Grid-based Support for Different Text Mining Tasks

Martin Sarnovský, Peter Butka, Ján Paralič

Centre for Information Technologies
Department of Cybernetics and Artificial Intelligence
Faculty of Electrical Engineering and Informatics
Technical University of Košice
Letná 9, 04200 Košice, Slovakia
E-mail: martin.sarnovsky@tuke.sk, peter.butka@tuke.sk, jan.paralic@tuke.sk

Abstract: This paper provides an overview of our research activities aimed at efficient use of Grid infrastructure to solve various text mining tasks. Grid-enabling of various text mining tasks was mainly driven by increasing volume of processed data. Utilizing the Grid services approach therefore enables to perform various text mining scenarios and also open ways to design distributed modifications of existing methods. Especially, some parts of mining process can significantly benefit from decomposition paradigm, in particular in this study we present our approach to data-driven decomposition of decision tree building algorithm, clustering algorithm based on self-organizing maps and its application in conceptual model building task using the FCA-based algorithm. Work presented in this paper is rather to be considered as a 'proof of concept' for design and implementation of decomposition methods as we performed the experiments mostly on standard textual databases.

Keywords: Grid services, text mining, clustering, classification, formal concept analysis

1 Introduction

The process of knowledge discovery is one of the most important topics in scientific and business problems. Nowadays, when the information overload means a big problem, knowledge discovery algorithms applied on very large text document collections can help to solve numerous problems and as text is still premier source of information on the web, the role of text mining is increasing. However, data are often geographically distributed in various locations. One approach to face this problem is distributed computing - distributed text mining algorithms can offer an effective way to mine extremely large document collections.

Motivation of this work is to use the Grid computational capabilities to solve text mining tasks. Grid is a technology, that allows from geographically distributed computational and memory resources create a universal computing system with extreme performance and capacity [1]. Nowadays the Grid projects are built on protocols and services that enable applications to handle distributed computing resources as s single virtual machine.

Some of the methods are time-consuming and use of the Grid infrastructure can bring significant benefits. Implementation of text mining techniques in distributed environment allows us to perform text mining tasks, such as text classification, in parallel/distributed fashion.

Knowledge discovery in texts is a variation of a field called knowledge discovery in databases, that tries to find interesting patterns in data. It is a process of semiautomatic non-trivial extraction of previously unknown, potentially useful and non-explicit information from large textual document collection. A key element of text mining is to link extracted information together to form new facts or new hypotheses to be explored further by more conventional means of experimentation. While regular data mining extracts the patterns from structured databases, text mining deals with problem of natural language processing. The biggest difference between data mining and text mining is in the preprocessing phase. Preprocessing of text documents is completely different than in the case of databases; in general, it is necessary to find a suitable way to transform the text into an appropriate internal representation, which the mining algorithms can work with. One of the most common internal representations of document collections is Vector Space Model [2]. Text mining phase is the core process of knowledge discovery in text documents. There are several types of text mining tasks as follows:

- Text categorization: assigning the documents with pre-defined categories (e.g. decision trees induction).
- Text clustering: descriptive activity, which groups similar documents together (e.g. self-organizing maps).
- Concept mining: modelling and discovering of concepts, sometimes combines categorization and clustering approaches with concept/logic-based ideas in order to find concepts and their relations from text collections (e.g. formal concept analysis approach for building of concept hierarchy).
- Information retrieval: retrieving the documents relevant to the user's query.
- Information extraction: question answering.

It is very usual that in any text mining process first three types are basic elements in order to support also information retrieval/extraction. Main goal of our work is

to show how well-known methods for text categorization, text clustering and concept mining could be adopted in Grid (distributed) environment in order to achieve more robust and faster application of text mining tasks. In our case we have implemented and tested three candidates, one per each type. After briefly presenting related work in Section 2, we describe every method with all modifications we have used in Section 3. In Section 4 we provide proposal and implementation details of Grid-based support in every case, and describe our experiments, results achieved and their evaluation (with emphasis on distribution aspects) in Section 5. Finally, we sum up the main results in conclusions section.

2 Related Work

In this section, we briefly describe related projects that utilize the Grid to perform advanced knowledge discovery in textual documents. DiscoveryNet[1] provides a service-oriented computing model for knowledge discovery, allowing the user to connect to and use data analysis software as well as document collection that are made available online by third parties. The aim of this project is to develop a unified real-time e-Science text mining infrastructure that leverages the technologies and methods developed by the DiscoveryNet and myGrid[2] projects. Both projects have already developed complementary methods that enable the analysis and mining of information extracted from biomedical text data sources using Grid infrastructures, with myGrid developing methods based on linguistic analysis and DiscoveryNet developing methods based on data mining and statistical analysis. National Centre for Text Mining[3] is also involved in research activities covering the Grid based text mining. Primary goal of this project is also focused to develop an infrastructure for text mining, a framework comprised of high-performance database systems, text and data mining tools, and parallel computing. Our work, presented in this article is complementary to the previous projects. Some of our algorithms (classification and clustering tasks) have been used within the GridMiner project[4]. Moreover, the FCA approach as far as we know has not been approached in any of the projects listed above.

