EFFICIENT DISTRIBUTION AND PROCESSING OF DATA FOR PARALLELIZING DATA MINING IN MOBILE CLOUDS

H.VenkataSubbaiah 1, MEDE CHITTI BABU 2, MATOORI KIRAN 3
4.CHANDRAKALA, 5. M.SIRISHA
1Assistant Professor, Department of Computer science &Engineering,
Ellenki college of Engineering and Technonlogy, patelguda (vi), near BHEL ameenpur (m),
Sangareddy Dist. Telangana 502319..

Abstract

We study different kinds of data distributions for improving the efficient, parallelized implementation of data mining in mobile cloud systems. Our formally-based approach ensures the correctness of the obtained parallel implementation. We apply our approach to parallel implementation of data min- ing algorithms in systems where a cloud is accessed via a mobile (wireless) network. Our approach derives a parallel implementation of a data mining algorithm that performs as much as possible computations at local servers of a mobile network, rather than transferring data for processing to a high-performance cluster in the cloud as it is done in the current cloud systems based on MapReduce. We implement our approach by extending the Java-based library DXelopes, and we illustrate our re- sults with the popular data-mining Normal Bayes classifier training algorithm. Our experiments on real-world data sets confirm that our approach significantly reduces the network traffic and the appli- cation run time.

Keywords: mobile cloud, wireless networks, parallel algorithms, distributed algorithms, distributed data mining, parallel data mining

1   Introduction

Mobile cloud computing systems unite a high number of mobile smart devices that produce large vol- umes of data with cloud resources for processing these data. The recent rise of mobile clouds is due to the convenience of their use: no need for cable installation, ease of connection, etc. However, mobile networks have several limitations, in particular - reduced bandwidth.

Figure 1 presents a typical mobile cloud computing architecture that comprises the following three layers of hierarchy [1] as shown in the figure:

• the mobile device includes hardware devices that produce large volume of data;the mobile network is responsible for transferring data from the mobile devices to the cloud for further processing;the cloud provides services or applications that integrate and/or analyze the data received from the all the mobile networks.
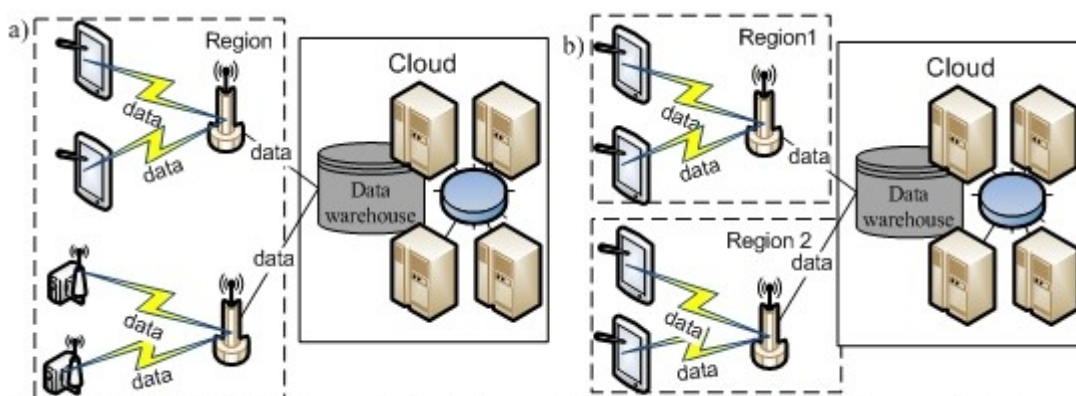
Figure 1: Example of mobile cloud with: a) vertically distributed data; b) horizontally distributed data

A system in Figure 1 receives data from different mobile devices. The mobile network connects these devices with a cloud where data from mobile devices are gathered and processed in a single data warehouse [1]. There are two different cases of how data are distributed in a mobile cloud system:

Vertical distribution shown in Figure 1 a): data produced at different mobile devices are distributed corresponding to the same kind of objects or events (for example, photo/video information, health parameters in the same region etc.).

Horizontal distribution shown in Figure 1 b): data are distributed according to the mobile devices where they are produced (for example, health parameters of people located in different regions).

Today, computations on the cloud often rely on scalable data processing systems based on the MapReduce programming model [2]. Modern MapReduce frameworks, such as Apache Hadoop [3], Apache Spark [4] etc., perform distributed data processing using a data warehouse, which is shown in both Figures 1 a) and b).

The important drawback of gathering data in a centralized data warehouse is that this increases the total network traffic. In case of mobile (wireless) networks with limited bandwidth, it may become especially critical. Fog computing [5] is a possible solution to this problem: data processing is moved nearer to the mobile devices where data are produced. However, the current Map Reduce frameworks do not yet support this idea.

Therefore, we aim at enabling distributed data mining for mobile cloud systems using the ideas of Fog computing. Our approach is to decompose a data mining algorithm, such that its parts are executed locally at the mobile devices (or nearby local servers), without transferring the raw data via the mobile network to a cloud. We expect that this will reduce network traffic in the mobile cloud system. We demonstrate that such optimization of a data mining algorithm strongly depends on the type of data distribution. The paper improves our previous work on parallelizing data mining algorithms for modern processors [6] and is an extended version of the recent conference contribution [7].

2   Related Work

Nowadays, there are several platforms providing data mining services for cloud computing, offered by leading IT vendors: Azure Machine Learning [8] from Microsoft, Amazon Machine Learning [9], Cloud Machine Learning platform [10] from Google, and Watson Analytics [11] from IBM.

The common feature of these data mining frameworks is that they are based on the MapReduce programming model [2]. This model uses the abstraction inspired by the functional primitives map and reduce that are popular as patterns in parallel and functional programming [12]. If these functions can be executed in parallel on the servers of a mobile cloud system then this brings potentially high performance of data mining.

