Fast Parallel Outlier Detection for Categorical Datasets using MapReduce

Anna Koufakou, Jimmy Secretan, John Reeder, Kelvin Cardona, and Michael Georgiopoulos

Abstract— **Outlier detection has received considerable attention in many applications, such as detecting network attacks or credit card fraud. The massive datasets currently available for mining in some of these outlier detection applications require large parallel systems, and consequently parallelizable outlier detection methods. Most existing outlier detection methods assume that all of the attributes of a dataset are numerical, usually have a quadratic time complexity with respect to the number of points in the dataset, and quite often they require multiple dataset scans. In this paper, we propose a fast parallel outlier detection strategy based on the Attribute Value Frequency (AVF) approach, a high-speed, scalable outlier detection method for categorical data that is inherently easy to parallelize. Our proposed solution, MR-AVF, is based on the MapReduce paradigm for parallel programming, which offers load balancing and fault tolerance. MR-AVF is particularly simple to develop and it is shown to be highly scalable with respect to the number of cluster nodes.**

I. Introduction

DETECTING outliers in data is a research field with many applications, such as credit card fraud detection, discovering criminal activities in electronic commerce, and network intrusion detection. Outlier detection approaches concentrate on detecting patterns that occur infrequently in the dataset, in contrast to traditional data mining techniques that attempt to find patterns that occur frequently in the data. Application examples where the discovery of outliers is useful include identifying irregular credit card transactions, indicating potential credit card fraud [1], or patients who exhibit abnormal symptoms due to their suffering from a specific disease or ailment [2].

Most of the existing research efforts in outlier detection concentrate on datasets with attributes that are either numerical or ordinal (can be directly mapped into numerical values). In the case where data with categorical attributes are present, these techniques map the categorical to numerical values, a task which is not always a straightforward process.

Another issue is that many of the above applications for the mining of outliers require the mining of very large datasets (e.g. terrabyte-scale data). This leads to the need for large, parallel machines and associated parallelizable outlier detection algorithms, which must scale well with the size and dimensionality of the dataset. The size of the datasets today also demands that the methods developed be computationally simple and easily balanced over a number of cluster nodes.

In this paper, we propose a parallel implementation of a fast and simple outlier detection method for categorical datasets, called *Attribute Value Frequency* (AVF). AVF [3] was shown to have a significant performance advantage over a number of other competitive outlier detection strategies that have appeared in the recent literature, and it was also shown to scale linearly as the dataset size increases, both in the number of points and number of dimensions. Also, AVF depends only on one user parameter (the number of desired outliers, *k*, needed to be identified), an important advantage, since it requires minimum user intervention. Furthermore, given the frequencies of each categorical value in the data, AVF performs only one dataset scan, thus lending itself to today's large and possibly geographically distributed data.

AVF is based on assigning a score to each point in the dataset using the frequency of each unique attribute value, thus it is easily parallelizable. In contrast, other techniques for outlier detection in categorical data (see [4], [5] and [6]) are much more complicated and cumbersome to parallelize, as they require several scans of the dataset, in order to extract frequently encountered, and sometimes lengthy, combinations of attribute values. Moreover, the parallel version of AVF that is proposed here is based on the MapReduce paradigm of parallel programming [7]. MapReduce provides the necessary simplicity of parallel development, while guaranteeing the necessary load balancing and fault tolerance for the implementation. It is worth noting that MapReduce has already been successfully used in parallelizing a number of machine learning approaches for data mining applications (e.g. see [8]).

Our contribution is that we introduce MapReduce-AVF (*MR-AVF)* a parallel outlier detection method for categorical datasets, geared towards identifying outliers in large data mining problems. MR-AVF is based on AVF, an outlier detection method that has been shown to perform favorably compared to other competitive but more complex outlier detection strategies. Due to its simplicity, AVF is an ideal method to parallelize, and using the MapReduce approach to parallelize it guarantees ease of development, load balancing and fault tolerance of the implementation. Our results show that MR-AVF exhibits close to ideal speedup with respect to number of processing nodes in the cluster.

