Distributed Data Mining Framework for Cloud Service

Ivan Kholod∗, Konstantin Borisenko, and Andrey Shorov
Saint Petersburg Electrotechnical University, St. Peterburg, 197376, Russia
{iholod, borisenkoforleti, ashxz}@mail.ru

Abstract
The article describes the data mining framework for building a cloud service. The framework can use different distributed execution environments. The article describes the mapping of a data mining algorithm (decomposed into functional blocks) on the set of distributed handlers. These handlers may be implemented as threads, actors and others. In addition, it describes the approach for creation of cloud service which uses this framework. As an example, it describes the experiments and comparison with other data mining systems for classification algorithm 1R.

Keywords: Data mining, Distributed Data Mining, Cloud Computing, Data Mining Cloud

1 Introduction
At present time cloud computing is used in different fields of services. Using complete analytic methods such as data mining and machine learning methods requires vast computing resources. This problem can be solved by integration of data mining and cloud computing technologies. Integration can be reached by mapping data mining algorithms to cloud computing resources by using different distributed execution environments. The MapReduce paradigm [12] and Apache Hadoop are often used to construct clouds for data mining applications. However, not any data mining algorithms can be easily decomposed into the functions map and reduce. Therefore, we propose an alternative framework to build cloud data mining services which are able to use different distributed execution environments. The next section is a review of research in the field of designing data mining cloud services. The third section contains the description how to map the decomposed algorithm onto functional blocks in the set of parallel and distributed handlers. The fourth chapter describes the implementation of the framework and the architecture cloud service to perform data mining algorithms, based on this framework. The last chapter discusses experiments with the 1R algorithm regarding to the proposed framework.

2 Related work
Few institutions and IT companies are actively working to create cloud services for data mining. The Chinese Mobile Institute was one of the first to work in this field. In 2007 it began research and development in the field of cloud computing. In 2009, it officially announced a platform for cloud computing, namely BigCloud, which includes tools for the parallel execution of the algorithms Data Mining Big Cloud-Parallel Data Mining (BC-PDM) [16]. BC-PDM is a SaaS platform that is based on Apache Hadoop. Users can upload data to the repository (hosted in the cloud) from different sources and apply a variety of applications for data management, data analysis and business applications. The application includes analysis of parallel applications, which perform: ETL processing, social network analysis, analysis of texts (text mining), data analysis (data mining), and statistical analysis. In 2009

∗ Corresponding author: Department of Computer Science, Saint Petersburg Electrotechnical University, ul. Prof. Popova 5, St. Peterburg, 197376, Russia
the company expanded its Amazon cloud services Amazon EC2 and Amazon Simple Storage Service (Amazon S3) with another PaaS service - the Amazon Elastic Mapreduce (EMR) \[1\]. This service is also built on the Apache Hadoop platform and provides a scalable infrastructure to perform user-created (restricted by certain rules) applications. The service allows you to download Amazon S3 in the desired application and/or data that will later be carried out on job sites of platform Hadoop. The composition of EMR includes examples of applications that can be downloaded to the service, including the crucial tasks of data mining. In 2012, Google released its new cloud service Google BigQuery \[3\]. This service allows to process vast amounts of data stored in the cloud. If you are using the web browser and console applications, the user must enter SQL-like query to process their data. It may include all of the same elements as in the Select request in SQL: FROM, JOIN, WHERE etc. Thus, the user can generate a quite flexible search query according to any data type. It should be noted here that this service does not directly solve the problems of data mining: clustering, classification, and others are still to be carried out. The main disadvantages of the described systems are:

- The inability to complete the entire cycle of the analysis without the need for complex configuration and refinement
- using as a platform for distributed computing MapReduce technology (and in particular the Apache Hadoop).

Amongst other drawbacks, this technology is adapted only for data processing functions, which have the property of list homomorphisms \[13\]. For example, Apache Mahout \[4\] is a project of the Apache Software Foundation to produce free implementations of distributed or otherwise scalable machine learning algorithms focused primarily in the areas of collaborative filtering, clustering and classification. Many of the implementations use the Apache Hadoop platform. The project is more than five years old. However, the last version (0.10 from 11 April 2015) of Apache Mahout includes less than ten data mining algorithms for MapReduce \[5\] (algorithms of collaborative filtering, clustering and classification). These algorithms are implemented by splitting on the “map” and “reduce” functions. Another example of using the MapReduce conception for data mining algorithms is described in the paper \[10\]. It is applicable to the algorithms corresponding to the Statistical Query Model (SQM) only. The paper contains only ten algorithms which have such a form. For this purpose it is divided into two phases:

- the first phase is dedicated to compute the statistic sufficiency by summing up all the data
- the second phase is the aggregation of the calculated statistics and the obtaining of the final solution.

