Data stream treatment using sliding windows with MapReduce

María José Basgall[1,2], Waldo Hasperué[2], and Marcelo Naiouf[2]

[1] *UNLP, CONICET, III-LIDI, La Plata, Argentina*
[2] *Instituto de Investigación en Informática (III-LIDI), Facultad de Informática - Universidad Nacional de La Plata*
{mjbasgall, whasperue, mnaiouf}@lidi.info.unlp.edu.ar

Abstract

Knowledge Discovery in Databases (KDD) techniques present limitations when the volume of data to process is very large. Any KDD algorithm needs to do several iterations on the complete set of data in order to carry out its work. For continuous data stream processing it is necessary to store part of it in a temporal window.

In this paper, we present a technique that uses the size of the temporal window in a dynamic way, based on the frequency of the data arrival and the response time of the KDD task. The obtained results show that this technique reaches a great size window where each example of the stream is used in more than one iteration of the KDD task.

Keywords: Big Data, MapReduce, Stream Processing.

1 Introduction

Stream processing (from now on *SP*) is an area extensively studied in recent years. SP permits carrying out certain works through the analysis of a continuous and potentially infinite data stream [1][2][3][4][5][6][7][8][9].

The aim of SP is to permit that the tasks analyze the data stream online, producing outputs at appropriate times. The main characteristic of this kind of processing is that the data of the stream arrive so fast that it is not possible to store them all and, if they can be stored, the data volume is so large that it is difficult to analyze it in short response times [10].

In order to carry out the analysis taking into account great volumes of data, it is extremely useful to use multiprocessor technologies, as well as algorithms parallelization techniques that take advantages of these technologies. Currently, it is possible to find several contributions where High Performance Computing (HPC) concepts are used in the stream processing area [1][2][3], at the same time, it is possible to find Cloud Computing papers. The latter introduces a new scope from the high performance computing point of view, since they give a "tailored" support for the use of applications without the need to purchase hardware [11][12][13][14].

Most of the techniques proposed for SP use the stream treatment model where each example is processed only once [1][15][16][17]. Other techniques implement a temporal and sliding window, where they store the last n data received or the most representative n [18][19][20][21][22].

The first approach has a great drawback: if the stream data distribution changes through time, it is very hard to build a model of the data that is able to learn from the new and old characteristics. This does not happen with those models that use a temporal window, since they store part of the stream and they can refeed the data model with the last example collected from the stream plus the n mentioned stored data.

For some problems, the goal is to find or adapt to the changes of the stream data distribution [8][18], therefore, the methods of the simple model of collect-use-discard an example are useful. For other situations, it is very interesting to build a model using all stream data. Like it is usually impossible due to the great volume of information, it is desirable to get the data model using most part of these. In this kind of cases, it is essential to have techniques that allow to use a temporal window as large as possible.

1.1 Data stream mining

Data stream mining area is in charge of processing data stream so as to get useful knowledge, in most of the cases, to make decisions. To get knowledge, it is necessary to apply some technique from the data stream mining area.

The traditional techniques used to get knowledge from a data set show limitations when the amount of data processed is very large, due to the execution time they need to perform the task. Any traditional algorithm, either clustering, classification, regression, learning by neural networks, among others, needs to perform several iterations on the complete data set to carry out its goal. It is impossible to apply the traditional techniques to a data stream since this cannot guarantee a

quick answer where data from a stream is still being collected.

To avoid the problem pointed out in the previous paragraph, there exists two alternatives. The first one consists in using a temporal window, the second one tries to control the amount of iterations of the algorithm that perform the knowledge extraction task. The answer is to find a balance between both alternatives, since by using a temporal window of a large size, the response times will be large, whereas with a small window, the response times will be quicker, but the obtained results will be based on a small amount of the stream. Therefore, it is interesting to have a technique that permits the treatment of the largest possible amount of data and short response times.

A technique that handles the amount of iterations made by the algorithm that performs the tasks, along with the temporal window size as a function of the stream frequency, the amount of stored data for processing, and the time it takes to obtain a partial result with the current data set, keeping the largest possible amount of stream available inside the window, is presented. In this way, the algorithm that performs the knowledge extraction uses each example during a meaningful number of iterations.