Figure 2 a) shows how a data mining algorithm is usually executed when using the MapReduce model: raw data from mobile devices are gathered via the mobile network in a single warehouse where they are then processed.
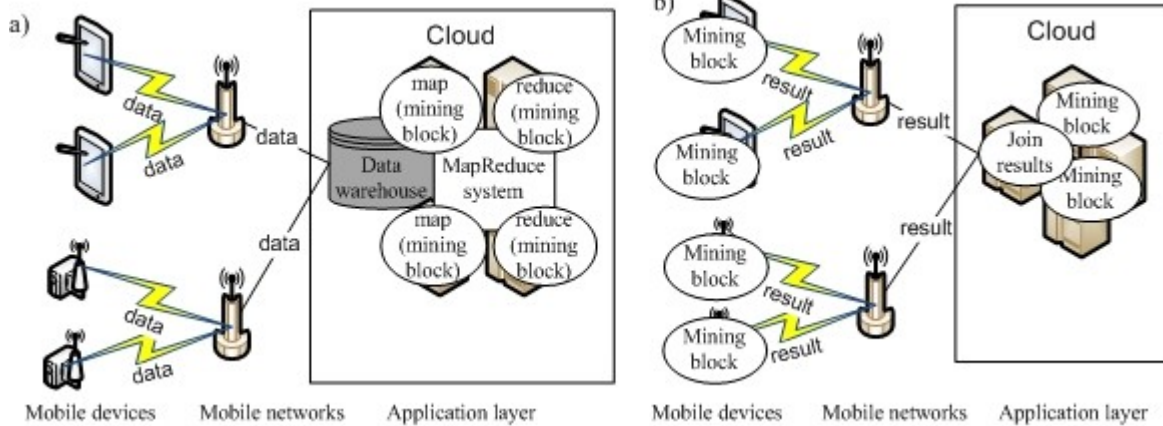

Figure 2: Variants of data mining algorithms applied to distributed data: a) traditional MapReduce pro- gramming model; b) suggested approach following Fog paradigm.

In case of using the MapReduce model in a mobile cloud system, the mobile network is responsible for connecting mobile devices (such as smartphones) with a cloud. These connections often negatively impact the performance of data mining, because they:

• create an intensive traffic in the network;

• increase the delay between data production at devices and data processing;

• increase the risk of an unauthorised access to personal data.

We aim at using the idea of Fog computing [5] for overcoming these problems. In the Fog, data are processed closer to their sources, thus enabling low latency and context awareness. Despite the popularity of Fog computing, there are still no ready solutions for its implementation in the context of data mining, especially in a mobile cloud.

Figure 2 b) shows the idea of our approach to data mining in accordance with the principles of Fog computing: we try to execute significant parts of a data mining algorithm at devices, instead of transferring data to the compute cluster and processing them there; the cluster receives intermediate results after data processing at mobile devices. Our approach optimizes the structure of the parallelized data mining algorithm according to the type of data distribution: horizontal or vertical.

3   The Formal Functional Approach

We develop our approach to be general, formally based, and covering a broad class of data mining algorithms. We briefly describe the approach below, with capital letters used for data types and lower- case letters denoting variables of these types and functions.

Representation of Data Mining Algorithms

We view a data mining algorithm as a function that takes a data set $d \in D$ as input and creates a mining model $m \in M$ as output:

$$dma : D \rightarrow M \qquad (1)$$

A data set d D usually contains characteristics (such as temperature, sound level, vibration, pressure, etc.) of objects (e.g., people, vehicles, etc.). We represent a data set as a 2-dimensional array (data matrix), e.g., for z objects that are described by p characteristics [13]:

$$d = (x_{j.k})_{z,p}$$

(2)

where $x_{j.k}$ is the value of kth characteristic of the jth object. The set of values of kth column is denoted as $T_k(x_{j.k}T_k): T_k = v_{k.1}, \ldots v_{k.u}$.

A mining model comprises elements that describe knowledge extracted by a data mining algorithm

from a data set. These elements can be, e.g., classification or association rules, cluster centers, decision tree nodes, etc. In (1), a data mining algorithm creates a mining model, without changing the input data. We represent a mining model $m \in M$ as an array of elements $e_i, i = 0, \ldots, u$:

$$m = [e_0, e_1, \ldots, e_u] \quad (3)$$

Since algorithms are usually a sequence of steps, we formally represent a data mining algorithm as a sequential composition of functions, as follows:

$$dma = f_n \circ f_{n-1} \circ \ldots \circ f_1 \circ f_0 \quad (4)$$

where $\circ$ is the composition operator that is applied from right to left.

In (4) function $f_0 : D \rightarrow M$ takes a data set $d \in D$ as an argument and returns a mining model $m_0 \in M$. The next functions $f_t, t = 1 \ldots n$ take the mining model $m_{t-1} \in M$ created by the previous function f_{t-1} and return the changed mining model $m_t \in M$.

$$f_t : M \rightarrow M \quad (5)$$

The functions $f_t, t = 1 \ldots n$ of type (4) are called Functional Mining Blocks (FMB). Some steps of a data mining algorithm use data matrix d to change the mining model m, i.e., they take the data matrix d as additional argument d:

$$f_{dt} : D \rightarrow M \rightarrow M \quad (6)$$

To use these functions in the composition (4), we exploit partial function application with the fixed first argument: $f_t = f_{dt} d$. Thus d is constant in the function f_t.
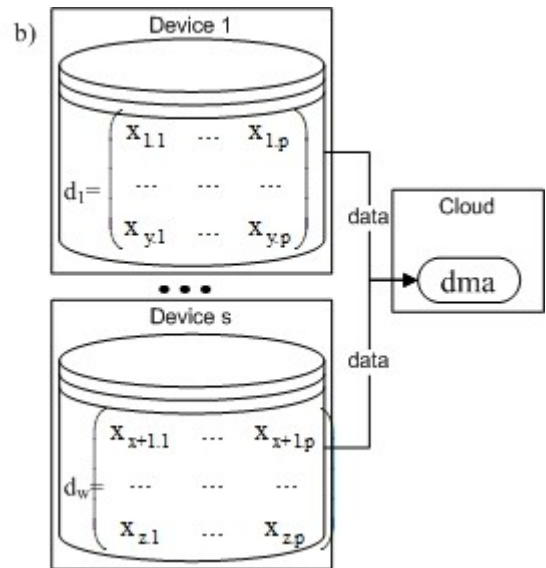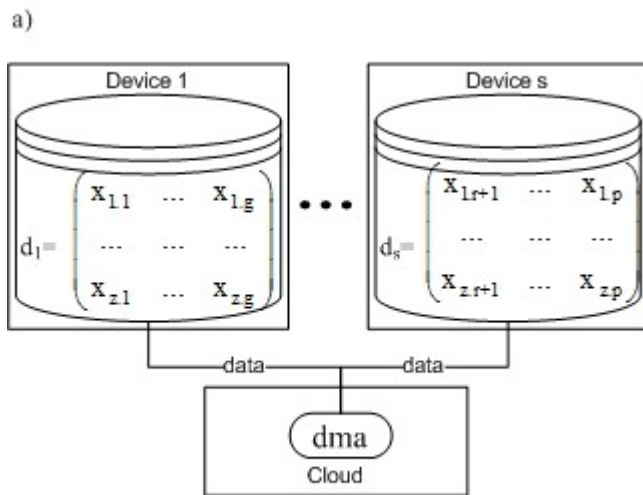
In order to apply some function f_{dt} to each element of the data matrix, we invoke it in a loop. We

introduce loops over columns and rows of the data matrix for iterative processing of data. We use an asterisk to refer to a whole row (e.g., d[j,] refers to the jth row) or whole column (e.g., d[, k] refers to the kth column) in a data matrix:

- loopc applies fdt to the columns of $d \in D$, starting from index is till index ie:

loopr : $I \rightarrow I \rightarrow (M \rightarrow M) \rightarrow D \rightarrow M \rightarrow M$ (8)

loopr is iefdt d m = (fdt d[ie, ∗]) ∘●●●∘ (fdt

d[is, ∗]) m; The first four arguments are fixed to use loopc and loopr in the composition (4).

loopc : $I \rightarrow I \rightarrow (M \rightarrow M) \rightarrow D \rightarrow M \rightarrow M$ (7)

loopc is iefdt d m = (fdt d[∗, ie]) ∘●●●∘ (fdt d[∗, is]) m;

- loopr applies fdt to the rows of $d \in D$ starting from index is till index ie:

Distributed data

Figure 3 represents a distributed storage: data matrix d is split between s devices: example in Figure 1 b).

$d = d1 \cup ●●● \cup ds$ (9)

where data sub-matrix dh, h = 1..s is located at hth device.

Figure 3: Traditional processing distribution of data matrix: (a) vertical; (b) horizontal The following two cases of data distribution are possible:

Figure 3 a) shows the vertical distribution of the data matrix. This case corresponds to the example

in Figure 1 a);

Figure 3 b) presents the horizontal distribution of the data matrix. This corresponds to the

Functions for data mining on distributed memory

Traditionally, data processing is performed by a data mining algorithm on a cloud as shown in Figure 2 a). For this, all original data taken at devices have to be transferred to the cloud, which is the critical problem addressed in this paper.

Our alternative to the traditional MapReduce approach is to execute FMBs of type (6) on the devices and transfer only mining models (as result) created by them, rather than large amounts of data as in the traditional MapReduce. In addition, the functions that are parts of the data mining algorithms will be executed in parallel in our approach, i.e., we

have parallel execution with distributed memory.

We introduce higher-order function paralleld that expresses the parallel execution of FMBs on a distributed memory:

$$paralleld : [(M \rightarrow M)] \rightarrow M \rightarrow M \quad (10)$$

paralleld [fr, . . . , fs] m = join m forkd [fr, . . . , fs] m,

where function forkd allows to invoke FMBs in parallel:

$$forkd : [M \rightarrow M] \rightarrow M \rightarrow [M] \quad (11)$$

forkd [fr, . . . , fs] m = [fr (copy m),..., fs(copy m)]

and function copy creates copies of the mining model m in separate areas of the distributed memory for parallel processing by FMBs:

$$copy : M \rightarrow M, \quad (12)$$

copy m = [m[0], m[1],..., m[v]].

Function join in (10) combines the mining models built by parallel FMBs in separate areas of distributed memory:

$$join : M \rightarrow [M] \rightarrow M \quad (13)$$

join m [mr, . . . , ms] = [m′[0], . . . , m′[g], . . . , m′[v]],

where m′[g] = m[g], if mi[g] = m[g] for all i = r . . . s

union m[g] [mr[g], . . . , ms[g]], otherwise

Here, function union merges elements of different mining models with the same index to a single mining model's element:

$$union : E \rightarrow [E] \rightarrow E \quad (14)$$

The implementation of function union depends on the mining model's elements.

Figure 4 shows the distributed execution of a data mining algorithm using function paralleld defined by (10). The definition of function paralleld does not take into account the type of data distribution. We show in the following that this is important for the ultimate performance.
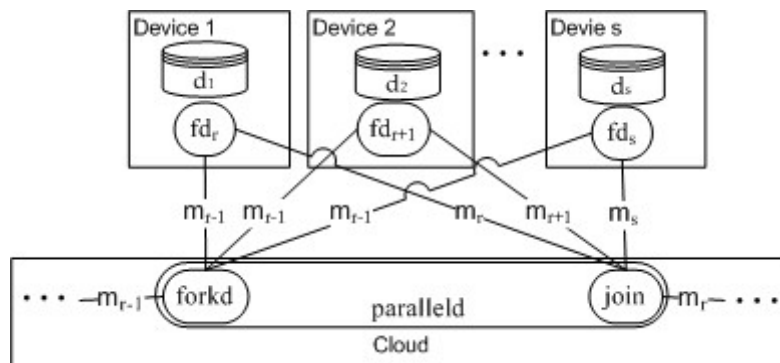

Figure 4: Execution of a data mining algorithm on distributed data.

