Rapid Parallel Detection of Distance-based Outliers in Time Series using MapReduce

Sorin N. Ciolofan, Florin Pop, Mariana Mocanu, Valentin Cristea

Computer Science Department, University Politehnica of Bucharest, Romania (e-mail: sorin.ciolofan@cs.pub.ro, florin.pop@cs.pub.ro, mariana.mocanu@cs.pub.ro, valentin.cristea@cs.pub.ro)

Abstract: Time series analysis is crucial in a large number of knowledge domains ranging from micro and macro economy, industry, tourism, health to hydrology, meteorology, agriculture, demography, etc. The interest in efficiently and meaningfully processing of time series data increased in the last decade with the spreading of sensor networks and Cyber-Physical Systems which produce huge amounts of measured data. The outlier detection is a key issue for Quality Assurance of time series data and its goal is to detect the objects that present a very different behavior from the expected one. Once identified, these objects are either removed or corrected. In this paper we propose a highly scalable parallel data processing algorithm for outlier ranking based on the distance between data objects. As opposed to the current existing sequential implementations, the provided algorithm is based on the parallel processing employed by the MapReduce paradigm. Using real monitored solar data for experimental validation we show the dramatically improvement of running time for large archives of time series (millions of records order).

Keywords: Time Series; Outliers; Distributed Processing; MapReduce; Data Mining.

1. INTRODUCTION

The proliferation of Internet searches, social media, Internet of Things (IoT), sensing devices in the context of smart cities as well as the unstructured data explosion (video, audio, images, etc.) require a shift in data management from the data warehouses and them distribute processing applications to a holistic approach combining distributed storage, distributed processing, artificial intelligence, multiprocessors systems, aspects mentioned by (Pasupuleti, 2014). The traditional systems failed to address the challenges posed by Big Data in terms of data availability, concurrency, fault tolerance and task scheduling. A solution for task scheduling for many task computing, which is useful for Big Data processing was presented by (Sfrent and Pop, 2015). Relational databases, especially commercial ones (Teradata, Netezza, Vertica, etc.) can handle to some extent the volume problem of Big Data but cannot deal with the other two (velocity and variety), as (Madden, 2012) state. On the other side, a theoretical solution for large-scale data-sets modeling using Petri nets was introduced by (Song et al., 2015). They proposed four new methods to meet the new features of data transmission among datacenters**.**

Apache Hadoop is a set of open source applications that are used together in order to provide a Big Data solution for both storage and parallel processing. The two main components mentioned above are in Hadoop, HDFS and YARN. Hadoop Distributed File System (HDFS) is organized in clusters where each cluster consists of a name node and several storage nodes. A large file is split into blocks and name node takes care of persisting the parts on data nodes. The name node maintains metadata about the files and commits updates to a file from a temporary cache to the permanent data node. The data node does not have knowledge about the full logical HDFS file; it handles locally each block as a separate file.

In Hadoop, fault tolerance is achieved through replication; optimizing the communication by considering the location of the data nodes (the ones located on the same rack are preferred). YARN is an acronym for MapReduce v2.0. YARN implements a master/slave execution of processes with a JobTracker master node and a pool of TaskTrackers which do the work. The two main responsibilities of the JobTracker respectively management of resources and job scheduling/monitoring are handled by two separate daemons. There is a global resource manager (RM) and a per application Application Master (AM). The slave is a per node entity named Node Manager (NM) which is doing the computations. The AM negotiates with the RM for resources and monitors task progress.

To create a Big Data ecosystem other components are added on top of Hadoop: configuration management (Zookeeper, presented by (Haloi, 2015)), columnar organization (HBase, discussed in (George, 2011)), data warehouse querying (Hive, with its highlights presented by (Thusoo et al., 2009)), easier development of MapReduce programs (Pig, presented by (Olston et al., 2008)), machine learning algorithms (Mahout, described in (Owen et al., 2011)). According with (Dean and Ghemawat, 2008), MapReduce is a programming model and its implementation for processing large data sets. The programs are automatically parallelized and run on clusters of machines.

However, as (Stefan , 2014) shown, this paradigm is not new in the Computer Science landscape, the theoretical roots of MapReduce can be found back in the 1936 in Stephens Kleene general recursive functions. The concept of **map()**

and **reduce()** functions is present in elementary circuits design, Backuss functional programming model, Lisp and Scheme and finally, nowadays, in Cloud Computing. The overall principle is presented in Fig 1.

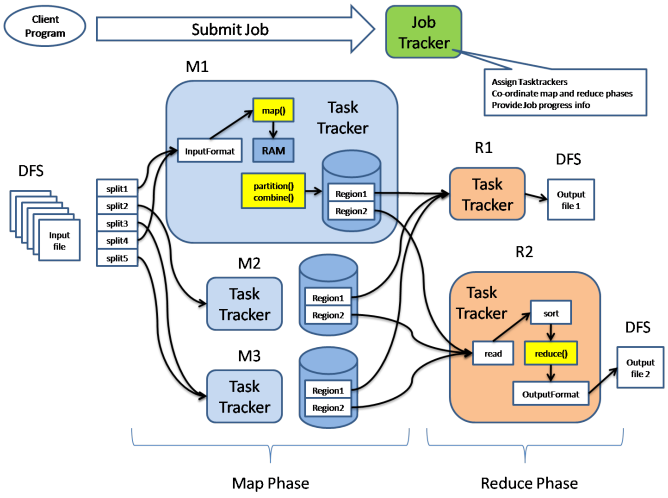

Fig. 1. MapReduce overview presented by (Dean and Ghemawat, 2010).