In this paper, a clustering technique was used to perform tests on a data stream. The algorithm used was K-means [23], to which the necessary modifications were performed in order to execute it using Hadoop MapReduce (framework for parallel and distributing processing) [24].

Hadoop MapReduce was chosen since it has a great computing capacity, which offers the possibility to use a wide data temporal window. It also facilitates the algorithms parallelization task.

This paper is organized as follows. In section 2 MapReduce paradigm is mentioned, in section 3 the proposed method is presented. In section 4, the experiments performed and the results obtained are shown. Finally, in section 5, conclusions and future work are presented.

2 MapReduce

One of the most used tools for the processing of large amounts of data is Hadoop MapReduce. From a paradigm originally developed by Google [25] and built on well-known principles of parallel and distributed processing. It is a framework for applications that processes large-scale data on clusters of servers commodity, which works on the HDFS file system.

Hadoop HDFS [24] offers efficiency, it is distributed, it gives fault tolerant storage and it is appropriate for applications that use large amount of data, since it gives a high performance access to them. It keeps large files with data access patterns in streaming. Its block size is much larger than the traditional filesystems size; this is so to reduce the number of disk searches.

The way a MapReduce works is based in two phases. The first phase (Map) consists in associating each example read with a key, i.e, it takes a group of input data and turns them into another group of data where the elements are turned into tuples (pair of key-value). When this phase ends, the second phase (Reduce) starts to work, where each reducer process receives all the values that are associated with a common key to, in the end, do the task, which solves specifically the problem and is written by the programmer.

3 Technique for handling the data temporal window

The proposed technique consists in two tasks. The first one is based on the capture of the data stream and its corresponding storage, that is executed continuously. The stream is read online, and the collected data is saved in the buffer. This buffer has no logical limits and it is organized into files which contain an amount of data (b); being b a parameter of the algorithm.

The second task performs the processing of the stream mining algorithm (in this particular case, a clustering technique), being executed in the MapReduce framework, using as input all the files inside the work directory in the HDFS. When the total size of the files is too big, the oldest files are deleted, reducing in this way the data window.

As data are collected from the stream, these are stored in a new file. The problem in accumulating files inside the directory is that, the more data there are, the longer time will need the MapReduce process to treat them.

3.1 Clustering on MapReduce

During the MapReduce phase, the clustering procedure is carried out, which is based on the K-means algorithm. The K number of clusters to find is determined at the beginning of the process and the initial centers are selected randomly.

The MapReduce job starts when the buffer stores a predetermined quantity of data of the stream. In that moment, all the accumulated data are stored into a file inside the work directory in the HDFS and its work starts. The algorithm proposed (algorithm 1) performs an iterative process which consists of stages. In each stage, the group of K centroids is passed to the Map process which looks for the closer c centroid, for each v input vector. As output, it writes tuples (c, v). Figure 1 shows the DAG of the proposed algorithm.

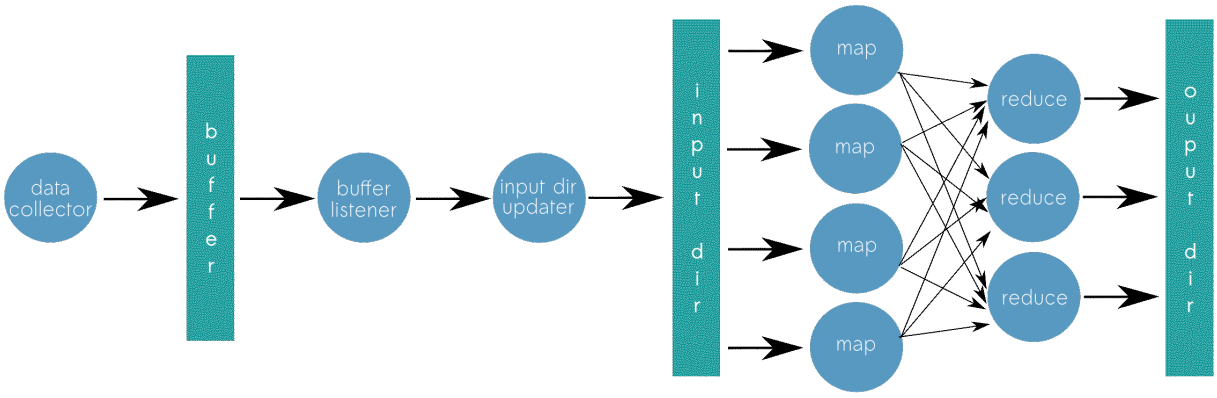

Figure 1: DAG

Algorithm 1 Pseudocode algorithm

{ K_{pre}: *previous centers,* K_{cur}: *current centers,* W_{size}: window size, S_{it}: iterative stage}