Manuscript received March 5, 2008. This work was supported in part by NSF grants: 0341601, 0647018, 0717674, 0717680, 0647120, 0525429, 0203446, as well as an NSF graduate research fellowship.

Anna Koufakou, Jimmy Secretan, John Reeder, and Michael Georgiopoulos are with the School of EECS at the University of Central Florida, Orlando, FL, 32816. Kelvin Cardona is with the Department of Computer Engineering at the University of Puerto Rico.

978-1-4244-1821-3/08/$25.00©2008 IEEE

The organization of this paper is as follows: In Section II, we provide an overview of the previous research related to outlier detection strategies, as well as a synopsis of the earlier work based on the MapReduce paradigm. In Section III, we present the MapReduce paradigm, while in Section IV we introduce our proposed algorithm for parallel outlier detection, i.e. the MapReduce-AVF. Finally, we present our experimental results in Section V, followed by our conclusions in Section VI.

II. Previous Work

The existing outlier detection methods can be grouped into the following categories.

Statistical-model based methods assume that a specific model describes the distribution of the data [9]. Limitations include obtaining the right model for each dataset and application, and lack of scalability with respect to data dimensionality [6].

Distance-based approaches (e.g. [10]) essentially compute distances among data points, thus becoming quickly impractical for large datasets (e.g., a nearest neighbor method has quadratic complexity with respect to the number of dataset points). Bay and Schwabacher [11] propose a distance-based method based on randomization and pruning and claim its complexity is close to linear in practice.

Clustering techniques can also be employed to cluster the data, and the points that do not belong in the formed clusters are designated as outliers. However, clustering-based methods are focused on optimizing clustering measures of goodness, and not on finding the outliers in the data [10].

Density-based methods estimate the density distribution of the data and identify outliers as those lying in low-density regions (e.g. [12], [13]). Although density-based methods detect outliers not discovered by the distance-based methods, they become problematic for sparse high-dimensional data [14].

Other outlier detection efforts rely on Support Vector methods (e.g. [15]), Replicator Neural Networks (RNNs) [16], or using a relative degree of density with respect only to a few fixed reference points [17].

Most of the aforementioned techniques are geared towards numerical data and thus are more appropriate for numerical datasets or ordinal data that can be easily mapped to numerical values [18]. Another limitation of previous methods is the lack of scalability with respect to number of points and/or dimensionality of the dataset

Outlier Detection techniques for categorical datasets have recently appeared in the literature (e.g. [5], [14], [19]). For instance, Otey et al. in [6] presented a distributed outlier detection method for mixed attribute datasets. Their approach is linear with respect to the number of data points; however their running time is exponential in the number of categorical attributes and quadratic in the number of numerical attributes.

Furthermore, Koufakou et al. [3] experimented with a number of representative outlier detection approaches for categorical data, and proposed AVF (Attribute Value Frequency), a simple, fast, and scalable method for categorical sets.

In this paper, we propose a parallel version of the AVF algorithm proposed in [3]. This parallel method is developed using MapReduce [7], which is a simplified parallel program paradigm for large scale, data intensive, parallel computing jobs. MapReduce hides the parallel machine from the programmer by simplifying the parallel programming model to two functions: the map function and the reduce function. Given a list of keys and associated values, the map function produces an intermediate set of keys and values. The reduce function then combines these intermediate values into a final result.

MapReduce has already found its way into several machine learning and data mining applications. Chu et al. [8] present many algorithms in MapReduce form, including Locally Weighted Linear Regression, k-means, Logistic Regression, Naive Bayes, Linear Support Vector Machines, Independent Component Analysis, Gaussian Discriminant Analysis, Expectation Maximization, and Backpropagation.