The input files are split into fragments (16-64Mb) and then each split is parsed according to InputFormat/RecordReader and pairs (*key, value*) are generated and presented to the custom **map()** function. JobTracker determines which TaskTrackers are available and based on their proximity to data sources, data is sent for processing. The **map()** function produces a new set of (*key, value*) elements which are written to the memory buffer. Optional **combine()** function can be used to reduce values on a key. The **partition()** function is applied to each key to determine the index of the reducer that will handle the corresponding value. When all records are finished, the memory buffer is flushed into the corresponding partitions built on local disk of the mapper machine. When JobTracker is notified that some map tasks finished their work it delegates reduce task trackers to download remote data from the files of the mappers and concatenate into one file (sorted by key) that will represent the input of the **reduce()** function which computes the aggregated value and writes it to the final file on HDFS. If initially MapReduce was used for Web data processing, later the areas where Hadoop proved its utility greatly extended to many domains including environmental sciences (e.g. hydrology), being subject to many optimization approaches, as (Nita et al., 2015; Voicu et al., 2014) proposed.

In (Jitkajornwanich et al., 2013), the authors speed up the storm identification using MapReduce instead of DFS (depth first search) traversal. For the locally storm the authors claim the new algorithm is 19 times faster and for hourly storms, 15 times faster. Also, they use as input directly the raw rainfall data file rather than storing it in ODM relational database, which is how they did in a previous approach and noticed significant disk I/O latency, results discussed by (Tarboton et al., 2007).

The paper is further organized as follows: Section 2 presents related work from outlier detection perspective, pointing advantages and drawbacks of each class of methods. Section 3 discusses the proposed algorithm for parallel distance based outlier detection while Section 4 presents the experimental validation of the algorithm and the results of the tests. Section 5 is dedicated to conclusions and future perspectives.

2. BACKGROUND AND RELATED WORK

Anomalies (or outliers) are data objects behaving far different from expected. Detection of such objects is very important in many practical applications: medical and healthcare, environmental resources management, public safety, fraud detection, industry processes, etc. Noise is a random error or variance which occurs in a measured variable and represents a different concept. Because it can look similar to outliers, often a confusion between the two concepts could arise, but in general the amount of error is not too far away from the expected value in the case of noise while in the case of outliers it is too far away. On the other hand, noise can alter the data set quality and make difficult to recognize outliers. It worth to be noted that outliers can have a very high impact on the accuracy of an analysis. Even the percentage of outliers is very low, they can play a key role in the application domain.

Anomalies can be detected in private or public Clouds using logging systems (see solutions presented by (Patrascu and Patriciu, 2014; Morariu et al., 2014), in real-time application monitoring (as is described by (Morariu et al., 2013) and represents an important aspect of Cyber Physical Systems security, approaches considered by (Wang et al., 2011).

Though the definition of outliers may seem simple, in practice there are major difficulties for detecting them, as Singh and Upadhyaya (2012) exemplify. There is often impossible to delimit a precise boundary between "normal" and outlier behavior. Malicious users can "dis-guise" outliers to appear like normal values. The notion of outliers is very different in respect to the application domain (in medicine a small variation of body temperature is considered an outlier while on stock market can be taken as normal). Because of these difficulties rather than trying to provide a universal method the detection techniques try to solve a specific problem.

According with (Han et al., 2011), outliers can be categorized in three main categories:

(1) *global outliers*: the value of the candidate differs significantly from all the other values;

(2) *contextual outliers*: the candidate can be an outlier only in some circumstances; and

(3) *collective outliers*: a group of values that together deviates significantly from the rest of the data set while the individual objects that compose the collective outlier may not be themselves outliers.

It worth to mention that a data object can present both behavioral attributes and contextual attributes. A global outlier can be viewed as a contextual outlier with an empty set of contextual attributes. Methods of outlier detection fit in one of the following categories:

A. supervised methods,

B. unsupervised methods, and

C. semi supervised methods.

In the first category, a domain expert can label the data, deciding which values are "normal" and which "abnormal". Based on this initial categorization a machine can learn further how to label future values. In the second category labels applied by a human expert are not available and to be able to decide which data can be considered outlier are used proximity based methods or clustering methods (described below). In the semi supervised methods labels are available only for a small subset of the initial data.

The data that has only one variable is called univariate while data involving two or more variables is called multivariate. According to the assumptions made regarding when a value is considered abnormal another classification can be made into: statistical methods (data is assumed to follow a certain model (e.g Gaussian distribution) so data that not fit into that model is considered "abnormal"; in our papers (Ciolofan et al., 2014; Pop et al., 2014) we discussed the parametric statistical methods in detail, proximity based methods (outliers are identified based on the distance between them and their neighbors) and cluster based methods (normal values are supposed to belong to large and dense clusters while outliers are not included in a cluster or form a low density small cluster, (see Breunig et al., 2000), angle based outlier detection (ABOD, presented by (Kriegel et al., 2008)), based upon the idea that if object *x* is an outlier then the angle between pairs of the remaining objects become small). The advantage of this last method is that the algorithm is parameter free but the drawback refers to its $O(n^2)$ complexity.

