
Xhafa, F.; Bogza, A.; Caballé, S. (2017) Performance evaluation of Mahout clustering algorithms using a twitter streaming dataset. *31st IEEE International Conference on Advanced Information Networking and Applications, IEEE AINA 2017, Taipei, Taiwan, March 27-29, 2017: proceedings.* [S.l.]: IEEE, 2017. Pp. 1019-1026 Doi: http://dx.doi.org/10.1109/AINA.2017.50.

Xhafa, F.; Bogza, A.; Caballé, S. (2017) Performance evaluation of Mahout clustering algorithms using a twitter streaming dataset. *31st IEEE International Conference on Advanced Information Networking and Applications, IEEE AINA 2017, Taipei, Taiwan, March 27-29, 2017: proceedings.* [S.l.]: IEEE, 2017. Pp. 1019-1026 Doi: http://dx.doi.org/10.1109/AINA.2017.50.

Performance Evaluation of Mahout Clustering Algorithms Using a Twitter Streaming Dataset

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

***Abstract*—Big Data has become commonplace in most Internet-based applications, which by delivering services to planetary scale numbers of users generate very large data sets. Such data sets are considered as a valuable source of analytics information and knowledge for many purposes and domains. It is claimed each time more that Big Data and machine learning, especially data mining, are the basis for developing advanced analytics platforms for turning data into valuable assets, gaining competitive advantage and make better decisions. At the same time, however, Big Data applications are showing to be *killer* applications for the state of the art machine learning and data mining algorithms. Indeed, traditional data mining frameworks such as WEKA, R, etc. and those from big companies such as IBM SPSS Modeler, SAS Enterprise Miner, Oracle Data Mining, etc. are facing the challenges of 1) coping with mining large data sets within short times and 2) under high rates of data generation. The way envisaged ahead to effectively deal with such challenges is to move to Cloud-based versions of such frameworks and development of new frameworks implemented using Cloud platforms. In either case, data mining and machine learning algorithms are being fully implemented in Cloud platforms under new requirements of Big Data for efficiency and performance. In the group of newly developed frameworks there is Apache Mahout, whose goal is *"to build an environment for quickly creating scalable performant machine learning applications"*. In this paper we analyse the performance of some clustering algorithms of Apache Mahout using a Twitter streaming dataset under a Hadoop MapReduce cluster infrastructure according to various evaluation criteria.**

Keywords: Big Data, Machine Learning, Data Mining, Apache Mahout, Performance, Hadoop Cluster.

I. Introduction

As Internet-based applications have embraced all application domains they generate all kinds of data, at very large quantities and at very high rates, known as Big Data and Big Data Streams. Traditional data analysis based on querying and reporting cannot discover the insights in such very large and often continuous data sets. Therefore, Big Data calls for more advanced methods and algorithms, which give quantitative information from data, and, on turn, when combined appropriately from various data sources yields to relevant information, knowledge and intelligence (in a variety of analytics such as business analytics, social analytics, environmental analytics, etc.). Such advanced methods are designed from a combination of a plethora of algorithms and techniques developed so far by different research and developing communities including statistics, data mining and machine learning, simulation and optimization [1], [2], [6], [10], [19]. Additionally, such advanced methods (broadly referred to as *"advanced analytics"*) need to be implemented on large scale platforms such as Cloud-computing ones, so that requirements for *unlimited* storage capacity, memory and high performance processing can be satisfied.

Big Data applications are showing to be *killer* applications for the state of the art machine learning, data mining algorithms and combinations of methods. The reason being that such new methods have to meet a variety of requirements:

- coping with mining large volumes of data, which in many case are of unprecedented scale (for instance data from billion node social networks, data streaming, etc.) [2], [6], [13], [19], [21].
- dealing with data processing under high rates of data generation [26], i.e., not only to be able to effectively and efficiently process and analyse data at present/short term by also to keep the efficiency according to high data generation at mid and long term.
- processing within short times so that important decisions can be taken in real or almost real time as could be needed by application domain (e.g. healthcare [22], businesses [23] to gain competitive advantage. etc.)
- cost-effective processing, i.e., while processing and analysing Big Data can lead to turning data into valuable assets, this should be kept at affordable costs for organisations, institution, enterprises, etc. [12].