III. MapReduce

MapReduce is a parallel programming paradigm, originally introduced by Google [7], whose central focus is to simplify the processing of large datasets on inexpensive cluster computers. These cluster computers often contain hundreds or thousands of nodes that both store and process the datasets in a distributed fashion. Typically, a single master server is used to schedule the data storage and computation on the nodes. The original MapReduce system was built on the Google File System (GFS), [20], which is optimized for storing large, infrequently changed datasets across standard disks on the cluster nodes. The MapReduce/GFS combination is built to tolerate regular node failures through replication of the data and speculative execution. This system also automatically provides for load balancing and scheduling associated with the parallel processing of the data.

Users design a MapReduce program by relying, almost entirely, on the map and reduce functions. As a consequence, the user is not forced to devise a parallelization strategy for the task at hand, but is only required to adapt it to a MapReduce model. The map function takes as input a set of key-value pairs, designated as k_1 and v_1, provided directly from the user-defined input files. Within the map function, the user specifies what to do with these keys and values. The map function outputs another set of keys and values, designated as k_2 and v_2. The reduce function sorts the key value pairs by k_2. All of the associated values v_2 are reduced and emitted as value v_3. The map and reduce functions are as follows:

$$map(k_1, v_1) \rightarrow (k_2, v_2)[]$$
$$reduce(k_2, v_2[]) \rightarrow (k_2, v_3)[]$$

At the MapReduce run-time level, the map operations are distributed by the master-server to the chunk-servers. The scheduler makes an effort to schedule computation on the same node where the data is stored. Meanwhile, other chunk-servers assigned to the reduce phase begin to take the (k_2, v_2) value pairs and sort them by k_2. These sorted arrays of v_2 values are passed to the reduce functions on these same assigned nodes. These outputs are finally saved on the GFS. It is quite common for an application to string together many simpler MapReduce operations.

Fault tolerance and load balancing are automatically provided by the software that supports MapReduce and the GFS. Because the GFS stores a user-specified number of copies (usually three) for each chunk of the data on different chunk servers, and because the GFS monitors the cluster to maintain these copies, losing a particular chunk of data should be relatively rare. For fault tolerance of the MapReduce operations, the master server keeps track of all running operations and can re-start failed tasks on other chunk servers that have a copy of the data. By the nature of operations that are put into the MapReduce framework (map operations that are independent on each element) they can be recomputed by any chunk server with the proper data.

A diagram of a typical MapReduce/GFS architecture is displayed in Figure 1.

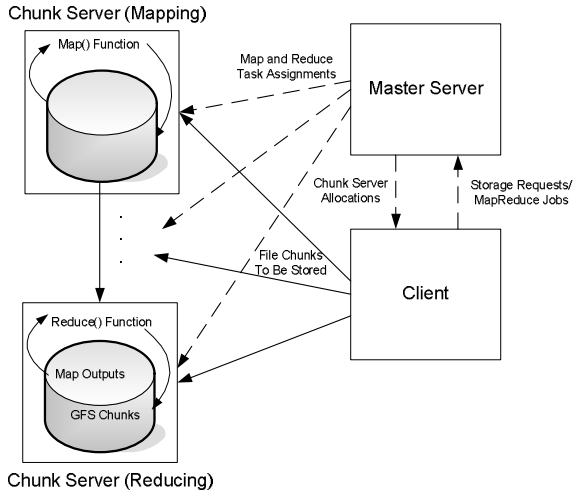

Figure 1: The flow of data in a MapReduce/GFS architecture for file storage and MapReduce operations. Dashed lines indicate control messages, and solid lines indicate data transfer.

An often cited MapReduce example is known as *WordCount* [7]. Suppose we need to obtain the number of occurrences of each unique word in a large file. In the MapReduce paradigm, this computation can be done easily and efficiently as follows: the map function receives as input a line from the large input file. Then the map function splits this line into its component words and emits the word as the key and '1' as the associated value.

