Journal of Advanced Research in Applied Sciences and Engineering Technology

Journal homepage:
https://semarakilmu.com.my/journals/index.php/applied_sciences_eng_tech/index
ISSN: 2462-1943

Development of Tools for Creating Parallel Data Mining Algorithms

Karshiyev Zaynidin[1,*], Sattarov Mirzabek[1]

[1] Department of Computer Systems, Faculty of Computer Engineering, Samarkand Branch of Tashkent University of Information Technologies named after Muhammad al-Kwarizmi , 140100 Samarkand, Samarkand, Uzbekistan

ARTICLE INFO

Article history:
Received 7 June 2023
Received in revised form 12 December 2023
Accepted 5 January 2024
Available online 7 February 2024

Keywords:
Data mining; classification; clustering; association rules; parallel algorithms

ABSTRACT

The purpose of this work is to develop tools for building parallel data mining algorithms for execution in a distributed environment. A formal model of a data mining algorithm is proposed, characterized by a representation of the algorithm in the form of a set of independent operations that change the state of the knowledge model and structural blocks that allow modifying the structure of the algorithm, including for parallel execution. A method is proposed for creating parallel algorithms for data mining, in contrast to existing ones, using a decomposition of the algorithm into thread-safe functional blocks and allowing parallelization, both by changing the structure of the parallel algorithm and by configuring its execution. A methodology is proposed for parallelizing data mining algorithms, which differs from those known in that the proposed method of creating parallel data mining algorithms taking into account the characteristics of a distributed environment is applied to sequential analysis algorithms. To create parallel data mining algorithms, software templates built on the basis of a formal model and separating the implementation of the algorithm from distributed execution tools are proposed. A library of parallel data mining algorithms has been developed for execution in a distributed environment, including the proposed templates.

1. Introduction

The greatest value and knowledge gained when using Data Mining (DM) algorithms is possible when analyzing significant amounts of data [1]. In this case, the following main problems of analysis arise:

i. performance - analysis of large volumes (measured in terabytes) requires large computing resources and can be performed in an unacceptable time for an analyst;
ii. distribution - due to the large amount of data, information can be stored in a distributed storage, in addition, due to the nature of the data, they can be stored in different sources.

Both problems can be solved by parallel and/or distributed data mining.

* *Corresponding author.*
E-mail address: zaynidin85@gmail.com

https://doi.org/10.37934/araset.39.1.2642

In the past few years, increasing the productivity of computing technology is associated both with the development of multi-core processors and with the increasing spread of cluster systems, including "cloud" systems [2]. However, modern software lags far behind the hardware and often uses the available computing resources inefficiently. This problem is primarily associated with the high complexity of solving the problem of parallelizing computational algorithms.

Data mining algorithms are no exception. A large number of studies are currently being carried out in this area. Separate directions in the field of DM are singled out (in foreign literature, this area is called Data Mining): parallel DM (Parallel Data Mining) and distributed DM (Distributed Data mining). Most of the efforts of researchers in the field of parallel DM algorithms are aimed at parallelizing individual analysis algorithms and their further optimization [3]. As a rule, the resulting solutions are focused on a certain computing environment, and when such a solution is transferred to other conditions, it becomes ineffective. In this regard, research in the field of general approaches to the parallelization of existing mining algorithms is an urgent task.

The purpose of this work is to develop tools for constructing parallel DM algorithms for execution in a distributed environment. To achieve the stated goal, the following tasks are solved in the work:

i. analysis of existing approaches to the creation of parallel DM algorithms;
ii. development of a formal model of the DM algorithm;
iii. development of a method for creating parallel DM algorithms based on thread-safe functional blocks;
iv. development of a methodology for constructing parallel DM algorithms for execution in a distributed environment;
v. development of software templates for the implementation of serial and parallel DM algorithms from thread-safe functional blocks;
vi. conducting experiments on the implementation of algorithms built in accordance with the proposed methodology.

As a result of the analysis of existing research areas in the development of parallel DM algorithms [4-17], two main approaches can be distinguished:

i. decomposition of the algorithm for possible parallelization - a generalized approach to the DM algorithm, which involves dividing it into some parts that can be executed in parallel;
ii. individual parallelization of algorithms - an individual approach to each DM algorithm and the choice of the most efficient parallel structure for given conditions.

Within the framework of the first approach, one can single out the commercial project NIMBLE by IBM. This project is aimed at developing an infrastructure that allows parallel execution of DM algorithms on tools that implement the MapReduce concept. It assumes that the algorithm is decomposed into separate parts (map and reduce parts) that can be executed in parallel. This project, firstly, is limited by the concept of MapReduce, and secondly, it does not contain a decomposition technique for the DM algorithm.

Within the framework of the second approach, the greatest efforts are concentrated, which took shape in two directions: parallel and distributed data mining. However, these efforts are aimed at optimizing the parallel structures of individual algorithms for a specific runtime environment. For this reason, the proposed parallel algorithms are not always efficient when changing the execution environment.

Most of the proposed parallel DM algorithms use data parallelism and are designed to run in distributed memory systems, as shown in Figure 1. Algorithms that use task parallelism are also mainly designed to work with distributed memory.

