Feature Ranking Based on Information Gain for Large Classification Problems with MapReduce

Conference Paper · August 2015

DOI: 10.1109/Trustcom.2015.580

6 authors, including:

Eftim Zdravevski
Ss. Cyril and Methodius University in Skopje
152 PUBLICATIONS **1,260** CITATIONS
SEE PROFILE

Petre Lameski
Ss. Cyril and Methodius University in Skopje
99 PUBLICATIONS **848** CITATIONS
SEE PROFILE

Andrea Kulakov
Ss. Cyril and Methodius University in Skopje
82 PUBLICATIONS **724** CITATIONS
SEE PROFILE

Sonja Filiposka
Ss. Cyril and Methodius University in Skopje
135 PUBLICATIONS **709** CITATIONS
SEE PROFILE

Feature ranking based on information gain for large classification problems with MapReduce

Eftim Zdravevski
Faculty of Computer Science and Engineering
Ss.Cyril and Methodius University, Skopje, Macedonia
Email: eftim.zdravevski@finki.ukim.mk

Petre Lameski
Faculty of Computer Science and Engineering
Ss.Cyril and Methodius University, Skopje, Macedonia
Email: petre.lameski@finki.ukim.mk

Andrea Kulakov
Faculty of Computer Science and Engineering
Ss.Cyril and Methodius University, Skopje, Macedonia
Email: andrea.kulakov@finki.ukim.mk

Sonja Filiposka
Faculty of Computer Science and Engineering
Ss.Cyril and Methodius University, Skopje, Macedonia
Email: sonja.filiposka@finki.ukim.mk

Dimitar Trajanov
Faculty of Computer Science and Engineering
Ss.Cyril and Methodius University, Skopje, Macedonia
Email: dimitar.trajanov@finki.ukim.mk

Boro Jakimovski
Faculty of Computer Science and Engineering
Ss.Cyril and Methodius University, Skopje, Macedonia
Email: boro.jakimovski@finki.ukim.mk

***Abstract*—In this paper we propose a parallel implementation of the algorithm for computation of information gain with the services of the Hadoop ecosystem: MapReduce, HDFS, HBase and Yarn (MapReduce). The proposed implementation can be used for ranking features of large datasets and furthermore for feature selection. We use the Cloudera distribution of Hadoop and Pig Latin with Python user-defined functions. We have tested the implementation for ranking features of the FedCSIS AAIA'14 dataset. The presented results show the speed-up of the parallelization compared to a one-node cluster. To demonstrate the portability of the implementation we present results using an on-premises and Amazon AWS clusters.**

***Keywords*—Hadoop, MapReduce, information gain, parallelization, feature ranking**

I. Introduction

In the recent years companies, organizations and governments collect, process and analyze enormous volumes of data. For most of them the data is not only generated from their normal work, rather it a prerequisite for their success. As a result many companies have followed different ideas on how cope with the Big Data challenge. One idea was to scale-up hardware so it has more processing power that can handles the larger volumes of data, and it has proven to work up to a certain point. However, after this point is reached, this idea can not work. That lead to the other idea of distributing the computation and data storage to clusters. Even though this is not so new idea in general, it was not until about ten years that it started to gain popularity. Inspired by Google's approach described in the 2004 MapReduce [1] and 2006 Big Table [2], many other companies and open-source projects followed similar pathways developing different distributed systems. One of the most popular such systems is Apache Hadoop. It is open-source software that contains a set of algorithms for distributed processing, scheduling and storage of large datasets on computer clusters. It is well established framework and Hadoop Wiki [3] lists some of its prominent users like Yahoo, Facebook, Ebay, Adobe etc.

The available data can be automatically analyzed using machine learning in order to derive various conclusions, make predictions for unseen data, find patterns within the data etc. Many learning algorithms such as decision trees [4], neural networks [5], Naive Bayes [6, 7] and many others notably experience degrading performance when the datasets contain redundant or irrelevant features. Be that as it may, other algorithms like Support Vector Machines can successfully overcome the redundant features, but this comes at a cost. Namely, if the number of redundant or relevant features is large, it can lead to enormous increase of the computational time to the extent that the algorithm can no longer run in reasonable time. This phenomenon is confirmed with theoretical and empirical evidence in plenty of research papers, some of which are [8], [9] and [10]. The problem of feature selection [11, 9, 12] can be defined as the task of selection of subset features that describe the hypothesis at least as well as the original set. In [13] are given guidelines for feature selection and are introduced the most widely used methods.

The rest of this paper is organized as follows. In the following subsection I-A we describe the services in the Hadoop ecosystem and in subsection I-B we review how information gain can be used for feature selection. Then in section II we propose a MapReduce implementation for computing information gain, and in section III we present the obtained results. Finally, in section IV we present the conclusion from our work and our plans for further research.

This work was partially financed by the Faculty of Computer Science and Engineering at the Ss.Cyril and Methodius University, Skopje, Macedonia

A. Hadoop

The MapReduce programming paradigm [1, 14] is essential to the distributed computation and storage that Hadoop achieves. It consists of two phases: map and reduce. The first phase, map, threats the data processing problems as embarrassingly parallel by splitting the data into distinct subsets that can be processed in parallel. The reduce phase is second and final aggregates the output from the map phase and produces the final result. In other words, the map procedure can perform variety of operations like: reading, projecting, filtering and sorting data. The output from this phase is an intermediate result usually comprised of a list of keys and values. These are mandatory for the reduce phase. Hadoop makes sure that the output gets to the reduce procedures in proper order so it can perform some summary or aggregate operation. Even though the MapReduce model is fairly restricted, its simplicity is making it very suitable and efficient for extremely large-scale implementations across thousands of nodes.