The listed phases can be realized through the Map-Reduce distributed computing model. There are some data mining algorithms libraries such as RapidMiner \[7\], Weka \[8\], R \[6\], etc. They include only solid implementations of the algorithms (implementations that have not been decomposed into separate interchangeable blocks). These implementations include task decomposition, which can be used to implement the analysis of different existing blocks. The most complicated block of data analysis and mining model construction is solid. As a result, these implementations of algorithms are not executed parallel or distributed and use all cloud computing resources.

### 3 The essence of the approach

#### 3.1 General concept

According to \[15\], a data mining algorithm can be written as a sequence of functional blocks (based on functional language principles). Classical functions in functional languages are pure functions. A data
Distributed Data Mining Framework for Cloud Service  
Kholod, Borisenko, and Shorov

A mining algorithm can be presented as a function (with two input arguments: input data set $d$ and a mining model $m$) that can be decomposed into a number of functions:

$$dma = f_{n}f_{n-1} \circ \ldots \circ f_{1}(d, m) = f_{n}(d, f_{n-1}(d, \ldots, f_{1}(d, \ldots, f_{1}(d, m), \ldots)), \ldots),$$

(1)

where $f_{i}$: is a pure function of the type $FB: D \rightarrow M \rightarrow M$, where

- $D$: is an input data set that is analyzed by the function $f_{i}$;
- $M$: is a mining model that is built by the function $f_{i}$.

We called this function - the functional block. According to Church-Rosser theorem reduction (execution) of such functional expressions (algorithm) can be done concurrently.

### 3.2 The function for parallel execution of a data mining algorithm

One of the main advantages of building algorithms from functional blocks is the possibility of parallel execution. For this task we need to transform the sequential expression (1) into a form, in which the functional blocks will be invoked as arguments. Therefore the high-order map function can be used to execute that task. It allows us to apply some functions to the elements of a list. The function can be executed parallel for different elements. The map function returns the list of results. To reduce the list into a single result the high-order fold function can be used. Thus, the function for the transformation of sequential expressions into a parallel form can be presented as follows:

$$parallel :: FB \rightarrow D \rightarrow M \rightarrow (M \rightarrow M) \rightarrow (FB \rightarrow [FB]) \rightarrow (D \rightarrow [D]) \rightarrow (M \rightarrow [M]) \rightarrow (FB \rightarrow D \rightarrow M \rightarrow H) \rightarrow (H \rightarrow M) \rightarrow M,$$

(2)

where

- $f$: (1st argument of the parallel function): a functional block is executed concurrently,
- $distrF$, $distrD$ and $distrM$ functions divide the functional block $f$, the input data set $d$ and the mining model $m$ into the lists:

$$distrF : (D \rightarrow M \rightarrow M) \rightarrow [(D \rightarrow M \rightarrow M]$$

(3)

$$distrD : D \rightarrow [D]$$

(4)

$$distrM : M \rightarrow [M]$$

(5)

- $start$: the function applies each functional block $f$ from the list $[F]$ to the elements of the lists $[D]$ and $[M]$ and returns a handler $h$ for the parallel execution of the parallel functional block $f$:

$$start : FB \rightarrow D \rightarrow M \rightarrow H$$

(6)

- $get$: the function reads the mining model from the handler $h$:

$$get : H \rightarrow M$$

(7)
join: the function joins the mining models from the list \([M]\) and returns the merged mining model \(M\):

\[
join : [M] \rightarrow M \tag{8}
\]

map: the function applies function start to elements of the lists \([F]\), \([D]\) and \([M]\):

\[
map : ((D \rightarrow M \rightarrow M) \rightarrow M \rightarrow M \rightarrow H) \rightarrow [D] \rightarrow [M] \rightarrow [M] \rightarrow [H] \\
map[h] = list((start(f[0], d[0], m[0])), ..., (start(f[k], d[k], m[k]))) \tag{9}
\]

call: the function reduces the list of mining models to a single result:

\[
call : FB \rightarrow D \rightarrow M \rightarrow M \\
call(f, d, m) = f(d, m) \tag{12}
\]

fold: the function reduces the list of mining models to a single result:

\[
fold : ([M] \rightarrow M) \rightarrow M \rightarrow (H \rightarrow M) \rightarrow [H] \rightarrow M \\
foldm = join(m, get(h[0]), get(h[1]), ..., get(h[k])) \tag{10}
\]

3.3 Mapping the parallel function on a distributed execution environment

A list of handlers parsing between the map and fold functions is part of some distributed execution environments. In general, an execution environment can be represented as a set of handlers:

\[
E = h_0, h_1, \ldots, h_j, \ldots, h_n, \tag{11}
\]

where

- \(h_0\): handler to execute sequential functional blocks of an algorithm,
- \(h_1 - h_n\): handlers to execute parallel functional blocks in the distributed environment.