The way envisaged ahead to effectively deal with such challenges is, on the one hand, to move to Cloud-based versions of existing frameworks (WEKA, R, etc. and those from big companies such as IBM SPSS Modeler, SAS Enterprise Miner, Oracle Data Mining, etc., which already count on with Cloud-based versions), and, on the other, to develop new frameworks fully implemented using Cloud platforms under new requirements of Big Data for efficiency

and performance. In the group of newly developed frameworks there is Apache Mahout [4], whose goal is *"to build an environment for quickly creating scalable performant machine learning applications*".

In this paper we analyse the performance of some clustering algorithms of Apache Mahout using a Twitter streaming dataset under a Hadoop MapReduce cluster infrastructure according to efficiency of processing, scalability, memory usage, etc. To that end, we design and implement a persistent data layer based on Yahoo! S4 stream processing engine [17] and Twitter Stream API. The persistence layer is in charge of gathering tweets in order to be processed later on in batches by the Mahout data mining algorithms. To enable processing by such algorithms, pre-processing, formatting etc. are accordingly done.

The rest of the paper is organized as follows. In Section II we overview some basic concepts from Hadoop and MapReduce, including some examples of applications. The conceptual model, requirements and architecture are presented in Section III. Section IV presents the Twitter stream processing using Yahoo!S4 and Section V describes the data layer of the system. The performance analysis is given in Section VI and some conclusions and future work in Section VII.

II. Hadoop MapReduce for Big Data processing

The need to handle, store and process large dataset is quickly expanding to many sectors, not just to IT-related fields. For example, the amount of patient data gathered over the years can lead in discovering better treatments in the health sector, but this amount of data is overwhelming to be processed by human power only. Hadoop is a commonly used framework for data mining and Big Data processing. There are many different fields in which the usage of Hadoop brings a considerable contribution in extracting relevant information from large datasets.

A. Hadoop and MapReduce

Hadoop is an open-source Apache software framework used for distributed processing of large datasets across large clusters. It is used for extensive data analytics and high performance computing. Hadoop at its core is made up of a distributed file system (Hadoop Distributed File System - HDFS), a processing framework called MapReduce and a job scheduling and resource manager framework, Hadoop YARN.

The Hadoop Distributed File System is designed to run over commodity hardware and ensures high fault tolerance and high throughput access at lowcost. Its architecture is based on the master-slave model. The master server is called the *NameNode*, while the slaves are *DataNodes*. The *NameNode* is the central machine of the cluster and it contains the file system metadata. Each *DataNode* manages the data stored on them and the computations over that data are made locally, in order to avoid data movement across the network. Usually files are divided in chunks of 64 to 512 MB (configurable) and the chunks are replicated in order to avoid data loss caused by system or network failures.

MapReduce is a programming model that it designed to be executed in parallel on large clusters over large datasets. The model is made up from two main functions: map and reduce. The map step can be seen as a preprocessing data step, while the reduce step can be associated with the actual computation and aggregation of results. Multiple map or reduce jobs are launched in parallel across multiple nodes in the cluster and each node receives a chunk of the input data to process, usually the one that is actually stored on that node. MapReduce follows the master-slave architecture also. The master node in this context is called the *ResourceManager* and it manages the existing jobs. Once a job is submitted, it is divided into tasks and the *ResourceManager* decides where to run each task and ensures continuous communication with each *NodeManager*. A *NodeManager* is associated with the slave and monitors the execution of the tasks on that node (there can be multiple tasks per node) and sends continuous feedback to the *ResourceManager*.

Hadoop was designed to be fault tolerant, which means that even if some of the nodes fail, there should be no data loss. This is possible through data replication across *DataNodes* and through task, job and nodes management. If a task fails, the *NodeManager* detects the failure, sends a message to the *ResourceManager*, which later reschedules the task. If an entire *DataNode* fails, the *NameNode* and the *ResourceManager* detect the failure, all the tasks that were running on that node are rescheduled and the data is already replicated on other nodes. The *NameNode* and the *ResourceManager* are single point of failures in this architecture, if the master fails all the cluster becomes unavailable, even if the slaves are running.