Hadoop with its different services schedules, distributes, orchestrates and monitors communications, data transfers, while providing redundancy and fault tolerance. There are many services (i.e. subsystems) in Hadoop that aid accomplishing the previous goals, but three of them are most notable: YARN (MapReduce2), HDFS and HBase [15] [16].

The fundamental idea of YARN (i.e. Yet Another Resource Negotiator) [17] is to take care of resource management and job scheduling/monitoring, by splitting these responsibilities into separate daemons: a global ResourceManager and per-application ApplicationMaster. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system. The per-application ApplicationMaster is responsible for negotiating resources from the ResourceManager and working with the NodeManagers to execute and monitor the component tasks. In other words, YARN is responsible for allocating resources to the MapReduce jobs, distributing them to the most appropriate nodes, etc.

Hadoop Distributed File System (HDFS) [18] is a file system that provides scalable, fault-tolerant, distributed storage system that works closely with MapReduce which was designed to span large clusters of commodity servers. The combined resources of the servers within the cluster can easily grow with the demand. An HDFS cluster is comprised of a NameNode which manages the cluster metadata and DataNodes that store the data. The file content is split into large blocks (typically 128 megabytes), and each block of the file is independently replicated at multiple DataNodes. The blocks are stored on the local file system on the DataNodes.

HBase is an open source, non-relational, distributed database modeled after Google's BigTable. It runs on top of HDFS (Hadoop Distributed Filesystem), providing BigTable-like capabilities for Hadoop [19, 20, 21]. In other words, it provides a fault-tolerant way of storing large quantities of sparse data. HBase is a NoSQL (Not Only SQL) database and is not a direct replacement for a classic Relational SQL databases. Unlike traditional databases where normalization of data and splitting it into related tables is the substance of the design, designing HBase tables takes a different approach which analyzes the usage patterns. Motivations for this approach include simplicity of design, horizontal scaling, and finer control over availability. The data structures used by NoSQL databases, including HBase, differ from those used in relational databases, making some operations faster in NoSQL and others faster in relational databases. The particular suitability of a given NoSQL database depends on the problem it must solve. Tables in HBase can serve as the input and output for MapReduce jobs run in Hadoop. In the parlance of Eric Brewers CAP theorem, HBase is a CP type system (i.e. Consistent and Partition tolerant) [22].

Because of its simplicity the MapReduce programming model has become popular. By some users this model is preferred over the traditional SQL which is a high-level declarative approach. Be that as it may, the extreme simplicity of MapReduce leads to much low-level hacking to deal with the many-step, branching dataflows that arise in practice. Furthermore, users must repeatedly code standard operations such as join by hand. These practices increase development time, introduce bugs, harm readability, and may obstruct optimizations [23]. A group at Yahoo motivated by these repeatable tasks on daily basis, has developed a scripting language called Pig Latin. Pig is a high-level dataflow system that is a compromise between SQL and MapReduce. Pig offers constructs for data manipulation similar to SQL, which can be integrated in an explicit dataflow. Pig programs are compiled into sequences of MapReduce jobs, and executed in the Hadoop MapReduce environment [24].

B. Information gain for feature selection

Feature selection is a phase that is performed prior training a machine learning algorithm. There are two approaches for feature selection: filter and wrapper approach. The filtering approaches rank the features based on some metric. These methods are generally characterized by simplicity, scalability and solid empirical background. Because they rely on relatively simple metrics, they are memory and computationally efficient and can be applied on datasets with tens or even hundreds of thousands of features. Filter methods are independent of the machine learning algorithm that is going to be applied later on. They can further be categorized into two groups. The first group consists of methods that rank the features based on some measure of their individual predictive power: information value [25], information gain [26, 27, 28, 29], information gain ratio [28, 29], RELIEF [30, 31], entropy [32] etc. The second type of filter approaches consists of methods which analyze the subset of features based on some metric that describes the performance of the whole subset and not only the individual features [33]. Namely, the correlation-based approaches described in [34] and [35] fall into this type of methods. Important to realize is that they search for subsets of features that have low inter-correlation between them and high correlation to the target classification [36].

The wrapper approaches search for subsets of features that are useful for the classification or regression task at hand. They are based on evaluating the performance of different subsets of features using a machine learning algorithm [37, 10, 38]. The contribution of the subset of features is taken into consideration and the whole process is black-box like. In order to apply a particular wrapper method, one has to define: how will be the space of all possible feature subsets traversed; how will be the performance of the learning algorithm evaluated in order

to guide the search; and which learning algorithm to be used. The greatest advantage of these methods is their universality and independence of the domain of the data and task, while their main drawback is their computational complexity.