while data stream **do**
  update working directory
  { *Begins iterative stage* }
    run MapReduce job
    update K_{pre}
    if K_{cur} do not change significantly from K_{pre} **then**
      activate cut S_{it}
    end if
    update K_{pre} with K_{cur}
    if full buffer **then**
      if not activated cut S_{it} **then**
        decrease W_{size}
        if *round* <= m **then**
          increase W_{size}
        else
          activate cut S_{it}
        end if
      end if
    else
      if activated cut S_{it} **then**
        increase W_{size}
      end if
    end if
  { *Ends iterative stage* }
end while

To take full advantage of the parallel computing, k reducers are created. Each one receives all the corresponding vectors to a same cluster and, with them, calculates the new centroid, that comes from the calculation of the average vector among all the received vectors. Each reducer writes as output the new centroid that was calculated apart from all the vectors that belong to the cluster. This last permits to know how the clusters are made up at the end of a MapReduce phase (this information is not used in this paper, but it will be useful in the future to study the own clustering result).

When the MapReduce job ends, the new C group of centroids is read from the work directory. Then, a new MapReduce job is carried out to refine the clustering procedure and, therefore, make the centroids converge. This iterative process continues until one of the two options described below occur.

The first one is when the job is executed a m maximum number of times, while the second condition happens when the clustering task reaches an equilibrium point and the centroids do not change meaningfully from one iteration to another, which is defined by the t convergence determination threshold.

While the MapReduce job works in the clustering algorithm, the streaming process continues with the data capture by filling a new buffer. In a new file, up to b data from this buffer, are stored. This condition exists in order to prevent files with a great amount of data from being written and to avoid the entire replacement of the window, thus achieving a slight displacement thereof.

During all this process, one of the parameters that is modified in a dynamic way is the size of the window that initially is proportional to the buffer size with a proportionality factor called f. The window increases when the algorithm reaches the convergence and, at the same time, the amount of data in the buffer is less than b. This means that the algorithm achieves to complete the data model and, therefore, the capacity of working with a larger volume of data is now possible. Otherwise, if the number of data in the buffer is larger than b and the algorithm did not get a convergence state, the size of the window is reduced in b, by eliminating the corresponding amount of files from the HDFS. When the amount of data in the buffer are larger than b, it means that the data stream speed increased, therefore, the b parameter increases as well, to make larger the amount of

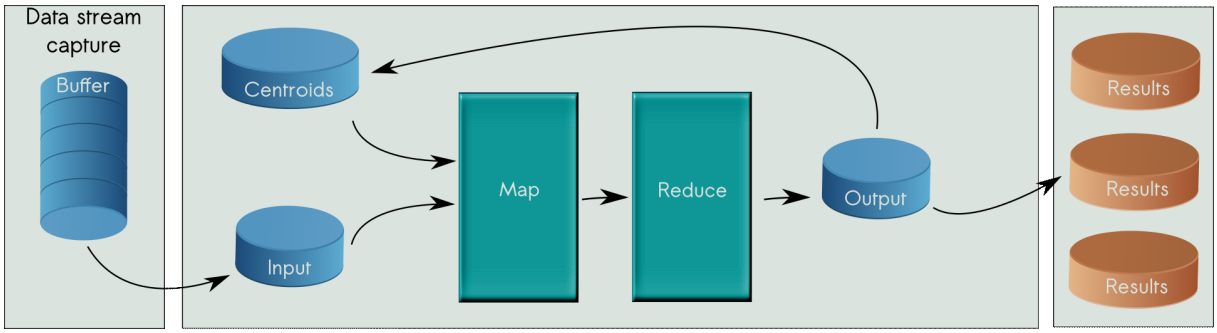

Figure 2: Data stream mining process applying an iterative task using MapReduce

data entered in the window. This implies a more abrupt displacement of the window.

Figure 2 shows the data stream mining process applying an iterative task using MapReduce.