There are multiple file systems that Hadoop can integrate with. These do not necessarily replace HDFS, but can be a source of data for Hadoop or a destination for Hadoop MapReduce jobs for example. One of them is FTPFS (File Transfer Protocol File Systems) which is a file system that supports access to a FTP server through standard file system APIs. Another example would be Amazon S3 (Amazon Simple Storage Server), which is mostly targeted for Hadoop clusters who are kept on Amazon EC2 (Amazon Elastic Compute Cloud) infrastructure.

Hadoop was designed to be deployed inside a cluster. Such clusters can be physically located in an on-site datacenter, but can also be deployed in a cloud infrastructure. There are multiple cloud vendors who offer the possibility of deploying a Hadoop cluster without having to acquire any hardware or needing specific setup expertise, like Amazon, Microsoft and Google.

B. Application examples

1) Log data processing: The processes that govern the world are getting more computerized every day, requiring tracking of the users' actions. There are many software applications that replaced human actions. Among many other advantages, like speed of completing a task or easiness of interaction, software application offer the possibility of tracking the actions completed through logs [18]. The problem that arises is that there is a lot of log data generated and it is difficult to extract relevant information from it. Hadoop can be used for large log data processing [8], [24], [25].

2) Healthcare and Bioinformatics: It is estimated [11] that the volume of medical data available worldwide provided by PACS (Picture Archiving and Communication Systems) vendors is around 150 Exabytes and is increasing at an approximate rate of x1.8 each year. PACS are only one of the sources of medical data, that contains different types of scans as images, such as ultrasounds, magnetic resonance (MR), computed tomography (CT), endoscopy and others. Besides PACS there are a lot more medical data sources, like patient medical history or treatments evaluation [15], [16] and therefore large scale processing is sought [7]. Given this high data volume, several experiments that used the Hadoop framework in order to extract relevant medical data have been conducted so far. One example would be the usage of the MapReduce algorithms to identify unproven cancer treatments on the health web, a study of great importance regarding to dealing with the dissemination of false and dangerous information to vulnerable health consumers [3]. Another example would be DNA sequencing [20]. CloudBurst is an algorithm for these types of studies that makes use of the MapReduce framework in order to parallelise computation. The experiment conducted proved that the time it takes to make the computation scales linearly with the number of nodes inside the Hadoop cluster. DNA fragment assembly algorithms have also been implemented using MapReduce [27].

3) Text Mining: The discovery of recurrent phrases in documents is an interesting research area in text mining and can be used for document summarization, clustering or topic search in a larger dataset. An experiment was conducted in [5] in order to use the MapReduce framework for developing an algorithm to discover such recurrent phrases. The MapReduce solution proved to scale well for this experiment and to be fault tolerant, but the challenge was the maintenance of a large distributed table in HBase that needs to be frequently read by map jobs. The results proved that the MapReduce solution decreased the application runtime up to six times, than a naive distributed implementation over HBase.

Text mining can be defined as a knowledge-intensive process in which a user interacts with a document collection over time by using a suite of analysis tools and seeks to extract useful information from data sources through the identification and exploration of interesting patterns [9]. In the case of text mining, the data sources are document collections and patterns are found in unstructured text data. For this reason, the preprocessing step is essential in the text mining process, because it transforms the unstructured text data into a more explicitly structured format.

The concepts the text mining operates with are document collection, document, character, word, term and concept. A document collection is a grouping of text-based documents. These documents can be grouped by any criteria and usually text mining techniques aim to discover patterns across such collections. A document collection can be either static, if the set of documents doesn't change over time, or dynamic, if documents can be added or updated frequently in that collection.

In our case, the document collection would be the collection of tweets we gather via the Yahoo!S4 application and it can be both static or dynamic, depending of the type of algorithm we are running against it. If we were to use regular clustering algorithms, the document collection would have to be static, that means that the stream of tweets received via Yahoo!S4 has to be processed and the tweets stored in some persistent manner in order to be able to retrieve them as a whole to be processed via Mahout algorithms. There are also streaming algorithms that can run on continuous streams of data and in that case the document collection would be dynamic, since new documents can be received constantly. In the second case, the Yahoo!S4 application would probably have to do some pre-processing on the data and then redirect the tweets received to the algorithm that processes them in order to extract valuable information. A document is the element that has to be processed by the text mining algorithms. It is an ordered collection of words that are usually constructed by a defined grammar and that make sense together. In our case, a tweet could be considered a document, since it is an entry inside the document collection which has to be processed to add value to the result.