Information gain is a synonym for KullbackLeibler divergence and it can be used as a metric for ranking individual features [26, 27]. In order to calculate information gain, first the entropy $H(X)$ of the dataset should be calculated. Let X denote a set of training examples, and each of them x_i is in the form $(x_i^1, x_i^2, ..., x_i^k, y_i)$. Let each column (i.e. feature) be a discrete random variable that takes on values from set $V^j, j = 1..k$. Let the set of possible labels (i.e. classes) is L, such as $y_i \in L$. Then the entropy of the dataset X can be calculated with equation (1), where $p(l)$ is the probability of instance x_i to be labeled as l (i.e $y_i = l$) and is defined with equation (2).

$$H(X) = -\sum_{l \in L} p(l) \log p(l) \tag{1}$$

$$p(l) = \frac{|\{x_i \in X | y_i = y\}|}{|X|} \tag{2}$$

The information gain of the j-th feature of the dataset X can be calculated with equation (3), where first part in the sum is the probability of the instance x_i to have value v of the j-th feature. The second part in the sum in equation (3) denotes the entropy of the subset of instances of X that have the value v of the j-th feature.

$$IG(X,j) = H(X) - \sum_{v \in V^j} \frac{\left|\left\{x_i \in X | x_i^j = v\right\}\right|}{|X|} H(\left\{x_i \in X | x_i^j = v\right\}) \tag{3}$$

As shown by equations (1),(2) and (3), calculation of information gain of all features boils down to counting the number of instances per feature, value and class. After we compute these counts, we can calculate the probabilities and consequently calculate the information gain. It only remains to sort the features based on their information gain in descending order and decide how many features we want to retain in the filtered dataset. In the following section II we propose parallel implementation for calculating the information gain of each feature j in the dataset X.

II. Parallel computation of information gain

In this section we demonstrate how we can leverage the power and flexibility of Pig Latin for implicit implementation of MapReduce jobs. In particular we are going to use it for calculation of the information gain of all features of a dataset. Writing parallel computer programs generally is more difficult than writing sequential ones, because parallelization introduces several new types of potential software bugs of which most common are race conditions, communication and synchronization between the different subtasks. Choosing Hadoop as environment for parallelization of algorithms overcomes many of those challenges without needing the programmer to put much effort for solving those kinds of challenges.

The implementation we propose in this paper consists of several phases, shortly described in the following subsections. All except the first and last step are Pig Latin scripts that compile into a set of MapReduce jobs. Note that all of them can be merged in one large Pig script, however we have chosen to run them in separate scripts so we can monitor the performance of each step better and to have the ability to repeat only specific step multiple times.

A. Copy data to HDFS

This phase is a trivial Linux command that copies the dataset files from the Linux file system to HDFS. There are more advanced Hadoop services like Flume and Sqoop for integration with other systems, but they are not of interest for the task at hand.

B. Preprocess, format and load data in HBase

. This step reads the files from HDFS, and during it we can perform normalization discretization, data cleansing etc. and finally store the dataset in a HBase table. This phase is a MapReduce job without a Reduce phase because during the Map phase the data is read from HDFS, processed and stored in HBase without needing to arrange similar records together.

Before loading the dataset in HBase, we need to define the table structure and create it. Because HBase tables, unlike SQL tables, cannot have secondary indexes, the primary key (row key) needs to be designed according to the usage patterns of the table. For scientific use and for parallelizing machine learning algorithms, we need a simple design that allows uniform data distribution across nodes. In most scientific datasets the data instances (i.e. rows) do not have ids for their instances, or if they do they are not used for the actual machine learning. Assuming the dataset has N rows, and the Hadoop cluster has M regions, then we would like to distribute the data uniformly so each region gets N/M rows. This in turn means that we need to specify $M - 1$ split points when creating the table. If we calculate the modulo number of the id of the row and the number of regions, then each region would get almost the same number of rows. This design of the row key allows fast random reads and writes, and additionally it facilitates addition of new data to the table at a later time without needing to redesign the table for equally dispersed load across regions. Usually each node is a multi-threaded machine so it can serve more regions.

C. Calculate entropy for the whole dataset

In order to calculate the entropy of the whole dataset we need to first count the number of instances per class (first MapReduce job), and then to sum the class probabilities (second MapReduce job). This step actually compiles as two separate MapReduce jobs. Notably this is a very simple step and does not require parallelization because its complexity is $O(N)$, where N is the number of instances in the dataset. Moreover, if we are interested in only sorting the features without having the real information gain for each of them, then the entropy can be omitted from equation (3) and use an arbitrary non-negative constant instead of it.

D. Calculate instance counts per feature index, feature value and class

This step is the most computationally expensive step in the algorithm. The source code of this step is shown in listing 1. Each parameter starting with $ can be passed to Pig script when invoking it. Such parameters are the table names, number of features, index of the class value, number of padding digits etc. First we need to load the dataset from a HBase table (lines 3 through 6). Then we need to expand each row of the dataset (denoted as dictionary r) to pairs: (feature index, feature value, class, 1). This is performed in lines 7 through 8 with the user-defined function $FeatureValueClassCounts$. If the dataset has M rows and N columns than MxN tuples will be generated. These tuples are afterwards grouped by the key $(featureindex, featurevalue, class)$ in lines 9 through 12, and finally the count is stored in another table (lines 13 through 15). All of the code in listing 1 is compiled in one MapReduce job.

```
1  register '$udf_path' using jython as
       paddingUDF;
2  set default_parallel $parallel;
3  pfdata_tmp = LOAD '$table_dataset' USING
       org.apache.pig.backend.hadoop.hbase.
4    HBaseStorage('r:*', '-loadKey=true') AS
5       (rowkey:tuple(prefix_padded:chararray,
            id_padded:chararray, id:int),
6        r:map[]);
7  pfdata_short = FOREACH pfdata_tmp GENERATE
8    FLATTEN(paddingUDF.FeatureValueClassCounts(r,
         ((int)$num_features),
         ((int)$num_features_digits), '$label'));
9  feature_value_class_counts_group = GROUP
       pfdata_short BY (featureIndexPadded,
       featureValue, class);
10 feature_value_class_counts = FOREACH
       feature_value_class_counts_group GENERATE
11   group as rowkey,
12   SUM(pfdata_short.instanceCount) as
         instanceCount:double;
13 STORE feature_value_class_counts INTO
       '$table_feature_index_tmp' USING
14       org.apache.pig.backend.hadoop.hbase.
15       HBaseStorage('r:instanceCount');
```

Listing 1. Counting number of instances per feature index, feature value and class with Pig Latin

E. Calculate information gain

Having the counts calculated in the previous step II-D, this step only calculates the probabilities and entropies in (3) and stores this result in HBase or HDFS. Nevertheless, it is usually the second longest running step from this list.