4 Performed tests and results

The tests were performed using a data stream from Twitter which has 830745 tweets. Figure 3 shows the arrival frequency. Since the collected data stream last more that five days, the data stream was transformed reducing the duration between the first and the last collected tweet, changing just the duration of the entire stream, keeping the relative frequency of the arriving tweets. All the test were performed using the same data stream.

The text of the tweets were processed to convert them into binary vectors of 90 dimensions, where each element represents the absence or presence of a particular topic. For all the tests the cosine distance was used as a similarity measure for the *k-means* algorithm execution, since is the most used in this kind of problems [26]. Tests with K from 4 to 11 have been performed. As an arbitrary example, the results for K = 5 have been presented in most cases. Bottlenecks have not been found.

Several tests were performed in which different parameters were modified. They are described below:

- Amount of data into a file (b): determines the data size being written in each file inside the work directory.
- Factor determining the initial window size (f): this factor is linked with b, sets the size with which the data window is initially defined. The window capacity is modified at runtime.
- Buffer size extension factor (e): defines the proportion with respect to the actual size of the buffer in which it will be increased.
- Maximum iterations (m): sets the maximum number of iterations to be carried out at each stage of the process.
- Convergence determination threshold (t): it is used to specify whether the centroids reach convergence from an iteration to another.

From the different values taken by the parameters detailed in table 1, the combination of them (720 in total) that best fits the analyzed data stream is empirically obtained, taking into account the number of interrupted stages and the values given by the renewal coefficient (r). This coefficient determines the proportion of data that are renewed in the window at each stage, taking the value 0 for the case where the data in the window are completely replaced, and value 1 otherwise. See equation 1.

$$r = \frac{tt_{i-1} - rt_i}{tt_i} \quad (1)$$

Where tt_i is the total quantity of tweets in the stage i and rt_i is the number of deleted tweets in the stage i.

Tests show that the best results are obtained from a large window size of 50000 (b x f = 1000 x 50 = 50000).

From all the tests performed, the two best were:

- Case I uses the following parameters: **s** = 1000, f = 50 , e = 0.5, m = 4 and t = 0.01
- Case II has the following configuration: **s** = 1000, f = 50 , e = 0.5, m = 6 and t = 0.01

In figures 4 and 5 the renewal coefficient for each stage of the cases I and II is shown. These charts show that in the first four stages, the algorithm is able to keep into the window more and more data from the previous stage. Then, in stages 5, 6 and 7 the algorithm renews completely the data in the window. Due to the nature of the data source, a high peak of them causes the algorithm to completely renew the data into the window. After that, the algorithm manages to keep the index greater than 0.6 in one case and 0.5 in the other, acquiring some stability in the r value, except in the stage 13 and stage 19, in each case.

For each stage, the window size and the runtime of the MapReduce job in the cases I and II are

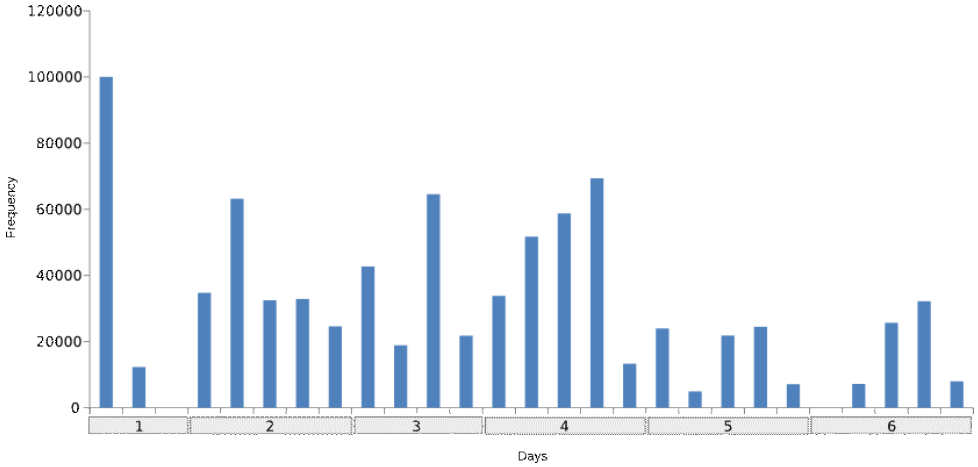

Figure 3: Tweets arrival frequency

Table 1: Parameters used for performing tests

Parameter	**Value**
Amount of data into a file (b)	{1000, 5000, 10000, 50000, 100000}
Factor determining the initial window size (f)	{2, 5, 10, 50}
Buffer size extension factor (e)	{0.1, 0.5, 1}
Maximum iterations (m)	{4, 6, 10, 30}
Convergence determination threshold (t)	{0.00001, 0.0001, 0.01}