[1] www.discovery-on-the.net
[2] www.myGrid.org.uk
[3] www.cse.salford.ac.uk/nactem
[4] www.gridminer.org

3 Text Mining Algorithms

3.1 Classification Using Decision Trees

Text Classification is the problem of assigning a text document into one or more topic categories or classes based on document's content. Traditional approaches to classification problems usually consider only the uni-label classification problem. It means that each document in collection has associated one unique class label. This approach is typical for data mining classification tasks, but in a number of real-world text mining applications, we face the problem of assigning the document into more than one single category. One sample can be labelled with a set of classes, so techniques for the multi-label classification problem have to be explored. Especially in text mining tasks, it is likely that data belongs to multiple classes, for example in context of medical diagnosis, a disease may belong to multiple categories, genes may have multiple functions, etc. In general there are many ways to solve this problem. One approach is to use a multinomial classifier such as the Naive Bayes probabilistic classifier that is able to handle multi-class data. But most of commonly used classifiers (including decision trees) cannot handle multi-class data, so some modifications are needed. Most frequently used approach to deal with multi-label classification problem is to treat each category as a separate binary classification problem, which involves learning a number of different binary classifiers and use an output of these binary classifiers to determine the labels of a new example. In other words, each such problem answers the question, whether a document should be assigned to a particular class or not. In the work reported in this paper, we used the decision trees algorithm based on the Quinlan's C4.5 [3]. A decision tree classifier is a tree with internal nodes labelled by attributes (words), branches departing from them are labelled by tests on the weight that attribute has in the document, and leafs represent the categories [4]. Decision tree classifies the unknown example by recursively testing of weights in the internal nodes, until a leaf is reached. While this algorithm is not suitable to perform multi-label classification itself, we use the approach of constructing different binary tree for each category. The process of building many binary trees can be very time consuming when running sequentially, especially on huge document collections. Due to the fact that these binary classifiers are independent on each other, it is natural to find a suitable way how to parallelize the whole process. Growing of these binary trees is ideal for parallel execution on a set of distributed computing devices. Such a distribution might be desirable for extremely large textual document collections or large number of categories, which e.g. can be associated with a large number of binary classifiers.

3.2 Clustering Using Growing Hierarchical Self-Organizing Maps

Self-organizing maps (SOM) [5] 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). Specific feature of SOM-based algorithms is realization of topology preserving mapping. Neurons are ordered in some regular structure (e.g. usually it is simple two-dimensional Grid) representing output space. A distance measure used in this space could be e.g. Euclidean distance based on the coordinates of weight vectors of neurons in the output space. Mapping created by learning of SOM then has a feature that two vectors which are closed each other in the input space are also mapped onto closely located neurons in the output space. Training consists of two steps: presentation of input document at the network input and adaptation of weight vectors. Based on the activation of the network best candidate from the neurons on the map is used as winner (e.g. Euclidean distance is used to find lowest distance to input vector). Next, weight vector of the winner and its neighbourhood neurons (with descending influence) are adapted in order to decrease its distance to the input vector.

One of the disadvantages of SOM algorithm is that structure of the whole output space is defined apriori. It is possible to avoid these using modifications, which dynamically expand map according to needs of the input feature space. The problem of adapting map is that it could expand to really large Grid (so we get same result like in case of SOM structure with predefined larger size) and in some applications it is hard to interpret results usefully. This leads us to go for another modification in "hierarchical" dimension – algorithm called GHSOM (Growing Hierarchical Self Organizing Map) [6], where map expands in two different ways:

- Hierarchically – according to the data distribution of input vectors some neurons on a map with large number of input documents assigned to them should be independently clustered in separate maps (each of these neurons expands into a submap on lower level), this provides hierarchical decomposition and navigation in submaps (Hierarchical SOM part).
- Horizontally – change of size of particular (sub)maps according to requirements of the input space (as it is done by Growing SOM).

Algorithm GHSOM consists of these steps:

1 First, *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 $m_0 = [\eta_{01}, \eta_{02}, ..., \eta_{0n}]^T$ contains average values for every attribute from the whole collection of input vectors. Mean quantization error of layer 0 is simply:

$$mqe_0 = \frac{1}{d} \cdot \|m_0 - x\|,$$

where d is the number of input vectors x.

2 Learning of GHSOM starts with the layer on level 1. This map is usually small (e.g. 2x2). For every neuron i we need n-dimensional weight vector

$$m_i = [\eta_{i1}, \eta_{i2}, ..., \eta_{in}]^T, m_i \in \Re^n,$$

which is initialized randomly. Dimension n of these vectors has to be the same as dimension of input vectors.

3 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 $h_{ci}(t)$, 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:

$$m_i(t+1) = m_i(t) \cdot \alpha(t) \cdot h_{ci}(t) \cdot [x(t) - m_i(t)],$$

where x is actual input vector, i is current neuron and c is winner in iteration t.

4 After some number of iterations (parameter λ) 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, 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 τ_m. The smaller the parameter τ_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 are setup to starting values and map is re-learned.

5 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 τ_u is parameter for controlling of hierarchical expansion.

6 Learning process follows for every new map identically with steps 2 to 5. 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).

7 GHSOM algorithm is finished when there is no neuron for expansion, or some predefined maximal depth of hierarchy is reached.

To summarize, the growth process of the GHSOM is guided by just two parameters. The parameter τ_u specifies the desired quality of input data representation at the end of the training process in order to explain the input data in more detail (if needed). Contrary to that, the parameter τ_m specifies the desired level of detail that is to be shown in one SOM. Hence, the smaller τ_m the larger will be the emerging maps. Conversely, the larger τ_m the deeper will be the hierarchy.

3.3 Use of Formal Concept Analysis in Text Analysis

Formal Concept Analysis (FCA, [7]) is a theory of data analysis that identifies conceptual structures among data sets. FCA is able to identify and describe all concepts (extensionally) and discover their structure inform of conceptual lattice that can be e.g. graphically visualized. These formal structures present inherent structures among data that (in our case) can be understand as knowledge model – e.g. ontology. It is an explorative method for data analysis and provides nontrivial information about input data of two basic types – concept lattice and attribute implications. Concept is cluster of "similar" objects (similarity is based on presence of the same attributes); concepts are hierarchically organized (specific vs. general). Standard usage of FCA is based on binary data tables (object has/has not attribute) – crisp case.