F. Order features by information gain and export the results

If one decides to store the results from the previous step in HDFS directly and not to keep it for further use in HBase, then this step is obsolete. In addition, sorting of the features is a simple operation that does not require parallelization, even if the number of features is several hundred thousand. Nevertheless, the export of the list of features and/or the reduced dataset that contains only the selected features is a simple operation. We can specify the number of features to be retained by a parameter that can be provided to the Pig script.

G. Copy the results from HDFS

The final step is a trivial step that consists only of copying the list of selected features or the reduced dataset from HDFS to the Linux file system.

III. Results

In order to test the parallel implementation we wanted a dataset that is relatively large, so we can monitor the impact of the number of nodes in the cluster. On the other hand, we wanted to avoid the need of preprocessing like discretization or transformation of values because that can modify the original features and then a comparison with other research would be more difficult. The FedCSIS AAIA'14 data mining competition dataset [39] met those criteria. It is a sparse matrix that has 50000 instances and 11852 numeric features, most of which are have the value 0 or 1. It is a multi-label problem that has 3 binary labels, that can be merged with the powerset technique as used in [40] into one one-label multi-class problem that has 8 (s^3) possible classes.

We have tested the same dataset on three completely different Hadoop clusters. Each of them was running the same version of Apache Hadoop 2.3.0 (integrated in Cloudera CDH 5.3.0).

The first cluster (denoted by *Amazon32* in the remaining of the paper) was deployed on Amazon AWS. It contained a total of 32 nodes, each of them a m1.xlarge instance with 15GB RAM and 8 compute units (4 cores with 2 compute units each). From the 32 nodes, 8 were hosting HBase Region Servers and HDFS Data Nodes, 3 were specifically dedicated to HDFS Data Nodes and 19 were running only YARN. We acknowledge that this configuration may not be optimal for the current task, but we were given access to this cluster without the ability to modify its configuration. Therefore we have decided to run tests using up to 8 nodes at a time, because when using more it would be difficult to estimate the speedup.

The second cluster (denoted by *FCSE24* in the remaining of the paper) was on-premises at the Faculty of Computer Science and Engineering (FCSE) at the Ss.Cyril and Methodius University, Skopje, Macedonia. It had a total of 24 nodes, each of them an Intel Xeon Processor E5640 with 12M Cache, 2.66 GHz, 24 GB RAM, 4 cores and 8 threads. From them 21 were configured to run the following services: HBase Region Servers, HDFS DataNodes and YARN MapReduce NodeManagers. The remaining nodes were used for other Hadoop and Cloudera management services.

The third cluster (denoted by *FCSE65* in the remaining of the paper) was also on-premises and it was an extended version of the second, containing a total of 65 nodes, of which 59 were running the following services: HBase Region Servers, HDFS DataNodes and YARN MapReduce NodeManagers.

During our tests none of these clusters was executing other tasks. On all of them we ran tests with different table structures in order to simulate clusters with smaller sizes. By pre-splitting the HBase tables to a specific number of regions we were able to force Pig Latin to compile that number of map tasks for each job. For all these configurations we are computing the speedup of the parallelization against a cluster with one node. We are simulating the one-node cluster by configuring the tables to

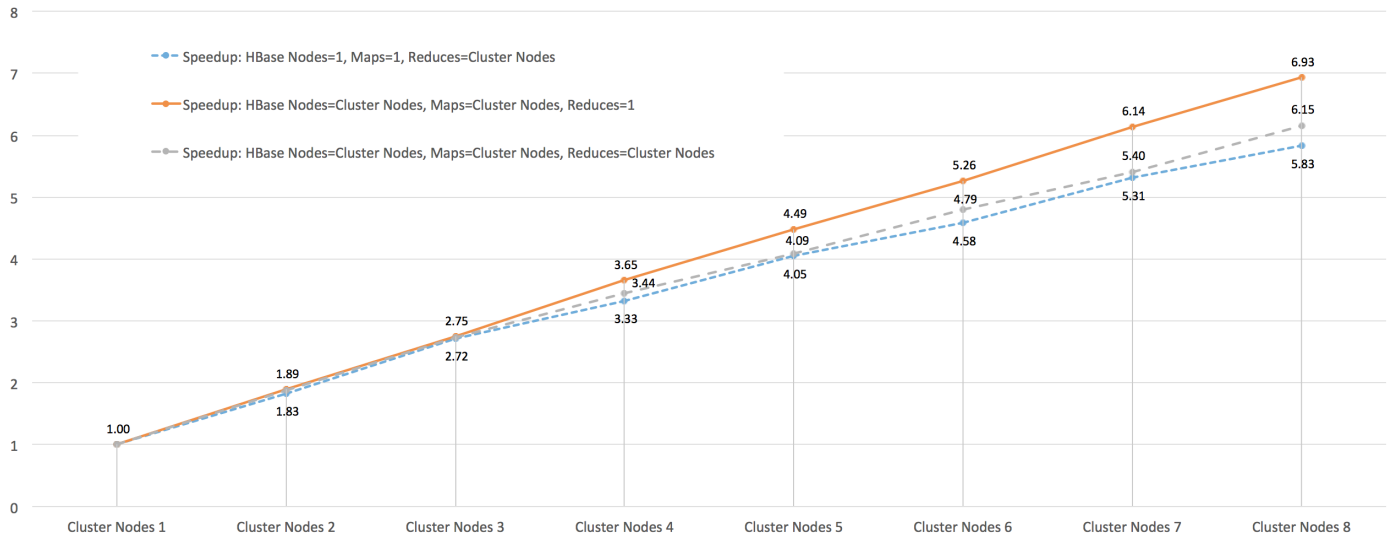

Fig. 1. Speedup depending on the number of active nodes, map and reduce tasks on the Amazon32 cluster