The reduce function takes these word-keys and '1' values as input. Because each '1' value is an occurrence of the same word, they are simply summed together to find the number of occurrences of the word. When the reduction operation is complete, there will be a list of words with their associated occurrence frequency. See Figure 2 for a pictorial illustration of the *WordCount* example.

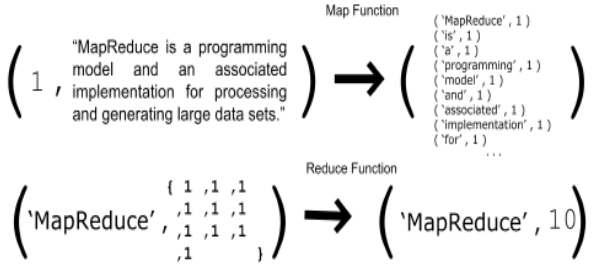

Figure 2: A pictorial illustration of the *WordCount* example.

IV. AVF AND MR-AVF ALGORITHMS

A. AVF: Attribute Value Frequency

The Attribute Value Frequency (AVF) algorithm is a simple and fast approach to detect outliers in categorical data, which minimizes the scans over the data, without the need to create or search through different combinations of attribute values or itemsets. Further details are omitted due to space limitations, and the reader is referred to [3] for further reading.

It is intuitive that that outliers are those points which are infrequent in the dataset, and that the 'ideal' outlier point in a categorical dataset is one whose each and every value is extremely irregular (or infrequent). The *infrequent-ness* of an attribute value can be measured by computing the number of times this value is assumed by the corresponding attribute in the dataset.

Let's assume that there are n points in the dataset, $\mathbf{x}_i$, $i = 1 \ldots n$, and each data point has m attributes. We can write $\mathbf{x}_i = [x_{i1}, \ldots, x_{il}, \ldots, x_{im}]$, where x_{il} is the value of the l-th attribute of $\mathbf{x}_i$. Following the reasoning given above, the AVF score below is a good indicator of deciding of whether $\mathbf{x}_i$ is an outlier:

$$AVF \quad Score(\mathbf{x}_i) = \frac{1}{m}\sum_{l=1}^{m} f(x_{il}) \qquad (1)$$

where $f(x_{il})$ is the number of times the l-th attribute value of $\mathbf{x}_i$ appears in the dataset. A lower AVF score means that it is more likely that the point is an outlier. Since (1) is essentially a sum of m positive numbers, the AVF score is minimal when each of the sum's terms is individually minimized. Thus, the 'ideal' outlier as defined above will have the minimum AVF score. The minimum score will be achieved when every value in the data point occurs just once.

Input: Dataset – $\boldsymbol{D}$ (n points, m attributes)
Target number of outliers – k
Output: k detected outliers

Label all data points as non-outliers;
Calculate frequency of each attribute value, $f(x_{il})$;
foreach point $\mathbf{x}_i$ ($i = 1..n$)
  foreach attribute l ($l = 1..m$)
    AVF Score ($\mathbf{x}_i$) + = $f(x_{il})$;
  end
  $Average_l$ (AVF Score ($\mathbf{x}_i$));
end
Return top k outliers with minimum AVF Score

Figure 3: AVF Pseudocode

Input: Dataset – $\boldsymbol{D}$ (n points, m attributes)
Target number of outliers – k
Output: k detected outliers

HashTable H;
map($k1 = i$, v1 = $\boldsymbol{D}_i = \mathbf{x}_i$, $i = 1..n$) **begin**
  foreach l in $\mathbf{x}_i$ ($l = 1..m$)
    collect(x_{il}, 1);
end
reduce($k2 = x_{il}$, v2) **begin**
  $H(x_{il})$ + = $\sum$ v2;
end
map($k1 = i$, v1 = $\boldsymbol{D}_i = \mathbf{x}_i$) **begin**
  AVF = $\sum_{l=1}^{m} H(x_{il})$;
  collect($k1$, AVF);
end
reduce($k2$ = AVF$_i$, v1 = i);

Figure 4: Parallel AVF Pseudocode – MR-AVF

As shown in the pseudocode of AVF (see Figure 3), once the AVF score is calculated for all the points, the k outliers returned are the k points with the smallest AVF scores. The complexity of AVF is $O(n*m)$, where n is the number of data points and m is the dimensionality of the dataset.