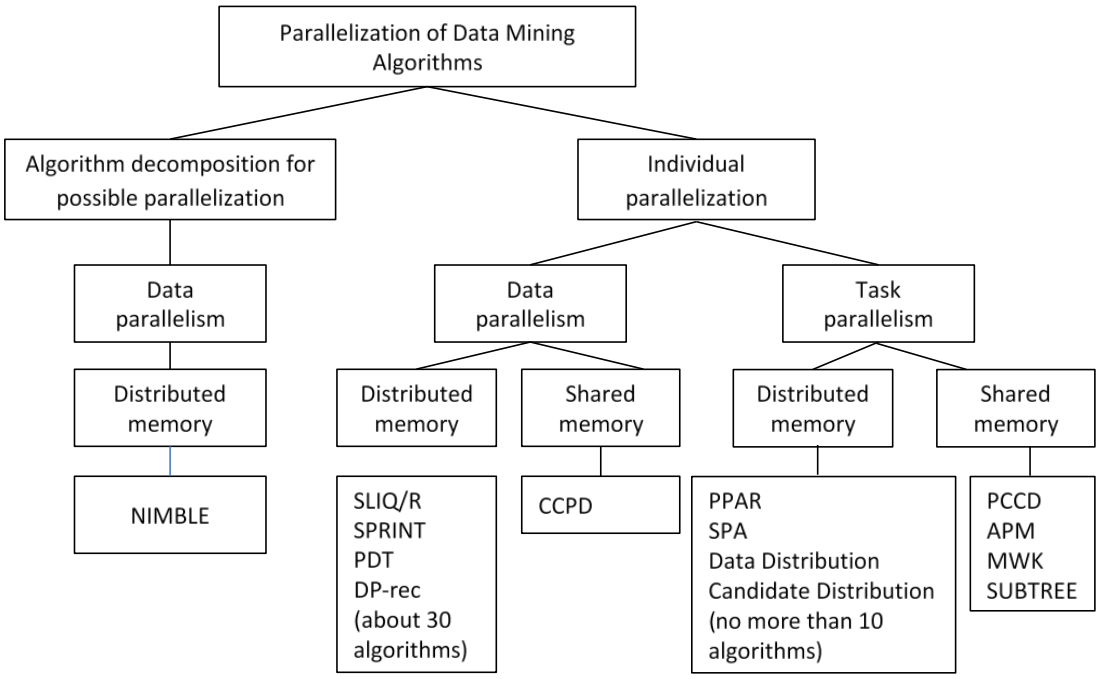

Fig. 1. Systematization of parallel data mining algorithms

Optimizing the DM algorithm for specific conditions is a laborious task. Adapting such algorithms to new execution conditions requires costs comparable to the creation of new algorithms. For this reason, this approach cannot be considered as a general approach to constructing parallel DM algorithms.

As a result, we can conclude that there is no known approach to creating parallel DM algorithms that is common to most algorithms and execution tools, which also determines the parallelization technique.

2. Methodology

The analysis of DM algorithms made it possible to identify a number of features of such algorithms [18]:

i. iterative processing of large amounts of data;
ii. the presence in the algorithms of typical blocks (cycles on vectors and attributes, etc.);
iii. formation on the basis of input data (data set, execution settings and initial knowledge model) of the result in the form of a knowledge model.

Based on this, the DM algorithm can be represented as some function that returns the constructed knowledge model. The input arguments of such a function are: a data set, settings, and an initial knowledge model (an empty structure that does not contain any regularities):

$$M = F\left(D, S, M_0\right) \tag{1}$$

The data set (D*)* formally as a set of three elements:

$$D(A,T,W), \tag{2}$$

where:
A – independent attributes of the data set:
$A=\{a_1,...,a_i,...,a_m\}$,
where a_i – data set attribute;

T – a set of target attributes defined as the attributes above:

$$T=\{a_{m+1},...,a_k,...,a_t\} \tag{3}$$

The pair (A,T) represents the metadata of the dataset.

$W=\{w_1,...,w_r,...,w_z\}$ – a set of data vectors representing sets of values of the corresponding attributes:

$$w_r=\{v_{1.r},...,v_{i.r},...,v_{m.r},v_{m+1.r},...,v_{k.r},...,v_{m+t.r}\} \tag{4}$$

where $v_{i.r}\in D(a_i)$ is the attribute value a_i vector $v_{i.r}$.

The knowledge model (M) can be formally described as follows:

$$M=\{R,K_t\} \tag{5}$$

where
$R=\{r_1,...,r_i,...,r_n\}$ - a lot of rules;
$K_t=\{k_1,...,k_p,...,k_g\}$,
where $k_p=\left|\{w_r : w_r\in W, v_{t.r}=v_{t.p}\}\right|$ is the number of vectors whose target attribute a_t has the value $v_{t.p}$, while $v_{t.1}\cup...\cup v_{t.p}\cup...\cup v_{t.z}=D(t)$.

The knowledge model (M) can discretely take one of the states:

- M_0 – initial state of the knowledge model;
- M_1 – model after the 1st change;
- ……
- M - complete knowledge model.

The discrete state is understood as the state of the knowledge model, in which it is holistic, i.e. satisfies all the restrictions that are imposed on it.

Algorithm settings can be represented as a set of pairs: a parameter and its value:

$$S=\left((s_1,p_1),...,(s_{y,}p_y),...,(s_z,p_z)\right) \tag{6}$$

where s_y is the *y*-th algorithm setting parameter, and p_y is the value of the *y*-th algorithm execution setting parameter.

Algorithm DM (*F*) can be represented as a set of sequentially executed operations:

$$F = \{O_1, ..., O_p, ..., O_q\} \quad (7)$$

where O_p is an operation that changes the knowledge model from one discrete state to another based on the data and settings given to it. Thus, the arguments of the operation should be the data D, the settings *S* and the knowledge model before the changes M_i, and its result is the knowledge model after the change M_{i+1}:

$$M_{i+1} = O(D, S, M_i) \quad (8)$$

Obviously, the knowledge model passed as an argument and the knowledge model returned as a result of the operation must be integral.

In accordance with this, the DM algorithm can be represented as a sequence (ordered set) of operations, the result of each of which is passed to the next operation in the sequence as an argument:

$$F = \{O_1(D, S, M_0), ..., O_p(D, S, M_p), ..., O_q(D, S, M_q)\} \quad (9)$$

Moreover, each such operation can also be decomposed into separate operations (functional blocks) that are performed sequentially:

$$M_{p+1} = O_p(D, S, M_p) = \{O_{p.1}(D, S, M_p), O_{p.2}(D, S, M_{p.1}), ..., O_{p.u}(D, S, M_{p.u}), ..., O_{p.y}(D, S, M_{p.y})\} \quad (10)$$

In addition to operations that change the knowledge model, the following elements have been added to the formal model [19] that determine the structure of the algorithm: conditional statements, loops (including loops over vectors and over data).