In the statistical parametric methods for univariate variables, based on the mean and standard deviation, one can estimate the probability that a certain measured value can occur in reality. If that probability is low (0.15%) the value can be labeled as outlier. Another straightforward test to check one outlier at a time is Grubbs test (also called "maximum normed residual test", according with (NIST/SEMATECH, 2015). In this approach the values are sorted from the lowest value to the highest and then it is checked if the maximum (or minimum) value is an outlier. Tietjen-Moore test is suitable if exactly k anomalies are suspected, either in the upper tail or in the lower tail or in both tails of the sorted array. The drawback is that k has to be specified in advance. If the value k is not known, then the ESD test (Extreme Studentized Deviate) can be applied assuming an upper limit for k is specified. For the multivariate variables, the problem can be reduced to the detection of outliers for univariate variable (e.g using Mahalanobis distance).

In the non parametric methods the statistical model is not known beforehand. Based on the input data, efforts are made to infer the model. In the histogram method first step is the creation of the histogram and second step is the mapping of the tested value to one of the histograms slots. Another method is to approximate the probability density function using a kernel function:

$$f_h(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

In the above equation, f_h is the estimated probability density for a variable x, n is the size of the dataset, h is a smoothing parameter and $K()$ is the kernel function (e.g uniform, triangular, Epanechnikov, biweight, triweight, Gaussian, etc.). In practice a Gaussian kernel function with mean 0 and standard deviation 1 is often used such as:

$$K(x) = \frac{1}{\sqrt{2\pi}} exp\left(-\frac{1}{2}x^2\right)$$

The kernel is a weighting function used in non-parametric estimation techniques with the following properties:

- Positive definition: $K(x) \geq 0$;
- $\int_R K(x)dx = 1$;
- Symmetry to the origin: $\int_R xK(x)dx = 0$;
- Finite second moment, written as: $\mu_2(K) = \int_R x^2K(x)dx < \infty$.

In general, any function with the mentioned properties can be used as kernel estimation. To select a scale that is appropriate for a specific set of data we can introduce ca scaling factor $\lambda > 0$ and re-define the kernel as:

$$\bar{K}(x) = \lambda K(\lambda x)$$

Using clustering for anomalies detection, we face with at least two disadvantages. First, clustering is a computationally very expensive and do not scale efficiently for large sets of input data. Besides that, one has to first process the majority of data (normal values which are not of primary interest here) in order to, finally, get to the outliers. Secondly, some methods of clustering (such as k-means technique) are not suitable for outlier detection since they are negatively affected by outliers and noise in the input data.

Historically, (Knorr and Ng, 1998) was the first article to define an algorithm for detection of distance based outliers. An object o is considered distance based outlier DB(M, D) if it has less than M objects located in its D-neighborhood. Formally this can be expressed as:

$$\|\{o' | dist(o, o') \leq D\}\| \leq M$$

where $dist(o, o')$ is the distance between the two objects (e.g euclidean distance), D and M are two user supplied parameters, M being far less than N (the total number of objects). Further we consider that data is a time series where at any instant in time *t* we get at most one real number value. Hence,

$$dist(o, o') = |o - o'|$$

The algorithm rather than verifying for each point how many neighbors it has in its D-neighborhood, groups the data into segments (cells) and asserts whether all objects in that segment are outliers or not. The data space is divided in segments of length $D/2$ and points are mapped to the corresponding segment according to their values (Fig 2). For a specific segment C we note $L_1(C)$ (level 1 of C) the two adjacent segments and $L_2(C)$ (level 2 of C) the segments situated at one segment distance from C. The bullets represent values of time series data and the stacked bullets

means the same value which occurs at different timestamps. Further is defined:

$$P(C) = C \cup L_1(C) \cup L_2(C)$$

the partition composed of the five segments and C is named the "*main segment of* P". The following properties hold:

(1) $(\forall x \in C) \wedge (\forall y \in C' | C' \in L_1) \Longrightarrow dist(x,y) \leq D$;
(2) if $(C' \notin L_1) \wedge (C' \notin L_2) \wedge (C' \neq C) \wedge (x \in C, y \in C') \Longrightarrow dist(x,y) > D$;

Based on these properties we can infer following rules which will help in process of determining the outliers:

- R1 : If C contains more than M points then none of the objects in C is outlier;
- R2 : If C and $L_1(C)$ contains more than M points then none of the objects in C is outlier;
- R3 : If C and $L_1(C)$and $L_2(C)$ contains less than M points then every point in C is outlier;
- R4 : A L_1 neighbor of a cell C as in rule R1 has none of the objects outlier because we can find more than M points at a distance less than D (the points in C).

To elaborate the algorithm and distinguish between segments of type $R1$ and segments of type $R4$ the first one are marked with red and the second with pink. This algorithm has two main drawbacks: it requires the parameters D and M to be supplied in advance (D is usually hard to estimate apriori) and, secondly, it does not provide a measure of outlierness.