B. Parallel AVF: MR-AVF

The original AVF algorithm calculates the AVF over each input record independently, making it amenable to easy parallelization. If the AVF can be expressed in terms of the MapReduce model, then the parallel algorithm can have the benefits of automatic load balancing and fault tolerance, with no additional effort from the user's perspective.

Using MapReduce, the Map function associates each distinct attribute value to the Map's output key. In the Reduce function, the frequency counts of each attribute value are computed. Finally, the AVF score of each point is calculated during a second Map function. The second reduce is simply a sorting operation of the computed AVF scores.

The pseudocode for MapReduce-AVF, or MR-AVF, is shown in Figure 4. In the first pair of map and reduce functions (first phase), the frequency of each attribute value is extracted from the data set. If the attribute values across each dimension are unique or (as in our code) the dimension is concatenated to the attribute value, this pair of functions is similar to the *WordCount* problem described in Section III.

In the second MapReduce phase, the attribute value frequency table resulting from the first phase is loaded into the hash table H by the map function. The map function then calculates and emits the AVF score for each individual input record, by iterating through the dimensions and adding the frequency of every attribute value. In order to sort the data points by their outlier score, the AVF score is emitted as the key, and the input point ID is emitted as the value. At the end of the MapReduce process, the result is a list of AVF scores sorted in ascending order with the listed point IDs. As a result, the top-k points represent the outliers of the dataset as they have the k minimum AVF scores.

V. Experiments

A. Experimental Setup

Since the original MapReduce/GFS implementation is proprietary, we used an open source MapReduce software called Hadoop [23]. Hadoop allows easy MapReduce implementation in Java, with support to connect it to other languages like C++ and Python. The software was installed on a 16-node cluster, where each of the nodes had dual Opteron processors, 3GB of RAM, and 73GB disks. We used Hadoop version 0.15 and Java to implement the parallel MR-AVF code.

B. Datasets Used

We used the following datasets from the UCI repository [21]:

- *Wisconsin Breast Cancer:* This dataset has 699 points and 9 attributes. Each record is labeled as either benign or malignant. Following the method in [16], we only kept every sixth malignant record, resulting in 39 outliers (8%) and 444 non-outliers (92%).
- *Lymphography*: This dataset contains 148 instances and 18 attributes. Classes 1 and 4 comprise 4% of the data, so they are considered as the outliers.
- *Post-operative*: This dataset is used to determine where patients should go to after a postoperative unit (Intensive Care Unit, home, or hospital floor). It contains 90 instances and 8 attributes. Class 1 and 2 are the outliers.
- *Pageblocks:* It contains 5,473 instances with 10 attributes. There are 5 classes, where one class is about 90% of the data, so the rest of the data can be thought of as outliers. We discretized the continuous attributes using an equal-frequency discretization approach, and removed half of the outliers so that we have a more imbalanced dataset.

TABLE 1. RESULTS ON THE UCI DATASETS

(a) Breast Cancer (39 outliers)

k	*AVF*	*Greedy*	*FPOF*	*Otey's*
4	4	4	3	3
8	7	8	7	7
16	14	15	14	15
24	21	22	21	21
32	28	29	27	28
40	32	33	31	33
48	36	37	35	37
56	39	39	39	39

(b) Lymphography (6 outliers)

k	*AVF*	*Greedy*	*FPOF*	*Otey's*
2	2	2	2	2
4	4	4	4	4
6	4	5	4	4
8	5	6	5	5
12, 13	6	6	5	5 (6)
15	6	6	6	6