We represent the conditional operator as an element of the following form:

$$M_{i+1} = D(D, S, M_i) = \{d(D, S, M_i), O_t(D, S, M_i), O_f(D, S, M_i)\} \quad (11)$$

where:

$d(D, S, M_i)$- a conditional function that returns true or false, based on data analysis D and the current model M_i, but does not change them;

$O_t(D, S, M_i)$–operation performed when the value-true, returned by the function $d(D, S, M_i)$;

$O_f(D, S, M_i)$–operation performed when the value-false, returned by the function $d(D, S, M_i)$;

We represent the cycle as follows:

$$M_{i+1} = C\left(D,S,M_i\right) = \left\{d\left(D,S,M_i\right), O_c\left(D,S,M_i\right)\right\} \quad (12)$$

where $O_c\left(D,S,M_i\right)$ is the operation that is performed while the conditional function $d\left(D,S,M_i\right)$ returns a value - true.

Thus, the algorithm can contain both operations and structural elements. We generalize them into the concept of a functional block of the algorithm:

$$b_i\left(D,S,M_i\right) = \begin{cases} O\left(D,S,M_i\right) \in \left\{O^{add}\left(D,S,M_i\right), O^{del}\left(D,S,M_i\right), O^{ch}\left(D,S,M_i\right)\right\} \\ E\left(D,S,M_i\right) \in \left\{D\left(D,S,M_i\right), C\left(D,S,M_i\right)\right\} \end{cases} \quad (13)$$

Function blocks described in this way have two properties:

- determinism - with the same input arguments (data, settings and initial state of the knowledge model), the returned knowledge model will be the same;
- has no side effects - i.e. when the function blocks are executed, the external elements are not changed.

The DM algorithm can be represented as an ordered sequence of functional blocks:

$$M_q = F\left(M_0,S,D\right) = \left(b_0\left(D,S,M_0\right),...,b_k\left(D,S,M_k\right),...,b_q\left(D,S,M_{q-1}\right)\right) \quad (14)$$

Thus, the DM algorithm itself also has no side effects and has the property of determinism.

To describe parallel executing branches of the algorithm using a formal model, we add the following structural element to it:

$$M_{i+1} = P\left(D,S,M_i\right) = \left\{s\left(D,S,M_i\right), b_d\left(D,S,M_i\right), b_1\left(D,S,M_i\right),...,b_x\left(D,S,M_i\right),...,b_z\left(D,S,M_i\right), j\left(D,S,M_i\right)\right\} \quad (15)$$

where is the $s\left(D,S,M_i\right)$ split operation that performs preparatory actions for parallel execution of operations; $j\left(D,S,M_i\right)$ – the join operation , which performs joining actions after the parallel execution of operations; $b_d\left(D,S,M_i\right)$ is a block that is a dispatcher for other blocks, if $b_d\left(D,S,M_i\right) = \varnothing$, then there is parallelization without a dispatcher.

If the blocks executed in parallel are the same $\left(b_1 = ... = b_x = ... = b_z\right)$, then there is data parallelism, otherwise (if $b_1 \neq ... \neq b_x \neq ... \neq b_z$) there is task parallelism.

In the case of data parallelism, the following condition must be met:

$$D_1 \neq D_2 \neq ... \neq D_x \neq ... \neq D_z \text{ and } D = D_1 \cup D_2 \cup ... \cup D_x \cup ... \cup D_z \quad (16)$$

those. all data processed by parallel blocks must be different and their union must form a complete data set.

In the case of task parallelism, the following condition must be met:

$$D_1 = D_2 = ... = D_x = ... = D_z \quad (17)$$

those. all datasets processed by parallel blocks are complete datasets.

For the interaction of parallel-executing sequences, we add two more structural elements to the model:

– send data block:

–

$$S(D,S,M_i) = \{O(D,S,M_i), s(m)\}, \quad (18)$$

where $O(D,S,M_i)$ – an operation to be performed before sending a message (for example, preparing a message); $s(m)$– an operation that sends a message *m*;

– data acquisition block:

–

$$R(D,S,M_i) = \{r(m), O(D,S,M_i)\}, \quad (19)$$

where $r(m)$– receiving message m ; $O(D,S,M_i)$– an operation that processes the received message.

Thus, the final form of the formal model of the DM (14) algorithm for parallel execution will be as follows:

$$b_i(D,S,M_i) = \begin{cases} O(D,S,M_i) \in \{O^{add}(D,S,M_i), O^{del}(D,S,M_i), O^{ch}(D,S,M_i)\} \\ E(D,S,M_i) \in \{D(D,S,M_i), C(D,S,M_i), P(D,S,M_i), S(D,S,M_i), R(D,S,M_i)\} \end{cases} \quad (20)$$

According to the formal model, the DM algorithm and blocks of its components do not have side effects and have the property of determinism. These properties ensure that such blocks are thread-safe, so they can be executed in parallel.

In accordance with the formal model, when splitting into blocks, the following requirements must be observed:

- the algorithm is an ordered sequence of functional blocks (1) that satisfy the conditions described below;
- each block can represent either some operation performed on the model or some structural element characteristic of the DM algorithm (20);
- the block that implements some operation must change the knowledge model submitted to the input in such a way that it remains integral (i.e., all restrictions imposed on the model must be met);
- inside the functional block there should be no access to external variables, all work should be performed on the basis of: data set D, settings S and the initial knowledge model M_{i-1} transferred to the functional block.

According to the formal model, a functional block can contain both executable code (an inseparable *O operation*) and a sequence of other functional blocks. In general, a block may contain more than one sequence.

As a result, the DM algorithm can be represented as a hierarchy of nested thread-safe functional blocks (Figure 2). At the same time, the algorithm itself, as a separate functional block, is also thread-safe.

Based on the blocks allocated in the formal model, the basic thread-safe functional blocks for building DM algorithms are defined:

- decision block $D(D,S,M_i)$;
- block cycle $C(D,S,M_i)$ and characteristic for DM algorithms:
  - block cycle over vectors $C_{bv}(D,S,M_i)$;
  - Attribute cycle block $C_{ba}(D,S,M_i)$;
- parallel execution block $P(D,S,M_i)$ and its variants:
  - data parallelization block $P_{bd}(D,S,M_i)$;
  - task parallelization block $P_{bt}(D,S,M_i)$;
- block sender $S(D,S,M_i)$;
- block recipient $R(D,S,M_i)$.