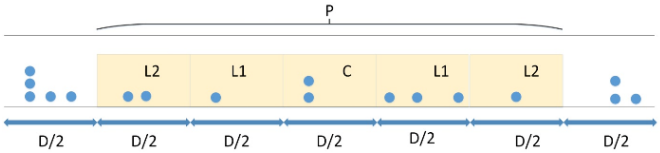

Fig. 2. Segmentation of the data space.

(Ramaswamy et al., 2000) proposed to measure the outlierness by the k^{th} nearest neighbor (k^{th}-NN) distance, giving a ranking of outliers rather than a binary classification. If the rank $q(x)$ is defined as being the distance between x and the k^{th} nearest neighbor then it can be made a connection with the Knorr-Ng algorithm and shown that the two algorithms are equivalent and

$$DB(M,D) = \{x | q(x) \geq D\}$$

The complexity of this algorithm is $O(n^2)$.

(Sugiyama and Borgwardt, 2013) introduced the novel idea of using an on-time random sample as a reference set to compute the ranks of outliers. Best results are obtained for a sample of $s = 20$ objects. The input parameter is s, the number of objects in the sample S. Each object has a rank associated:

$$q_{s_p} = min\{d(x,x') | x' \in S(s)\}.$$

Their algorithm is $\Theta(nms)$ time complexity (n is total number of objects, m is dimension number, and s is sample size) and is much faster than the actual state of the art (2 to 6 orders of magnitude) being the most effective from the detection capability point of view. All these algorithms presented above are sequential, missing the benefits of parallel processing. To overpass this major disadvantage, we discuss in the next section a MapReduce algorithm based on the idea of outlier detection by sampling.

Detecting outliers can be applied for specific performance evaluation of heterogeneous systems, solution presented by (Barbierato et al., 2011; Barbierato et al., 2013), modeling Apache Hive based applications in Big Data architectures, where outliers' tests must be removed from data sets, solution presented by (Barbierato et al., 2013), or estimation of the energy consumption of mobile sensors, model presented by (D'Arienzo et al., 2013). Related to the last application, an overview of energy efficiency techniques in datacenters was presented by (Valentini et al., 2013).

Facing with wide distributed data, a MapReduce framework that aims to enable large-scale distributed computing across multiple clusters was presented by (Wang et al., 2012). Outliers detection can be done on local time series or on distributed data coming from different sources (sensors, drones, etc.). Using Hadoop for detecting outliers in time series, we face with a big data computing computing across distributed cloud data centres, where data access and security is very important. A solution that describe a security framework for Hadoop processing was introduced by (Zhao et al., 2014), with design and implementation presented by (Wang et al., 2013).

4. THE OUTLIERS DETECTION ALGORITHM IN TIME SERIES BASED ON MAPREDUCE

We propose an outlier detection algorithm using MapReduce, composed of two chained map-reduce tasks (Fig 3). First problem is to compute in parallel a unique random sample of size s from the entire data set of n objects and to pass it to each mapper.

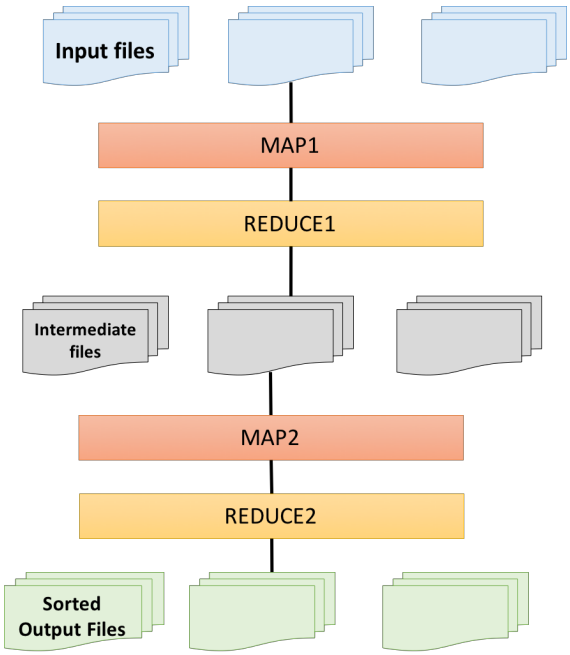

Fig. 3. Chained MapReduce tasks to compute the outliers ranks.

The naive approach would be to gather all n data objects in a Reducer and then randomly pick s elements. However, this has the disadvantage of the amount of data sent by mappers.

Algorithm 1 Outliers Detection in Time Series

```
1: counter ←1;                                   ▷ global file counter
2: function MAP1(fileName, fileContent)
3:     HashMap hMap ←createNewHashMap();
4:     List topList ←createNewList();
5:     key ←getCounter();
6:     for all contentLine,line do
7:         dataObject.timestamp ←line.timestamp;
8:         dataObject.value ←line.value;
9:         dataObject.isSamplePoint ←false;
10:        hMap.PUT(generateRandomNo(),dataObject);
11:        EMIT(key,dataObject);
12:    end for
13:    key ←key+1;
14:    SETCOUNTER(key);
15:    Sort hMap on keys;
16:    Store the top s objects in topList;
17:    for i ← 1; i ≤nf; i ← i + 1 do
18:                                  ▷ nf, number of input files
19:        for j ← 1; j ≤s; j ← j + 1 do
20:            dataObject ←topList.GET(j);
21:            dataObject.isSamplePoint ←true;
22:            EMIT(i,dataObject);
23:        end for
24:    end for
25: end function
26: function REDUCE1(key, [val1, val2, ...])
27:    List samplePool ←createNewList();
28:    List observationsPool ←createNewList();
29:    List globalSample ←createNewList();
30:    for dataObject in [val1,val2...] do
31:        if dataObject.isSamplePoint()=true then
32:            samplePool.ADD(dataObject);
33:        else
34:            observationsPool.ADD(dataObject);
35:        end if
36:    end for
37:    Sort samplePool;
38:    Retain and the top s values in globalSample;
39:    for dataObj1 in observationsPool do
40:        for dataObj2 in globalSample do
41:            d_k ← dist(dataObj1, dataObj2);
42:        end for
43:        r ← min {d_k};
44:        EMIT(r, dataObj1);
45:    end for
46: end function
47: function MAP2(fileName, fileContent)
48:                                      ▷ identity mapper
49: end function
50: function REDUCE2(key, [val1, val2, ...])
51:                                      ▷ identity reducer
52: end function
```