(c) Post-Operative (26 outliers)

k	*AVF*	*Greedy*	*FPOF*	*Otey's*
10	3	4	3	1
20	7	7	7	7
30	10	8	9	9
40	11	12	10	10
50	12	13	12	13
60	16	20	17	18
70	21	21	21	21
80	24	24	24	24

(d) Pageblocks (280 outliers)

k	*AVF*	*Greedy*	*FPOF*	*Otey's*
100	40	45	19	19
200	84	81	42	42
300	120	130	63	63
400	168	157	74	74
500	189	177	80	80
600	201	183	94	94
700	206	213	96	96
800	214	237	110	110
900	223	242	116	116
1000	233	242	121	121

TABLE 2. RUNTIME IN SECONDS FOR THE SIMULATED DATASETS WITH VARYING DATA SIZE, *N*, FROM 1K TO 800K DATA POINTS

Data Size ***(thousands)***	**Greedy**	**AVF**	**FPOF**	**Otey's**
1	0.27	0.00	0.81	4.58
10	2.72	0.03	8.13	44.72
30	8.53	0.06	24.02	134.30
50	14.31	0.09	40.19	222.88
100	26.42	0.19	81.06	445.39
200	52.75	0.39	165.08	891.28
300	79.39	0.58	241.61	1337.06
400	106.14	0.80	323.97	1781.78
500	131.75	0.94	404.45	2233.74
600	158.70	1.16	484.00	2678.73
700	184.94	1.33	564.80	3127.22
800	212.08	1.56	667.55	3568.55

Most importantly, we conducted experiments with simulated data in order to showcase the speedup and the associated scalability of MR-AVF, the parallel algorithm that we propose in this paper. For that purpose, we created a sizeable, simulated, categorical dataset that contains 10 million data points, 64 attributes, and 10 categorical values per attribute. The simulated dataset used in our experiments was created using available software by Cristofor [22], which allows for the generation of categorical datasets with different values for the number of data points, the dimensionality of the data, and the number of categorical values per dimension.

C. Results

The outlier detection accuracy and scalability of the serial version of AVF was shown in [3]. We reiterate some of the results from [3] in Table 1 for completeness. The algorithms against which we compared AVF are the Greedy algorithm [19], the FPOF algorithm from [5], and Otey's algorithm from [6]. In Table 1, we denote by *k* the number of target or desired outliers that the algorithm is attempting to detect. Given *k*, the desired number of outliers, the accuracy of an outlier detection algorithm is determined by the number of actual outliers detected by the algorithm.

As can be seen from Table 1, AVF's outlier detection accuracy compares very well with the outlier detection accuracy of the other competitive outlier detection strategies proposed in the literature. For example, in Table 1(a), all algorithms converge to the 39 outliers for *k* equal to 56. The same goes for Table 1(b), (c) and (d). In Table 1(d), AVF's accuracy is higher than two other methods, FPOF and Otey's; Greedy's performance is slightly better, mainly because it is based on multiple dataset scans, in which outliers are successively taken out of the dataset one after another. Given the frequency of each attribute value, AVF needs only one dataset scan and is based on very simple computations using the frequencies. In comparison, the three other methods in Table 1 require multiple dataset scans and complicated computations for each data point.

Table 2 contains the runtime performance of all algorithms using a simulated dataset, with varying number of data points, *n*. For example, using a simulated dataset of 700,000 data points and 10 attributes (generated with software from [22]), Greedy took about 185 seconds, Otey's about 3,127 seconds, FPOF 564 seconds, while AVF had a running time of 1.33 seconds. Further experiments conducted in [3] (omitted due to space limitations) confirmed AVF's advantage over the other methods with respect to running time, and similar results were observed from experiments in [3] for higher values of dimensionality, *m*, and desired number of outliers, *k*.