described in figures 6 and 7. As to the size of the window is observed that it grows till the end of the experiment. This growth is possible because each stage begins its execution with the centroids of the previous stage. Upon achieving certain equilibrium, each stage requires fewer iterations to keep it, what causes that the following stages have the capacity to work with larger windows. In the same chart the runtime of the MapReduce job is shown, from where it follows that it was less than two minutes in all stages.

For experimentation, a sheet was used from a Blade of 8 sheets, with two 2.0 GHz quad core Intel Xeon e5405 processors, in each of them. Each sheet has 10Gb of RAM (shared between both processors) and L2 cache 2 x 6Mb between pairs of cores.

5 Conclusions and future work

In this paper, a technique for the use of a sliding window of data stream, where the main feature is to maximize the size of such window, allowing each collected example from the stream to be used by the knowledge extraction algorithm as many times as possible, is presented. From the obtained results, it is possible to conclude that the proposed technique can use windows of great size and still fulfilling the aim of the work.

Although the tests were done with a clustering task using a K-means algorithm [23] version in the MapReduce paradigm [24], it is possible to use the proposed technique with any iterative task just by using its corresponding implementation in MapReduce.

As a future work, the continuity of these tests for different data streams is proposed to try getting an intelligent handling of the window size based on changes to the stream frequency, and response time of the task being performed. Another objective for the future is to implement this technique in specific environments for the treatment of streaming such as Spark Streaming [27].

References

[1] N. Takahashi *et al.*, "A parallelized data stream processing system using dynamic time warping distance," in *2009 International Conference on Complex, Intelligent and Software Intensive Systems, Fukuoka, Japan, March 16-19, 2009*, pp. 1100–1105.

[2] Y. Noh *et al.*, "Real-time data stream processing for ubiquitous home network systems," in *4th International Conference on Multime-*