In our implementation we will use a reservoir sampling method where each mapper associates a random number with each data object it has access to, and then select the local top s objects and sent to a reducer. The reducer will choose the final random sample as being the top s objects among all sets of local top s objects.

The Outliers Detection in Time Series Algorithm is presented using MapReduce paradigm in Alg. 1. MAP1() function is responsible for preparing the local top s objects. In lines 6-9 the input file is parsed line by line and data objects are initialized. Then, in a hash map data structure (*hMap*) the object is put as a value, the key being the random number generated (line 10). The observed data is emitted (line 11) along with the local sample (line 21) after the prior sorting of the hash map on keys (line 15).

REDUCE1() uses three utility data structures: *samplePool* which consists of all sample objects sent by all mappers, *observationsPool* which stores the corresponding measured objects and *globalSample* which uses the *samplePool* to extract the global random sample of size *s* with respect to the entire set of measured data. The function has two main responsibilities:

(1) to create the global random sample retaining the top *s* objects from *samplePool* list (line 36), and
(2) to compute for each observed data object the distances from this object to all objects included in the sample (lines 37-41). The minimum of these distances is then emitted as key for value data object (line 43). The intermediate files contain the data objects and their rank, without being sorted on rank.

MAP2() and REDUCE2() are just two identity map/reducers used to sort the data from the intermediate files based on the rank. The final output files contain data sorted on the rank.

4. EXPERIMENTAL RESULTS

To test the performance of algorithm described above we used a Hadoop 1.2.1 cluster deployed on Google Compute Engine Cloud facilities. Each node is *n1-standard-2* type having 2 VCPUs, 7GB RAM and 500 GB disk size.

The Hadoop cluster described in Fig. 4 has one Google Compute Engine instance used as the Hadoop master node which contains the HDFS *NameNode* and the MapReduce JobTracker. Also is possible to have Hive and Pig installed here. The workers in the cluster are Google Compute Engine instances which can be configured depending on the requirements for RAM and CPU's. Once created, the worker nodes will have the Hadoop HDFS *DataNode* and MapReduce *TaskTracker* software installed. Google Cloud Storage (GCS) provides the storage support for input/output files used in the MapReduce jobs. The link between the worker instances and GCS is possible via the Google Cloud Storage Connector for Hadoop. GCS is used for file storage instead of HDFS because it provides quicker startup, less maintenance, high availability and interoperability with other services, as specified in (Google, 2014).

We deployed two configurations, the first one having 1 master node and 1 worker node and the other having 1 master and 10 workers. The input consisted of real observations of solar data made publicly by Vignola (2015c) such as direct/diffuse solar radiation, spectral data, meteorological

data (total rainfall, barometric pressure, humidity, snow depth, *etc.*). Input data for our experiment consists of files of size approximately 1.2Mb each file containing data measured during one month using a 5 minutes interval.

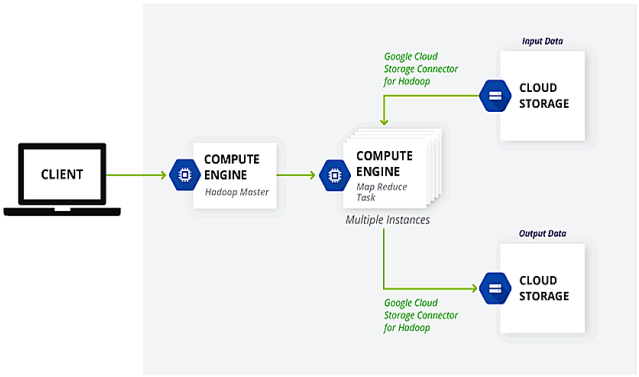

Fig. 4. Hadoop on Google Cloud Platform, described by (Google, 2014).

Table 1. Running time.

Masters	Workers	Files	Records	Running time(sec)
1	1	1	8.928	198
1	10	1	8.928	200
1	1	4	34.560	220
1	10	4	34.560	201
1	1	8	69.984	248
1	10	8	69.984	211
1	1	12	105.120	268
1	10	12	105.120	223
1	1	60	525.600	521
1	10	60	525.600	281
1	1	120	1.051.200	846
1	10	120	1.051.200	289

The data used in our experiment is raw data acquired through Eugene monitoring station (Vignola, 2015a). The structure of the file and the code meaning is provided at (Vignola, 2015b).

