Barkhordari and Niamanesh *J Big Data* (2018) 5:25
https://doi.org/10.1186/s40537-018-0129-4

Journal of Big Data

RESEARCH Open Access

Kavosh: an effective Map-Reduce-based association rule mining method

Mohammadhossein Barkhordari* and Mahdi Niamanesh

*Correspondence:
Barkhordari@ictrc.ac.ir
Information
and Communication
Technology Research Center,
No 5, Saeedi Alley, Colledge
Intersection, Enghelab Street,
Tehran 1599616313, Iran

Abstract

The immense amount of data generated on a daily basis by various devices and systems necessitates a change in data analysis methods. As an important part of analytics, data mining methods require a paradigm shift to solve problems because the old methods cannot manage massive data. Association rule mining is a data mining algorithm used to solve various domain problems. Because of the immense volume of data, one-node solutions are no longer useful, and it is necessary to solve problems by using a distributed and shared-nothing architecture such as Map-Reduce. However, when association rule mining is transferred to these architectures, new problems appear. The main problems are lack of data locality and iteration support and process skewness. In this paper, a method is proposed that solves these problems. Kavosh converts data into a unified format that helps nodes perform their tasks independently without the need to exchange data with other nodes. In addition, the proposed method compresses input data to facilitate data management. Another advantage is the lack of process skewness because it is possible to allocate a predefined amount of data to each node. Kavosh omits iterations required for finding frequent itemsets by changing the Map-Reduce architecture. The proposed method is implemented using Hadoop, and the results are compared with open-source products in terms of three aspects: execution time, load balancing and data compression. The results show that Kavosh outperforms other methods in these aspects.

Keywords: Big data, Data mining, Map-Reduce, Association rules

Introduction

With the growth of information, traditional analysis methods must be modified because they cannot handle immense amounts of data. Data mining algorithms are analytics that require a paradigm shift for algorithm execution and changes for deployment over nodes. Data mining algorithms must be modified so they can be executed over scalable and distributable environments. However, modifying data mining algorithms for distributed architecture is not easy. One problem with distributed architecture is data locality. This occurs when the required data for processing do not exist on the processor node.

One of the most prominent methods used to solve big data problems over shared-nothing architecture is Map-Reduce. Map-Reduce [1] is used in open-source solutions such as Hadoop and Spark. There are many products in addition to Hadoop and Spark that can be used to solve data mining problems. However, pure Map-Reduce suffers from the data locality problem and does not support iteration. Because nearly all data

mining algorithms are iterative in nature, it is necessary to solve the iteration problem in Map-Reduce to solve data mining problems by using Map-Reduce-based methods.

Association rule mining is a data mining algorithm that requires iteration to find frequent itemsets. Methods such as Apriori [2] and FP-Growth [3] solve association rule mining problems. There are two main parameters for association rule mining: confidence and support. A rule in an association rule is defined as $x \rightarrow y$ (if x, then y), where x and y are itemsets. The support of a rule is defined as the frequency of an itemset in a database. The confidence of a rule is defined as

$$Confidence(x \rightarrow y) = \frac{\text{Support}(x \cap y)}{\text{Support}(x)}$$

In this paper, Kavosh, which is a method for association rule mining, is proposed. This method is designed for a shared-nothing architecture and can properly solve the association rule mining problem in Map-Reduce.

This method solves the data locality problem completely. Thus, the nodes do not require data from other nodes. By changing and unifying the data format, data compression and data load balancing are supported. In addition, iteration is omitted in the proposed method; therefore, it is well-suited for the Map-Reduce architecture.

The remainder of this paper is structured as follows: In "Related works" section, related works are discussed. In the third section, Kavosh is presented. The fourth section describes an evaluation, and the final section presents the conclusions.

Related works

Map-Reduce is used to solve many problems over a shared-nothing architecture. This method is also used to solve association rule mining. There are four main problems with association rule mining on a shared-nothing architecture, namely, lack of support for the following:

- Data locality,
- Iteration,
- Load balancing, and
- Data compression.

