
Xhafa, F. [et al.] (2016) Apache mahout's k-means vs. fuzzy k-means performance evaluation. *2016 International Conference on Intelligent Networking and Collaborative Systems, IEEE INCoS 2016, 7-9 September 2016, Ostrava, Czech Republic: proceedings*. [S.l.]: IEEE, 2016. Pp. 110-116. Doi: http://dx.doi.org/10.1109/INCoS.2016.103.

Xhafa, F. [et al.] (2016) Apache mahout's k-means vs. fuzzy k-means performance evaluation. *2016 International Conference on Intelligent Networking and Collaborative Systems, IEEE INCoS 2016, 7-9 September 2016, Ostrava, Czech Republic: proceedings*. [S.l.]: IEEE, 2016. Pp. 110-116. Doi: http://dx.doi.org/10.1109/INCoS.2016.103.

Apache Mahout's $k-$Means *vs*. Fuzzy $k-$Means Performance Evaluation

Fatos Xhafa*, Adriana Bogza
Universitat Politècnica de Catalunya
Barcelona, Spain
Email: fatos@cs.upc.edu

Santi Caballé
Universitat Oberta de Catalunya
Barcelona, Spain
EMail: scaballe@uoc.edu

Leonard Barolli
Fukuoka Institute of Technology
Fukuoka, Japan
EMail: barolli@fit.ac.jp

***Abstract*—The emergence of the Big Data as a disruptive technology for next generation of intelligent systems, has brought many issues of how to extract and make use of the knowledge obtained from the data within short times, limited budget and under high rates of data generation. The foremost challenge identified here is the data processing, and especially, mining and analysis for knowledge extraction. As the *old* data mining frameworks were designed without Big Data requirements, a new generation of such frameworks is being developed fully implemented in Cloud platforms. One such frameworks is Apache Mahout aimed to leverage fast processing and analysis of Big Data. The performance of such new data mining frameworks is yet to be evaluated and potential limitations are to be revealed. In this paper we analyse the performance of Apache Mahout using large real data sets from the Twitter stream. We exemplify the analysis for the case of two clustering algorithms, namely, $k-$Means and Fuzzy $k-$Means, using a Hadoop cluster infrastructure for the experimental study.**

Keywords: Data Mining Algorithms, Apache Mahout, Big Data, $k-$Means, Fuzzy $k-$Means, Performance, Hadoop Cluster.

I. Introduction

Big Data is considered as a game changer for many application domains. It is based on the premise that generating Big Data is cheap and feasible for enterprises, institutions and organizations widely. However, data processing and knowledge extractions are computationally challenging, especially for small and medium size enterprises, which in most cases cannot cope with the data storage, processing and mining due to high data generation rates, short time frames and high costs. Indeed, Big Data hinders obstacles, which if not overcome, can impede extracting the knowledge in the data and its use in business processing, decision-making, etc. Obviously, existing data mining techniques and frameworks are the premier source for processing and analysing the data, finding (structured, frequent, approximate, etc.) patterns in data [3], [5], discovering association rules [1], [2], [10], grouping/clustering/bi-clustering data according to one or more criteria, etc. Most such data mining frameworks (DMFs) are moving to Cloud to alleviate the user from the burden of installing, configuring and running such frameworks locally. We can distinguish here Cloud based versions of existing DMFs like Weka and R and new DMFs properly designed for Cloud platforms such as Mahout, in either case running on Hadoop clusters. The later frameworks aim at overcoming limitations and failure of existing DMF frameworks to cope with more demanding requirements of data mining techniques for efficiency and scalability [2], [3], [6], [10], [11].

As the *old* data mining frameworks were designed without Big Data requirements, a new generation of such frameworks is being developed fully implemented in Cloud platforms. One such frameworks is Apache Mahout aimed to leverage fast processing and analysis of Big Data. The performance of such new data mining frameworks is yet to be evaluated and potential limitations are to be revealed. In this paper we analyse the performance of Apache Mahout using large real data sets from the Twitter stream. We exemplify the analysis for the case of two clustering algorithms, namely, $k-$Means and Fuzzy $k-$Means, using a Hadoop cluster infrastructure for the experimental study. The study of such new implementations is interesting not only for processing time and scalability but also for in-memory usage, given that for large data sets it is not possible to load into memory the whole data set.