Data Mining on Distributed Data

For iterative processing of rows and columns of data matrix d, both loops loopc (7) and loopr (8) are used as nested. There are two possible orders of applying an FMB fdt to the elements of data matrix d:

• by columns (i.e., in the following order: x1.1, . . . , xz.1, x1.2, . . . , xz.2, . . .):

$$(loopc\ 1\ p\ (loopr\ 1\ z\ fdt))\ d; \quad (15)$$

• by rows (i.e., in the following order: x1.1, . . . , x1.p, x2.1, . . . , x2.p, . . .):

(loopr 1 z (loopc 1 p fdt)) d. (16)

For parallel processing of the distributed data matrix (9), we decompose expressions (15) and (16) in the FMBs that are executed on the devices and process only submatrices stored in them locally. For this, the loops are decomposed as follows:

• for horizontal data distribution (distribution of data matrix's rows), the loop loopr is decomposed;

• for vertical data distribution (distribution of data matrix's columns), the loop loopc is decomposed. Function paralleld is used to execute the FMBs of the composition distributed on the devices. We further consider how the number of functions paralleld depends on the type of data distribution and the

order of nesting the loops (15) or (16).

We first consider the case when data matrix d is distributed vertically (Figure 3 a)). In expression (15), we decompose the outer loop loopc:

(loopc 1 p (loopr 1 z fdt)) d = ((loopc r + 1 p (loopr 1 z fdt)) ds) ∘••• ∘((loopc 1 g (loopr 1 z fdt)) d1).

In this case, we use the single function paralleld to execute FMBs of the composition in a parallel manner:

paralleld [(loopc 1 g (loopr 1 z fdt)), . . . , (loopc r + 1 p (loopr 1 z fdt))].

In expression (16), we decompose the inner loop loopc:

(loopr 1 z (loopc 1 p fdt)) d = (loopr 1 z (((loopc r + 1 p fdt) ds) ∘ • • • ∘ ((loopc 1 g fdt) d1)).

Therefore, function paralleld is invoked m

times from the outer loop loopr:

(loopr 1 z paralleld[((loopc 1 g fdt) d1), . . . , ((loopc r + 1 p fdt) ds)].

When data matrix d is distributed horizontally (Figure 3 b)) in expression (15), we decompose the inner loop loopr:

(loopc 1 p (loopr 1 m fdt)) d = loopc 1 p (((loopr x + 1 z fdt) ds) ∘...∘ ((loopr 1 y fdt) d1)).

As the result, function paralleld is invoked from the outer loop p times:

(loopc 1 p paralleld[((loopr 1 y fdt) d1), . . . , ((loopr x + 1 z fdt) ds)].

However, in expression (16) for horizontal data distribution (Figure 3 b)), we decompose the outer loop

loopr:

(loopr 1 z (loopc 1 p fdt)) d = ((loopr x + 1 z (loopc 1 p fdt)) ds)∘••• ∘((loopr 1 y (loopc 1 p fdt)) d1)),

so for parallel execution of FMBs in the composition we can employ one function paralleld: paralleld [((loopr 1 y (loopc 1 p fdt)) d1), . . . , (loopr x + 1 z (loopc 1 p fdt)) ds)].

Table 1 summarizes the number of invoking function paralleld for different kinds of data distributions

and order of nesting the loops (15) and (16). Summarizing, we see that a non-optimal order of nesting loops and the type of data distribution increases the number of functions paralleld. Each invocation of paralleld leads to executing functions forkd and join. This increases the total run time of the whole algorithm, so it is important to reduce the number of invocations of paralleld. This can be achieved by adjusting the order of these loops according to the type of data distribution, as described in the next section.

Typeofdistribution	Orderofnestingtheloops	Numberofinvokingfunction*paralleld*

| VerticalHorizontal | $(loopc1 \; p \; (loopr1 \; z \; fd_t))$ $d(loopr1 \; z \; (loopc1 \; p \; fd_t))$ $d(loopc1 \; p \; (loopr1 \; z \; fd_t))$ $d(loopr1 \, z(loopc1 \, pfd_t))d$ | 1 z(thenumberofrowsindatamatrix$d$) p(thenumberofcolumnsindatamatrix$d$)1 |

Table 1: Depending the number of functions paralleld from data distribution and nesting the loops

Adaptation of Loops to the Type of Data Distribution

Adjusting the execution order of the nested loops loopr and loopc according to the type of data distribu- tion can be achieved by executing the loop interchange operation which is often used in compilers [14]. It changes the order of iteration variables in a group of nested loops. For example, the iteration variable used in the inner loop can be moved to the outer loop and vice versa:

loopc 1 p (loopr 1 z fdt) = loopr 1 z (loopc 1 p fdt)

In the simplest case, a single FMB fdt is applied to each of the elements of the data matrix d. In a more complex case, FMB called from the outer loop can be a composition of several FMBs and not all of them process data matrix elements.

Let us look at the following example:

loopc 1 p (fdt−1 ∘ (loopr 1 z fdt) ∘ fdt+1).

Here, operation interchange cannot be applied, because outer loop loopc invokes not only inner loop loopr but also FMBs fdt−1 and fdt+1. Therefore, it is necessary to perform the loop fission operation prior to changing the order. This operation is also widely used in compilers [14]. It breaks down a loop to several loops, each of which has the same index boundaries but contains only a part of initial loop's body:

loopc 1 p (fdt−1 ∘(loopr 1 z fdt)∘ fdt+1) = (loopc 1 p fdt−1)∘(loopc 1 p (loopr 1 z fdt))∘(loopc 1 p fdt+1).

As a result, the operation interchange is applied only to the second function of the composition:

(loopc 1 p fdt−1) ∘ (loopc 1 p (loopr 1 z fdt)) ∘

(loopc 1 p fdt+1) = (loopc 1 p fdt−1) ∘ (loopr 1 z (loopc 1 p fdt)) ∘ (loopc 1 p fdt+1).

Applying at first the operation f ission and then the operation interchange to the nested loops loopc and loopr allows us to adjust the data mining algorithm's structure to the type of data distribution in order to reduce the number of invocations of the function paralleld and the total run time of the algorithm.

Summarizing, a data mining algorithm is parallelized for distributed execution on the mobile devices in the cloud by following the steps below:

1.   represent the algorithm as a composition (4) of functions ft, t=0..n and the mining model as an array of elements (3);

2.   adjustloopc and loopr loop calls according to the type of data distribution using transformations

fission and interchange:

(a)   if the type of data distribution is horizontal (vertical) then

i.   if outer loop is not loopr (loopc) then

A.   if inner FMB is a composition several FMBs then execute operation f ission

B.   execute operation interchange

3.   convert the sequential execution of FMBs, which can be performed in parallel with distributed, into parallel execution by paralleld function.