As mentioned above, the data locality problem is a lack of required data on the processor node, which creates data dependency among nodes. This problem causes network congestion and increases the algorithm execution time. Another problem is lack of iteration support. When intermediate results are created by the Reducers, they must be fed as input to the Mappers for another iteration. This problem also increases the execution time because intermediate data must be written on the disk by the Reducers and read by the Mappers to initiate another iteration. The third problem is the load balancing problem. This occurs when the processes are not allocated to the Mappers fairly. This problem reduces the algorithm speed because all nodes must wait for a busy node after job completion. The last problem is lack of support for data compression. Because of the large amount of data, data volume reduction is necessary for more rapid iterations and intermediate result storage.

In some methods, traditional Apriori methods are used in Map-Reduce. Transactions are allocated to Mappers and frequent k-itemsets are extracted from each Mapper before the results are shuffled through combiners and the final k-itemsets are extracted according to support and confidence thresholds. In Oruganti et al. [4], Kovacs et al. [5], Li et al. [6], Mappers and Reducers are used, but in [7, 8], in addition to Mappers and Reducers, Combiners are used for better shuffling and to address performance issues.

In Lin et al [9], three methods are proposed: Single Pass Counting (SPC), Fixed Passes Combined-counting (FPC) and Dynamic Passes Combined-counting (DPC). In Apriori, each iteration must generally wait until the results of all the Reducers from the previous iteration have been generated.

The FiDoop [10] method uses three Map-Reduce phases to generate frequent itemsets. FiDoop claims to support automatic parallelization, load balancing, data distribution, and fault tolerance. ScaDiBino [11] extracts rules with the maximum length and one target field. This method omits iterations. In Yu et al. [12], the Distributed Parallel Apriori (DPA) algorithm is proposed and metadata are stored in the form of Transaction Identifiers (TIDs). In this method, a single scan of the database is required and a balanced workload among nodes is created.

Various proposed methods use FP-Growth on a shared-nothing architecture. In Li et al. [13], parallel FP-Growth is proposed for independently executing a group of tasks on a node. In Bechini et al. [14], MRAC and MRAC+ are proposed, which are Map-Reduce-based and FP-Growth-based methods, respectively, and FP-Growth was modified to overcome performance issues. In Yang et al. [15], a Hadoop-based method is proposed that uses a distributed DH-TRIE frequent pattern algorithm and tries to solve FP-Growth problems for big data. In Tlili et al. [16], a partitioning method is used to achieve load balancing problems for association rule mining. PARMA [17] creates multiple small random samples of the input data and runs a mining algorithm on them. Because it is implemented on small samples, the algorithm can be run in parallel and independently. The results are aggregated to produce the final results. In Yu and Zhou [18], Tidset-based Parallel FP-tree (TPFP-tree) and Balanced Tidset-based Parallel FP-tree (BTP-tree) are proposed. In the proposed method, a transaction identification set (Tidset) is used to provide direct access to transactions instead of a full database scan. In Moens et al. [19], two methods are proposed: Dist-Eclat and BigFIM. Dist-Eclat has three steps: finding the frequent items, generating frequent itemsets of size k and sub-tree mining. BigFIM also has three steps: generating frequent itemsets of size k, finding potential extensions and sub-tree mining. The Sequence-Growth algorithm is designed according to the concepts of the lexicographical sequence tree and the lazy mining pruning strategy and is implemented in the MapReduce framework for a distributed execution [20]. In Liang et al. [21], an algorithm for lexicographic frequent itemset generation is proposed, which claims to find the maximum information from a database regarding frequent itemsets and their respective frequencies in a single database scan.

Methods

The proposed method uses the Map-Reduce architectural structure for association rule mining. The first step in the proposed method is to convert the input data items to a Kavosh format and the second step is to extract rules with variable lengths. In this paper,

the number of fields after the "if" conditions are applied is defined as the length of the rule. In the proposed method, data are converted into Kavosh format, which helps nodes execute their processes independently. This format is a unified format and other input data formats are converted to this format. The proposed unified format helps create <key, value> pairs for use in the Map-Reduce method.

• Converting input data items to Kavosh format

In this section, the Kavosh format is defined

$\Theta = \{\theta_1, \theta_2, \ldots, \theta_n, \dagger\}$.

The input data table is denoted as Θ, the columns of the table as θ and the table key as †

$\theta_k = \{\mu_1, \mu_2, \ldots, \mu_m\}$.

The distinct values of each column are denoted as μ. In this paper, it is assumed that all θ values are discrete and that a continuous θ must be converted to discrete values.