A *character* is the basic element from a document and can be a letter, a number, a special symbol or a white-space. One or multiple characters can form words. Words are the element that provides meaning to letters grouped together and delimited by other types of characters. In order to facilitate the document processing step, a document could be represented in a more structured manner as a set of all the words in the document for example. It is important to note however that it is recommended to optimize the set of words generated for that document in order to ignore stop words (common words that bring no value to the meaning of the document), symbolic characters and numerics.

A *term* can be a single word or a multi-word phrase that has a specific meaning within the collection of documents in which is encountered.

Concepts are "features generated for a document by means

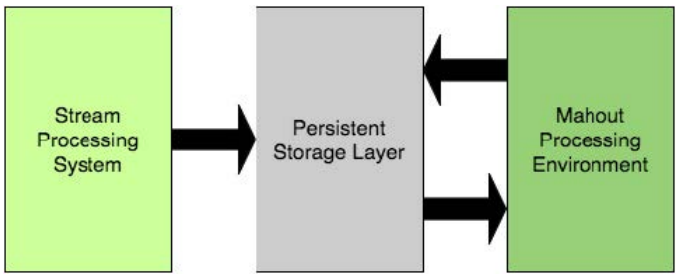

Figure 1: Conceptual model.

of manual, statistical, rule-based, or hybrid categorization methodologies" [9]. It is not unusual that the concepts describing a particular document collection are not actually frequent words in that collection.

III. Conceptual model, requirements and architecture

The system is made of three basic conceptual entities (see Fig. 1):

- The stream processing system that listens to the Twitter streaming API and processes the events received
- The persistent storage layer that gathers the tweets
- The processing environment inside which the Mahout algorithms are to be evaluated

The stream processing system: it has to be able to listen to the Twitter Streaming API and process the events that are received. Processing these events implies executing some data clean-up operations over the text content of a tweet and write them to the persistent storage layer. It needs to be able to handle the rate of the events as sent by Twitter via the Streaming API. In our case, the stream processing system will be deployed inside a cluster with multiple nodes. One node will receive the Twitter events and several others will process them.

The persistence storage layer: it has to store the tweets in a format that is easily readble using the Mahout framework. Since we are handling large datasets, the persistence storage layer should also be scalable and distributed. For the clustering algorithms we will evaluate, the output of the preprocessing steps should be the TF-IDF vectors in a SequenceFile format stored in HDFS (see later).

The processing environment: it should be scalable in order to be able to handle larger batches of data. It should also ensure quick access to the persistent storage layer, since the data mining algorithms can be considered I/O intensive. All data must be passed through at least once and intermediary results need to be stored sometimes directly on disk which can lead to a considerable number of read/write operations. Since the system needs to handle large datasets, computation could be divided into independent chunks to be executed in parallel and then the partial results could be merged into a final one. This means that we could use multiple workers to execute the Mahout algorithms on chunks of data.

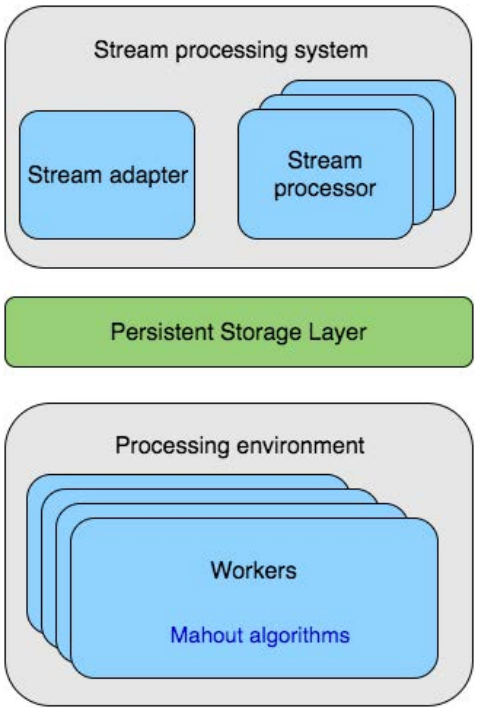

Figure 2: Enhanced conceptual model.

