Cloud-based clustering of text documents using the GHSOM algorithm on the GridGain platform

M. Sarnovsky*, Z. Ulbrik*

* Department of cybernetics and artificial intelligence, Faculty of electrical engineering and informatics, Technical University in Kosice, Letna 9/A, 042 00 Kosice, Slovak Republic
martin.sarnovsky@tuke.sk, zdeno.ulbrik@student.tuke.sk

Abstract—This paper provides an overview of our research activities aimed on efficient use of distributed computing concepts for text-mining tasks. Work presented within this paper describes the GHSOM (Growing Hierarchical Self-Organizing Maps) algorithm for clustering of text documents and proposes the design and implementation of distributed version of this approach. Proposed implementation is based on JBOWL framework as a base for text mining. For distribution we used MapReduce paradigm implemented within the GridGain framework, which was used as a cloud application platform. Experiments were performed on standard Reuters dataset and for testing purposes we decided to use a simple private cloud infrastructure.

I. INTRODUCTION AND RELATED WORK

Self-organizing maps [1] algorithm is one of the methods for non-hierarchical clustering of objects based on the principles of unsupervised competitive learning paradigm. This model provides mapping from high-dimensional feature space into (usually) two-dimensional output space called map, which consists of neurons characterized by n-dimensional weight vector (same dimension as input vectors of the objects). Many of the clustering text-mining systems are based on Self-organizing maps (SOM). Fundamental feature of SOM (and related architectures) is their topology preserve mapping. The basic problem of SOM is that they have fixed architecture. The model being closest to the SOM is the Growing Grid, introduced by Fritzke, where a SOM-like neural network grows dynamically during training. Another possibility is to use a hierarchical structure of independent SOM, where for every unit of a map a SOM is added to the next layer. This means architecture called Hierarchical Feature Map. The Growing hierarchical self-organizing map (GHSOM) [2] combines the benefits of this neural network models. It means hierarchical architecture where each layer is composed of independent SOM that adjust their size according to the requirements of the input data. More about other algorithms and related works and papers can be found in [6].

Particular methods of text clustering are time-consuming (especially when performed on large textual data collections) therefore use of the distributed computing paradigms can offer significant benefits. Implementation of these techniques allows us to perform text-mining tasks, such as text classification and text clustering in parallel/distributed fashion. Several approaches exist in this area with main aim to re-implement typical workflows into the distributed computing environment. Discovery Net[1] and GridMiner[2] projects dealt with similar types of tasks in the past, their goal was to design and implement data mining system based on service oriented architecture. WEEP[3] (Workflow Enactment Engine Project) was following project which result was a platform for data mining on the grid, easy to use workflow engine was implemented for WSRF (Web Services Resource Framework) services using WS-BPEL (Web Services Business Process Execution Language) as formalism for processing and execution of services. Pilot application was data mining on the medical data. Moreover, several other similar approaches already exists in this area, such as AWARD framework based on MapReduce approach (a programming model developed by Google for processing large data sets of data), software implementations of scientific workflows theory Pegasus[4], or Taverna[5], combination of Kepler approach for workflows management and Hadoop environment as an implementation of MapReduce method.

Use of grid and cloud infrastructure as a platform for distributed clustering and classification of text documents has been already investigated. Various methods including self-organizing maps for text clustering are described in [3]. In particular, GHSOM algorithm in particular was used in formal concept analysis for creation of generalized one-sided concept lattices [4, 10]. Computation complexity of such approach leads to distribution of GHSOM algorithm, which was introduced in [5].

Current trends within this area lead to interactive analysis of big data. Leader in this field is project Dremel[6] by Google that uses querying engine based on aggregation trees. This approach is used by services such as BigQuery and open source tools Drill. Combined with Pregel tool for processing of large quantities of graph data, these tools set current trends and to a great extent

1 http://www.discovery-on-the.net/
2 http://www.gridminer.org/
3 http://weep.gridminer.org/
4 http://pegasus.isi.edu/
5 http://www.taverna.org.uk
6 http://code.google.com/p/dremel/
overcome established approaches to processing of large amounts of data.