The next sections describe an example of such transformation.

Illustration for the Normal Bayes

Classifier Training Algorithm

We use the Normal Bayes classifier (NBC) training algorithm [15] as an example data mining application to illustrate our approach. We choose the NBC training algorithm because it belongs to the Top 10 data mining algorithms [16]. In the classification task, a data set d is a set of objects, each of which belongs to a known class, and each of which has a known vector of attributes. The column that defines class of object is called a dependent attribute ap (for example, health status of men). The columns with other characteristics of the object are independent attributes, (for example: temperature, pressure, age, and etc.). The NBC training algorithm based on applying Bayes' theorem with the "naive" assumption of independence between every pair of attributes. The classic Bayes method to approximate $P(x_{j.k}\ x\ j.p)$ for a continuous variable $x_{j.k}$ is assuming that it follows a known distribution for which parameters can be estimated from the data. Gaussian distribution (normal distribution) is the most popular and most-used statistical continuous distribution method. It has two parameters: mean μ and variance σ^2. For each of the class values $v_{p.r} \in T_p$, we assume that:

$P(x_{j.k} = v_{k.q}|x_{j.p} = v_{p.r}) = f(x_j, \mu, \sigma)$.

where $f(x_j, \mu, \sigma)$ is the normal distribution with mean μ and variance σ :

$$f(x_j, \mu, \sigma) = \sigma \sqrt{2\pi}\ e$$

$$\frac{(x-\mu)^2}{2\sigma^2}\ .$$

The means μ and variances σ are estimated from the values of d in rows for which $x_{j.p} = v_{p.r}$ simply through unbiased estimates:

$$\mu \quad \frac{1}{z} \quad 2$$

$$\frac{1}{z}\ 2$$

$$= \quad \sum_{j=1} x_j \text{ and } \sigma$$

$$= z - 1 \sum(x_j - \mu)\ .$$

where z is the number of rows in d. The benefit of the normal method is that it performs well if the un- derlying distribution indeed follows a normal distribution [15], is fast in both learning and classification time and requires little memory.

To optimize the algorithm, we apply the following transformations:

$$\sigma^2 \quad \frac{1}{z} \quad 2 \qquad z \qquad 2 \qquad 2$$

$$= z - 1 \sum(x_j - \mu)$$

$$= \sum_{j=1} x_j \qquad (z \qquad 1)\mu$$

This optimization allows us to perform NBC training algorithm for two phases:

1. calculate the sum and the sum of squares within one iteration through matrix d to obtain the total calculation for each class;

2. calculate μ and σ^2 for each class, which are saved in mining model m and used to calculate possibility of belonging of a new row to one of the classes.

```
1. forj=1...z//loopforeachvector
2.    fork=1...p-1//loopforeachattribute
3.        sv[d[j.p]][k]=sv[d[j.p]][k]+d[j.k];//calculatesumofvaluesforeachclass
4.        ssv[d[j.p]][k]=ssv[d[j.p]][k]+d[j.k]2;//calculatesumofsquareofvaluesforeachclass
5.    endfor;
6. endfor;
7. forr=1...c//loopforeachclasses
8.    fork=1...p-1//loopforeachattribute
9.        µ[r][k]=sv[r][k]/z;//calculatemean
10.       σ2[r][k]=ssv[r][k]-(z-1)·sv[r][k]·sv[r][k];//calculatevariance
11.    endfor;
12. endfor;
```

Figure 5: The Normal Bayes classifier training algorithm: pseudocode.

Figure 5 shows a pseudocode of the NBC training algorithm.

The average value μ and deviation σ calculation functions do not have the properties of list homo- morphism. Thus, the average values and deviations calculated for each distributed data sources d_h set individually cannot be correctly united (unable to obtain result equal to calculation based on the total data d).

Hence, the NBC training algorithm cannot be executed completely for each of data sources d_h with the further joining of the results because the joint result may be incorrect. As an alternative to the com- plete execution of the algorithm per each device, the algorithm can be distributed between these devices. To do this, we parallelize the NBC construction algorithm using the described approach. According to it, the algorithm is represented as a composition of FMB.

The mining model for NBC training algorithm comprises:

• mining model's element m[0] is number of classes (m[0] = |Tp|);

• mining model's elements m[1],..., m[s] are the sums of values (m[q] = a[d[j.p]][k], 1 ≤ q ≤ s, s =

|Tp| • p);

• mining model's elements m[s+1],..., m[u] are the sums of square of values (m[g] = b[d[j.p]][k], s <

g ≤ u, u = s + |Tp| • p);

• mining model's elements m[u + 1], . . . , m[h] are the means (m[i] = μ[r], u < i ≤ h, h = u + |Tp| • p);

• mining model's elements m[h +1],..., m[w] are the means (m[l] = σ 2[r], h < l ≤ w, w = h +|Tp|• p). The NBC classifier can be represented as a composition of the FMBs as follows:

• fd1 is the loop for rows of data d (line 1 in the pseudocode):

fd1 d m = loopr 1 z fd2 d m;

• fd2 is the loop for columns of data d (line 3 in the pseudocode):

fd2 d m = loopc 1 p fd4 ∘ fd3 d m;

• fd3 calculate sum of values for each class (line 3 in the pseudocode):

fd3 d m = m[d[j.p] • k] = m[d[j.p] • k] + d[j.k];

• fd4 calculate sum square values for each class (line 4 in the pseudocode):

fd4 d m = m[s + d[j.p] • k] = m[s + d[j.p] • k]

+ d[j.k]2

f5 is the loop for classes (line 7 in the pseudocode) applies a function f6 to list of values some column from index is till index ie:

f5 m = loopv 1 m[0] f6 m loopv : I → I → (M → M) → M → M

loopv is ieft m = (f6 m[ie]) ∘ • • • ∘ (f6 m[is])

• f6 is the loop for columns (line 8 in the pseudocode):

f6 m = loopc 1 p f8 ∘ f7 m;

• f7 calculate μ for each class (line 9 in the pseudocode):

f7 m = m[u + r • k] = m[r • k]/z;

• f8 calculate σ 2 for each class (line 10 in the pseudocode):

f8 m = m[h + r • k] = m[s + r • k] − (z − 1)• m[r • k] • m[r • k].