Based on the above requirements, the conceptual can be enhanced further (see Fig. 2).

Architecture: For the stream processing system, the Apache Yahoo! S4 was chosen. This is a distributed stream processing solution, which uses Apache Zookeeper for clusters management. The architecture of the S4 environment is based on the actor model, where there are several nodes with different responsibilities that communicate with each other via messages, or in our case events. Inside the S4 environment there are two types of nodes, which are deployed inside two different clusters. The first one receives events from the Twitter Streaming API and converts them to S4 events that are to be used internally. The second one gathers the S4 events and processes them and then stores them to the persistence storage layer.

For processing the twitter stream we used three logical nodes for the S4 environment and deployed all of them on the same physical node. One node acts as a twitter-adapter and two others as twitter-processors. We discovered that a rate of approximately 900 tweets to process per second (as observed through our experiments), having two twitter-processor nodes is enough to handle the load.

For the processing environment, we used the Hadoop environment. The Hadoop architecture is based on the master-slaves model and used HDFS for the persistent storage layer. In HDFS data is stored into chunks and distributed across the

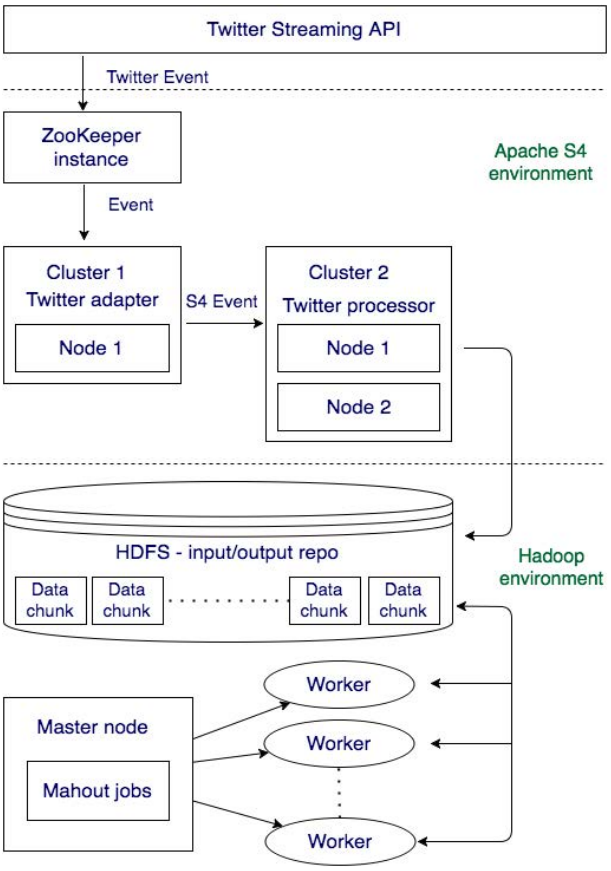

Figure 3: System architecture.

nodes inside the Hadoop cluster. This brings the advantage of data proximity when performing computing operations. Inside our Hadoop cluster, we used a variable number of nodes throughout the experiments we conducted. An overall architecture of the system is presented in Fig. 3.

IV. Processing the Twitter stream with Yahoo!S4

There are two application which are deployed inside two different clusters: twitter-adapter and twitter-processor. The overall model of the Yahoo!S4 system for processing Twitter stream can be seen in Fig. 4.

A. Twitter-adapter

This is the adapter application, used to convert the Twitter stream into a stream of S4 events. Filtering the tweets that are received by the twitter-adapter module by the language in which they are written could be possible, via the Twitter Stream API, since the request can receive filtering parameters for language and other attributes, but in practice it doesn't seem to work as expected. In our attempt of filtering the Twitter Stream using the *language=en* parameter using the HTTP POST API, the stream continued receiving tweets in Japanese or other non-English languages. So we decided

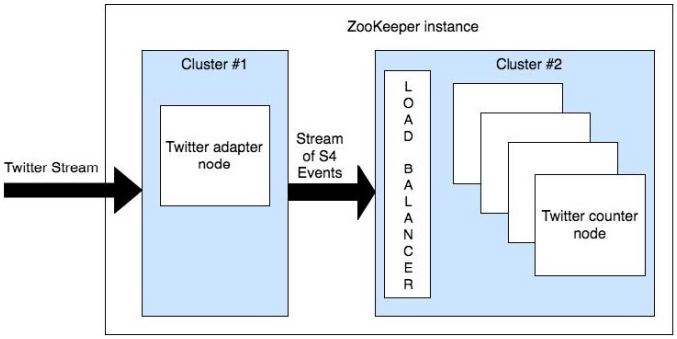

Figure 4: S4 model.

