: 978-3-319-24306-1_62

[5] GeorgiosSkourletopoulos, Constandinos X. Mavromoustakis, George Mastorakis, Jordi MongayBatalla, CiprianDobre, Spyros Panagiotakis and EvangelosPallis: Big Data and Cloud Computing."A Survey of the State-of-the-Art and Research Challenges" SpringerDoi: 9783319451435c2

[6] M. S. Paukkeri, A. P. García-Plaza, V. Fresno, R. M. Unanue, and T. Honkela, "Learning a taxonomy from a set of text documents," Applied Soft Computing, vol. 12, no. 3, pp. 1138–1148, 2012.

[7] R. Sujatha, R. Bandaru, and R. Rao, "Taxonomy Construction Techniques–Issues and Challenges," Indian Journal of Computer Science and Engineering, vol. 2, no. 5, pp. 661-671, 2011.

[8] H. Hedden, The Accidental Taxonomist, Information Today, Inc., 2016.

[9] D. Sánchez, and A. Moreno, "Automatic Generation of Taxonomies from the WWW," In International Conference on Practical Aspects of Knowledge Management, pp. 208-219, Vienna, Austria, December 2004.

[10] H. Delgado, Taxonomy Organization of information of Web Content, 2019. https://disenowebakus.net/en/taxonomyinformation-web-content

[11] V. Kashyap, C. Ramakrishnan, C. Thomas, and A. Sheth, "TaxaMiner: an experimentation framework for automated taxonomy bootstrapping," International Journal of Web and Grid Services, vol. 1, no. 2, pp. 240–266, 2005.

[12] D. Boley, "Principal Direction Divisive Partitioning," Data Mining and Knowledge Discovery, vol. 2, no. 4, pp. 325–344, 1998.

[13] L. E. Anke, H. Saggion, and F. Ronzano, " TALN-UPF: Taxonomy Learning Exploiting CRF-based Hypernym Extraction on Encyclopedic Definitions," In Proceedings of the 9th International Workshop on Semantic Evaluation, pp. 949–954, Denver, Colorado, June 2015.

[14] P. Velardi, S. Faralli, and R. Navigli, "Ontolearn Reloaded: A Graphbased Algorithm for Taxonomy Induction," Computational Linguistics, vol. 39, no. 3, pp. 665–707, 2013.

[15] Yao, B. Cui, G. Cong, and Y. Huang, "Evolutionary Taxonomy Construction from Dynamic Tag Space," World Wide Web, vol. 15, no. 5, pp. 581–602, 2012.

[16] L. Tang, H. Liu, J. Zhang, N. Agarwal, and J. J. Salerno, "Topic Taxonomy Adaptation for Group Profiling," ACM Transactions on Knowledge Discovery from Data (TKDD), vol. 1, no. 4, pp. 1–28, 2008.

[17] R. M. Marcacini, and S. O. Rezende, "Incremental Construction of Topic Hierarchies using Hierarchical Term Clustering.," In Software Engineering and Knowledge Engineering (SEKE), pp. 553–558. Redwood City, California, USA, July 2010.

[18] R. Irfan, S. Khan, K. Rajpoot, and A. M. Qamar, "TIE Algorithm: a Layer over Clustering-based Taxonomy Generation for Handling Evolving Data," Frontiers of Information Technology Electronic Engineering (FITEE), vol. 19, no. 6, pp. 763-782, 2018.

[19] X. Wu, X. Zhu, G. Q. Wu, and W. Ding, "Data Mining with Big Data," IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 1, pp. 97–107, 2013.

[20] A. McAfee, E. Brynjolfsson, T. H. Davenport, D. J. Patil, and D. Barton, "Big data: The Management Revolution," Harvard Business Review, vol. 90, no. 10, pp. 60–68, 2012.

[21] M. J. Embrechts, C. J. Gatti, J. Linton, and B. Roysam, "Hierarchical Clustering for Large Data Sets," In Advances in Intelligent Signal Processing and Data Mining, vol 410, pp. 197—233, Springer, 2013.