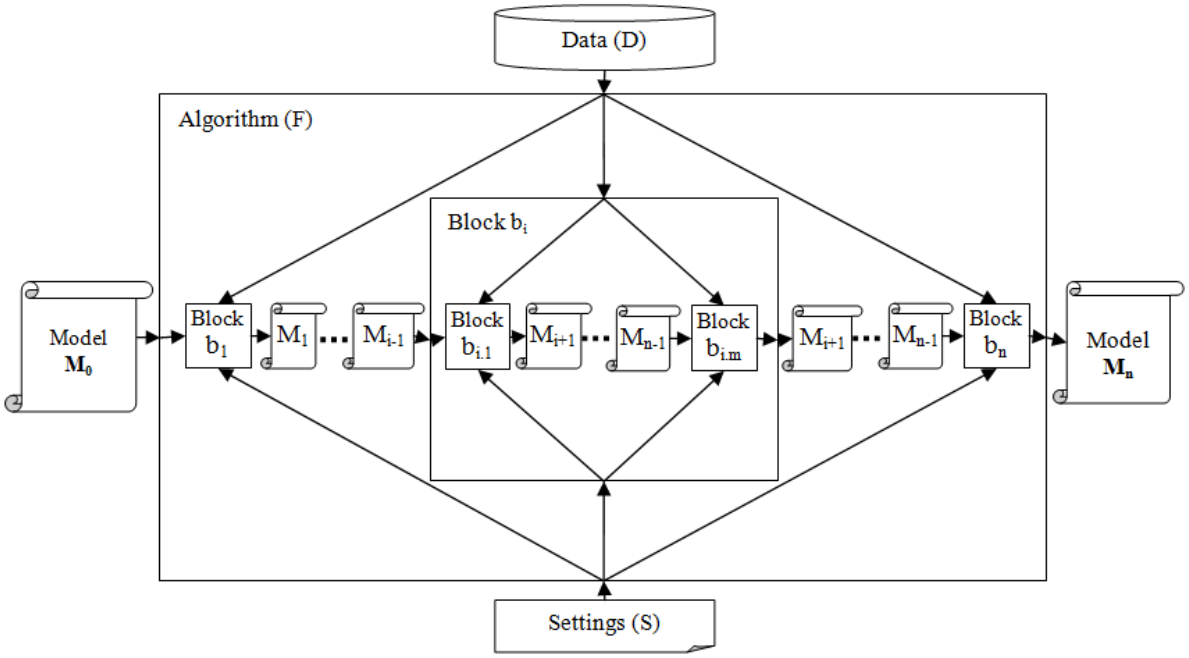

Fig. 2. Block structure of the algorithm

Based on the representation of the algorithm as a hierarchy of nested functional blocks, a method for constructing parallel algorithms DM is proposed. To build a parallel algorithm, the DM method involves the following actions [20]:

- decompose the DM algorithm into elementary actions;
- group elementary actions into functional blocks that perform a discrete change in the knowledge model;
- select structural elements and nested sequences of functional blocks;

- add functional blocks for parallel execution to the structure of the DM algorithm.

Using the functional blocks listed above, the proposed method makes it possible to construct the following types of parallel structures of DM algorithms:

- with task/data parallelization;
- with/without interaction between parallel branches for distributed memory systems;
- with/without controller.

Setting up parallel execution blocks allows you to implement the following types of parallel DM algorithms:

- with single data and single model (SDSM) - in this case, parallel sequences of functional blocks work with one data source and one knowledge model;
- with multiple data and single model (MDSM) - in this case, each sequence works with a separate source, but with one knowledge model;
- with single data and multiple models (SDMM) - in this case, each sequence works with a common data source, but each builds its own knowledge model, which are subsequently combined into one;
- with multiple data and multiple models (MDMM) - in this case, each sequence works with its own data source and each builds its own knowledge model, which are subsequently combined into one.

Can be described software templates in the form of classes for building DM algorithms from separate blocks. Classes define both operations and blocks corresponding to structural elements: decision blocks and loops. In addition, classes are proposed for the implementation of parallel DM algorithms.

Let's consider each of the classes in more detail.

The Step class is a class that implements a block - an operation and is base class for all blocks of the algorithm. The input parameters for it are data, settings and model. The most significant class method is the execute() method, which starts the execution of the block. This method is overridden by child classes depending on their purpose.

The SequenceOfSteps class is a class that implements an ordered sequence of blocks. The class stores a sequence of objects of the Step class and redefines the execute() method in such a way that it sequentially launches all blocks for execution and ensures that the knowledge model changed by the previous block is transferred to the next block.

CyclicStep class – abstract class for loop implementation. It stores a sequence of blocks (as an object of the SequenceOfSteps class). At each iteration of the loop, the stored sequence of blocks is executed.

The DecisionStep class is an abstract condition checking class. Includes two sequences of blocks (objects of the SequenceOfSteps class) and an abstract condition() function that returns true or false. Depending on the result of this function, one or another sequence of blocks is executed.

The CycleByVectors class is a class that implements a cycle over vectors. This class extends the CyclicStep class and cycles through the elements (vectors) of the input data.

The MiningAlgorithm class is a class that implements the algorithm itself as an ordered sequence of functional blocks. The abstract class MiningAlgorithm contains the main sequence of blocks of the algorithm (as an object of the SequenceOfSteps class). It performs the necessary actions to initialize the knowledge model and objects of the Step class , after which it starts the execution of the main sequence of blocks.

The ParallelStep class is abstract (it does not implement the execute() method described in the Step class) and contains an array of handlers for parallel execution of algorithm branches - handlers (and access methods to them), as well as parallel execution settings paralellAlgSettings.

The ParallelBlock class is responsible for executing the parallel branches of the algorithm and is inherited from the ParallelStep class. In the ParallelBlock class, in addition to the inherited methods, two more methods are defined - split() and join(). In the split() method, operations are performed to launch parallel branches of the algorithm. The join() method combines threads and results from individual branches.