Problem is that classic data table from textual documents contains real-valued attributes. Then we need some fuzzification of classic crisp method. One approach to one-sided fuzzification was presented in [8]. Concept lattice created from real-valued (fuzzy) attributes is called *one-sided fuzzy concept lattice*. The proposed algorithm for FCA discovery is computationally very expensive and provides a huge amount of concepts (if we use definition). One approach to solve the issue is based on the problem decomposition method (as was described in [9]). This paper describes one simple approach to creation of simple hierarchy of concept lattices. Starting set of documents is decomposed to smaller sets of similar documents with the use of clustering algorithm. Then particular concept lattices are built upon every cluster using FCA method and these FCA-based models are combined to simple hierarchy of concept lattices using agglomerative clustering algorithm. For our experiments we used GHSOM algorithm for finding of appropriate clusters, then ´Upper Neighbors´ FCA algorithm (as defined in [10]) was used for building of particular concept lattices. Finally, particular FCA models were labelled by some characteristic terms and simple agglomerative algorithm was used for clustering of local models, with the metric based on these characteristic lattices terms. This approach is easy to implement in distributed manner, where computing of local models can be distributed between nodes and then combined together. This will lead to reduction of time needed for building the concept model on one computer (sequential run). Next, we will shortly describe the idea of one-sided fuzzy concept lattice, process of data pre-clustering, concept lattices creation and algorithm for combination of concept lattices (introduced in [18]).

3.3.1 One-sided Fuzzy Concept Lattice

Let A (attributes) and B (objects) are non-empty sets and R is fuzzy relation on their Cartesian product, R: $A \times B \rightarrow [0,1]$. This relation represents real-valued table data with rows and columns as objects and attributes, respectively. In case of texts, object is document and attribute is term (word). Then $R(b,a)$ express a grade

in which the document b contains term a (or in text mining terminology – weight of term a in vector representation of document b).

Now we can define mapping τ: $P(B) \rightarrow {}^{A}[0,1]$ which assigns to every set X of elements of B function $\tau(X)$ with value in point $a \in A$ (P – power set):

$$\tau(X)(a) = \min\{R(a,b): b \in X\},$$

i.e. this function assigns to every attribute the least of such values. This means that objects from X have this attribute at least in such grade.

Another (backward) mapping σ: ${}^{A}[0,1] \rightarrow P(B)$ then simply assigns to every function f: $A \rightarrow [0,1]$ a set:

$$\sigma(f) = \{b \in B: (\forall a \in A)\, R(a,b) \geq f(a)\},$$

i.e. those attributes, which have all values at least in grade set by the function f (these attributes the function of their fuzzy-membership to objects dominates over f). From properties of mappings we can see that the pair $<\tau,\sigma>$ is Galois connection, i.e. $\forall\, X \subseteq B$ and $f \in {}^{A}[0,1]$ holds $f \leq \tau(X)$ iff $X \subseteq \sigma(f)$.

Now we can define mapping cl: $P(B) \rightarrow P(B)$ as the composition of the mappings τ and σ, i.e. $\forall\, X \subseteq B$: $cl(X) = \sigma(\tau(X))$. Because conditions $X \subseteq cl(X)$, $X_1 \subseteq X_2 \rightarrow cl(X_1) \subseteq \mathrm{cl}(X_2)$ and $cl(X) \subseteq cl(cl(X))$ are fulfilled, cl is a closure operator.

As in crisp case, concepts are subsets $X \subseteq B$ for which $X = cl(X)$. Such pair $<X,\tau(X)>$ is called *(one-sided) fuzzy concept* (X – extent of this concept, $\tau(X)$ – intent of this concept). Then the set of all concepts, i.e. $L = \{X \in P(B): X = cl(X)\}$, ordered by inclusion is a lattice called *one-sided fuzzy concept lattice*, operation of which are defined as following: $X_1 \wedge X_2 = X_1 \cap X_2$ and $X_1 \vee X_2 = cl(X_1 \cup X_2)$. In next parts we will use only term concept lattice, but we mean always one-sided fuzzy concept lattice presented above.

3.3.2 Proposed Approach for Using of Problem Decomposition Method

In our case, FCA can be used to create hierarchy of concepts and relations between these concepts. Problems with use of this method in textual documents domain is time-consuming computation of concepts and hard interpretability of huge amounts of concepts. Solution can be combination with other algorithms like clustering algorithms. Problem decomposition approach can be seen on Fig. 1 [9].

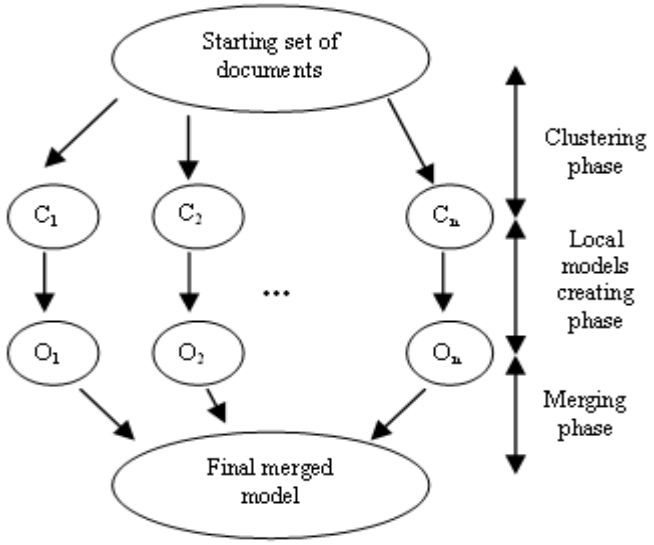

Figure 1

This diagram presents general scheme of the reduction-based conceptual model creation step. In clustering phase input dataset is divided in many smaller training sets. Then particular model O_i using FCA is created from every cluster C_i. Finally, all local models are merged together in the last phase of model generation step.