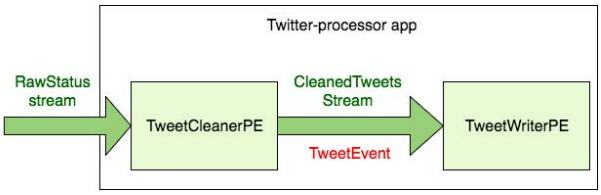

Figure 5: Data flow in the Twitter Processor S4 module.

to receive all tweets in all languages and do a filtering before sending the tweets to the twitter-processor application, using the language attribute of the Tweet. The twitter-adapter application communicates with the twitter-processor one via a remote data stream. The process is pretty straight-forward, when an application inside the ZooKeeper cluster creates a new output stream, it is exposed in ZooKeeper and other applications that define input streams with the same name are automatically connected.

B. Twitter-processor

The twitter-processor application receives the Tweets as S4 events and uses two Processing Elements in order to forward data to the Hadoop cluster: TweetCleanerPE and TweetWriterPE (the data flow through the twitter-processor application can be seen in Fig. 5).

V. Data sets and persistence layer

The dataset is obtained from tweets received *via* the Twitter Streaming API, which is further transformed into 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.

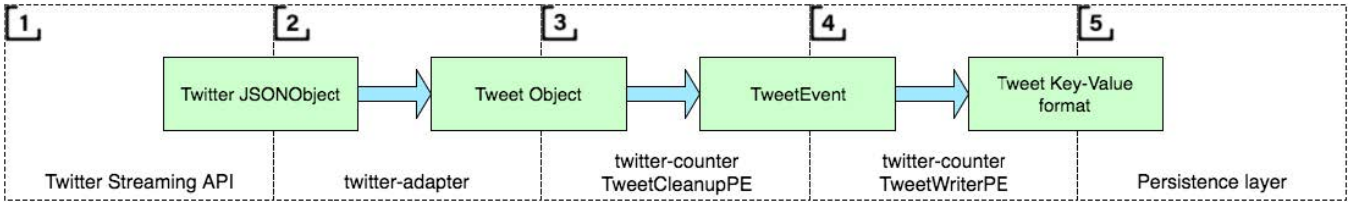

Figure 6: Sequence of changes applied to tweet structure.

- 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.

In the twitter-processor module there are two processing elements, as discussed above. In the TweetCleanerPE there are some data cleaning operations executed over the text content and then the tweet id and the cleaned tweet content data is sent further on to the TweetWriterPE. The other metadata information that was extracted from the original tweet content in the tweet objects we previously described reaches the twitter-processor module, but currently it is not persisted anywhere.

Given the simplified data structure we have obtained, the information could be stored in a key-value format, where the key is the tweet id and the value is the tweet text content. Data changes to tweet structure throughout several steps before being permanently stored in the persistence layer can be seen in Fig. 6.

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

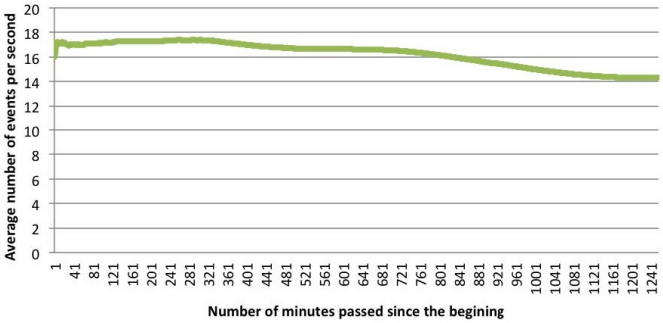

Figure 7: Tweet events rate over 20 hours window.

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

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.