The header line in the file contains the location code, the year, the measured element code. Each following line contains the day of the year, the time of day in military format, the value measured by sensor. The data input was chosen N files where N = 1; 4; 8; 12; 60; 120 (data corresponding to 1 month, 4 months, 8 months, 1 year, 5 years, 10 years). Table 1 summarizes the running time in seconds for each scenario.

In Fig 5 are represented three plots: Running Time for 1 worker (rt_{01}) vs. 10 workers (rt_{10}), Processing Rate and Speedup, according with the following definitions:

$$ProcessingRate = \frac{inputSize}{runningTime}$$

$$Speedup = \frac{rt_{01}}{rt_{10}}$$

For an input consisting of up to 12 files, the difference between the running time on 1 worker vs. 10 workers is not significant. That can be explained by the fact that the load is not big enough to exploit the computational capacity of the 10 workers cluster and even 1 worker is enough for this input dimension. In a 10 workers cluster there is an additional payload for network transfer of data between mappers and reducers. The performance dramatically increases with the input getting bigger (example: for 120 files which corresponds to observations of one parameter at 5 minutes interval for 10 years) we observed a running time almost 3 times lower on the 10 workers cluster, since 1 worker does not have enough resources to handle this input faster. It is expected that increasing much more the input we will see a higher ratio between running time on 10 workers vs. running time on 10 workers.

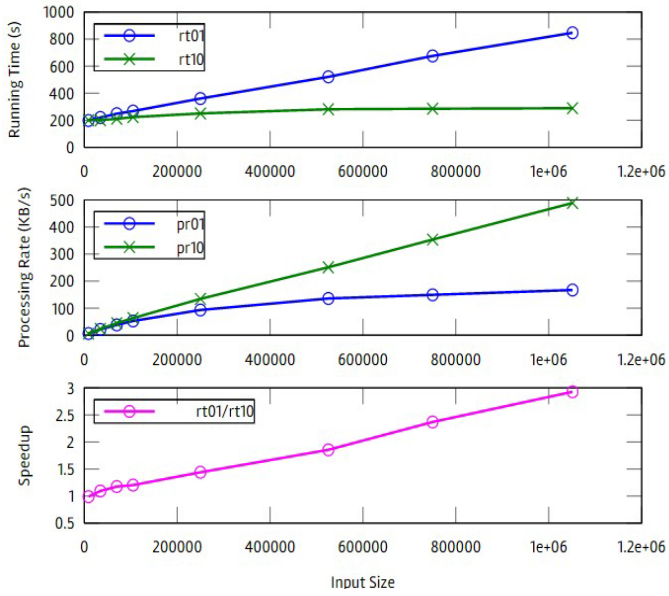

Fig. 5. Experimental results: Running Time, Processing Rate and Speedup.

We define efficiency as:

$$E = \frac{N_i}{N_{m1}}$$

where N_i is the number of input records for Map1() and N_{m1} is the number of intermediate files that are produced after the execution of Map1(). If we consider that we have a number of n_f input files, each file having n records and the sample size is s we obtain:

$$E = \frac{n * n_f}{n_f * (n + n_f * s)} = \frac{1}{1 + s * \frac{n_f}{n}}$$

For our experiments, we obtained an efficiency E in the range 78.81% (for $n_f = 120$) to 99.77% (for $n_f = 1$). Given a number n_f which corresponds to a certain efficiency E_1 it is useful to know the increase Δn_f in the number of input files that will correspond to an efficiency $E_2 > \delta$ where δ is a target minimum efficiency. Using the equation above, we obtain:

$$\Delta n_f < \frac{n * (1 - \delta) - \delta * s * n_f}{\delta * s}$$

For example, if $n_f = 50$ input files we would like to know how many additional input files we can add so finally we get an efficiency over δ = 80%. We obtain in this case $\Delta n_f < 61$ so we can add up to 60 more files.

5. CONCLUSIONS

In this paper we discussed the importance of using Hadoop storage and distributed processing framework for large data sets acquired in various scientifically domains including environmental sciences. We also presented the related work for detecting outliers and then proposed an algorithm for chained MapReduce tasks to compute distance based ranks for outliers in time series. Due to its distributed nature, the proposed algorithm is highly scalable for large volumes of data. By processing in parallel, there is achieved a significant speed up compared with the original sequential algorithm. Our implementation provides also fault tolerance and load balancing. For future improvements of performance, we consider aggregating more input files and presenting to Map1() fewer files but with a bigger size, ideally each input file being

$$\frac{3}{4} * blockSize \leq fileSize \leq blockSize$$

where $blockSize$ refers to the Hadoop block size (by default 64MB). Having less files of bigger size addresses problems of Name Node (such as bigger RAM consumption, each block corresponding to an object in memory, increased delay when Name Node starts, Network impact) and also of MapReduce (disk I/O, busy queues of map tasks).

The second possible optimization refers to the known problem that using default *InputFormat* implementation in case of compute-intensive applications, the fairness in scheduling is neglected and leads to over-usage of the nodes where the data is physically located while the rest are left underutilized. In HDFS the location is a set of nodes where the block resulting from the split phase is physically located.