"Pre-clustering" of input set of documents can be viewed as reduction step where interesting groups of similar documents are found. The reduction step is based on filtering of terms that these objects (inside cluster) do not contain cooperatively. This step is top-down reduction problem (divide-and-conquer) approach to conceptual model extraction phase. Every cluster has independent training set (with reduced cardinality of weight vector), for each one a small concept lattice is built with a help of fuzzy FCA approach. Small models are merged then together and whole conceptual model from tested collection is finally created.

Important steps of our implementation are (more detailed description of every step is provided in [18]):

1 We use GHSOM clustering method for dividing the initial large set of documents into a hierarchy of clusters.

2 Find local concept lattices for every cluster of similar documents in the resulted GHSOM ´leafs´ (neuron without sub-map, in the end of expansion), i.e. in created hierarchy of maps particular one-sided concept lattices are built upon documents using ´Upper Neighbors´ algorithm (as presented in [10], updated for real-valued attributes). Before creation of the whole concept lattice documents are tested through attributes, if value of some attribute is lower than some threshold, value of attribute is set to zero.

This is inspired by work presented in [11] and is very useful for reduction of number of terms in concept lattice description. If we have higher concept in hierarchy of lattices, the number of concept's terms and weights is smaller. Terms with non-zero weights can be used as characteristic terms of actual concept (set of documents in concept).

3 Every concept lattice then can be presented as hierarchy of concepts characterized by some terms. Because we needed some description of lattice for merging of lattice to one model, we extracted terms from particular lattices and created their representation based on these terms. A weight of descriptive terms was based on level of terms in hierarchy (of course, important was highest occurrence of term). Then terms can be used for characterization of particular lattices and for clustering based on some metric.

4 Merging phase is based on clustering of lattices. First, we created one node for every local hierarchy (for every concept lattice), which contains list of documents, list of characteristic terms (sorted by value of weights), vector of terms weight's values (also in normalized type). Particular nodes are then compared using vectors of terms' weights, so vectors are normalized into interval <0,1>. After this step differences between numbers of documents in particular nodes are respected. Comparison of lattices is used in process of agglomerative clustering of these nodes (for detailed description of algorithm see [18]).

Final hierarchy contains nodes with list of documents in it and the sorted list of characteristic terms of nodes. Every node has link to upper node and list of lower nodes. ´Leaf´ nodes of hierarchy contain link on the particular local concept lattices.

4 Distributed Support for Text Mining Algorithms

4.1 Tools and Technologies for Grid-based Support

JBOWL - (Java Bag-of-Words Library) [12] is an original software system developed in Java to support information retrieval and text mining. 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 supports the document preprocessing, building the text mining model and evaluation of the model. It provides a set of classes and interfaces that enable integration of various classifiers. JBOWL

distinguishes between text mining algorithms (SVM, SOM, linear perceptron) and text mining models (rule based classifiers, classification trees, maps, etc.).

GridMiner [13] is a framework for implementing data mining services in the Grid environment. It provides three layered architecture utilizing a set of services and web applications to support all phases of data mining process. The system provides a graphical user interface that hides the complexity of the Grid, but still offers the possibility to interfere with the data mining process, control the tasks and visualize the results. GridMiner is being developed on top of the Globus Toolkit.

4.2 Distributed Trees Induction

The interface of the sequential and distributed versions of the service defines two main methods needed to build final model: *BuildTextModel* and *BuildClassificationModel*. While the first one is implemented as a pure sequential method, the second one can build the final model distributing the partial binary classifiers [14, 15]. This behaviour of the service depends on its configuration. Moreover, other methods were implemented to provide term reduction and model evaluation, but these methods were not used during the performance evaluation experiments discussed in the next section.

1 *BuildTextModel* - This method creates the Text Model from the documents in the collection. The model contains a document-term matrix created using TF-IDF weighting, which interprets local and global aspects of the terms in collection. The input of the method is a parameter specifying the text model properties and the location of the input collection.

2 *BuildClassificationModel* - The Classification Model, as the result of the decision tree classifier, is a set of decision trees or decision rules for each category. This service method creates such a model from the document-term matrix created in the previous method. The sequential version builds the model for all categories and stores it in one file. The process of building the model iterates over a list of categories and for each of them creates a binary decision tree (based on tree algorithm described in Chapter 3). The distributed version performs the same, but it distributes the work of building individual trees onto other services, so called workers, where partial models containing only trees of dedicated categories are created. These partial models are collected and merged into the final classification model by the master node and stored in the binary file, which can be passed to a visualization service.

4.3 Distributive Approach in Learning of GHSOM Model

Distributed algorithm GHSOM is implemented as service in GridMiner system Grid layer using Jbowl (Java 1.5). Implementation contains distributed algorithm together with preprocessing of text documents and visualization of output model. After creation of first map several new clustering processes are started – building of hierarchical sub-GHSOMs, which consist of hierarchically ordered maps of Growing SOM. Main idea is parallel execution of these clustering processes on working nodes of Grid. Approach can be described easily (scheme of distribution is shown on Fig. 2) [16]:

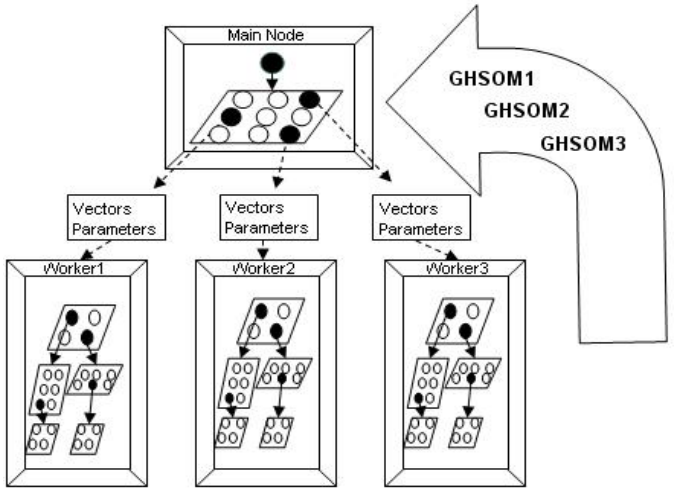

Figure 2
Distribution scheme of GHSOM algorithm in Grid environment

1 On master node deviation of input data and layer 1 map is computed, and neurons for expansions are chosen (as described in Section 3.2). Important is that before start 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 GridMiner methods current list of available working nodes is retrieved. Number of nodes is important parameter for distribution.