The rest of the paper is organized as follows. In Section II we briefly describe Apache Mahout data mining framework. The high performance computing needs arising from data pre-processing are presented in Section III. The performance analysis of two clustering algorithms, namely, $k-$Means and Fuzzy $k-$Means using real big data sets from Twitter stream is addressed in Section IV. We end the paper in Section V with some conclusions and remarks for future work.

II. Apache Mahout and Clustering Algorithms

The Apache Mahout project is an open source project under the Apache umbrella, which provides a framework for building scalable algorithms and also offers built-in algorithms that can be run on top of Hadoop MapReduce as well as on top of Apache Spark, H2O or Flink. The main focus in this paper is on using the MapReduce algorithms that are implemented in Mahout, which can be run in-memory, but for large datasets they need to be executed in

*SmartLearn Group, Universitat Oberta de Catalunya, Spain

a Hadoop environment.

In latest releases of Apache Mahout there is a clear shift in focus from Hadoop MapReduce implementations to more comprehensive platforms. From version 0.12.0 the MapReduce clustering algorithms became deprecated and the project seems to be oriented towards Apache Flink, which is a streaming dataflow engine that provides data distribution, communication and fault tolerance for distributed computations over data streams. Apache Flink project integrates stream processing and batch processing, which makes it an important choice for both batch and stream data mining. Evaluating the performance of Mahout's implementation over the MapReduce framework can provide valuable insights for the great number of projects that already have the necessary infrastructure in place, given the popularity of Hadoop MapReduce.

Apache Mahout provides multiple types of algorithms: recommendations, clustering, classifications and others. The algorithms that are of interest for the purpose of this study are the clustering ones, namely, $k-$Means and Fuzzy $k-$Means. We briefly describe them next.

A. Mahout $k-$Means

The $k-$Means algorithm is one of the most commonly used clustering algorithms because of its simplicity. Most implementations of $k-$Means take as input the following:

- A set of points that are to be distributed into clusters.
- A set of initial centroids for the clusters or expected number of clusters (depending on the implementation).
- A distance measure method.
- A maximum number of iterations to be performed.
- A convergence *delta* parameter, which is an indicator that the clusters have been identified and no more iterations are needed.

The algorithm consists of two steps that are executed multiple times, until the clusters have converged (based on the convergence delta parameter) or until the maximum number of iterations has been reached. It starts by using the initial centroids received as input or randomly chooses n centroids. The two steps that are performed repeatedly are:

1) Assign all points to the cluster with the nearest centroid.
2) Recompute the centroid for each cluster.

The complexity of this algorithm is $O(nktd)$, where n is the number of items in the dataset, k is the number of clusters, d is the number of dimensions for each item and t the number of iterations.

The main advantage for using $k-$Means is that it is easy to understand and implement. It also leads to accurate results when the data points can be grouped into well separated clusters. A disadvantage would be that if the clusters cannot be well defined over the dataset, then the results might not be very accurate, since it allows a point to be part of only one cluster. Another disadvantage is that it requires previous knowledge about the number of clusters that are to be identified, which on large datasets is difficult to estimate and usually these are identified via other faster algorithms (like Canopy). $k-$Means is also unable to identify noisy data or outliers, precisely because it is based on a static number of clusters.

B. Mahout Fuzzy $k-$Means

The Fuzzy $k-$Means algorithm is an enhancement of $k-$Means. The main difference is that it allows a data point to be part of multiple clusters. It assigns to each point probability values for being part of every cluster and then the centroids of the clusters are computed based on the positions and the probabilities of each point. Assuming that each point is defined as a vector of coordinates $v_i = (a_1; a_2; a_3; \ldots; a_n)$ and that the probability values for that point belonging to each cluster are expressed in a matrix U where u_{ij} is the probability of the point v_i to belong to cluster c_j, the centroid of each cluster is computed based on the following formulae:

$$c_j = \frac{\sum_{i=1}^{n} u_{ij}^m \cdot v_i}{\sum_{i=1}^{m} u_{ij}}$$

where m represents the accepted level of fuzziness, $m > 1$, and n represents the number of points in the cluster.