Thus, we can represent the NBC training algorithm as the following composition of functions:

NBC = f5 ∘ fd1 ∘ f0 = (loopv 1 m[0] (f6)) ∘ (loopr 1 z (fd2)) ∘ f0 =   (17)

(loopv 1 m[0] (loopc 1 p (f8 ∘ f7))) ∘ (loopr 1 z (loopc 1 p (fd4 ∘ fd3) d)) ∘ f0

Parallelization of the Normal Bayes Classifier Training Algorithm

After representing the NBC training algorithm as a composition of functions (17), in the second step we verify the corresponding outer loop and the type of data distribution. In expression (17), the outer loop is loopr. It corresponds to the horizontal data distribution,

therefore no transformations are needed.

In the final step, we transform the sequential form of the NBC training algorithm into a parallel form for distributed memory by inserting function paralleld. However, to use function paralleld, we need to define function union (14) for mining model's elements of the NBC training algorithm. These elements are sums of values and of square of values, therefore function union must sum up the values which are calculated in parallel in distributed memory:

union : E → [E] → E(18)

union m[q] [m1[q], . . . , mr[q]] = m[q] + m1[q] + • • • + mr[q]

Thus, function paralleld can be applied to loop loopr in expression (17) to execute it in parallel on the servers for horizontally distributed data:

NBCHPar = (loopv 1 m[0] (loopc 1 p f8 ∘ f7))∘(paralleld [loopr 1 z (loopc 1 p fd4 ∘ fd3 d)])∘ f0 (19)

Figure 6 shows deploying the FMBs of composition (19) on servers. We distribute loop loopr between the servers in according with distribution of data matrix d, i.e. on each server loop loopr process all rows only locally submatrix. For example, in the first device all rows of submatrix d1 is processed by loop loopr 1 y (loopc 1 p fd4 fd3 d). For this, the initial mining model m0 is copied and sent from the cloud to all devices. The computed mining models m4 are collected from all devices and are combined by function join into mining model m5 which is the final result of the NBC training algorithm.

Figure 6: Distributed execution of the NBC training algorithm for horizontally distributed data.
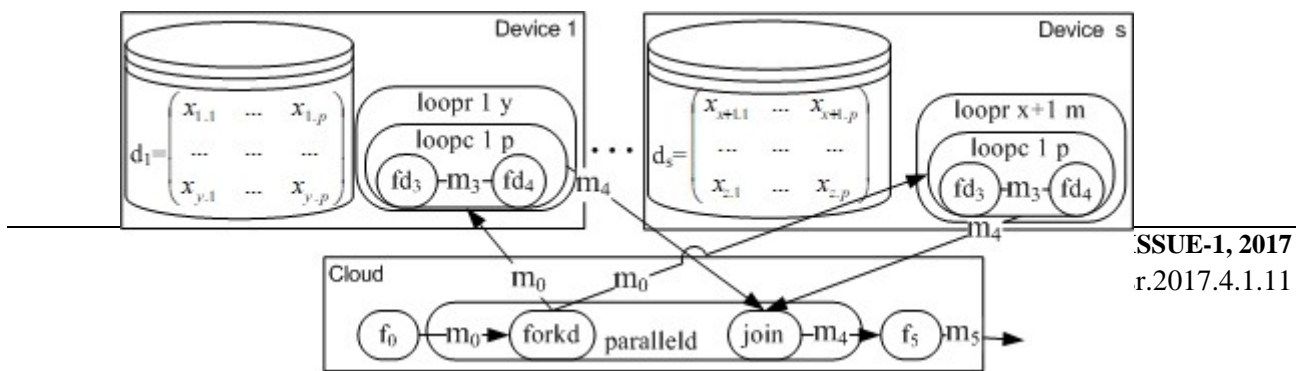
For processing vertically distributed data, we transform expression (17), because outer loop loopc does not agree with the type of data distribution. The outer loop comprises single FMB fd2, therefore executing f ission is not needed. However, we apply operation interchange to change the order of nested loops loopr and loopc:

loopr 1 z (loopc 1 p f d4 ∘ f d3 d) = loopc 1 p (loopr 1 z f d4 ∘ f d3 d).

As the result, NBC training algorithm for vertically distributed data can be represented as the following composition of functions:

NBC′ = (loopv 1 m[0] (loopc 1 p f8 ∘ f7)) ∘ (loopc 1 p (loopr 1 z fd4 ∘ fd3 d)) ∘ f0. (20)

Function union (18) can also be used to combine the elements of mining models built by FMBs on devices. Thus function paralleld can be applied to the loop loopc of expression (20) in order to execute it in parallel on the devices with distributed memory for vertically distributed data:

NBCVPar = (loopv 1 m[0] (loopc 1 p f8 ∘

f7))∘(paralleld [loopc 1 p (loopr 1 z fd4 ∘ fd3 d)])∘ f0. (21) Figure 7 shows the deployment of the FMBs of composition (21) on the servers. Here, we distribute loop

loopc and loop loopr between them in according with the distribution of data matrix d. The loop loopc is executed on servers with submatrix d1 that contains only pth depended attribute. The loop loopc is distributed between the other servers where it processes all columns of local submatrices. The interaction between the cloud and the devices is organized similarly to the case with the horizontally distributed data (Figure 6).

As a result, we derived two parallel versions of the NBC training algorithm: one for the data mining of the horizontally and one for vertically distributed data.