2 Distributed vectors are then used as inputs for GHSOM algorithm which runs on particular nodes in order to create hierarchical submodel. When end condition is reached (maximal depth or nothing to expand), particularly created GHSOM submodels are returned to the main node.

3 Returned parts of GHSOM model are merged (on the main node) into one complete model.

Every clustering task contains identifier of neuron within layer 1 map, list of (identifiers of) input vectors mapped on this neuron and parameters of GHSOM algorithm. Assignment of clustering tasks to Grid nodes is following:

Let h is number of neurons to be expanded and u is number of available working nodes on Grid, 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 node finishes all tasks in queue, his work is finished. If all submodels are returned to the master node, merging of model finishes whole process. 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).

4.4 Combination of Local FCA Models Using Distributed Service-based Architecture

The implemented algorithm for distribution of FCA-based algorithm (presented in Section 3.3) on the Grid can be divided into two basic fragments - the server and client (worker) side. The method that we have designed, implemented and tested is sketched on Figure 3. The method works as follows.

In the first step, master node performes the following tasks: document pre-processing (tokenization, elimination of stop-words, stemming, term selection), document clustering (use of GHSOM algorithm), assigning each cluster (each map including clusters, neurons) to client. Second step is based on the client side (worker nodes) where each client will get particular clusters, and then local FCA model is created on each client followed by sending of local FCA hierarchies to master node. Last step is again performed on server side (master node), in this case server receives local FCA hierarchies from all clients and final merged FCA-based model is built.

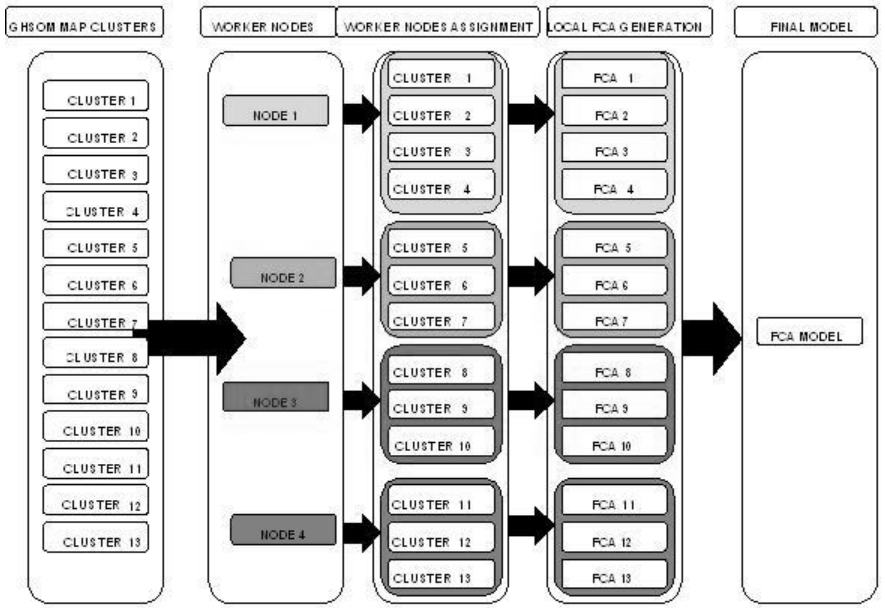

Figure 3
Distribution scheme for FCA-based problem decomposition method

5 Experiments

Two different data collections of text documents were used in this part of work. We used the collection "TIMES 60" which contains 420 articles from Times newspaper and Reuters ModApte dataset, which is standard corpus of textual documents that contains 12,902 documents. After pre-processing it contained 7769 documents and 2401 terms. We performed the experiments with the distributed algorithms on different workstations. In general, all of the workstations were Sun machines with different performance and different memory. Differences between types of text mining tasks will be emphasised.

5.1 Experiments with Decision Trees Induction

In this section, we present experiments performed on the local area network of the Institute of Scientific Computing in Vienna. As the experimental test bed, we used five workstations Sun Blade 1500, 1062 MHz Sparc CPU, 1.5 GB RAM connected by a 100 MBit network.

The main goal of the experiments was to prove, that the distribution of processes mentioned above, can reduce the time needed to construct the classification model. We started the experiments using the sequential version of the service, in order to compare the sequential version with the distributed one. The time to build the final classification model on a single machine using the ModApte dataset was measured three times and its mean value was 32.5 minutes. Then we performed the first series of the distributed service tests without using any optimization of distribution of categories to the worker nodes. According to the number of worker nodes, the master node assigned the equal number of categories to each worker node. The results show us the speedup of building the classification model using multiple nodes (see also Figure 4).

The detailed examination of the results and of the document collection proved that the time to build a complete classification model is significantly influenced by the working time of the first node. Examination of the dataset and workload of particular workers showed us that the first node always received a set of categories with the highest frequency of occurrences in the collection. It means that other worker nodes always finished the building of their partial models in a shorter time than the first one. It is caused by non-linear distribution of category occurrences in this collection. The most frequent category (category number 14) occurs in 2780 documents and it was always assigned to the first worker node. That was the reason, why the first worker node used much longer time to build-up the partial model.