After computing new centroids for each cluster, the probabilities matrix is recomputed based on the new centroids.

$$u_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{|x_i - c_j|}{|x_i - c_k|} \right)^{\frac{2}{m-1}}}$$

where c represents the total number of clusters.

C. Centroid Generation

The $k-$Means and Fuzzy $k-$Means algorithms take as input an initial set of centroids for the clusters. These can be computed using another clustering algorithm implemented in Mahout, namely Canopy [7]. This is often used as a pre-processing step for $k-$Means algorithms on large datasets. Datasets can be considered large based on several features: large number of entries in the data, large number of dimensions of the data and large number of clusters that can be derived. The canopy algorithm is proven to reduce the clustering computation time in each of these cases by an order of magnitude with no impact on the accuracy of the results. In our experiments, the dataset, namely the data sets of tweets from Twitter stream (see Sect. IV) can be considered to be large from all these three viewpoints, since there is a large number of tweets retrieved, the number of dimensions is considered infinite since we are dealing with text documents and there is a large number of clusters that can be derived since there are many topics that are being discussed on the Twitter platform.

The Canopy algorithm can use an approximate distance measure method for quickly distributing data across approximate canopies. A canopy is a collection of items that are relatively similar to one another. It is important to note that one item from the dataset can be part of multiple generated canopies. There are two threshold values used within this algorithm, T_1 and T_2, where $T_1 > T_2$. If the distance between a canopy (which can be viewed as the center of a cluster) and an uncategorized point is smaller than T_1, it is probably that point is part of the canopy, but it might be part of others too. If the same distance is smaller than T_2, then the point is definitely part of the canopy and there is no need to try to place it in other canopies too. The recommendation is to use an approximate and cheap distance measure to evaluate if the points are at distance smaller than T_1 from the canopy and then use a more advanced one to evaluate if that distance is actually smaller than T_2. By doing this, most of the distance computations will be done using the cheaper distance measure and then the accuracy of the results can be improved by using a more advanced distance measure for the second part.

There are multiple distance measures that could be used, including Euclidean distance, Squared Euclidean distance, Manhattan distance and Cosine distance, among others. As these are standard similarity measures in the literature, we have omitted their definitions here.

III. Pre-processing

Data preprocessing is a necessary step for all data mining methods and it consists of multiple actions of data cleaning and structuring/formatting in order to prepare it for applying the actual algorithms. The $k-$Means and Fuzzy $k-$Means algorithms implemented in Mahout take as input a list of *TF-IDF* vectors and vectorizing the text content is a preprocessing step for the data mining algorithms. We recall here that TF-IDF stands for Term Frequency – Inverse Document Frequency, and is defined as a numerical statistics that is intended to reflect how important a word is to a document in a collection. It is often used as a weighting factor in information retrieval and text mining. The TF-IDF value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

In our case the data set entries are tweets –we refer here to as document indistinctly–therefore, in order to convert the raw text content to TF-IDF vectors, which can be considered overall as a data transformation step, there are multiple actions that need to be performed. The first one is generating the dataset dictionary, with the Document Processor implemented in Mahout and using the Lucene Standard Analyzer. A Lucene Analyzer is used in order to extract indexable tokens (or words) from texts. There are multiple analyzers provided by the Lucene library: Standard, Whitespace, Stop or Snowball. The Lucene Standard Analyzer is capable of handling names, email addresses, special characters like punctuation marks. It also contains a default list of stopwords and eliminates them if encountered in the document that is being analyzed.

The Document Processor from Mahout launches MapReduce jobs over Hadoop. A document tokenization job involves only map actions and at the end it generates the tokenized content (each tweet is associated with a set of tokens or words which are part of that tweet).

After the tokenized documents are obtained, the term-frequency (*TF*) vectors need to be generated. Term-frequency vectors can be viewed as a map in which the key is a word found in the document and the value is the number of occurrences of that word in the document. The DictionaryVectorizer from the Mahout library takes as arguments the maximum size of *ngrams*, which in our case will be one, since we want to take into consideration independent words only.