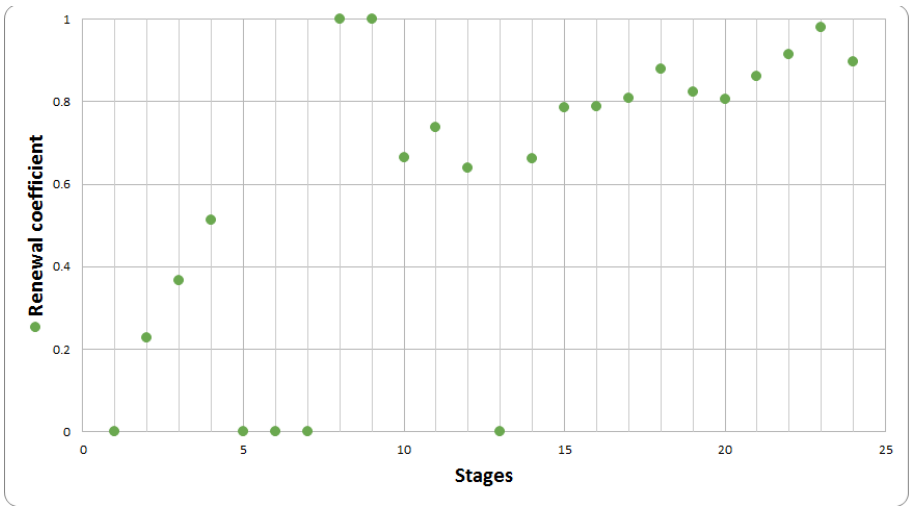

Figure 4: Renewal coefficient for each of the stages. Case I

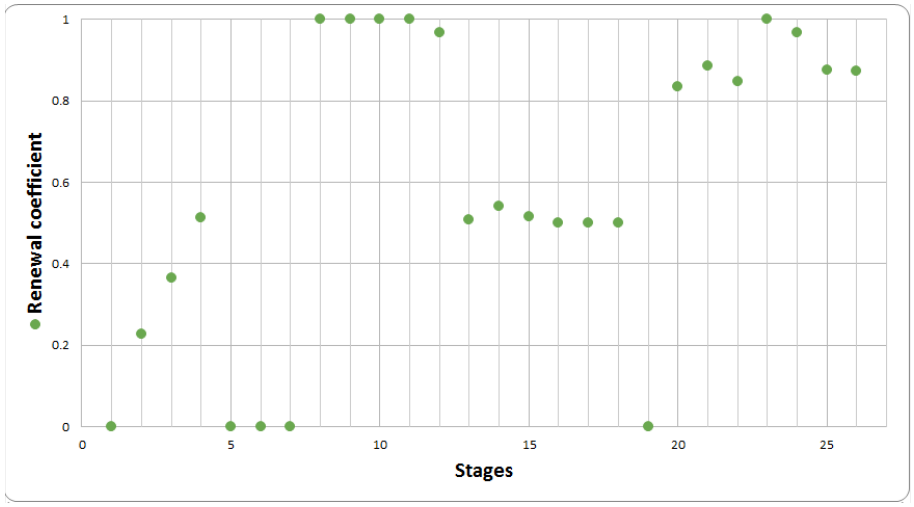

Figure 5: Renewal coefficient for each of the stages. Case II

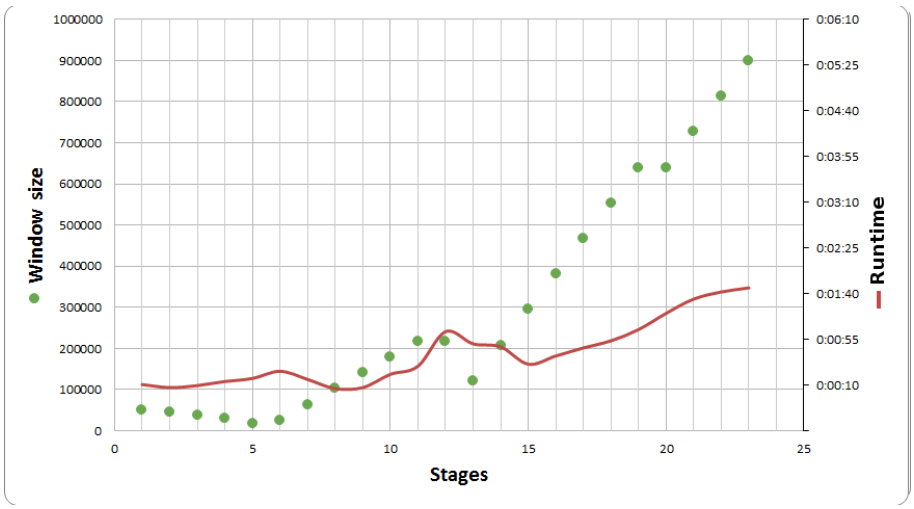

Figure 6: Window size and MapReduce job runtime for each of the stages. Case I

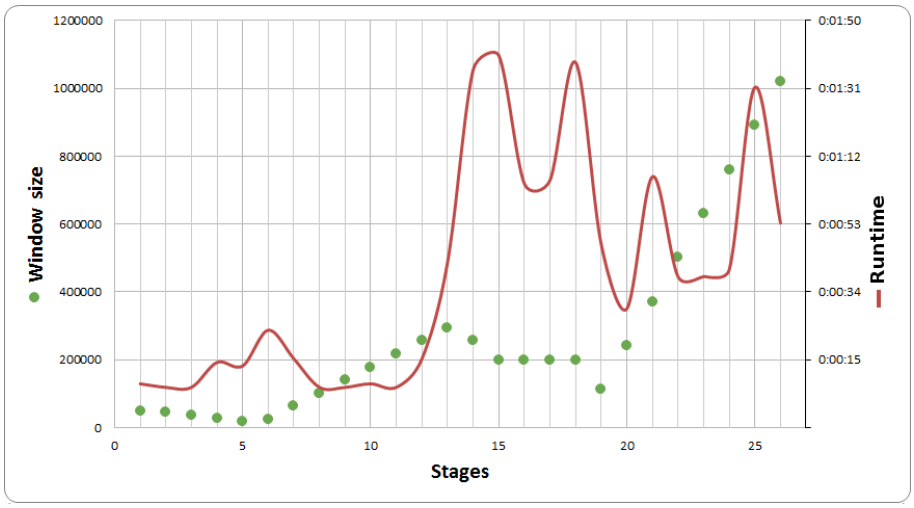

Figure 7: Window size and MapReduce job runtime for each of the stages. Case II

dia and Ubiquitous Engineering, MUE 2010, Cebu, Philippines, 11-13 August, 2010.