Based on the above definitions, the Kavosh format is defined as follows:

$$\digamma = \{\theta_1 \rightarrow \mu_1, \theta_2 \rightarrow \mu_2, \ldots, \theta_n \rightarrow \mu_m\}$$

where $\digamma$ is the Kavosh table and each $\theta_p \rightarrow \mu_q$ is equal to zero or one. Each input column value is converted to a Boolean column if there are more than two values for the column.

The input data (**Θ**) are divided into equal segments. For fragmentation, the function ψ is used, which is defined as follows:

$$\Psi\ (\digamma, \dagger_{start}, \dagger_{finish}, \eta_k)$$

where η_k is the Mapper node to which the data $\digamma$ are allocated from key $\dagger_{start}$ to key $\dagger_{finish}$. From the previous conversion steps, a Boolean matrix is created on each Mapper. Each field $\digamma$ can be selected as a target field. If a binary position is assigned to each field $\digamma$ (except target fields), each row can be converted into a decimal number. Therefore, a <Key, Value> pair is created in which the Key is the decimal number of each row (Rule-Key) and the Value is the decimal value of the target field. In the next step, the Kavosh triple is created, which is defined as follows:

<Rule-Key, Rule count, Target field count with the specified value>

According to the generated <Rule-Key, Target fields>, the count of each *Rule-Key* and target field with the specified value is calculated and the Kavosh triple set is created. To convert Mapper data into the Kavosh triple and Rule-Key aggregation, the function ☨ is used.

$$☨(\eta_k, \rho_k)$$

where ρ_k is the result of computation of the input data. After calculating ρ_k for each Mapper, all ρ_k are reduced to $\rho_{Reducer}$ by the function £.

$$£(\rho_1, \rho_2, \ldots, \rho_p, \rho_{Reducer}, ɜ)$$

where ɜ is the Reducer node and p is the number of Mappers.

• Rule extraction

In this section, a *Map-Key* is added to the *Rule-Key* (ϻ). The *Map-Key* is used for rule generation, and its length in binary format is equal to the *Rule-Key*. Here, $\wp_{\text{Reducer}}$ is sent to the rule generation nodes. Each rule generation node has one or more *Map-Key*s. The function ϖ allocates $\wp_{\text{Reducer}}$ to each rule generation node.

where ι is the *Map-Key,* q is the number of *Map-Key*s allocated to a rule generation node, and $\underline{Ъ}$ is the rule generation node.

$$\varpi\left(\wp_{\text{Reducer}}, \iota_1, \iota_2, \ldots, \iota_q, \underline{Ъ}_r\right)$$

Each *Map-Key* is concatenated to all *Rule-Key*s. If the length of the *Rule-Key* in binary format is equal to L, then there are $2^L - 1$ *Map-Key*s. They start from zero to $2^L - 1$ and are concatenated with the *Rule-Key*s to create various condition combinations of fields Ғ. The concatenation (Υ) of the *Map-Key* with the result of the "logical and" *(ϰ)* of the *Map-Key* and the *Rule-Key* is called *MHB_Key*, and τ is the *MHB_Key.*

$$\tau = \Upsilon\left(\iota_i, \varkappa\left(ϻ, \iota_i\right)\right)$$

There are two important factors for rule extraction in association rule mining: confidence and support. In this paper, Д (τ) is defined as the frequency of a specified *MHB_Key* (τ) and ζ (τ, σ, ρ) is defined as the frequency of a specific decimal value (ρ) that is equal to the target field (σ) for a specified *MHB_Key* (τ). In addition, ς is the total number of generated Kavosh triple sets. Thus, the support (ƨ) is calculated using the following formula:

$$ƨ(\tau) = \frac{Д(\tau)}{ς}$$

Moreover, the confidence (C) is calculated using the following formula:

$$C(\tau, \sigma, \rho) = \frac{\zeta(\tau, \sigma, \rho)}{Д(\tau)}$$

After ƨ and C have been calculated, they are compared with the defined thresholds for the association rules and the final results are produced. The rules are extracted using Д.

$$Д\left(ƨ(\tau), C(\tau, \sigma), \alpha, \beta, \underline{Ъ}_r\right)$$

where α is the support threshold and β is the confidence threshold. Each rule generation node creates its final result separately because the results do not have common rules with other nodes. Figure 1 shows the Kavosh architecture. It consists of three layers: the Mapper layer, the Reducer layer and the rule generation layer.