To implement data and task parallelization, two classes ParallelByData and ParallelByTask are inherited from the ParallelBlock class.

The ParallelByData class implements data parallelization. It contains the branche parameter, a sequence of steps of the parallel branches of the algorithm, which is cloned when the algorithm is run for each available parallel branch handler. To work with this parameter, the class defines: the getBranche() method - to get a sequence and the setBranche() method - to form a sequence of steps.

The ParallelByTask class implements task parallelization. It, unlike the ParallelByData class, contains an array of sequences of steps for each branch of the parallel algorithm - branches. To work with this parameter, the class defines: the getBranches() method for getting a list of sequences and the addBranche() method for adding a new sequence of steps.

The SenderStep class is a class that implements the sender block. In the execute () method, it sends messages through handlers to all parallel executing branches of the DM algorithm. The message formation algorithm, like the message itself, depends on the DM algorithm and is implemented in a class that inherits from SenderStep and implements the corresponding functional block of the algorithm.

The ReceiverStep class is a class that implements a receiver block. It defines a message variable containing the message to receive. The execute() method of this class directly receives the message and initializes the message variable. The message is received in a waiting loop, i.e. the loop is exited only after the message has been received. The message variable is available to child classes and can be further processed according to the DM algorithm.

The described classes make it possible to implement any structure of the parallel algorithm from those given in the second chapter.

The implementation of the split and join operations in the parallel block class allows you to perform SDSM, SDMM, MDSM and M DMM type parallelization without changing the structure of the algorithm. Changing the behaviour of an algorithm is done by customizing its execution.

references to handlers that exist in the ParallelStep class are instances of the ExecutionHandler class. – common to all means of distributed execution. Currently, distributed computing tools are widely used: multi-threaded execution, service-oriented execution, actor model, execution based on the MapReduce concept, multi-agent execution, etc.

In order for the implementation of the algorithm and its parallel structure to be independent of the means of execution, the "Adapter" design pattern is used. The adapter uses classes that inherit from the ExecutionHandler class (referenced by the ParallelStep class) for each implementation.

The generalization of the obtained results is the proposed method for constructing parallel DM algorithms from thread-safe functional blocks for execution in a distributed environment. When solving the problem of constructing such an algorithm, the initial data are shown in Figure 3:

- parallel algorithm execution environment (Figure 3a);
- type of data storage (Figure 3b);
- view of the DM function (Figure 3c).

The technique consists of two main steps [21]:

i. determination of the structure of the parallel algorithm DM;
ii. setting up the parallel DM algorithm.

At the first stage, the structure of the DM algorithm is analyzed in order to identify typical elements and other features inherent in the algorithm. Based on the results of the analysis, the structure of the parallel algorithm is formed.

To determine the structure of a parallel algorithm, we must perform the following steps:

i. decompose the DM algorithm into elementary operations;
ii. group into functional blocks actions that perform a discrete change in the model;
iii. perform a software implementation of the block structure of the sequential algorithm;
iv. perform an analysis of the resulting structure, taking into account the remaining cycles and conditional transitions, and determine the structure of the parallel algorithm;
v. perform a software implementation of the parallel structure of the DM algorithm based on previously implemented functional blocks.

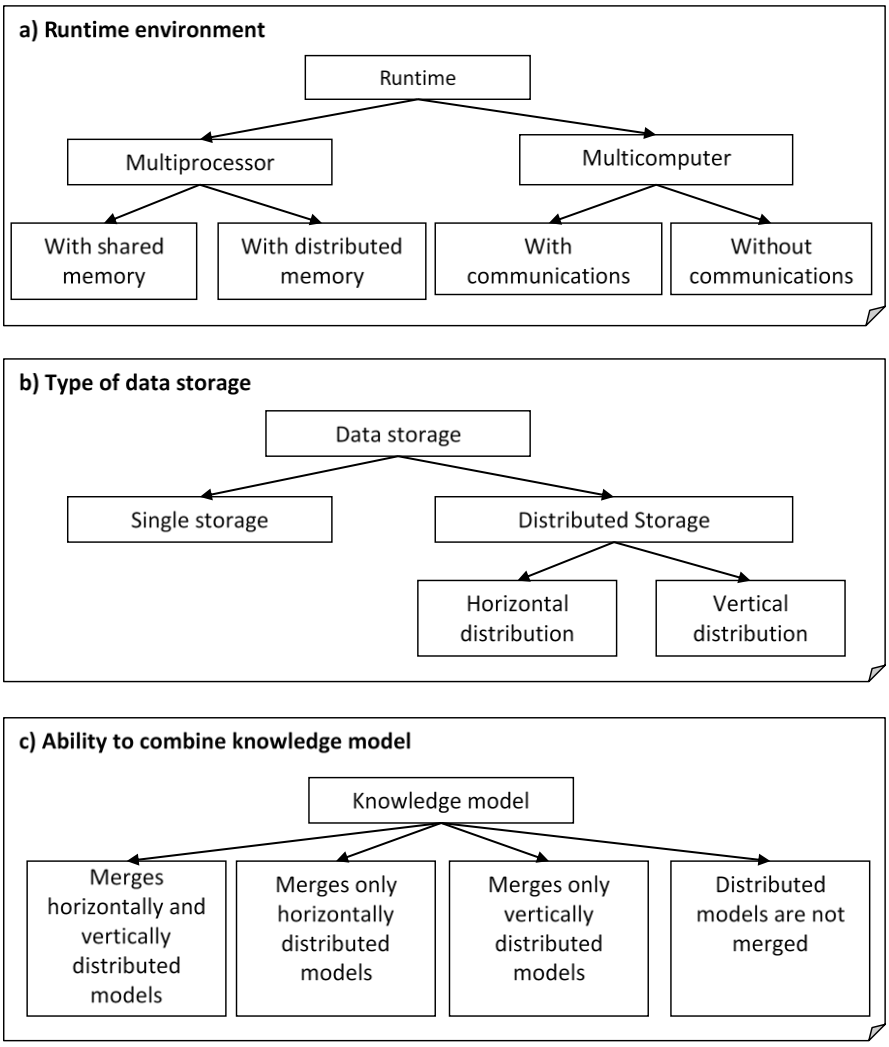

Fig. 3. Initial data for the technique of parallelization of DM algorithms

The first two steps are performed in accordance with the proposed method for constructing the parallel DM algorithm described above.