Each handler must implement the start and get functions to use them in the parallel function. The handler calls functional blocks between these functions. It can be represented as the function:

\[
call : FB \rightarrow D \rightarrow M \rightarrow M \\
call(f, d, m) = f(d, m) \tag{12}
\]

The implementation of the start and get functions is specified by the execution environment which includes these handlers. Examples of these handlers are threads, actors, web services and others.

4 Implementation of the distributed data mining framework

4.1 Base classes for a data mining algorithm

We extended the Xelopes library [9] (data mining algorithms library) and implemented all functions as classes of the object-oriented language Java. Figure 1 shows the class diagram of these blocks.

Here the FB-type function is described by the class Step. The condition DecisionStep and the cycle CyclicStep are functions of FB type, so they are inherited from the class Step. VectorsCycleStep class and AttributesCycleStep class were made for vector and attributes cycles. MiningAlgorithm was made to from the target algorithm. It has attribute the Steps containing the sequence of all algorithm steps (1).

The InitSteps method occurs formation of the algorithm’s structure by creating steps which determines of sequence and nests their execution. The ParallelStep class (see Figure 2) is needed for parallelization of algorithms. It implements parallelization of function expression (2). As a step of the algorithm, it also inherits from the Step class. It contains a step sequence to be executed in parallel and main methods: \(distrF\) (3), \(distrD\) (4), \(distrM\) (5), map (9), fold (10) and join (8). Parallelization by data is the most common parallelization type for data mining algorithms. The ParallelByData class was added for parallelization of algorithms by data. The ParallelByTask class was added for parallelization of algorithms by tasks. They inherit from the class ParallelStep and both classes are markers of parallelizing point in an algorithm. The library elements were represented in detail at the conference [14].
4.2 Base classes for parallel execution of a data mining algorithm

For the parallel and distributed execution we extended the framework. The user needs to create tasks as an object of the class EMiningBuildTask (inheritor of MiningBuildTask from CWM specification [2]) to execute data mining algorithm and to build a data mining model. This class is the entry point for the data mining cloud service. Follow entities should be set (Figure 3):

- parallel data mining algorithm (object of class inheritor of MiningAlgorithm class)
- execution environment (object of ExecutionEnvironment class)
- parameters specific to mining functions and the data mining algorithm (objects of class inheritor of the MiningFunctionSettings class and of the MiningAlgorithmSettings class accordingly also of CWM specification)
- input data set (object of class inheritor of MiningInputStream class). A parallel data mining algorithm is created as inheritor of the MiningAlgorithm class. The detailed creation of a parallel data mining algorithm was presented at the conference [14] and some examples will be shown below.

4.3 Base classes for handlers

The ExecutionEnvironment class describes an execution environment to run data mining algorithm. It is the entry point to execute a data mining algorithm. The following parameters can be set:
Figure 2: Class diagram for parallel execution of a data mining algorithms

- numberHandlers is number of handlers on which the parallel algorithm will be executed;
- systemType is type of system fulfilling the parallel algorithm execution (multithreads, actor model, mapreduce and other);
- memoryType is memory type of the system which will be fulfilling the execution of the parallel algorithm.

An object of the MiningAlgorithm class is deployed onto the environment (object of ExecutionEnvironment class) by deploying the method - deployAlgorithm. The class ParallelStep (see Figure 2) executes the parallelization of the functional sequential blocks on the execution environment by calling the methods of handlers, which are implemented as ExecutionHandler classes:

- handler initialization (start method (6)) – allocation for every handler particular data and data model
- handler execution (call method (12))
- handler idling (getModel method (7)) – awaiting results from every handler and receiving the data model
- handler completion (fold method of ParallelStep class (10))
- data model integration (join method of ParallelStep class (8))

Handlers (11) are used for the parallel execution of parallelized functional blocks (Figure 4). They run a sequence of functional blocks - sequence. Classes ParallelByDataProxy and ParallelByTaskProxy
establish the link between a handlers and a executable sequence. They have abstract the methods start and join, they run and finish handlers. Those methods’ implementations are in descendants that make particular handler types of parallel and distributed execution: threads, actors and so on.

We implemented two distributed execution environments: multithreads (see figure 5) and actors model (see figure 6).

### 4.4 Cloud service based on the distributed data mining framework

Possible architecture of data mining cloud service are shown in figure 7. It has three layers. First layer is GUI – user interface. Second layer is API – program interface. Third layer is Data Mining Algorithms Library, Cloud Environment and Data Transform.

Figure 8 shows the Data Mining Cloud Service page for the execution of environment adjustments.

The environment initialization is located in the center of the page. A list of the created environments is shown on the right side. Environment settings table forms dynamically according to type of chosen environment. Figures 9 and 10 show exemplary the settings table with environment adjustments.