After the first series of tests, we implemented the optimization of distribution of the categories to the worker nodes according to the frequency of category occurrences in the documents. Categories were sorted by this frequency and distributed to the worker nodes according to their frequency of occurrence, what means that each node was assigned with equal number of categories, but with a similar frequency of their occurrences. We run the same set of the experiments as in the first series and the results showed us more significant speedup using less worker nodes, see optimized bars in Figure 4. The best performance results were achieved using optimized distribution on 5 worker nodes (5.425 minutes), which was comparing to single machine computing time (32.5 minutes) almost 6 times faster. The minimal time to complete classification model is limited by the time of processing of the most frequent category - if this is assigned to a single worker node.

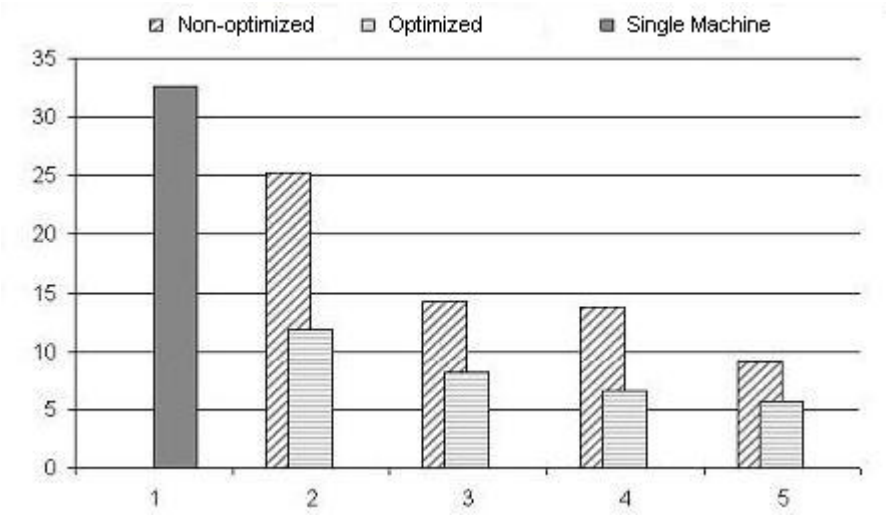

Figure 4
Experiments with distributed decision trees induction

5.2 Experiments with Distribution of GHSOM Maps Creation

The goal of experiments was in comparison of time complexity between sequential and distributed version of GHSOM algorithm. Experiments were realized in network of servers and workstations under usual working conditions (they were at the same time used also by other services). In order to get more precise results experiments are averaged from three identical tests. Number of computing nodes and parameter τ_m of GHSOM were changed during experiments.

Distributed version worked in testing Grid environment, which consisted of master server (4 x UltraSPARC-III 750 MHz, 8 GB RAM) and 6 SUN workstations, 100 Mbit/s network, data collections Times60 (420 documents) and Reuters-21578 (12 902 documents).

Experiments on Times collection were realized with τ_m 0.3, 0.6 and 0.8. First we started with sequential runs on one node. And then we tested distributed version for 0.3 τ_m with 2, 3, 4, 5 and 6 nodes (12 expanded neurons on layer 1 map). For parameter set to 0.6 and 0.8 only 2, 3 and 4 Grid nodes, because there were only 4 expanded neurons on layer 1 map. Graphical results of resulting computation times are shown on Figure 5.

For experiments with Reuters collection only τ_m with 0.6 and 0.8 was used and maximum number of nodes was 4. Results are shown on Figure 6.

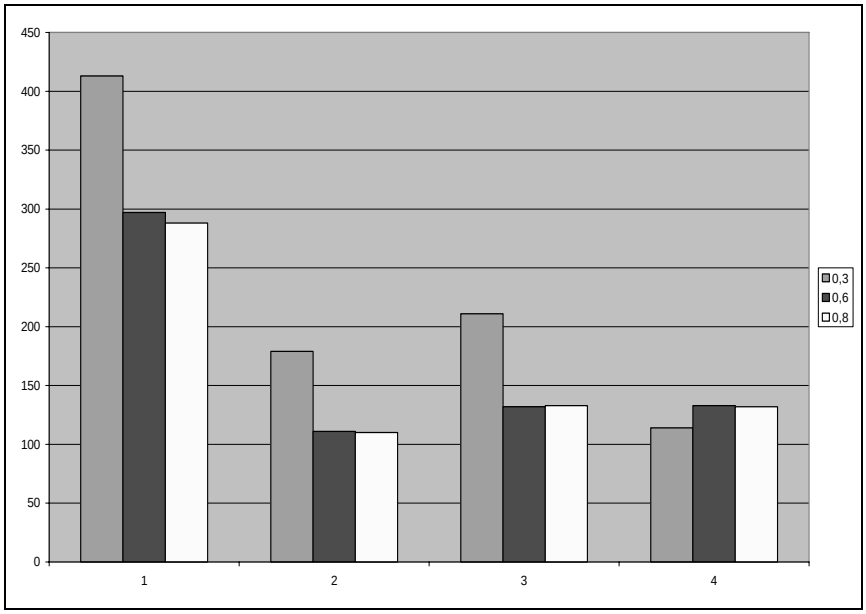

Figure 5
Graph of times (in seconds, y axis) for Times 60 collection for different number of nodes (x axis, max 4 nodes) with different values of τ_m parameter (legend – values 0.3, 0.6, 0.8)