At the second stage of the parallelization of the algorithm, the execution of the parallel algorithm is tuned and its efficiency is analyzed, followed by debugging. The stage includes the following steps:

II.1. setting up the execution of the parallel DM algorithm in accordance with the environment parameters and the type of data distribution;

II.2. analysis of the parallel execution of the DM algorithm on test data, by evaluating the efficiency and acceleration;

a. return to step II .1. if the results are unsatisfactory and not all settings have been tested;

b. return to step I .4. if the results are unsatisfactory and all settings have been tested.

The described technique can be represented as a block diagram shown in Figure 5. The dotted line on it marks the boundary of the technique steps that affect the structure of the algorithm and do not affect it (changing the characteristics of parallel execution due to tuning).

The advantage of the technique is that the most time-consuming steps associated with the implementation of functional blocks are not repeated when debugging the algorithm and increasing its efficiency. All changes in the algorithm in this direction are associated either with the restructuring of the algorithm without changing the functional blocks or with its reconfiguration.

Further development of the results obtained can be directed to the software implementation of the efficiency assessment method and automation of the stage associated with choosing the most efficient structure of the parallel DM algorithm.

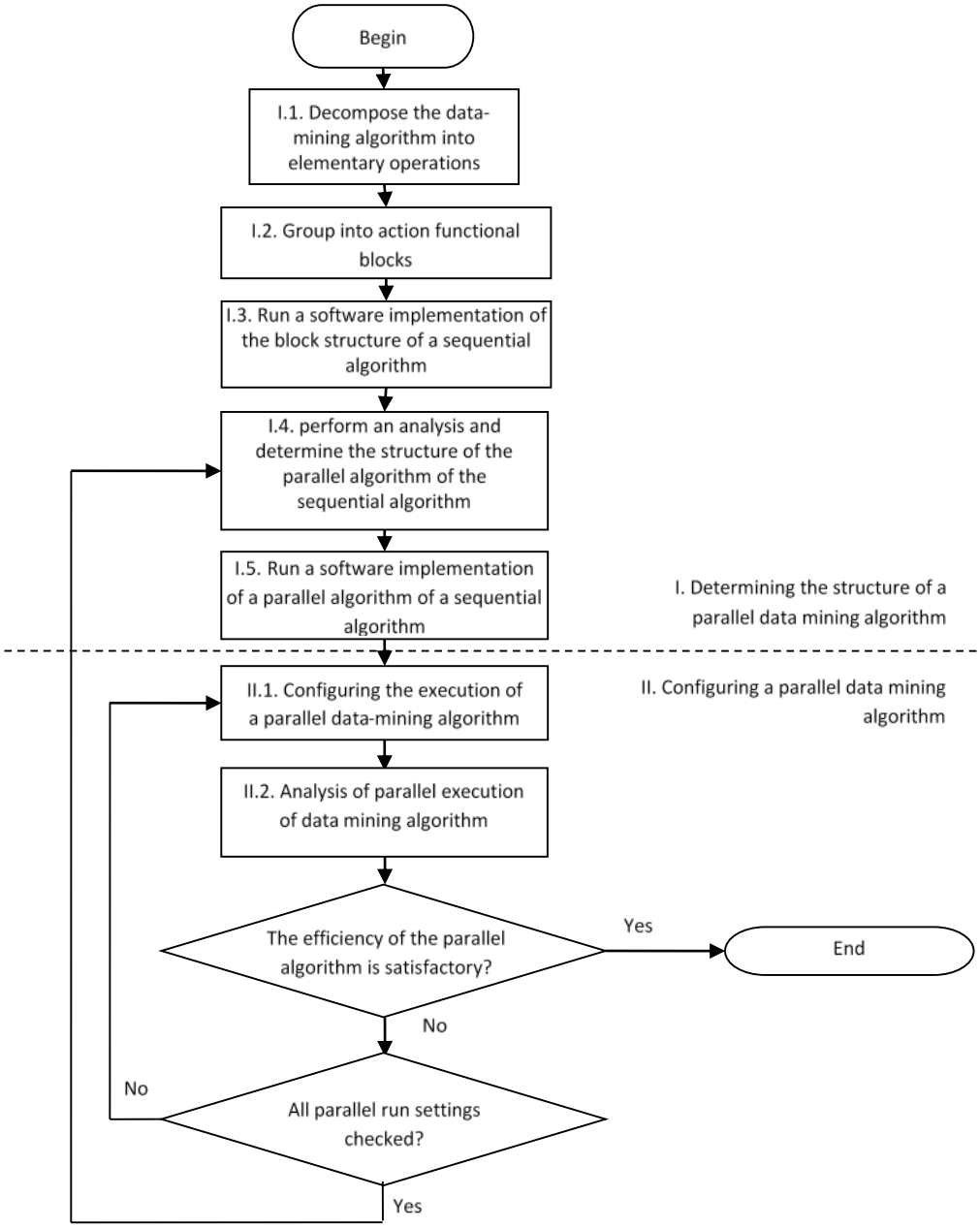

Fig. 4. Methodology parallelization of the DM algorithm

3. Result

A library of DM algorithms has been developed, which implements the block approach of constructing DM algorithms and their parallelization. The core of the library are classes implemented in accordance with the proposed DM algorithm model and allowing to implement thread-safe functional blocks of DM algorithms and execute them in parallel.

To perform a full cycle of analysis, in addition to classes for implementing algorithms, classes have been added to the library that describe: initial data and their metadata; knowledge model; algorithm level settings and DM function.

The DM algorithm library is divided into components:

i. CMW Classes - CWM standard classes;
ii. Library Core - base classes of the library;
iii. Parallel Core - classes for parallel execution;
iv. Clustering Algorithms - clustering algorithms;
v. Classification Algorithms - classification and regression algorithms;
vi. Association Algorithms - algorithms for searching for association rules.