II. GHSOM ALGORITHM

Within this section we will describe GHSOM learning process. At the beginning of the learning process mean quantization error (deviation of all input vectors) is computed on at layer 0 (it could be seen as mean deviation of all input vectors with respect to a map with just one neuron - cluster). Then weight vector

\[ m_0 = [\eta_{01}, \eta_{02}, \ldots, \eta_{0d}]^T \]  

contains average values for every attribute from the whole collection of input vectors. Mean quantization error of layer 0 is:

\[ mqe_0 = \frac{1}{d} \cdot \| m_0 - x \| \]  

where \( d \) is the number of input vectors \( x \). Learning of GHSOM starts with the layer on level 1. This map is usually small. For every neuron \( i \) we need \( n \)-dimensional weight vector

\[ m_i = [\eta_{i1}, \eta_{i2}, \ldots, \eta_{in}]^T, m_i \in \mathbb{R}^n \]  

which is initialized randomly. Dimension \( n \) of these vectors has to be the same as dimension of input vectors. Learning of SOM is competitive process between neurons for better approximation of input vectors. Neuron with weight vector nearest to the input vector is the winner. Its weight vector as well as weight vectors of neurons in its neighborhood is adapted in order to decrease their difference to input vector. The grade of adaptation is controlled by learning parameter \( \alpha(t) \), which is decreasing during time of learning. Number of neighboring neurons, which are also adapted, is also decreasing with time. At the beginning of the learning process many of winner’s neighbors are adapting, but near the end of learning only the winner is adapted. Which neurons and how much are adapted is defined by the neighborhood function \( h_i(t) \), which is based on distance between winner \( c \) and current neuron \( i \) (in output space). As a combination of these principles we have the following learning rule for computing of weight vector \( m_i \):

\[ m_i(t + 1) = m_i(t) \cdot \alpha(t) \cdot h_i(t) \cdot [x(t) - m_i(t)] \]  

After some number of iterations, mean quantization error of map is computed using:

\[ MQE_m = \frac{1}{u} \cdot \sum_i mqe_i \]  

where \( u \) is number of neurons \( i \) at map \( m \) and \( mqe_i \) is mean quantization error of neuron \( i \) at the map \( m \). Every layer of the GHSOM is responsible for explaining some portion of the deviation of the input data as present in its preceding layer. This could be achieved by adding of new neurons into map on every layer in order to have suitable size. Maps on every level grow until the deviation present in the unit of its preceding layer is reduced to at least a fixed percentage \( \tau_m \). The smaller the parameter \( \tau_m \) is chosen, the larger will be the size of SOM. If for current map condition

\[ MQE_m \geq \tau_m \cdot mqe_0 \]  

is fulfilled, new row or column of neurons is added into map. It is added near the error neuron (neuron with largest error). Addition of row or column depends on position of most distant neighbor neuron to error neuron (new row or column is inserted between them; distance is computed in input space – weight vectors of neurons). Weight vectors of new neurons are usually initialized as average values of neighboring neurons. After such a neuron addition-learning parameters is setup to starting values and map is re-learned. When the learning of map on level 1 (or any other level) is finished, it means that

\[ MQE_m < \tau_m \cdot mqe_0 \]  

it is a time to expand neurons of the map to another level (if needed). Neurons, which have still high mean quantization error (comparing with \( mqe_0 \)), should be expanded and new map in next hierarchical level is created. Every neuron \( i \), which fulfils next condition, have to be expanded:

\[ mqe_i > \tau_u \cdot mqe_0 \]  

where \( \tau_u \) defines the rate of the variability of the neuron and variability of all input documents and controls the hierarchical expansion. Learning process follows for every new map identically. Only difference is that in every new submap only inputs from one expanded neuron of parent map are used for learning of its submap, and only fraction of quantization error of the parent map is going to be analyzed (concretely error of expanded neuron). GHSOM algorithm is finished when there is no neuron for expansion, or some predefined maximal depth of hierarchy is reached. Architecture of trained model then can be viewed as on Fig.1.