The Document Frequency Convertor from Mahout also launches MapReduce jobs over Hadoop and generates the dictionary of the dataset and the Document-Frequency (*DF*) Vector. The document frequency vector contains for each word the number of documents in which the word is present. This is useful in order to select the words that appear in a larger number of documents and to assign them a smaller weight when it comes to clustering, because their meaning doesn't bring many insights to the topic of each document. The number of times a word appears in a document is not taken into consideration when computing the document frequency value. After the *DF* value is computed for each word, the Inverse Document Frequency (*IDF*) value is generated, according to the following formula:

$$IDF = \log\left(\frac{N}{DF}\right)$$

where N represents the number of documents in the collection and DF represents the Document-Frequency value previously described.

The multiplication with N is used for normalizing the values. This IDF value is used in order to assign smaller weights to more frequent words across the collection. It will not influence however the frequent words inside the same document.

The $TF - IDF$ processor simply computes the $TF - IDF$ values starting with the *TF* and *IDF* values, based on the following formula:

$$TD - IDF = TF * IDF = TF * \log\left(\frac{N}{DF}\right)$$

The preprocessing steps discussed herewith can be observed in the Fig. 1. On top of the sequence of steps there is the Apache Yahoo! S4 [8] (denoted *S4 processor* in the

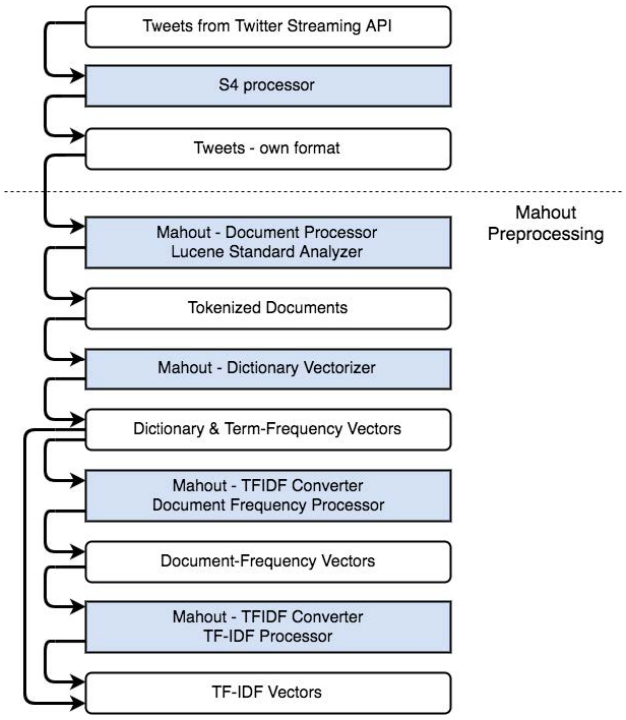


Figure 1: Data pre-processing steps.

figure), which is the stream processing used for building the dataset of tweets via the Twitter Streaming API.

IV. Performance evaluation

A. The HPC Infrastructure

We used RDLab as distributed infrastructure[1] for processing and analyzing the data. The RDLab infrastructure aggregates hardware resources for research and project development:

- Over 160 physical servers.
- Over 1000 CPU cores and more than 3 TBytes of RAM memory.
- Over 130 TBytes of disk space.
- High speed network at 10Gbit.

The RDLab High Performance Cluster (HPC) offers several software packages such as Lustre High Performance Parallel file system, Hadoop support, SMP and MPI parallel computation, etc. We have used up to 32 nodes in the cluster during the experimental study.

B. Performance evaluation criteria

Several tests were conducted in order to extract quantifiable metrics. For the performance evaluation for the Mahout library, we have considered three main indicators:

- Processing time
- In-memory usage

Other questions aimed to answer regarding the performance of Mahout throughout these experiments are if Mahout scales well with large datasets and if the required processing time increases linearly with the data set size.

Throughout the experiments, we were able to evaluate several steps from the process of clustering data, namely:

1) Data preprocessing –converting raw text to TF-IDF vectors;
2) Centroid generation – using the Canopy algorithm in order to generate some centroids from the data set to be used by other algorithms, and
3) $k-$Means and Fuzzy $k-$Means data mining clustering algorithms, which were considered suitable for the evaluation due to unstructured data format.

C. Data sets, structure and size

The dataset is constructed based on tweets received via the Twitter Streaming API, which is further transformed to suitable formatting in order to be used later on by the Mahout data mining algorithms.