[3] C. Kuka, "Processing the uncertainty: Quality-aware data stream processing for dynamic context models," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pp. 560–561, March 2012.

[4] D. Bonino and F. Corno, "spchains: A declarative framework for data stream processing in pervasive applications," *Procedia Computer Science*, vol. 10, 2012.

[5] J. Stefanowski *et al.*, "Processing and mining complex data streams," *Inf. Sci.*, vol. 285, pp. 63–65, 2014.

[6] R. Agerri *et al.*, "Big data for natural language processing: A streaming approach," *Knowledge-Based Systems*, vol. 79, 2015.

[7] Y. Ma *et al.*, "Remote sensing big data computing," *Future Gener. Comput. Syst.*, vol. 51, pp. 47–60, Oct. 2015.

[8] P. ZareMoodi *et al.*, "Novel class detection in data streams using local patterns and neighborhood graph," *Neurocomput.*, vol. 158, pp. 234–245, June 2015.

[9] D. Desai and A. Joshi, "A deviant load shedding system for data stream mining," *Procedia Computer Science*, vol. 45, 2015. International Conference on Advanced Computing Technologies and Applications (ICACTA).

[10] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*. New York, NY, USA: Cambridge University Press, 2011.

[11] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*. ("Chapman and Hall/CRC" Computational Science), CRC Press, 2010.

[12] P. Pacheco, *An Introduction to Parallel Programming*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st ed., 2011.

[13] S. Zhang *et al.*, "Cloud computing research and development trend," in *Future Networks, 2010. ICFN '10. Second International Conference on*, Jan 2010.

[14] S. S. Saurabh Bilgaiyan and S. S. Sahu, "Cloud computing: Concept, terminologies, issues, recent technologies," *Research Journal of Applied Sciences*, vol. 9, pp. 614–618, 2014.

[15] S.-S. Kim and H.-K. Ahn, "An improved data stream algorithm for clustering," *Computational Geometry*, vol. 48, no. 9, 2015.

[16] E. Lughofer and M. Sayed-Mouchaweh, "Autonomous data stream clustering implementing split-and-merge concepts - towards a plug-and-play approach," *Inf. Sci.*, vol. 304, May 2015.

[17] A. S. Asensio *et al.*, "Improving data partition schemes in smart grids via clustering data streams," *Expert Systems with Applications*, vol. 41, no. 13, pp. 5832 – 5842, 2014.

[18] Y. Li, D. Li, S. Wang, and Y. Zhai, "Incremental entropy-based clustering on categorical data streams with concept drift," *Know.-Based Syst.*, vol. 59, Mar. 2014.

[19] Z. Miller *et al.*, "Twitter spammer detection using data stream clustering," *Information Sciences*, vol. 260, pp. 64 – 73, 2014.

[20] R. Mythily *et al.*, "Clustering models for data stream mining," *Procedia Computer Science*, vol. 46. Proceedings of the International Conference on Information and Communication Technologies, December 2014, Kochi, India.

[21] PhridviRaj *et al.*, "Clustering text data streams - a tree based approach with ternary function and ternary feature vector," *Procedia Computer Science*, vol. 31, 2014. 2nd International Conference on Information Technology and Quantitative Management, {ITQM}.

[22] M. Z. ur Rehman *et al.*, "Hyper-ellipsoidal clustering technique for evolving data stream," *Knowledge-Based Systems*, vol. 70, 2014.

[23] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 281–297, Oakland, CA, USA., 1967.

[24] Apache Hadoop. `https://hadoop.apache.org/`. Accessed 08/2016.

[25] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, Jan. 2008.

[26] E. Rasmussen, "Information retrieval," ch. Clustering Algorithms, pp. 419–442, Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1992.

[27] Spark Streaming. `http://spark-project.org/`. Accessed 08/2016.