4 Experiments and Results

We implement the two parallel NBC training algorithms (19) and (21) in the Java-based library DXelopes [17]. With them, we perform the experiments described in the following.
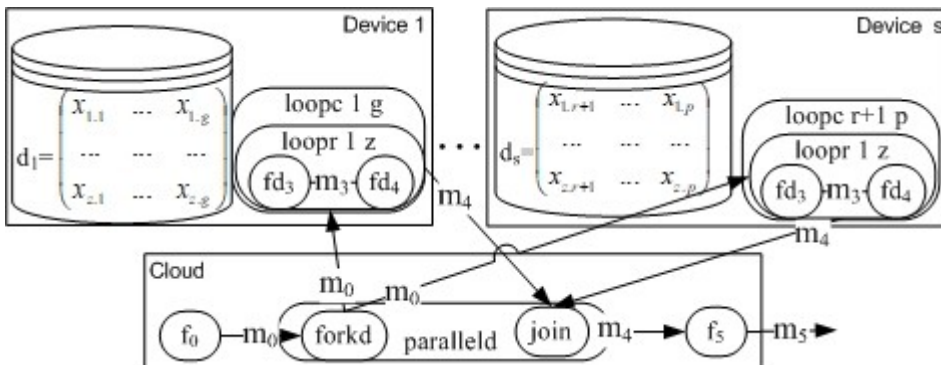

Figure 7: Distributed execution of the NBC training algorithm for vertically distributed data.

In our experiments, we use the real-world data set "Predict Outcome of Pregnancy" from the Kaggle Datasets [18]. This data set contains data from the Annual Health Survey: Woman Schedule. The data set contains 68 independent attributes related to birth, including: birth

history; type of medical attention at delivery; details of maternal health care (antenatal/natal/postnatal, immunization of children, etc.), and one dependent attribute that contains information about the outcome of pregnancy: live birth/still- birth/abortion. The data set comes as a file in CVS format of 2 Gb in size.

Table 2: Distributed data sets

Typeofdistribution	Numberv of de-ices	Numberofrows	Numberc of olumns	Sizeofdataset(Mb)
Horizontal	4	3402670	68	500
Vertical	4	14461451	17	500
Horizontal	2	6805350	68	1000
Vertical	2	14461451	34	1000
Singledataset	1	14461451	68	2000

Table 2 shows how the data are partitioned in our experiments. The data set is split into 2 and then into 4 parts and distributed between two and then between four devices, correspondingly. In each of the cases, the data set is distributed both horizontally and vertically. A non-divided data set (for a single device) is used for comparison. Each devices has the following configuration: CPU Intel Xenon, 2.90 GHz, 4 Gb. The cloud is equipped with a CPU Intel Xeon (12 physical cores), 2.90 GHz, 4 Gb. To imitate communication channel limitation between devices and cloud, we use channel throttling with level 75Mbps, that matches 4G wireless systems.

Figure 8 shows the network traffic and figure 9 shows the run time for horizontally (Figure 8 a) and Figure 9 a)) and vertically (Figure 8 b) and Figure 9 b)) distributed data. We compare the two variants of the distributed NBC training algorithm adapted to horizontally (NBCHPar) and vertically (NBCVPar) distributed data with traditional approach when data are gathered into cloud.

The traditional approach with gathering data generates larger traffic (more 10 times), because for it all data are transferred by network. When processing distributed data in our approach without gathering data, only processing mining models are transferred over the network. We observe that increasing the number of devices leads to increasing the volume of network traffic, because a higher number of results are being sent over the network.

We observe that the run time in our approach is lower than with traditional approach. These resultscan be explained by processing a smaller volume of data on each devices in parallel and also by a smaller network traffic. Also we can view that run time of NBCHPar is lower than of NBCVPar for horizontally distributed data and the run time of NBCVPar is lower than of NBCHPar for vertically distributed data. This is explained by the non-corresponding structure of the algorithm and the type of data distribution, that increases the over-head of distributed execution (number of functions paralleld). We observe that the difference between both variants is larger for vertically than for horizontally distributed data. This can be explained by larger numbers of rows than columns, so the number of invocations of functions copy and join in NBCHPar for vertically distributed data is very large. We also observe that the distinction between the two variants is reduced with a smaller number of devices. These results can also be explained by the decreased number of invocations of functions copy and join for the distributed execution of the algorithm on a smaller number of devices.
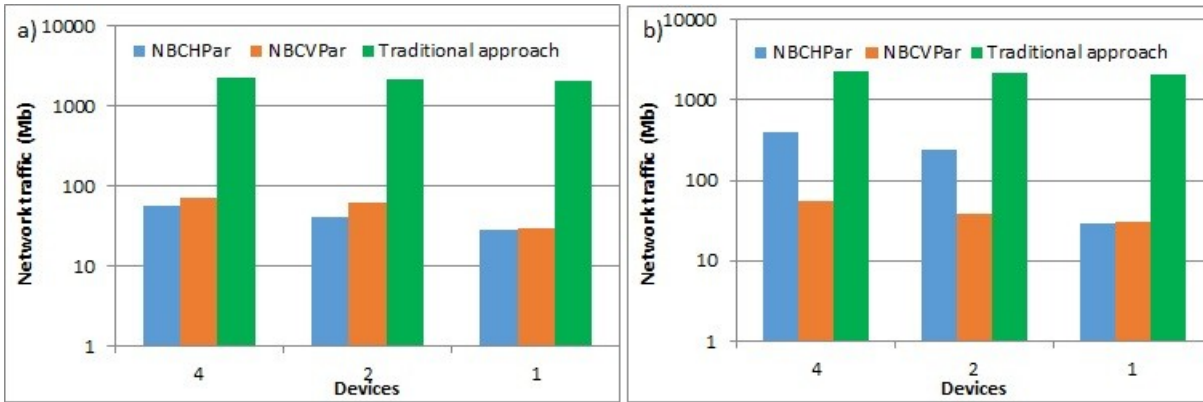
Figure 8: Network traffic of executing NBC training algorithm for: a) horizontally distributed data; b) vertically distributed data.
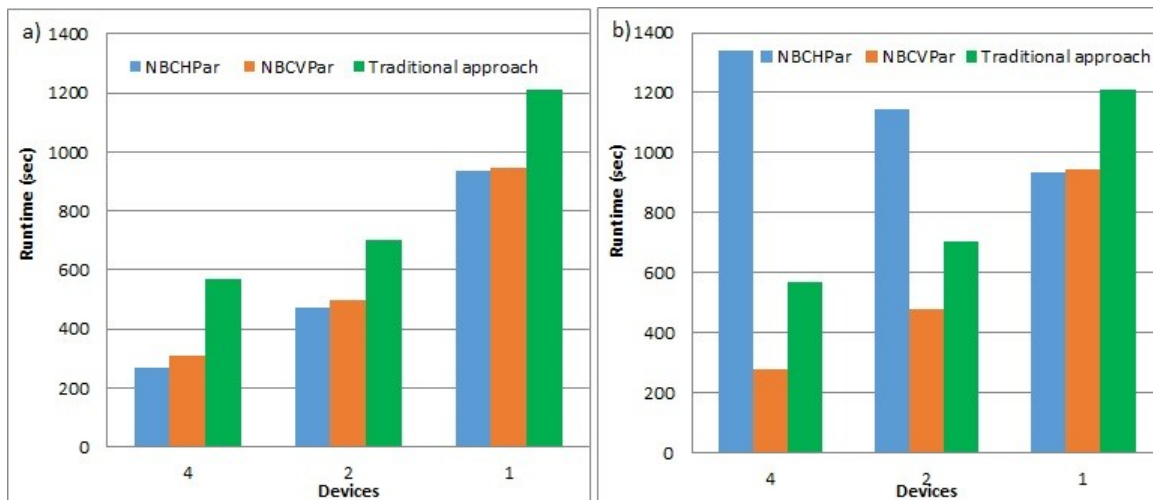