Data reception rate: There may be multiple events coming from via the Twitter Streaming API to the twitter-adapter module, but only tweets written in English are passed further on to the twitter-processor module. The average number of events per second is 16.21 (see Fig. 6 for a graphical representation).

VI. Performance evaluation

A. 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

Table I Cluster processors

Processor model	Number of cores	Frequency
Intel(R) Xeon(R) CPU X3230	4	2.66GHz
Intel(R) Xeon(R) CPU 3070	2	2.66 GHz
Intel(R) Xeon(R) CPU X5550	4 (8 threads)	2.66 GHz
Intel(R) Xeon(R) CPU X3363	4	2.83GHz
Intel(R) Xeon(R) CPU X5670	6 (12 threads)	2.93 GHz
Intel(R) Xeon(R) CPU E5450	4	3.00 GHz
Intel(R) Xeon(R) CPU X3220	4	2.40 GHz
Intel(R) Xeon(R) CPU X3350	4	2.66 GHz
Intel(R) Xeon(R) CPU 5130	2	2.00 GHz
Intel(R) Xeon(R) CPU 5160	2	3.00 GHz
Intel(R) Xeon(R) CPU 5110	2	1.60 GHz

that are implemented in Mahout, which can be run in-memory, but for large datasets they need to be executed in a Hadoop environment. In this study we consider clustering algorithms, namely, $k-$Means and Fuzzy $k-$Means.

B. The HPC Infrastructure

We used RDLab as distributed infrastructure[1] for processing and analysing 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 52 nodes in the cluster during the experimental study (see Table I).

C. Computational results

At first, we run several experiments on 4 nodes in the Hadoop cluster (four LXC containers deployed on the same physical machine, so there was no network latency) to observe the processing time and in-memory usage of the clustering algorithms according to file sizes, starting from a smallest size of 256MB. The processing time and in-memory usage showed to be very large even for such a small data set (see Table II), the reason being that the 256MB file contained about 3 million entries. The clustering algorithms have to iterate through all the points and compute their distance to the centres of the cluster, etc.; the complexity of these algorithms is obviously influenced by the number of entries in the data set.

We summarise here some computational results measuring the scalability of the Mahout library with regard to different

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

Table II Processing time and in-memory usage for 256MB dataset size.

Evaluated steps	Processing time	In-memory usage
Data pre-processing	73 min.	168 GB
Centroid generation	294 min.	6.17 GB
kMeans Algorithm	27.28 hours	2.93 GB

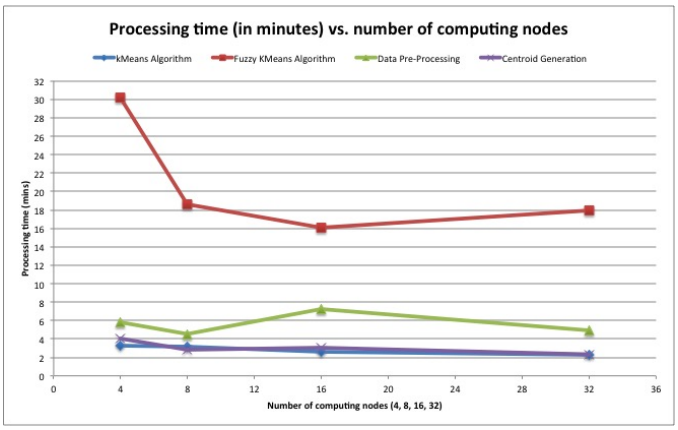

Figure 8: Performance comparison of Data pre-processing, Centroid generation, kMeans and Fuzzy kMeans

numbers of nodes inside the Hadoop cluster. Several independent runs were conducted and processing time for 1) data pre-processing, 2) centroid generation, 3) kMeans algorithm and 4) Fuzzy kMeans algorithm were measured (see Fig. 8). The experiment was conducted for 160k and 320k data set sizes.

As can be seen from Fig. 8, the processing time significantly reduces for both the kMeans the Fuzzy kMeans (about 40% reduction when comparing 4 nodes processing with 32 node processing). Also, it can be observed that the processing time of Fuzzy kMeans is order of magnitudes larger than that of kMeans.