Data structure: The tweets received via the Twitter Streaming API contain, besides the text content, a lot of meta-data that can provide additional information about the popularity of the tweet or the context in which it was published. Examples of meta-data for a tweet are, among others:

- language – described in a BCP 47 format or equal to “und" if the language could not be detected.
- coordinates – the geo-location from which the tweet was published.
- creation date.
- entities – special entities, which are extracted from the tweet content: urls, hashtags, user mentions.
- re-tweeted – true/false, indicates if the tweet has been re-tweeted or not.
- re-tweet counter – the number of times this tweet has been re-tweeted.
- user – the profile of the author of the tweet, which contains: id, creation time for the user account, description.
- followers counter – indicates the number of followers the user has.
- friends counter – indicates the number of friends the user has (which is equivalent to the number of accounts the user is following).
- profile image.
- status – the most recent tweet that the user has published.
- statuses count – the total number of tweets that the user has published over time.

Data size: Given the length limitation of a tweet content, the size of one data entry is very small. Tweets

[1] http://rdlab.cs.upc.edu/index.php/en/

contain UTF-8 characters which can be represented on 32 bits (i.e. 4 bytes). A maximum length of 140 characters means a maximum size of 560 bytes. A tweet id can be represented as a long number, so it requires up to 8 bytes. Based on this values, the maximum memory space size required for storing a tweet is around 568 bytes.

Taking into consideration the clean up operations performed over the text content of a tweet before storing it, the size of a data entry is most likely smaller than this amount. So we decided to compute the average size of an actual tweet based on the disk space size the dataset occupies and the total number of stored tweets:

$$avg\ tweet\ size = \frac{used\ disk\ space}{total\ number\ of\ stored\ tweets}$$

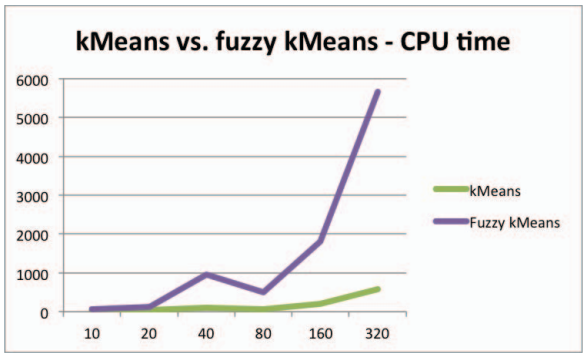
Using the above formulae, the average memory size required for storing a single data entry resulted to be around 91 bytes, which is around 6 times smaller than the worst case scenario we assumed initially and one of the reasons that explains this is the fact that the urls and user mentions in the tweet content can take more than 50% of the entire text.

D. Computational results

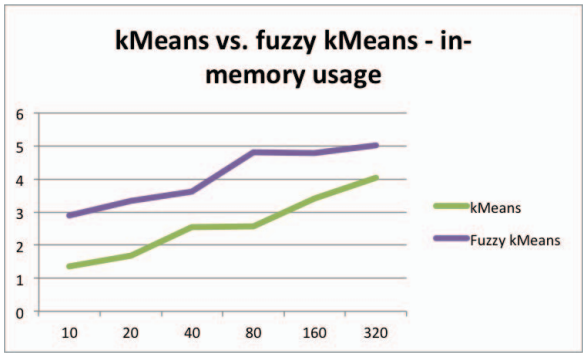
We have summarized some computational results about CPU processing time in Fig. 2 when using four computing nodes. In the figure, the x-axis represents the thousands of entries in the dataset, while the y-axis represents the processing time expressed in seconds. Regarding the results, we can observe that the pre-processing time is doubled when the data size is doubled, while for the computation of centroids and the finalization of the algorithms $k-$Means and Fuzzy $k-$Means the processing time grows exponentially with the data size.

We have summarized some computational results about in-memory usage in Fig. 3 when using four computing nodes. In the figure, the x-axis represents the thousands of entries in the dataset, while the y-axis represents in-memory usage expressed in GigaBytes. Regarding the results, we can observe that for the pre-processing in-memory usage is doubled when the data size is doubled, while for the computation of centroids and the finalization of the algorithms $k-$Means and Fuzzy $k-$Means the in-memory usage grows linearly with the data size.