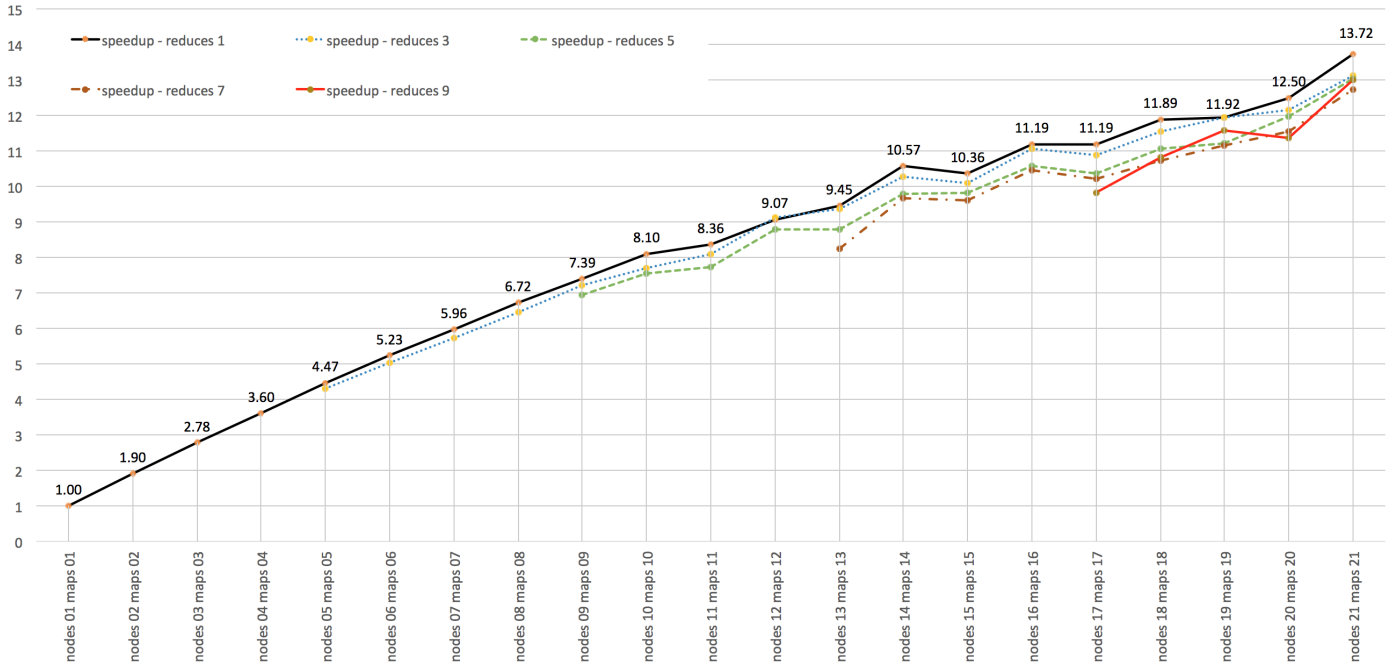

Fig. 2. Speedup depending on the number of active nodes, map and reduce tasks on the FCSE24 cluster

have only one region, thus all MapReduce jobs that read from those tables have only one map task. We have tested using different number of reduce task by setting a configuration property in the Pig scripts. The remaining of this section is divided in two, III-A containing summary information for all steps that are fast and did not benefit significantly from the parallelization, and III-B containing detailed information about the step described in II-D, which was the most computationally expensive. Table I shows the information gain of the top 50 features.

A. *Fast steps*

The dataset was stored in two files: one containing the data in EAV (entity attribute value) format, and one containing the labels. The total size of the files is 72 MB, so the first step described in subsection II-A was couple of seconds on all configurations. The step described in II-B was actually two MapReduce jobs. The first one was for loading the labels which took 58 to 70 seconds, and one for loading the data took 130 to 145 seconds on the on-premises and 175 to 195 seconds on the Amazon cluster. Calculating the entropy of the dataset, described in section II-C, took 118 to 152 seconds on both clusters. The step described in subsection II-D is analyzed in more detail in the following subsection III-B.