![Figure 1. Architecture of a trained GHSOM](image-url)
The size and structure of final model is influenced by two abovementioned parameters. The parameter $\tau_u$ specifies the desired quality of data representation while $\tau_m$ specifies the desired level of detail (higher values of this parameter leads to larger maps and deeper hierarchy).

Additional parameters of GHSOM algorithms are more user-oriented. If we want to have some visually good view of maps and whole hierarchy, we need to have chance to parameterised some additional things – maximal depth of the hierarchy (too deep hierarchy is sometimes very difficult to understand) and minimal number of instances for creation of new sub-map (map from 5 documents is usually not useful).

III. DESIGN AND IMPLEMENTATION OF DISTRIBUTED CLUSTERING ALGORITHM BASED ON GHSOM

Distributed implementation of algorithm GHSOM is built on top of the JBOWL library and uses the GridGain as a cloud application framework.

JBOWL (Java Bag-of-Words Library) is an original software system developed in Java to support information retrieval and text mining [6], [7]. The system is being developed as open source with the intention to provide an easy extensible, modular framework for pre-processing, indexing and further exploration of large text collections, as well as for creation and evaluation of supervised and unsupervised text-mining models. Jbowl is a Java library, which contains methods for preprocessing, classification, clustering (including GHSOM algorithm) and evaluation techniques. It provides a set of classes and interfaces that enable integration of various classifiers and clustering algorithms. Jbowl distinguishes between clustering algorithms (GHSOM, k-means) and clustering models (centroid-based clustering models, etc.). We used implementation of GHSOM algorithm, which was described in [8].

GridGain is Java based middleware for development of data processing applications in distributed environments [9]. It supports development of scalable data-intensive and high-performance distributed applications. Main benefit of GridGain is the fact, that it is independent from infrastructure and platform. This allows applications developed using the framework to be deployed on various types of cloud infrastructures. GridGain provides native support for Java, Scala and Groovy programming languages. Framework integrates two core technologies:

- Compute grid
- In-memory data grid

Computational grid technology provides in-memory MapReduce implementation for handling distribution of process logic. In-memory data grid on the other hand presents the capability to parallelize the data storage by storing partitioned data in memory closer to the application. In comparison to Hadoop implementation of MapReduce, GridGain uses its own implementation of this computing paradigm. It is designed specifically for real-time in-memory processing use cases and its main goal is to split a task into multiple sub-tasks, load balance those sub-tasks among available cluster nodes, execute them in parallel, then aggregate the results from those sub-tasks and return them to user. We used GridGain MapReduce as a platform for implementation of distributed classification trees for text classification based on Jbowl [11] and proposed more complex information system for text mining based on this framework [12].

Our distributed GHSOM implementation uses MapReduce model and is based on building of hierarchical sub-GHSOMs, which consist of hierarchically ordered maps of Growing SOM. Main idea is parallel execution of these clustering processes on worker nodes. Approach can be described easily (scheme of distribution is shown on Fig. 2).

1. On the master node deviation of input data and layer 1 map is computed, and neurons for expansion are chosen. Important is that before the beginning of learning, all necessary preprocessing steps are done and input collection is ready in vector representation based on tfidf terms weighting scheme. Then from the input collection related vectors are selected (which are needed for particular expanded neurons) and distributed on working nodes. Using GridGain methods current list of available worker nodes is retrieved.

2. Vectors are distributed and used as inputs for GHSOM algorithm, which runs on particular nodes in order to create hierarchical sub-model. When end condition is reached (maximal depth or nothing to expand), particularly created GHSOM sub-models are returned to the master node.