To check the results submitted for protection using the library, the clustering and classification algorithms were parallelized - k-means and 1R in accordance with the proposed methodology.

When parallelizing the k-means algorithm, the first structure of the parallel algorithm (assuming partial parallelization) was found to be ineffective for all types of parallel execution of the DM algorithm (Table 1).

Table 1
Execution of parallel k-means algorithm (assuming partial parallelization)

Number of vectors	150	1500	3000	6000
Sequential algorithm	27.834 ms	36.918 ms	553.274 ms	1035.382 ms
Parallel algorithm				
SDSM	40.893ms	118.984 ms	1219.746 ms	2760.848 ms
SDMM	68.804 ms	112.436 ms	1451.084 ms	3783.674 ms
MDSM	35.516 ms	45.218 ms	619.356 ms	1466.711 ms
MDMM	49.449 ms	77.365 ms	811.494 ms	2138.980 ms

To improve efficiency in accordance with the methodology, it was decided to restructure the algorithm and execute the entire algorithm in parallel. The restructuring was done with a minor code change. This algorithm is implemented in 528 lines of code, and only four of them, that is, 0.75% of the code, were changed to change the structure of the algorithm. Repeated experiments showed the effectiveness of the new structure for all types of parallel execution except SDMM for data with the number of vectors 150 and 1500 (Table 2).

Table 2
Execution of parallel k-means algorithm (assuming execute the entire algorithm in parallel)

Number of vectors	150	1500	3000	6000
Sequential algorithm	27.834 ms	36.918 ms	553.274 ms	1035.382 ms
Parallel algorithm				
SDSM	19.445 ms	27.182 ms	341.904 ms	676.175 ms
SDMM	27.327 ms	41.651 ms	372.165 ms	722.527 ms
MDSM	15.669 ms	20.612 ms	303.576 ms	587.922 ms
MDMM	15.585 ms	21.106 ms	289.781 ms	613.402 ms

The models obtained as a result of the execution of serial and parallel k-means algorithms do not differ, that is, when the algorithm is parallelized, the result of its work has not changed.

When parallelizing the 1R algorithm, two parallel structures were also experimentally tested: with partial parallelization and with parallel execution of the entire algorithm. The first structure of the algorithm gives efficient results for all types of parallel execution except SDMM and SDSM for data with 150 vectors (Table 3). The second structure of the algorithm gives efficient results for all types of parallel execution except SDMM (Table 4). The restructuring of the parallel 1R algorithm also did not require much effort and was done with a minor code change. This algorithm is implemented in 613 lines of code, and only four of them, that is, 0.65% of the code, were changed to change the structure of the algorithm.

Table 3
Execution of parallel 1R algorithm (assuming partial parallelization)

Number of vectors	150	1500	3000	6000
Sequential algorithm	18.114 ms	33.668ms	42.704 ms	60.97 ms
Parallel algorithm				
SDSM	18.665 ms	31.919 ms	41.446 ms	58.316
SDMM	19.005 ms	35.365 ms	44.428 ms	63.674
MDSM	13.238ms	27.103 ms	33.795 ms	47.813
MDMM	13.719 ms	26.762 ms	33.902 ms	46.265

Table 4
Execution of parallel 1R algorithm (assuming execute the entire algorithm in parallel)

Number of vectors	150	1500	3000	6000
Sequential algorithm	18.114 ms	33.668ms	42.704 ms	60.97 ms
Parallel algorithm				
SDSM	11.789 ms	20.803ms	41.512 ms	58.789 ms
SDMM	18.258 ms	34.917 ms	44.014 ms	60.632ms
MDSM	13.706 ms	27.983ms	35.569 ms	47.812 ms
MDMM	13.506 ms	26.883 ms	32.352ms	43.917 ms

The models obtained as a result of the execution of sequential and parallel algorithms 1R also do not differ, that is, when the algorithm is parallelized, the result of its work has not changed.

The experiments carried out proved the effectiveness of the proposed solutions and confirmed their correctness.

4. Conclusion

Thus, the following results were obtained in the work:

i. A formal model of the DM algorithm is proposed, which differs in the representation of the algorithm as a set of independent operations that change the state of the knowledge model and structural blocks that allow modifying the structure of the algorithm, including for parallel execution. The main typical elements of algorithms, including those for parallel execution, are singled out.
ii. A method for constructing DM algorithms is proposed based on thread-safe function blocks, which allows parallelization both at the structural level and at the configuration level without the need to change such blocks.
iii. Proposed software templates for parallel DM algorithms based on a formal model, allowing to build parallel algorithms, both by structural changes and by customizing the execution, and allowing to separate the implementation of the algorithm from the means of distributed execution.
iv. A method of decomposition and construction of parallel data mining algorithms from thread-safe functional blocks for execution in a distributed environment is proposed, which allows not repeating the most time-consuming steps when optimizing the algorithm. All changes in the algorithm in this direction are associated either with the restructuring of the algorithm without changing the functional blocks or with its reconfiguration.
v. A library of DM algorithms has been built for execution in a distributed environment.

References

[1] Fayyad, Usama, Gregory Piatetsky-Shapiro, and Padhraic Smyth. "From data mining to knowledge discovery in databases." AI magazine 17, no. 3 (1996): 37-37. https://doi.org/10.1609/aimag.v17i3.1230

[2] Kupriyanov M. and other. Intelligent analysis of distributed data based on cloud computing. Monograph. - St. Petersburg: Publishing house of St. Petersburg Electrotechnical University "LETI", 2011. 148 p.

[3] Kupriyanov M. and other. Data Mining in Distributed Systems. Monograph. - St. Petersburg: Publishing house of St. Petersburg Electrotechnical University "LETI", 2012. 110 p.

[4] Kupriyanov, M. S., and Karshiev, Z. A. "An Overview of Parallel Data Mining Algorithms." Problems of Computational and Applied Mathematics (2017): 127-139.