The purpose of this paper is to further amplify the significant computational advantages of AVF, compared to other competitive outlier detection strategies. In our efforts for parallelizing AVF (MR-AVF), we relied on the algorithm's simplicity and inherent parallelism, as well as the ease of parallelization that the MapReduce paradigm offers. To that extent, we ran MR-AVF for the simulated

dataset for a different number of cluster nodes, i.e., for nodes equal to 1, 2, 4, 8, and 16. The 'ideal' speedup is linear, i.e. when running an algorithm with linear speedup, doubling the number of nodes doubles the speed.

As we can see from the algorithm description in Section IV, there are two main phases in the MR-AVF computation: the first phase extracts the attribute value frequencies from the data, and the second phase calculates the AVF scores for each data point as in Eq. (1) based on the frequencies obtained from the first phase. The speedup of both phases, as well as the speedup of the entire algorithm is illustrated in Figure 5, as the number of cluster nodes increases from 2 to 16.

In the first phase of MR-AVF, i.e. the extraction of categorical attribute value frequency, the speedup is very close to linear. In the second phase, where AVF scores are calculated, there is a sub-linear speedup that levels off quickly at 8 nodes. This can be explained by the fact that the frequency extraction involved more keys as the output of the map phase (one for each dimension of every point, rather than one per point for the frequency extraction phase) and thus was more computationally intensive. For the simulated dataset with which we experimented, the AVF frequency extraction phase required about 80% of the total time, and the AVF score phase required 20% of the total time. Therefore, the overhead of scheduling the map and reduce tasks was greater for the AVF score phase in comparison to the amount of time needed to compute the actual scores, resulting in the lower observed speedup for stage two of the MR-AVF, shown in Figure 5. It is expected that for larger datasets, the AVF score phase will exhibit better scaling. Note though, that because the task is weighted toward the frequency extraction phase, the speedup of the final algorithm was relatively close to linear as exhibited in the corresponding curve of Figure 5.

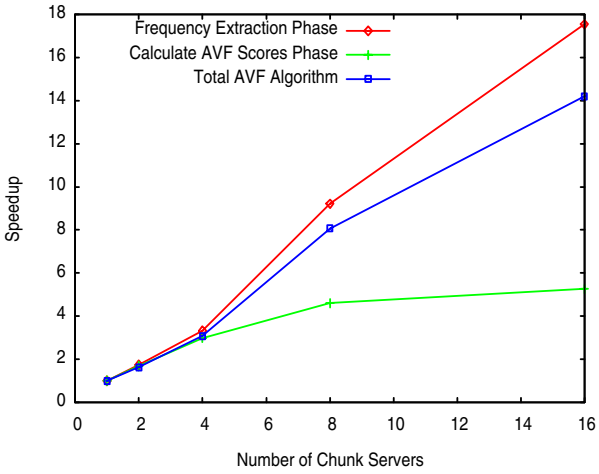

Figure 5: Speedup of our parallel AVF algorithm (MR-AVF) as the number of servers in our cluster increases from 1 to 16 nodes.

VI. Conclusions

We have presented a parallel outlier detection method for categorical datasets, that exhibited high scalability and speed-up. Our proposed parallel outlier detection approach, MapReduce-AVF, or MR-AVF, is built on AVF, an outlier detection method that is fast, scalable, makes minimal dataset scans and requires minimum user intervention (i.e., specification of the number of outliers that need to be extracted).

We based our parallel approach on the MapReduce paradigm and thus MR-AVF is extremely simple and easy to develop. MapReduce ensures load balancing and fault tolerance, and has already been used for parallelization of many other data mining approaches. We experimentally showed that the speedup of MR-AVF is very close to linear as the number of nodes in the cluster increases, thus making it amenable for large data mining applications.

References

[1] Bolton, R.J., Hand, D.J., "Statistical fraud detection: A review", *Statistical Science*, 17, pp. 235–255, 2002.