3. Returned parts of GHSOM model are merged (on the master node) into one final model. Assignment of clustering tasks to Grid nodes is following:

![Figure 2. Using MapReduce for GHSOM training](http://www.gridgain.com)
Let $h$ is number of neurons to be expanded and $u$ is number of available worker nodes, then

1. If $h \leq u$, into tasks queues of first $h$ nodes exactly one clustering task per queue is assigned.

2. If $h > u$, in first iteration $u$ clustering tasks are assigned to first $u$ nodes, in next iteration rest of the tasks is assigned similarly while is needed.

After assigned tasks are distributed, particular submodels are created on separate nodes. When worker node finishes all assigned tasks in queue, its work is finished. All submodels are returned to the master node and merged into the final model. Reference to “parent” map node of level 1 is set correctly to main map created at start of the process as well as references to "children" are correctly set to maps created on particular nodes. Then final merged model is saved as persistent serialized Java object (important for next usage).

### IV. EXPERIMENTS AND RESULTS

The main goal of the experiments was to prove, that the distribution process using MapReduce in GridGain could reduce the time needed to build the classification model. Experiments were performed using the Reuters-21578 dataset. It is a standard corpus for evaluation of text-mining methods. Dataset contains 12,902 documents. In the experiments we used the subset of the dataset containing documents published in year 1987. This corpus consisted of 7769 documents. Dataset was indexed and preprocessed using JBOWL methods including stopwords filtering, and prefixes and suffixes removal. Preprocessed dataset contained 2401 terms. Our approach was tested and evaluated on our testing cloud infrastructure. The testbed serves for testing purposes only and consists of twelve networked computers with deployed GridGain framework. Computers have Intel® Xeon® Processor W3550, 3.06 GHz CPU installed, 4 GB RAM memory and are connected via 1 GBit local network.

Experiments were performed with parameter $\alpha$ as 0.5, value of parameter $\tau_{m}$, was set to 0.6 (which enabled richer GHSOM structure). Main goal was to prove the scalability of implemented approach. We performed series of experiments using various number of worker nodes involved in the computation process. Particular experiments were repeated three times and values in graphs represents mean value of registered times achieved with particular number of nodes.

Achieved results of computation times are shown on Fig. 3. The experiments were performed on total six involved worker nodes. The results show improvements in computation times of the algorithm in distributed version, but also the fact that the time reduction is not linear. This is the result of unbalanced data distribution, different values of variance error in learning from particular data parts. Better optimization of worker nodes workload is interesting topic and will be investigated in the future experiments. Critical point in proposed distributed approach of GHSOM algorithm is the process of creation of level 1 map. This limit the reduction of computation times with current distribution strategy. Combination of parallel building of first layer map (e.g. using computational cluster) and then distribution of these maps on the worker nodes could be interesting approach.

### V. CONCLUSIONS AND FUTURE WORK

In the future our plan is to build on our previous results in area of distribution of classification and clustering tasks in distributed environment and its following solutions respectively, which dealt with implementation of cloud applications using the GridGain framework. Our goal is to design and implement the optimization methods for tasks distribution in cloud environment based on holistic approach. We will use the information about current state of cloud infrastructure as well as values of qualifying data factors. It is possible to monitor the data containing information about the state of cloud infrastructure and use them within the process of their distribution for task distribution and resource usage optimization. In this area, studying of approaches and criterions for choosing of computation distribution appear as interesting field of research. Those usually depend on numerous conditions and factors, including size of the corpus, available capacities, access to distributed resources (quantity of computational nodes, size of available system memory, network topology, etc.).

In scope of this task, our methodology will be based on definition of generalized workflow that will be driven at each step by the actual values of data determining factors, available environment, and target characteristics of particular tasks. On such basis it will be possible to determine which model, settings and other parameters of particular analytic task solution respectively (for example methods of data preprocessing, selection of algorithms and their settings, volume and attributes of computing distribution etc.) are the most suitable.

**ACKNOWLEDGMENT**

This work was supported by the Slovak VEGA Grants No. 1/1147/12.
REFERENCES