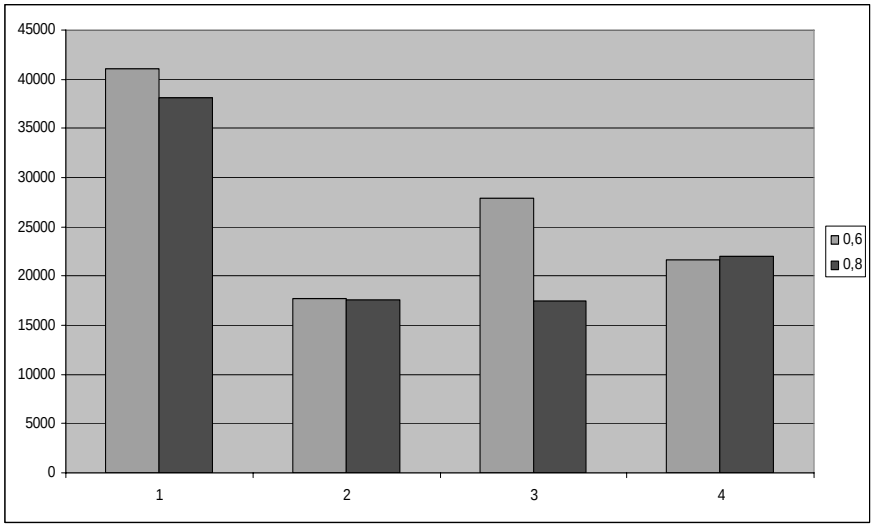

Figure 6
Graph of times (in seconds, y axis) for Reuters collection for different number of nodes (x axis, max 4 nodes) with different values of τ_m parameter (legend – values 0.6, 0.8)

The results of the experiments show interesting improvements in computation times of the algorithm in distributed version, but also the fact that addition of more worker nodes sometimes does not lead to better time reduction. The reasons could be unbalanced distribution of data for worker nodes, different values of variance error in learning from particular data parts and different computational power of Grid worker nodes. Better optimization of workers usage should be interesting for the next experiments. Critical point in such distributed version of GHSOM algorithm is creation of level 1 map (very often it is more then 50% of computing time). This means that further reductions of computation times are not possible with current distribution strategy. Combination of parallel building of first layer map (e.g. using computational cluster) and then distribution of this maps on the Grid could be helpful for another reduction of time complexity.

5.3 Experiments with FCA-based Distributed Approach

Again, both data collections of text documents were used in this part of work. Our main goal was to compare time consumption of the algorithm depending on the number of worker nodes. We used 9 different workstations deployed on the Grid. We performed various experiments with different values of *threshold* parameter 0.03, 0.05, 0.07 and 0.1. For each value of the threshold parameter we performed three runs of algorithm, final execution time of the algorithm was computed by averaging times of all three runs. The results of the experiments are depicted in the graphs.

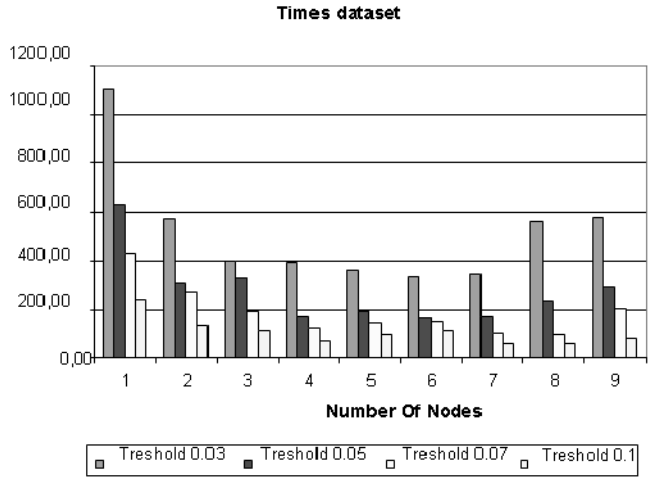

Figure 7
Experiment results on the Times dataset

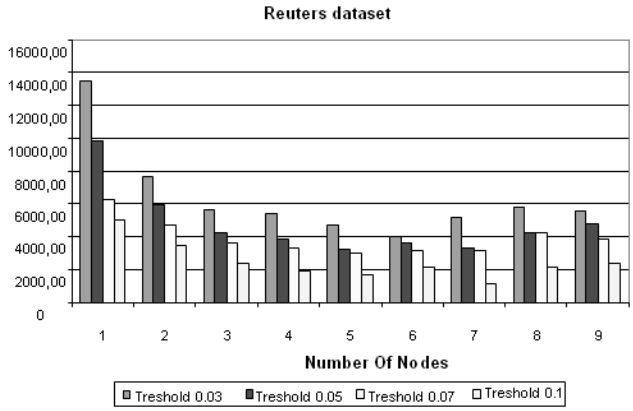

Figure 8
Experiment results on the Reuters dataset

We have compared number of the working nodes and its influence to the computing time of the merged FCA model generation. The results of experiments show that our distributed version has leads to clear reduction of computation time (even for small number of nodes). But the experiments in real environment also showed that increasing number of nodes leads (from some moment) to decreasing time complexity reduction. It is caused by heterogeneous computational power of involved worker nodes as well as their concurrent use by other, non-experimental tasks. More effective should be to optimize distribution of work according to the actual performance of particular nodes, i.e. dynamic distribution.