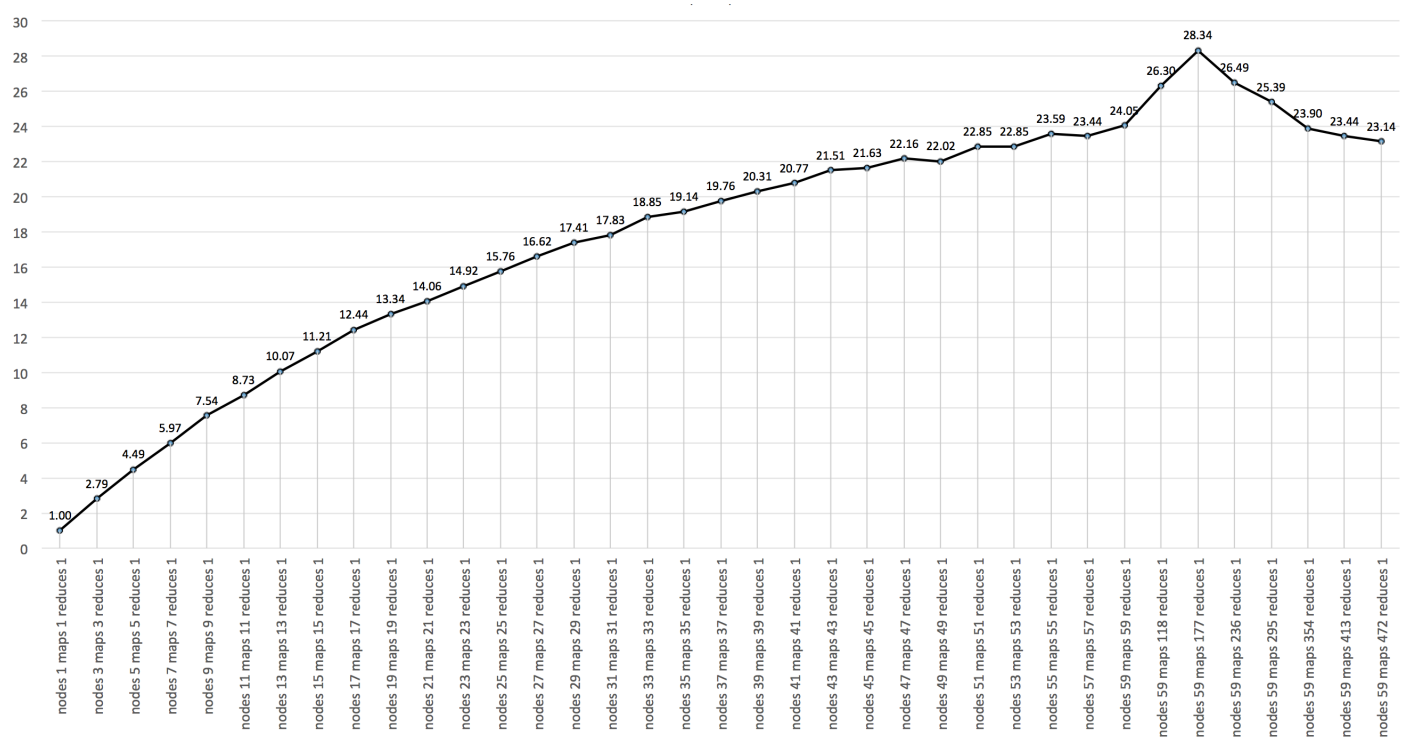

Fig. 3. Speedup depending on the number of active nodes and map tasks on the FCSE65 cluster

After it completed and stored the results in a pre-splitted table, calculating the information gain of each feature, described in subsection II-E, took 69 to 97 seconds on both clusters. The final step, the export of the list of information gain of all features, took 46 to 70 seconds. All of the MapReduce tasks had an overhead of up to 60 seconds for compilation of the Pig script, generating JAR files, distributing them on the cluster and negotiating resources.

B. Computationally expensive step - Calculating counts

The step described in subsection II-D was the most complicated and the speedup for it varied significantly depending on the cluster size and configuration. The remaining of this subsection describes details of the impact of the parallelization of this step and all listed speedups and durations are only for it.

We started testing on the Amazon32 cluster, trying to determine the impact of the number of nodes, maps and reduces. We have tried three options when trying to utilize the nodes of the cluster: use as much as possible nodes to run map tasks and have only one reduce task; use as much as possible nodes to run both map and reduce tasks; and use only one node for one map task and use all available nodes for reduce tasks. The speedup compared to the one-node cluster depending on the available nodes using these three options are shown on Fig. 1. It indicates that for this dataset is best to go have only one reduce phase, but use as many nodes as possible for the map tasks. This, in fact, makes sense because the work is performed during the map phase and during the reduce phase these results are grouped together. Having more than the default of one reduce task actually increases the duration because the partial results in each reduce task need to be merged together. The total duration of this step on the one-node cluster was 4732 seconds, while the quickest solution with speedup of 6.83 took 693 seconds.

Then we continued testing on the FCSE24 cluster, using the same approach. Additionally we tried using 1,3,5,7 or 9 reduce tasks, depending on the number of available nodes. Our intent was to confirm that using only one reduce task (the default value in Pig Latin) will be more appropriate for a dataset of this size. The charts shown on Fig. 2 indeed confirm this assumption. The greatest speedup was always achieved when using only one reduce task, regardless of the number of available nodes. The total duration of this step on the one-node cluster was 3637 seconds, while the quickest solution with speedup of 13.72 took 265 seconds.

Finally, we conducted experiments on the largest FSCSE65 cluster. Given that using only one reduce task gave best results on all configurations on the other two clusters, we have decided to use only one reduce task for all experiments on the FCSE65 cluster. Additionally, on this cluster we wanted to examine if using more map tasks than actual nodes will be beneficial. The motivation behind this is that all nodes are multi-core machines, so we wanted to see what happens when one HBase Region Server serves multiple table regions. The chart on Fig. 3 indeed shows this. When we have two or three times more map tasks (118, 177) than actual nodes (59), the performance is increased. However, when we further increase the number of map tasks to 236 (59 x 4), 295 (59 x 5), 354 (59 x 6), 413 (59 x 7) and 472 (59 x 8), the performance gradually degrades. One reasonable explanation for this is that as the number of map tasks gets larger, the operating system on the nodes needs to spend more time on task switching, swapping, while also needing to run many Hadoop and other

services in the background. In fact, using a reasonable amount of regions per region server is mentioned in [41] in Section 9.7 and also in [19] in Chapter 11. The total duration of this step on the one-node cluster was 3656 seconds, while the quickest solution with speedup of 28.34 took 129 seconds.