On the other hand, caring only about fairness leads to scarifying data locality (thus performance) because a job might be scheduled to execute on a node that is far away from its data. As described by (Zaharia et al., 2010) the fair scheduler should be delayed to address the conflict between locality and fairness. A job that is scheduled to execute on a node that does not have local data for it waits for a small amount of time (few seconds) until an opportunity to be scheduled appears on a node that has local data for it.

One direct use of proposed method is in the CyberWater systems, which is a cyber-infrastructure with the main goal to offer a solution for water quality in respect to the pollution phenomena studied on rivers network, first introduced by (Ciolofan et al., 2013). In this system an alert service was introduced, being a typical Publish/Subscribe application. The users can subscribe to receive notifications on their mobile devices (mobile phones, smartphones, tablets, *etc*.) or on their computers, via email. The Alerts Service depends directly on the other Services, like Propagation Analysis Service and Prediction Service, since the analyzed and predicted values are used as a basis for the action of sending notifications. Here, the main role of detecting outliers is to prevent false-positive situation and to avoid false alarms. For the prediction module, where different time series are used, we need to eliminate all outliers from the training set. Only in this way we can minimize the error for predicted values.

Another applications of detecting outliers can be found to a variety of domains including mobile computing, smart cities, forensics and eHealth. The proposed processing model can be integrated in all these domains as a preprocessing phase for all involved time series.

ACKNOWLEDGMENT

The research presented in this paper is supported by projects: *CyberWater* grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012; *clueFarm*: Information system based on cloud services accessible through mobile devices, to increase product quality and business development farms PN-II-PT-PCCA-2013-4-0870.

We would like to thank the reviewers for their time and expertise, constructive comments and valuable insight.

REFERENCES

Barbierato, E., Dei Rossi, G.L., Gribaudo, M., Iacono, M., and Marin, A. (2013). Exploiting product forms solution techniques in multiformalism modeling. *Electron. Notes Theor. Comput. Sci.*, 296, 61–77. doi:10.1016/j.entcs. 2013.07.005.

Barbierato, E., Gribaudo, M., and Iacono, M. (2011). Defining formalisms for performance evaluation with simthesys. *Electron. Notes Theor. Comput. Sci.*, 275, 37–51. doi:10.1016/j.entcs.2011.09.004.

Breunig, M., Kriegel, H.P., Ng, R.T., and Sander, J. (2000). Lof: Identifying density-based local outliers. In *Proceedings of the 200 SIGMOD International Conference on Management of Data*, 93–104. ACM.

Ciolofan, S., Mocanu, M., and Ionita, A. (2013). Cyber Infrastructure architecture to support decision taking in natural resources management. In *Control Systems and Computer Science (CSCS), 2013 19th International Conference on*, 617–623. doi:10.1109/CSCS.2013.17.

Ciolofan, S., Mocanu, M., Pop, F., and Cristea, V. (2014). Improving quality of water related data in a Cyber Infrastructure. In *IWOCPS - Third International Workshop on Cyber Physical Systems*, Romanian Academy.

D'Arienzo, M., Iacono, M., Marrone, S., and Nardone, R. (2013). Estimation of the energy consumption of mobile sensors in WSN environmental monitoring applications. In *Proceedings of the 2013 27th International Conference on Advanced Information Networking and Applications Workshops*, WAINA '13, 1588–1593. IEEE Computer Society, Washington, DC, USA. doi:10.1109/ WAINA.2013.33.

Dean, J. and Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1), 107–113.

Dean, J. and Ghemawat, S. (2010). Mapreduce: a flexible data processing tool. *Commun. ACM*, 53(1), 72–77.

George, L. (2011). HBase: the definitive guide. *O'Reilly Media, Inc.*

Google (2015). *Architecture: Hadoop on google cloud platform*. URL https://cloud.google.com/solutions/architecture/hadoop.

Haloi, S. (2015). Apache ZooKeeper Essentials. Community experience distilled. *Packt Publishing*.

Han, J., Kamber, M., and Pei, J. (2011). Data Mining: Concepts and Techniques. *Morgan Kaufmann Publishers Inc.*, San Francisco, CA, USA, 3rd edition.

Jitkajornwanich, K., Gupta, U., Shanmuganathan, S., Elmasri, R., Fegaras, L., and McEnery, J. (2013). Complete storm identification algorithms from big raw rainfall data using mapreduce framework. In *Big Data, 2013 IEEE International Conference on*, 13–20.

Knorr, E.M. and Ng, R.T. (1998). Algorithms for mining distance-based outliers in large datasets. In A. Gupta, O. Shmueli, and J. Widom (eds.), VLDB'98, Proceedings of *24rd International Conference on Very Large Data Bases*, August 24-27, 1998, New York City, New York, USA, 392–403. Morgan Kaufmann.

Kriegel, H.P., S hubert, M., and Zimek, A. (2008). Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, 444–452. ACM, New York, NY, USA.

Madden, S. (2012). From databases to big data. IEEE Internet Computing, 16(3), 4–6.

Morariu, O., Morariu, C., and Borangiu, T. (2013). Transparent real time monitoring for multi-tenant j2ee applications. *Journal of Control Engineering and Applied Informatics*, 15(4), 37–46.

Morariu, O., Morariu, C., Borangiu, T., and Raileanu, S. (2014). Smart resource allocations for highly adaptive private cloud systems. Journal of Control Engineering and Applied Informatics, 16(3), 23–34.