[22] R. Babbar, I. Partalas, E. Gaussier, M. R. Amini, and C. Amblard, "Learning Taxonomy Adaptation in Large-scale Classification," The Journal of Machine Learning Research, vol. 17, no. 1, pp. 3350–3386, 2016.

[23] A. Muller, J. Dorre, P. Gerstl, and R. Seiffert, "The TaxGen Framework: Automating the Generation of a Taxonomy for a Large Document Collection," In Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences HICSS-32, Hawaii, USA, 1999.

[24] E. A. Dietz, D. Vandic, and F. Frasincar, "Taxolearn: A Semantic Approach to Domain Taxonomy Learning," In 2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, pp. 58-65, Macau, China, 2012.

[25] M. Steinbach, G. Karypis, and V. Kumar, "A Comparison of Document Clustering Techniques," In TextMining Workshop at KDD2000, May 2000.

[26] H. Schütze, C. D. Manning, and P. Raghavan, Introduction to Information Retrieval, Cambridge University Press, 2008.

[27] A. K. Jain, "Data Clustering: 50 years beyond K-means," Pattern Recognition Letters, vol. 31, no. 8, pp. 651–666, 2010.

[28] R. Irfan, S. Khan, M.A. Abbas, and A. A. Shah, "Determining Influential Factors and Challenges in Automatic Taxonomy Generation: A Systematic Literature Review of Techniques 1999-2016," Information Research: An International Electronic Journal, vol. 24, no. 2, 2019.

[29] V. Subramaniyaswamy, V. Vijayakumar, R. Logesh, and V. Indragandhi, "Unstructured Data Analysis on Big Data using MapReduce," Procedia Computer Science, pp. 456–465, 2015

[30] A. S. Shirkhorshidi, S. Aghabozorgi, T. Y. Wah, and T. Herawan, "Big Data Clustering: A Review," In International Conference on Computational Science and its Applications, Springer, 2014, pp. 707– 720.

[31] D. Moulavi, P. A. Jaskowiak, R. J. Campello, A. Zimek, and J. Sander, "Density-based Clustering Validation," In Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 2014.

[32] B. Zerhari, A. A. Lahcen, and S. Mouline, "Big Data Clustering: Algorithms and Challenges," In Proc. of Int. Conf. on Big Data, Cloud and Applications (BDCA'15), Tetuan, Morocco, May, 2015.

[33] J. Dean, and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Communications of the ACM, vol. 51, no. 1, pp. 107-113, 2008.

[34] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," In 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), Incline Village, NV, May, 2010.

[35] J. Lin, "Mapreduce is good enough? if all you have is a hammer, throw away everything that's not a nail!," Big Data, vol. 1, no. 1, pp. 28-37, 2013.

[36] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave et al., "Apache Spark: A Unified Engine for Big Data Processing," Communications of the ACM, pp. 56–65, 2016.

[37] K. Aalijah and R. Irfan, "Scalable Taxonomy Generation and Evolution on Apache Spark," 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), 2020, pp. 634-639, doi: 10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00110.

[38] A. G. Jivani, "A comparative study of stemming algorithms," International Journal of Computer Technology and Applications, vol. 2, no. 6, pp. 1930-1938, 2011.

[39] F. Murtagh, and P. Legendre, " Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion?," Journal of Classification, vol. 31, no. 3, pp. 274-295, 2014.

[40] E. K. Molloy, and T. Warnow, "TreeMerge: A New Method for Improving the Scalability of Species Tree Estimation Methods," Bioinformatics, vol. 35, no. 14, pp. 417-426, 2019.

[41] E. K. . W. T. Molloy, "NJMerge: A Generic Technique for Scaling Phylogeny Estimation Methods and its Application to Species Trees," In RECOMB International conference on Comparative Genomics, Magog-Orford, QC, Canada, 2018.

[42] S. Petrovic, "A Comparison between the Silhouette Index and the Davies-Bouldin Index in Labelling IDs Clusters," In Proceedings of the 11th Nordic Workshop of Secure IT Systems, Linköping, Sweden, October 2006.

[43] J. Xiao, J. Lu, and X. Li, "Davies Bouldin Index based Hierarchical Initialization K-means," Intelligent Data Analysis, vol. 21, no. 6, pp. 1327-1338, 2017.