VII. Conclusions and Future Work

Several Cloud-based implementations of machine learning (ML) and data mining (DM) algorithms are emerging after the Big Data. Such implementations aim to overcome limitations of traditional ML and DM frameworks to handle Big Data. Mahout is one such Cloud-based implementation of ML and DM algorithms to efficiently deal with Big Data. Among interesting algorithms there are the clustering algorithms whose performance is affected by the number of entries in the data set. We have presented some performance evaluation results for Mahout cluster algorithms using Twitter Stream data set. We observed that significant reduction in processing time can be achieved for clustering algorithms executed on Hadoop MapReduce.

In our future work we plan to study the performance of some classification algorithms from Mahout library.

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)*, 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)*, pp. 487-499, 1994.

[3] Aphinyanaphongsa, Y., Fua, L.D.and Aliferisa, C. F. *Identifying unproven cancer treatments on the health web: Addressing accuracy, generalizability and scalability*, IMIA and IOS Press, 2013.

[4] Apache Mahout: http://mahout.apache.org

[5] Balkir, A. S., Foster, I. and Rzhetsky, A. A distributed look-up architecture for text mining applications using MapReduce, *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–11, 2011.

[6] Bayardo, R.J. Efficiently mining long patterns from databases. In Proceedings of *ACM-SIGMOD International Conference Management of Data (SIGMOD'98)*, pp. 85-93, 1998.

[7] Boyd, T., Brian Lee, B., Savel, Th., Stinn, J. and Kesarinath, G. An example of the use of Public Health Grid (PHGrid) technology during the 2009 H1N1 influenza pandemic. *International Journal of Grid and Utility Computing*, Vol. 2 No. 2, pp. 148-155, 2011.

[8] 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)

[9] Feldman R. and J. Sanger, J. *The Text Mining Handbook. Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, 2006.

[10] Han, J., Pei, J., Yin, Y. and Mao, R. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. *Data Mining and Knowledge Discovery* vol. 8, no. 1 , 53-87, 2004.

[11] Hudges, G. How big is 'big data' in healthcare? 2011. [Online]. Available: http://blogs.sas.com/content/hls/2011/10/21/how-big-is-big-data-in-healthcare/

[12] Kiran, M., Murphy, P., Monga, I., Dugan, J. and Baveja, S. Lambda architecture for cost-effective batch and speed big data processing. *IEEE International Conference on Big Data (Big Data)*, 2785 - 2792, 2015.

[13] 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: 268-273, IEEE CPS, 2014.

[14] 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.

[15] Philip Moore, Ph., Qassem, T., Xhafa, F. 'NoSQL' and Electronic Patient Record Systems: Opportunities and Challenges. *3PGCIC 2014*: 300-307, IEEE CPS, 2014.

[16] Moore, Ph., Thomas, A.M., Tadros, G., Xhafa, F. and Barolli, L. Detection of the onset of agitation in patients with dementia: real-time monitoring and the application of big-data solutions. *International Journal of Space-Based and Situated Computing (IJSSC)* 3(3): 136-154, Inderscience, 2013.

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

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

[19] 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.

[20] Schatz, M. C. "High performance computing for DNA sequence alignment and assembly," Master's thesis, Faculty of the Graduate School of the University of Maryland, College Park, US, 2010.

[21] 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.

[22] Ta, V.D., Liu, Ch.M., and Nkabinde, G.W. Big data stream computing in healthcare real-time analytics. *IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, pp. 37 - 42, 2016.

[23] van der Schaar, M. Real-time discovery and decision making from big data. *IEEE International Conference on Consumer Electronics*. pp. 1 - 3, 2014.

[24] 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)*, pp. 181-189, IEEE CPS, 2012.

[25] Xhafa, F., Garcia, D., Ramirez, D. and Caballé, S.: Performance Evaluation of a MapReduce Hadoop-Based Implementation for Processing Large Virtual Campus Log Files. *3PGCIC 2015*: 200-206, IEEE CPS, 2015.

[26] Xhafa, F., Naranjo, V. and Caballé, S.: Processing and Analytics of Big Data Streams with Yahoo!S4. *AINA 2015*: 263-270, IEEE CPS, 2015.

[27] Xu, B., Gao, J. and Li, C. An efficient algorithm for DNA fragment assembly in MapReduce, Biochemical and biophysical research communications, vol. 426, no. 3, pp. 395–398, 2012.