TABLE I. TOP 50 FEATURES ORDERED BY INFORMATION GAIN

Rank	Feature	InfoGain	Rank	Feature	InfoGain
1	11701	0.07422	26	7407	0.0256033
2	143	0.07000	27	11825	0.0249701
3	11832	0.06009	28	4505	0.0249698
4	1509	0.05154	29	11100	0.0249225
5	5909	0.04936	30	10331	0.0247915
6	8635	0.04539	31	7529	0.0247519
7	2182	0.04012	32	2274	0.0247061
8	865	0.03817	33	10261	0.0246147
9	6523	0.03817	34	7592	0.0245778
10	5827	0.03795	35	4319	0.0245677
11	5188	0.03467	36	1349	0.0245448
12	5513	0.03296	37	7405	0.0245288
13	6162	0.03294	38	11463	0.0245111
14	5967	0.03271	39	11000	0.0244753
15	2835	0.03223	40	6779	0.0240003
16	139	0.0318404	41	10428	0.0236240
17	9306	0.0318030	42	460	0.0235250
18	1772	0.0296594	43	7291	0.0233440
19	3257	0.0283169	44	8853	0.0232071
20	9848	0.0283169	45	2883	0.0232064
21	675	0.0282140	46	5925	0.0231852
22	73	0.0273487	47	8114	0.0225087
23	7275	0.0266788	48	5330	0.0223354
24	7419	0.0266100	49	1156	0.0219374
25	1244	0.0262854	50	2701	0.0218273

IV. CONCLUSION AND FUTURE WORK

In this paper we have proposed a parallel implementation of the metric information gain that can be used for feature selection. We have demonstrated how can we manually set the degree of parallelism by pre-splitting the HBase tables so they have optimal number of regions and even data distribution across regions. In order to verify the proposed approach based on Pig Latin and some Python-based user-defined functions we have performed some tests and have analyzed the speedup. In order to verify the correctness of the implementation we have compared the ranked features with the results obtained from WEKA.

In order to affirm the proposed parallelization, we need to measure its impact on different cluster configurations and with other publicly available datasets. In that manner, we also need to propose valid data transformation and normalization techniques so we can generalize the approach and make it available for datasets that contain non-discretized continuous or nominal features. Additionally we plan to parallelize other more advanced feature selection algorithms using a similar framework.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251254.1251264

[2] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 15–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267308.1267323

[3] "Hadoop wiki: List of institutions that are using hadoop for educational or production uses, howpublished = https://wiki.apache.org/hadoop/poweredby, note = Accessed: 2015-01-29."

[4] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN 1-55860-238-0

[5] T. M. Mitchell, *Machine Learning*, 1st ed. McGraw-Hill Science/Engineering/Math, 3 1997. ISBN 9780070428072. [Online]. Available: http://amazon.com/o/ASIN/0070428077/

[6] D. Mladenic and M. Grobelnik, "Feature selection for unbalanced class distribution and naive bayes," in *Proceedings of the Sixteenth International Conference on Machine Learning*, ser. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. ISBN 1-55860-612-2 pp. 258–267. [Online]. Available: http://dl.acm.org/citation.cfm?id=645528.657649

[7] R. O. Duda, *Pattern classification*, 2nd ed. New York: Wiley, 2001. ISBN 0471056693

[8] H. Almuallim and T. G. Dietterich, "Learning with many irrelevant features," in *Proceedings of the Ninth National Conference on Artificial Intelligence - Volume 2*, ser. AAAI'91. AAAI Press, 1991. ISBN 0-262-51059-6 pp. 547–552. [Online]. Available: http://dl.acm.org/citation.cfm?id=1865756.1865761

[9] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, vol. 97, no. 1–2, pp. 245 – 271, 1997. doi: http://dx.doi.org/10.1016/S0004-3702(97)00063-5 Relevance. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0004370297000635

[10] P. Langley, *Elements of machine learning*. San Francisco, Calif: Morgan Kaufmann, 1996. ISBN 1558603018

[11] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," in *Machine Learning: Proceedings of the Eleventh International Conference*. Morgan Kaufmann, 1994, pp. 121–129.

[12] B. Raman and T. R. Ioerger, "Instance based filter for feature selection," *Journal of Machine Learning Research*, vol. 1, no. 3, pp. 1–23, 2002.

[13] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, Mar. 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=944919.944968

[14] D. Miner, *MapReduce design patterns*. Sebastopol, CA: O'Reilly, 2013. ISBN 9781449327170

[15] A. Holmes, *Hadoop in practice*. Shelter Island, NY: Manning, 2012. ISBN 9781617290237 1617290238

[16] T. White, *Hadoop: the definitive guide*, 3rd ed. Beijing: O'Reilly, 2012. ISBN 9781449311520

[17] "Apache hadoop nextgen mapreduce (yarn)," http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html, accessed: 2015-01-29.

[18] "Hdfs architecture guide," http://hadoop.apache.org/docs/

r1.2.1/hdfs_design.html, accessed: 2015-01-29.