5.4 Discussion

Several questions could arise from the description of our grid-based approach. When discussing the difference between sequential and parallel running of tasks we have to emphasis that our approach cannot produce any information loss due to character of the decomposition and computing of the models. Text-mining results of the sequential and parallel run are therefore identical. Main aim of work presented in the paper is to provide the proof of concept for potential speedup of tasks computing that we have implemented. In this case proof of concept means that our approach is demonstrated on rather smaller number of computing units and applied on standard collections of text documents (Reuters) in order to prove that presented approach to distribution is scalable. Scalability of distributed algorithms is important issue, especially if applying them within large-scale distributed environment. Our experiments were aimed at algorithms behaviour in the testbed environment by increasing number of involved computing resources for which the selected datasets are sufficient. We assumed, that if our approach is proved to be scalable on our testbed, it can be applied in more large-scale fashion

(on larger datasets and using more computing resources) gaining similar results (speedup). Of course, bounds of scalability of these methods are data and environment-dependent. Our assumption based on experiments is that if data collections have approximately normal distribution of documents among the categories (in case of classification task), the scalability of our approach should be maintained also in larger scale. Similar assumptions can be expected also in case of clustering and FCA approaches. We have chosen three completely different text mining tasks, trying to cover text classification, text clustering and formal concept analysis tasks. Particular algorithms were chosen with respect to possibility of distributed implementation as our approach cannot be applied on several other algorithms.

Conclusions

Main aim of this article was to present the idea of suitable modifications and implementation of text mining services into the distributed Grid environment. Integration of the text mining services into the distributed service oriented system enables a plenty of various possibilities for building the distributed text mining scenarios. Using the Grid as a platform it is possible to access different distributed document collections and perform various text mining tasks. In this paper we focused on how to effectively use the Grid infrastructure by means of suitable decomposition of algorithms into the distributed fashion. We proposed three data-driven distributed methods for text mining: induction of the decision trees, GHSOM clustering algorithm and FCA method. One of the main goals of this work was to provide the proof of concept that proposed approach is well suited for distribution on the Grid, and results showed, that Grid-enabling of text mining process should considerably decrease time costs in comparison with sequential versions of these algorithms.

Acknowledgement

The work presented in the paper is supported by the Slovak Grant Agency of Ministry of Education and Academy of Science of the Slovak Republic within the project No. 1/4074/07 ”Methods for annotation, search, creation, and accessing knowledge employing metadata for semantic description of knowledge”, and by the Slovak Research and Development Agency under the contracts No. RPEU-0011-06 (project PoZnaŤ) and No. APVV-0391-06 (project SEMCO-WS).

References

[1] Foster I., Kesselman, C.: Computional Grids, The Grid – Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999

[2] Luhn, H. P.: A Statistical Approach to Mechanized Encoding and Searching of Literary Information, in IBM Journal of Research and Developement, 4:309-317, 1957

[3] Quinlan, J. R.: Learning First-Order Definitions of Functions, in Journal of Artificial Intelligence Research, 5:139-161, 1996

[4] Apte, C., Damerau, F., Weiss, S. M.: Towards Language Independent Automated Learning of Text Categorisation Models, in Research and Development in Information Retrieval, pp. 23-30, 1994

[5] Kohonen, T.: Self-Organizing Maps, Springer-Verlag, Berlin, 1995

[6] Ditttenbach, M., Rauber, A., Merkl, D.: The Growing Hierarchical Self-Organizing Map, in Proceedings of International Joint Conference on Neural Networks, Como, Italy, 2000

[7] Ganter, B., Wille, R.: Formal Concept Analysis, Springer Verlag, 1997

[8] Krajci, S.: Clustering Algorithm Via Fuzzy Concepts, in Proceedings of DATESO 2003 workshop, Ostrava, Czech Republic, 2003, pp. 94-100

[9] Butka, P. Combination of Problem Reduction Techniques and Fuzzy FCA Approach for Building of Conceptual Models from Textual Documents (in Slovak), in Znalosti 2006, 5th annual conference, Ostrava, Czech Republic, 2006, pp. 71-82

[10] Belohlavek, R.: Concept Lattices and Formal Concept Analysis (in Czech), in Znalosti 2004, 3rd annual conference, Brno, Czech Rep., 2004, pp. 66-84

[11] Quan, T. T., Hui, S. C., Cao, T. H.: A Fuzzy FCA-based Approach to Conceptual Clustering for Automatic Generation of Concept Hierarchy on Uncertainty Data, in Proceedings of CLA conference, Ostrava, Czech Republic, 2004, pp. 1-12

[12] Bednar, P., Butka, P., Paralic., J.: Java Library for Support of Text Mining and Retrieval, in Proceedings of Znalosti 2005, 4th annual conference, Stara Lesna, Slovakia, 2005, pp. 162-169

[13] Brezany, P., Janciak, I., Woehrer, A., Tjoa, A. M.: Gridminer: A Framework for Knowledge Discovery on the Grid - From a Vision to Design and Implementation, in Cracow Grid Workshop, Cracow, Poland, 2004

[14] Brezany, P., Janciak, I., Sarnovsky, M.: Text Mining within the GridMiner Framework, in 2nd Dialogue Workshop, Edinburg, GB, 2006

[15] Janciak, I., Sarnovsky, M., Tjoa, A. M., Brezany, P.: Distributed Classification of Textual Documents on the Grid, in High Performance Computing and Communications, HPCC 2006, LNCS 4208, Munich, Germany, September 13-15, 2006, pp. 710-718

[16] Sarnovský, M., Butka, P., Safko, V.: Distributed Clustering of Textual Documents in the Grid Environment (in Slovak), in Znalosti 2008, 7th annual conference, Bratislava, Slovakia, 2008, pp. 192-203

[17] Butka, P., Sarnovský, M., Bednár, P. One Approach to Combination of FCA-based Local Conceptual Models for Text Analysis - Grid-based

Approach, in Proceedings of SAMI 2008, IEEE conference, Herlany, Slovakia, 2008, pp. 131-135

[18] Butka, P., Zeher, M. Simple Approach to Combination of FCA-based Local Conceptual Models for Text Analysis. In Proceedings of the 7th International Workshop on Data Analysis, WDA 2006, Košice, Slovakia, 2006, pp. 1-10