[5] Ghoting, Amol, Prabhanjan Kambadur, Edwin Pednault, and Ramakrishnan Kannan. "NIMBLE: a toolkit for the implementation of parallel data mining and machine learning algorithms on mapreduce." In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 334-342. 2011. https://doi.org/10.1145/2020408.2020464

[6] Kholod, Ivan, Andrey Shorov, Maria Efimova, and Sergei Gorlatch. "Parallelization of Algorithms for Mining Data from Distributed Sources." In *Parallel Computing Technologies: 15th International Conference, PaCT 2019, Almaty, Kazakhstan, August 19–23, 2019, Proceedings 15*, pp. 289-303. Springer International Publishing, 2019. https://doi.org/10.1007/978-3-030-25636-4_23

[7] Kholod, I. I., M. S. Kuprianov, E. V. Titkov, A. V. Shorov, E. V. Postnikov, I. G. Mironenko, and S. S. Sokolov. "Training normal Bayes classifier on distributed data." *Procedia Computer Science* 150 (2019): 389-396. https://doi.org/10.1016/j.procs.2019.02.068

[8] Zaki, Mohammed J. "Parallel and distributed data mining: An introduction." In *Large-scale parallel data mining*, pp. 1-23. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. https://doi.org/10.1007/3-540-46502-2_1

[9] Zaki, Mohammed J., and Ching-Tien Ho, eds. *Large-scale parallel data mining*. No. 1759. Springer Science & Business Media, 2000. https://doi.org/10.1007/3-540-46502-2

[10] Talia, Domenico. "Parallelism in knowledge discovery techniques." In *International Workshop on Applied Parallel Computing*, pp. 127-136. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. https://doi.org/10.1007/3-540-48051-X_14

[11] Mehta, Manish, Rakesh Agrawal, and Jorma Rissanen. "SLIQ: A fast scalable classifier for data mining." In *Advances in Database Technology—EDBT'96: 5th International Conference on Extending Database Technology Avignon, France, March 25–29, 1996 Proceedings 5*, pp. 18-32. Springer Berlin Heidelberg, 1996. https://doi.org/10.1007/BFb0014141

[12] Shafer, John C., Rakesh Agrawal, and Manish Mehta. "A scalable parallel classi er for data mining." In *Proc. 22nd International Conference on VLDB, Mumbai, India*. 1996.

[13] Dafir, Zineb, Yasmine Lamari, and Said Chah Slaoui. "A survey on parallel clustering algorithms for big data." *Artificial Intelligence Review* 54 (2021): 2411-2443. https://doi.org/10.1007/s10462-020-09918-2

[14] Dhillon, Inderjit S., and Dharmendra S. Modha. "A data-clustering algorithm on distributed memory multiprocessors." In *Large-scale parallel data mining*, pp. 245-260. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. https://doi.org/10.1007/3-540-46502-2_13

[15] Zaiane, Osmar R., Mohammad El-Hajj, and Paul Lu. "Fast parallel association rule mining without candidacy generation." In *Proceedings 2001 IEEE international conference on data mining*, pp. 665-668. IEEE, 2001. https://doi.org/10.1109/ICDM.2001.989600

[16] Niu, Xinzheng, Mideng Qian, Chase Wu, and Aiqin Hou. "On a parallel spark workflow for frequent itemset mining based on array prefix-tree." In *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, pp. 50-59. IEEE, 2019. https://doi.org/10.1109/WORKS49585.2019.00011

[17] Waghamare, Bhagyashri, and Yogesh Bodhe. "Data mining technique for reduction of association rules in distributed system." In *2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT)*, pp. 415-418. IEEE, 2016. https://doi.org/10.1109/ICACDOT.2016.7877619

[18] Zaynidin, Karshiev, and Ivan Kholod. "Block structure of data mining algorithms." In *2016 International Conference on Information Science and Communications Technologies (ICISCT)*, pp. 1-3. IEEE, 2016. https://doi.org/10.1109/ICISCT.2016.7777378.

[19] Kholod, Ivan, Zaynidin Karshiyev, and Andrey Shorov. "The formal model of data mining algorithms for parallelize algorithms." *Soft Computing in Computer and Information Science* (2015): 385-394. https://doi.org/10.1007/978-3-319-15147-2_32

[20] Karshiev Z.A., Kupriyanov M. "Method of Building Parallel Data Mining Algorithms by Thread-Safe Functional Blocks." *International Journal on Recent and Innovation Trends in Computing and Communication* 4, no. 10 (2016): 180-83. https://doi.org/10.17762/ijritcc.v4i10.2581

[21] Karshiev Z.A. and Sattarov M. "Technique of Building Parallel Data Mining Algorithms." *International Journal of Innovative Technology and Exploring Engineering* 9, no. 2 (2019): 3331-3336. https://doi.org/10.35940/ijitee.B7930.129219

[22] Amin, Zamree, and Roslina Mohammad. "Bowtie analysis for risk assessment of confined space at sewerage construction project." *Progress in Energy and Environment* (2023): 22-34. https://doi.org/10.37934/progee.24.1.2234

[23] Kiew, Peck Loo, Nur Ainaa Mohd Fauzi, Shania Aufaa Firdiani, Man Kee Lam, Lian See Tan, and Wei Ming Yeoh. "Iron oxide nanoparticles derived from Chlorella vulgaris extract: Characterization and crystal violet photodegradation studies." *Progress in Energy and Environment* (2023): 1-10. https://doi.org/10.37934/progee.24.1.110

[24] Kek, Hong Yee, Huiyi Tan, Desmond Daniel Chin Vui Sheng, Yi Lee, Nur Dayana Ismail, Muhd Suhaimi Deris, Haslinda Mohamed Kamar, and Keng Yinn Wong. "A CFD assessment on ventilation strategies in mitigating healthcare-associated infection in single patient ward." *Progress in Energy and Environment* (2023): 35-45. https://doi.org/10.37934/progee.24.1.3545

[25] Ha, Chin Yee, Terh Jing Khoo, and Jia Xuan Loh. "Barriers to green building implementation in Malaysia: A systematic review." *Progress in Energy and Environment* (2023): 11-21. https://doi.org/10.37934/progee.24.1.1121