[19] L. George, *HBase the definitive guide*. Sebastopol, CA: O'Reilly, 2011. ISBN 9781449315771 1449315771. [Online]. Available: http://public.eblib.com/choice/publicfullrecord.aspx?p=769368

[20] Y. Jiang, *HBase administration cookbook master HBase configuration and administration for optimum database performance*. Birmingham: Packt Publishing, 2012. ISBN 9781849517157 1849517150 1849517142 9781849517140. [Online]. Available: http://site.ebrary.com/id/10598980

[21] N. Dimiduk and A. Khurana, *HBase in action*. Shelter Island, NY: Manning, 2013. ISBN 1617290521 9781617290527

[22] E. A. Brewer, "Towards robust distributed systems (abstract)," in *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '00. New York, NY, USA: ACM, 2000. doi: 10.1145/343477.343502. ISBN 1-58113-183-6 pp. 7–. [Online]. Available: http://doi.acm.org/10.1145/343477.343502

[23] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, "Building a high-level dataflow system on top of map-reduce: The pig experience," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1414–1425, Aug. 2009. doi: 10.14778/1687553.1687568. [Online]. Available: http://dx.doi.org/10.14778/1687553.1687568

[24] A. Gates, *Programming Pig*. Sebastopol: O'Reilly Media, 2011. ISBN 9781449317690 1449317693 9781449317683 1449317685. [Online]. Available: http://public.eblib.com/choice/publicfullrecord.aspx?p=801461

[25] R. Anderson, *The credit scoring toolkit: theory and practice for retail credit risk management and decision automation*. Oxford: Oxford University Press, 2007. ISBN 9780199226405

[26] T. M. Cover and J. A. Thomas, *Elements of information theory*, 2nd ed. Hoboken, NJ: Wiley, 2006. ISBN 9780471241959 0471241954 9780471241959

[27] C. Shang, M. Li, S. Feng, Q. Jiang, and J. Fan, "Feature selection via maximizing global information gain for text classification," *Knowledge-Based Systems*, vol. 54, no. 0, pp. 298 – 309, 2013. doi: http://dx.doi.org/10.1016/j.knosys.2013.09.019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950705113003067

[28] C. Lee and G. G. Lee, "Information gain and divergence-based feature selection for machine learning-based text categorization," *Inf. Process. Manage.*, vol. 42, no. 1, pp. 155–165, Jan. 2006. doi: 10.1016/j.ipm.2004.08.006. [Online]. Available: http://dx.doi.org/10.1016/j.ipm.2004.08.006

[29] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, pp. 79–86, 1951.

[30] K. Kira and L. A. Rendell, "A practical approach to feature selection," in *Proceedings of the Ninth International Workshop on Machine Learning*, ser. ML92. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992. ISBN 1-5586-247-X pp. 249–256. [Online]. Available: http://dl.acm.org/citation.cfm?id=141975.142034

[31] I. Kononenko, "Estimating attributes: Analysis and extensions of relief," in *Machine Learning: ECML-94*, ser. Lecture Notes in Computer Science, F. Bergadano and L. De Raedt, Eds. Springer Berlin Heidelberg, 1994, vol. 784, pp. 171–182. ISBN 978-3-540-57868-0. [Online]. Available: http://dx.doi.org/10.1007/3-540-57868-4_57

[32] T. Jebara and T. Jaakkola, "Feature selection and dualities in maximum entropy discrimination," in *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000. ISBN 1-55860-709-9 pp. 291–300. [Online]. Available: http://dl.acm.org/citation.cfm?id=2073946.2073981

[33] H. Liu and H. Motoda, *Feature Extraction, Construction and Selection a Data Mining Perspective*. Boston, MA: Springer US, 1998. ISBN 9781461557258 1461557259. [Online]. Available: http://dx.doi.org/10.1007/978-1-4615-5725-8

[34] M. A. Hall, "Correlation-based feature selection for machine learning," Ph.D. dissertation, The University of Waikato, 1999.

[35] L. Yu and H. Liu, "Feature selection for high-dimensional data: A fast correlation-based filter solution," in *ICML*, vol. 3, 2003, pp. 856–863.

[36] M. Dash, H. Liu, and H. Motoda, "Consistency based feature selection," in *Knowledge Discovery and Data Mining. Current Issues and New Applications*, ser. Lecture Notes in Computer Science, T. Terano, H. Liu, and A. Chen, Eds. Springer Berlin Heidelberg, 2000, vol. 1805, pp. 98–109. ISBN 978-3-540-67382-8. [Online]. Available: http://dx.doi.org/10.1007/3-540-45571-X_12

[37] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artif. Intell.*, vol. 97, no. 1-2, pp. 273–324, Dec. 1997. doi: 10.1016/S0004-3702(97)00043-X. [Online]. Available: http://dx.doi.org/10.1016/S0004-3702(97)00043-X

[38] P. Yang, W. Liu, B. Zhou, S. Chawla, and A. Zomaya, "Ensemble-based wrapper methods for feature selection and class imbalance learning," in *Advances in Knowledge Discovery and Data Mining*, ser. Lecture Notes in Computer Science, J. Pei, V. Tseng, L. Cao, H. Motoda, and G. Xu, Eds. Springer Berlin Heidelberg, 2013, vol. 7818, pp. 544–555. ISBN 978-3-642-37452-4. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37453-1_45

[39] "AAIA'14 data mining competition," https://fedcsis.org/2014/dm_competition, accessed: 2014-05-30.

[40] E. Zdravevski, P. Lameski, A. Kulakov, and D. Gjorgjevikj, "Feature selection and allocation to diverse subsets for multi-label learning problems with large datasets," in *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, Sept 2014. doi: 10.15439/2014F500 pp. 387–394.

[41] A. H. Team, "Apache HBase reference guide," http://hbase.apache.org/book.html, accessed: 2015-03-29.