We also compared the performance of $k-$Means and Fuzzy $k-$Means algorithms and observed that the $k-$Means algorithm has a much faster execution time than Fuzzy $k-$Means, whose execution time grows exponentially as the number of tweets in the data set increases. Similarly, $k-$Means algorithm uses less virtual memory than Fuzzy $k-$Means, however, in both cases the in-memory usage is linear with increase of data size (see Fig. 4 for these differences).



(a) Processing time



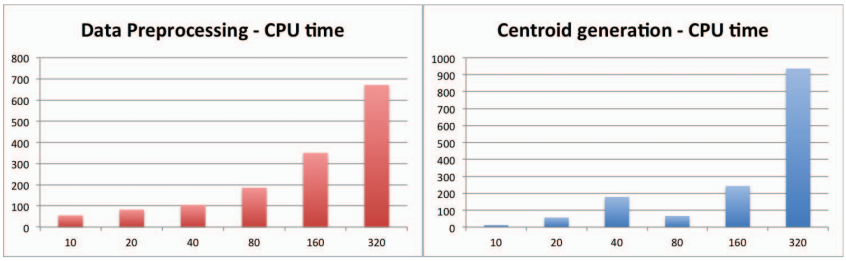
(b) In-memory usage

Figure 4: $k-$Means *vs.* Fuzzy $k-$Means processing time and in-memory usage when using 4 nodes and various data set sizes.

V. Conclusions and Future Work

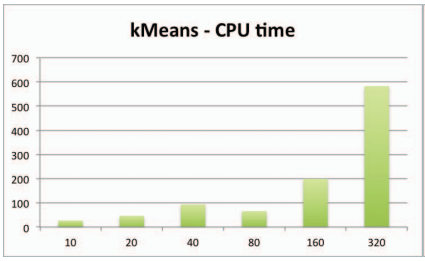
In this paper we have presented a study on the performance evaluation of some clustering algorithms implemented in Apache Mahout data mining framework. With the ever increasing amounts of data, most of *old* data mining algorithms and frameworks, such as WEKA, are showing limitations to cope with such data set sizes increase. As an alternative, new implementations, such as Mahout, fully implemented for Cloud deployment and executions, aim to deal efficiently with Big Data. Yet, the achieved performance of such new implementations is to be drawn from studies on various data sets. In our study we have analysed the performance of $k-$Means and Fuzzy $k-$Means algorithms using data sets from Tweeter stream and executed in a Hadoop cluster. The study showed that the processing time of the algorithms grows fast, especially for the Fuzzy $k-$Means algorithms, whose execution time grows exponentially. On the positive size, the in-memory usage for both algorithms grows linearly with increase in data sets size.

In our future work we would like to further extend this study to other algorithms of Mahout framework as well

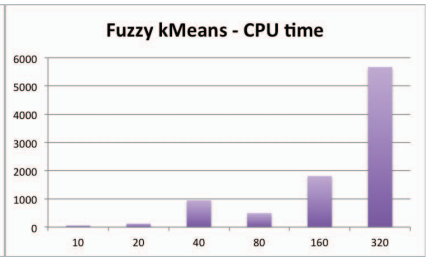


(a) Data Preprocessing

(b) Centroid generation

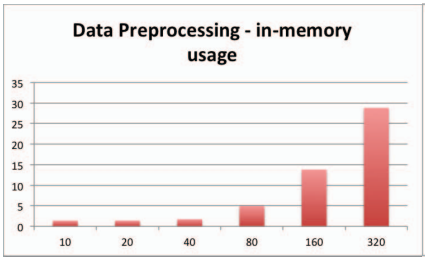


(c) kMeans

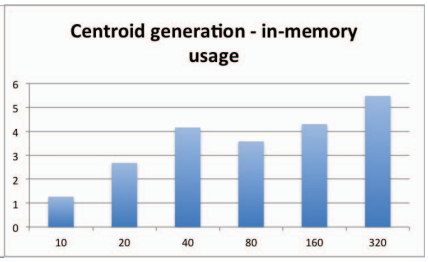


(d) fuzzy kMeans

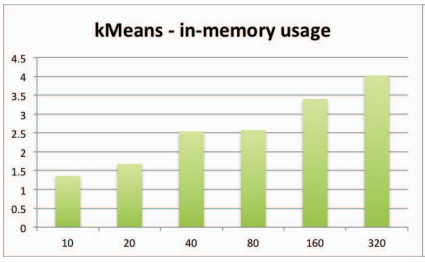
Figure 2: Performance processing metrics when using 4 nodes and various data set sizes.