Figures 9 and 10 show the dynamic tables formation. At first the Multithreads-type was chosen. This environment type has no extra settings. That is why the table is empty. The next type is Actors. It has two extra settings: nodes in cluster amount and seeds nodes amount (see Figure 10). According to that, parameters were added to the settings table.
5 Experiments

Series of experiments were done to prove the effectiveness of our implemented algorithm 1R and its parallelizations. The computations were executed on a Quad Core PC: CPU: Intel(R) Core(TM) i5-4460, 3.2GHz, RAM: 4 GB, OS: Windows 10, JDK 1.7 (x64). To execute the algorithms, Java virtual machine was launched with next settings: –Xmx20148m and –Xms512m. The settings provide up to 2 GB RAM. The experiments were executed in the multithreads environment for 1, 2 and 4 threads. Datasets with different size were used for experiments. Table 1 shows their settings. The datasets are stored in ARFF and Excel formats.

These are settings for the experiments of the algorithm 1R:

- without decomposition on functional blocks (1R Xelopes) — algorithm is based on the Weka’s library algorithm and adapted to the Xelopes library;
- sequential with decompositions on functional blocks (1R DXelopes) — we made decomposition into blocks
paranellized on vectors with decomposition on functional blocks (1R Parallel) — we converted the sequential form to a parallelized form and executed it using 2 and 4 threads.

We also made experiments using the most popular open-source Data Mining libraries RapidMiner and WEKA to compare results with our approach. Table 2 shows those results.

Figure 11 shows plots of the experiment evaluation time in relation to the amount of data.

According to the experimental results, we conclude:

- The Xelopes library algorithm takes slightly less amount of time than using the WEKA library and significantly less using RapidMiner.
Algorithm 1R using decomposition on functional blocks is slower by 20-30% than algorithm using Xelopes and WEKA without decomposition, but still it is significantly faster than RapidMiner. Performance decreases because of additional steps called by interface blocks unification: saving and extracting transitional results in data models;
parallelized algorithm (with decomposition on functional blocks) using 2 handlers is faster by 70\% average than sequential using decomposition and by 40\% than sequential without using decomposition;

- parallelized algorithm using 4 handlers is faster by 20-30\% more. Slightly performance increase is related to the increasing time for the models combination, that are made in different threads;

- performance in actor model execution environment is faster than in multithreads execution environment for large data sets;

- parallelized algorithm performance increases with analyzing data vectors increase. Amount of time
6 Conclusion

Thus, the representation of a data mining algorithm as functional expression makes it possible to divide the algorithm into functions of FB type (functional blocks). Such a splitting of data mining algorithms into blocks helps to map it to handlers and therefore to different distributed environment. We implemented this approach as a Java framework. It is possible to construct different variants of parallel data mining algorithms and to execute them in different distributed execution environments. Single entry points allows to manage setting and execution of a data mining algorithm. This framework can be used to build a cloud service for distributed data mining. We implemented 1R algorithm for multithread execution environments and for actor model execution environments. The experiments show the efficiency of our framework.
Acknowledgments

The paper has been prepared within the scope of the state project “Organization of scientific research” of the main part of the state plan of the Board of Education of Russia, the project part of the state plan of the Board of Education of Russia (task 2.136.2014/K), supported by grant of RFBR 16-07-00625, supported by Russian President’s fellowship, as well as with the financial support of the Foundation for Assistance to Small Innovative Enterprises in the scientific and technical spheres 10134gy2015.

References


Distributed Data Mining Framework for Cloud Service

Kholod, Borisenko, and Shorov


Author Biography

Ivan Kholod is an expert in the fields of parallel and distributed data mining. In 2003, he defended his Ph.D. degree thesis in the area of mobile agent systems. Since 2008, he is working in the position as a lecturer in the Department of Computer Engineering, Saint Petersburg State Electrotechnical University “LETI”, Saint Petersburg, Russia. Ivan has published more than 30 articles in areas related to the data mining, parallel and distributed data mining.

Konstantin Borisenko is PhD student in Saint Petersburg State Electrotechnical University “LETI” and is working there in the position of Junior Researcher in the Department of Computer Engineering. He is an expert in computer security, cloud computing, data mining and machine learning. Konstantin is co-author of six papers and book chapters in areas related to the computer security, cloud computing and data mining.
Andrey Shorov is an expert in the fields of computer security, cloud computing, data mining and machine learning. In 2012, he defended a Ph.D. degree thesis in the area of computer security. Since 2014, he is working in the position of Senior Researcher in the Department of Computer Engineering, Saint Petersburg State Electrotechnical University “LETI”, Saint Petersburg, Russia. Andrey has published more than 30 articles in areas related to the modeling and simulation, network security, cloud computing, data mining and machine learning.