NIST/SEMATECH (2015). *e-handbook of statistical methods*. URL http://www.itl.nist.gov/div898/handbook/.

Nita, M.C., Pop, F., Voicu, C., Dobre, C., and Xhafa, F. (2015). Momth: multi-objective scheduling algorithm of many tasks in hadoop. *Cluster Computing*, 1–14.

Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. (2008). Pig latin: A not-so-foreign language for data processing. In Proceedings of the 2008 ACM SIGMOD *International Conference on Management of Data*, SIGMOD '08, 1099–1110. ACM, New York, NY, USA.

Owen, S., Anil, R., Dunning, T., and Friedman, E. (2011). Mahout in *Action. Manning Publications Co*., Greenwich, CT, USA.

Pasupuleti, P. (2014). Pig Design Patterns. *Packt Publishing*.

Patrascu, A. and Patriciu, V.V. (2014). Logging system for cloud computing forensic environments. *Journal of Control Engineering and Applied Informatics*, 16(1), 80–88.

Pop, F., Ciolofan, S., Negru, C., Mocanu, M., and Cristea, V. (2014). A bio-inspired prediction method for water quality in a cyber-infrastructure architecture. In *2014 Eighth International Conference on Complex, Intelligent and Software Intensive Systems* (CISIS), 367–372. IEEE.

Ramaswamy, S., Rastogi, R., and Shim, K. (2000). Efficient algorithms for mining outliers from large data sets. *SIGMOD Rec.*, 29(2), 427–438.

Sfrent, A. and Pop, F. (2015). Asymptotic scheduling for many task computing in big data platforms. *Information Sciences*.

Singh, K. and Upadhyaya, S. (2012). Outlier detection: applications and techniques. *International Journal of Computer Science Issues*, 9(1), 307–323.

Song, W., Wang, L., Ranjan, R., Kolodziej, J., and Chen, D. (2015). Towards modeling large-scale data flows in a multidatacenter computing system with petri net. *Systems Journal*, IEEE, 9(2), 416–426. doi:10.1109/ JSYST.2013.2283954.

Stefan, G.M. (2014). Mapreduce - an integrative paradigm in cyber-physical systems. In *IWOCPS - Third International Workshop on Cyber Physical Systems, Romanian Academy*.

Sugiyama, M. and Borgwardt, K. (2013). Rapid distance-based outlier detection via sampling. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger (eds.), Advances in *Neural Information Processing Systems* 26, 467–475. Curran Associates, Inc.

Tarboton, D.G., Horsburgh, J.S., and Maidment, D.R. (2007). Cuahsi community observations data model (odm) version 1.0 design specifications.

Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., and Murthy, R. (2009). Hive: A warehousing solution over a map-reduce framework. Proc. *VLDB Endow.*, 2(2), 1626–1629.

Valentini, G.L., Lassonde, W., Khan, S.U., Min-Allah, N., Madani, S.A., Li, J., Zhang, L., Wang, L., Ghani, N., Kolodziej, J., Li, H., Zomaya, A.Y., Xu, C.Z., Balaji, P., Vishnu, A., Pinel, F., Pecero, J.E., Kliazovich, D., and Bouvry, P. (2013). An overview of energy efficiency techniques in cluster computing systems. *Cluster Computing*, 16(1), 3–15.

Vignola, F. (2015a). Oregon srml, eugene monitoring station info.

URL http://solardat.uoregon.edu/ Eugene.html.

Vignola, F. (2015b). Oregon srml, the structure of archive file.

URL http://solardat.uoregon.edu/ ArchivalFiles.html.

Vignola, F. (2015c). Solar radiation monitoring laboratory, university of oregon, solar data archives. URL http:// solardat.uoregon.edu/SelectArchival.html.

Voicu, C., Pop, F., Dobre, C., and Xhafa, F. (2014). Momc: Multi-objective and multi-constrained scheduling algorithm of many tasks in hadoop. In *P2P, Parallel, Grid, Cloud and Internet Computing* (3PGCIC), 2014 Ninth International Conference on, 89–96. IEEE.

Wang, J., Abid, H., Lee, S., Shu, L., and Xia, F. (2011). A secured health care application architecture for Cyber Physical systems. *Journal of Control Engineering and Applied Informatics*, 13(3), 101–108.

Wang, L., Tao, J., Ma, Y., Khan, S.U., Kolodziej, J., and Chen, D. (2013). Software design and implementation for mapreduce across distributed data centers. *Appl. Math*, 7(1L),85–90

Wang, L., Tao, J., Marten, H., Streit, A., Khan, S.U., Kolodziej, J., and Chen, D. (2012). Mapreduce across distributed clusters for data-intensive applications. In Proceedings of the *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, IPDPSW '12, 2004–2011. IEEE Computer Society, Washington, DC, USA.

Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., and Stoica, I. (2010). Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, 265–278. ACM, New York, NY, USA. doi:10.1145/1755913.1755940.

Zhao, J., Wang, L., Tao, J., Chen, J., Sun, W., Ranjan, R., Kolodziej, J., Streit, A., and Georgakopoulos, D. (2014). A security framework in g-hadoop for big data computing across distributed cloud data centres. *Journal of Computer and System Sciences*, 80(5), 994–1007.