Figure 9: Runtime of executing NBC training algorithm for: a) horizontally distributed data; b) vertically distributed data.

5   Conclusion

In this paper, we improve the parallelization and implementation of data mining algorithms in modern mobile cloud systems. Our approach formally transforms a high-level functional representation of a data mining algorithm into a parallel implementation that performs as much as possible computations at local nodes, rather than transferring data for processing at a central compute cluster as it is done in the current systems based on MapReduce. We study different kinds of data distributions between the servers of mobile networks and we adapt the structure of the algorithm correspondingly. Thereby our parallel implementation of data mining avoids the main disadvantages of MapReduce in the context of mobile cloud: increased total processing time, high network traffic, and a risk of unauthorized access to the data.

We develop a functional formalism as a formal base of our approach: it enables formally ensuring the correctness of the formal program transformations and of the obtained parallel implementation. The compiler-level transformations interchange and fission adapt the structure of a data mining algorithm to the type of data distribution.

We implement our approach in the Java-based data mining library DXelopes, and we illustrate the approach with the popular data-mining NBC training algorithm.

Our experiments on real-world data set from the Kaggle collection confirm that the flexible adaptation of the distributed implementation of the NBC training algorithm to the type of data distribution by using our approach significantly reduces the net-work traffic and the application run time.

References

[1] H. T. Dinh and e. a. Chongo Lee, "A survey of mobile cloud computing: architecture, applications, and approaches," Wireless Communications and Mobile Computing, vol. 13, no. 18, pp. 1587–1611, October 2011.

[2] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in Proc. of the 6th Symposium on Operating System Design and Implementation (OSDI'04), San Francisco, California, USA. USENIX, December 2004, pp. 137–150.

[3] "Apache hadoop," http://hadoop.apache.org [Online; Accessed on March 25, 2020], Apache, 2020.

[4] "Apache spark." http://spark.apache.org [Online; Accessed on March 25, 2020], Apache, 2020.

[5] FlavioBonomi and Rodolfo Milito, "Fog Computing and its Role in the Internet of Things," in Proc. of the 1st SIGCOMM Workshop on Mobile Cloud Computing (MCC'12), Helsinki, Finland. ACM, August 2012, pp. 13–16.

[6] K. Ivan, S. Andrey, T. Evgeni, and G. Sergei, "A formally based parallelization of data mining algorithms for multi-core systems," The Journal of Supercomputing, vol. 75, no. 12, p. 7909–7920, July 2018.

[7] K. Ivan, S. Andrey, and G. Sergei, "Improving data mining for differently distributed data in iot systems," in Proc. of the 2019 International Symposium on Intelligent and Distributed Computing (IDC'19), Petersburg, Russia, ser. Studies in Computational Intelligence, vol. 868. Springer, Cham, October 2019, pp. 75–85.

[8] G. C. J., "Introduction to machine le'arning on microsoft azure," https://azure.microsoft.com/en-gb/ documentation/articles/machine-learning-what-is-machine-learning [Online; Accessed on March 25, 2020], June 2019.

[9] B. J., "Amazon machine learning – make data-driven decisions at scale." https://aws.amazon.com/blogs/ aws/amazon-machine-learning-make-data-driven-decisions-at-scale [Online; Accessed on March 25, 2020], April 2015.

[10] "Google cloud machine learning at scale," https://cloud.google.com/products/machine-learning [Online; Ac- cessed on March 25, 2020], Google.

[11] L. A. and et al., "Question analysis: How watson reads a clue." IBM Journal of Research and Development, vol. 56, no. 3.4, pp. 2:1–2:14, April 2012.

[12] G. Sergei and C. Murray, Parallel Skeletons.In: Padua D. (eds). Springer,Boston, MA, 2011.

[13] H. T., T. R., and F. J., The elements of statistical learning: Data Mining, Inference, and Prediction. Springer- Verlag New York, 2001.

[14] A. R. and K. K., Optimizing Compilers for Modern Architectures. Morgan Kaufmann, October 2001.

[15] C. Georg and B. R. L., Statistical Inference. Cengage Learning Publisher, June 2001.

[16] X. Wu and V. Kumar, "Top 10 algorithms in data mining," Knowledge and Information Systems, vol. 14, pp. 1–37, December 2007.

[17] "Prudsysxelopes." Prudsys, 2019, https://de.wikipedia.org/wiki/XELOPES.

[18] "Kaggle. dataset: Predict outcome of pregnancy." https://prudsys.de/en/knowledge/technol ogy/ prudsys-xelopes/ [Online; Accessed on March 25, 2020], Prudsys, June 2019.