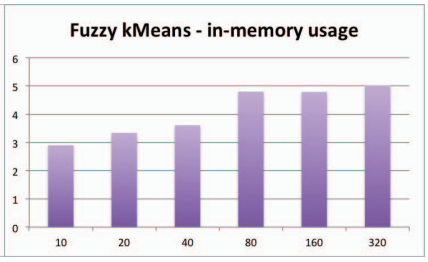
(a) Data Preprocessing



(b) Centroid generation



(c) kMeans



(d) fuzzy kMeans

Figure 3: In-memory usage metrics when using 4 nodes and various data set sizes.

as evaluating the accuracy of the algorithms on large data sets. Likewise, we would like to consider other data sets from other application domains such as Virtual Campus or autonomic computing systems [4], [9], [12].

REFERENCES

[1] Agrawal, R., Imielinski, T. and Swami, A. Mining association rules between sets of items in large databases. In *Proceedings 1993 ACM-SIGMOD International Conference on Management of Data (SIGMOD'93)*, Washington, DC, pp. 207-216, 1993.

[2] Agrawal, R. and Srikant, R. Fast algorithms for mining association rules. In *Proceedings of 1994 International Conference on Very Large Data Bases (VLDB'94)*, Santiago, Chile, pp. 487-499, 1994.

[3] Bayardo, R.J. Efficiently mining long patterns from databases. In Proceedings of *ACM-SIGMOD International Conference*

Management of Data (SIGMOD'98), Seattle, WA, pp. 85-93, 1998.

[4] Caballé, S. and Xhafa, F. Distributed-based massive processing of activity logs for efficient user modelling in a Virtual Campus. *Cluster Computing* 16(4): 829-844 (2013)

[5] Han, J., Pei, J., Yin, Y. and Mao, R. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. *Data Min. Knowl. Discov.* 8, 1 (2004), 53-87.

[6] Kolici, V., Xhafa, F., Barolli, L., Lala, A. Scalability, Memory Issues and Challenges in Mining Large Data Sets. In *Proceedings of 6th International Conference on Intelligent Networking and Collaborative Systems (NCoS 2014)*, Italy, September 10 - 12, 2014: 268-273, IEEE CPS

[7] McCallum, A., Nigam, K. and Ungar L.H. Efficient Clustering of High Dimensional Data Sets with Application to Reference Matching, *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 169-178, 2000. doi:10.1145/347090.347123

[8] Neumeyer, L., Robbins, B., Nair, A. and A. Kesari. S4: Distributed Stream Computing Platform. I n *2010 IEEE Data Mining Workshops (ICDMW)*, Sydney, Australia, Dec. 2010, pp. 170 –177.

[9] Salfner, F., Tschirpke, S. and Malek, M. Comprehensive Logfiles for Autonomic Systems. *18th International Parallel and Distributed Processing Symposium (IPDPS'04)* - Workshop 11, 2004

[10] Savasere, A., Omiecinski, E., and Navathe, S. 1995. An efficient algorithm for mining association rules in large databases. In *Proceedings of International Conference on Very Large Data Bases (VLDB'95)*, Zurich, Switzerland, pp. 432-443, 1995.

[11] Silverstein, C., Brin, S., Motwani, R. and Ullman, J. Scalable techniques for mining causal structures. In *Proceedings of International Conference on Very Large Data Bases (VLDB'98)*, New York, NY, pp. 594-605, 1998.

[12] Xhafa, F., Lopez Martinez, A., Caballé, S., Kolici, V. and Barolli, L. Mining Navigation Patterns in a Virtual Campus. In *Proceedings of the 3rd International Conference on Emerging Intelligent Data and Web Technologies (EIDWT 2012)*, September 19 - 21, University Politehnica of Bucharest, Bucharest, Romania: 181-189, IEEE CPS, 2012.