[2] Penny, K.I., Jolliffe, I.T., "A comparison of multivariate outlier detection methods for clinical laboratory safety data", *The Statistician, Journal of the Royal Statistical Society*, 50, pp. 295–308, 2001.

[3] Koufakou, A., Ortiz, E., Georgiopoulos, M., Anagnostopoulos, G., Reynolds, K., "A Scalable and Efficient Outlier Detection Strategy for Categorical Data", *Int'l Conference on Tools with Artificial Intelligence ICTAI*, October, 2007.

[4] Agrawal, R., Srikant, R., "Fast algorithms for mining association rules", *Proc. of the Int'l Conference on Very Large Data Bases VLDB*, pp. 487–499, 1994.

[5] He, Z., Xu, X., Huang, J., Deng, J., "FP-Outlier: Frequent Pattern Based Outlier Detection", *Computer Science and Information System*, pp. 103-118, 2005.

[6] Otey, M.E., Ghoting, A., Parthasarathy, A., "Fast Distributed Outlier Detection in Mixed-Attribute Data Sets", *Data Mining and Knowledge Discovery*, 2006.

[7] Dean, J., and Ghemawat, S., "Mapreduce: Simplified data processing on large clusters," *Proceedings of OSDI'04: Symposium on Operating System Design and Implementation,* 2004.

[8] Chu, C.-T., Kim, S., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A. Y., and Olukotun, K., "Map-reduce for machine learning on multicore," *Proceedings of NIPS*, 19, 2006.

[9] Barnett, V., Lewis, T. *Outliers in Statistical Data*. John Wiley, 1994.

[10] Knorr, E., Ng, R., and Tucakov, V., "Distance-based outliers: Algorithms and applications", *Very Large Databases VLDB Journal*, 2000.

[11] Bay, S.D. Schwabacher, M., "Mining distance-based outliers in near linear time with randomization and a simple pruning rule", *Proc. of ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, 2003.

[12] Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J., "LOF: Identifying density-based local outliers", *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, 2000.

[13] Papadimitriou, S., Kitawaga, H., Gibbons, P., Faloutsos, C., "LOCI: Fast outlier detection using the local correlation integral", *Proceedings of the International Conference on Data Engineering*, 2003.

[14] Wei, L., Qian, W., Zhou, A., Jin, W., "HOT: Hypergraph-based Outlier Test for Categorical Data", *Proc. of 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining PAKDD*, pp. 399-410, 2003.

[15] Tax, D., Duin, R., "Support Vector Data Description", *Machine Learning*, pp. 45–66, 2004.
[16] Harkins, S., He, H., Williams, G., Baster, R., "Outlier Detection Using Replicator Neural Networks", *Data Warehousing and Knowledge Discovery, 4th International Conference, DaWaK*, pp. 170-180, 2002.
[17] Pei, Y., Zaiane, O., Gao, Y., "An Efficient Reference-based Approach to Outlier Detection in Large Dataset", *IEEE Int'l Conference on Data Mining*, 2006.
[18] Hodge, V., Austin, J., "A Survey of Outlier Detection Methodologies", *Artificial Intelligence Review*, pp. 85, 2004.
[19] He, Z., Deng, S., Xu, X., "A Fast Greedy algorithm for outlier mining", *Proceedings of PAKDD*, 2006.
[20] Ghemawat, S., Gobioff, H., and Leung, S.-T., "The google file system," In *Proceedings of 19th ACM Symposium on Operating Systems Principles,* October, 2003.
[21] Blake, C., Merz, C. *UCI machine learning repository:* www.ics.uci.edu/~mlearn/MLRepository.html.
[22] Cristofor, D., and Simovici, D., "Finding Median Partitions Using Information-Theoretical Algorithms", *Journal of Universal Computer Science*, pp. 153-172, 2002 (software at http://www.cs.umb.edu/~dana/GAClust/index.html)
[23] Hadoop, "Welcome to hadoop!", http://lucene.apache.org/hadoop/, 2007.