For clearer illustration, pseudocodes are provided. Table 1 lists the functions used in the code.

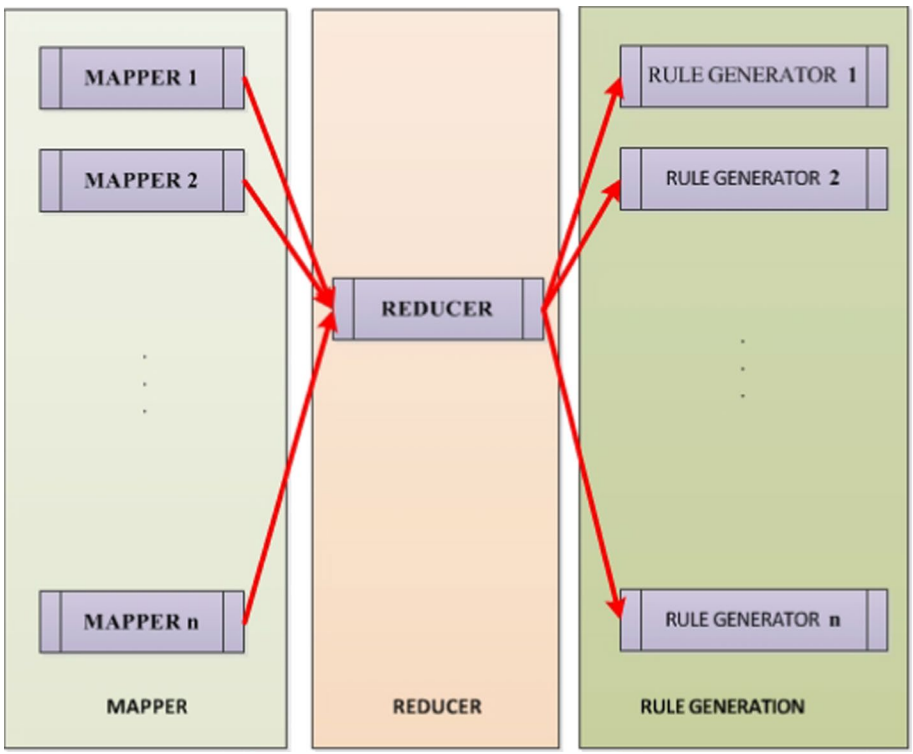

Fig. 1 Kavosh architecture

Table 1 Pseudocode functions

Function	Description
NOC(TableName)	*Returns the number of columns*
NOR(TableName)	*Returns the number of rows*
NODV(TableName, ColumnName/ColumnID)	*Returns the number of distinct values in the specified column*
NaOC(TableName, k)	*Returns the name of the* kth *column in the specified table*
DVC(TableName,ColumnName/ColumnID, k)	*Returns the* kth *distinct value of the specified column*
CT(ColumnsArray[],TableName)	*Creates a table with an input column array with a specified table name*
RVT(TableName, ColumnName/ColumnID,RowID)	*Returns the value of the specified table name, column name/ column ID, and row ID*
EXEC (Query, ResultTable)	*Executes a query, creates a results table and puts the results into the results table* *The results table can be created in RAM or HDD, according to the node hardware specification.*
&	*& is a logical operator (And)*
I2B(i)	*Converts the integer value i to a binary value*

The following code shows the binominal conversion.

```
Convert_To_Binominal()
{
        For i=1 to NOC(InputTableName)
          For j=1 to NODV(InputTableName, i)
                ColumnsArray[]=ColumnsArray[]+DVC(InputTableName,i, j);
         Next j
        Next i
        CT(ColumnsArray[],"Kavosh_Convert");
        For i=1 to NOR(InputTableName)
         Kavosh.InsertNewRow();
         For j=1 to NOC(InputTableName)
            For k=1 to NODV(InputTableName, j)
              If (InputTableName (i,j)==DVC(InputTableName,j, k))
                    RVT(Kavosh_Convert,i,j)=1    ;
              Else
                    RVT(Kavosh_Convert,i,j)=0    ;
            Next k;
         Next j;
        Next i;
}
```

The following pseudocode shows the Mapper's functionality:

```
Mapper()
{
        CT("[Rule_Key],[TargetFieldCount],[ TargetFieldSum]","Kavosh_Mapper");
        For i=1 to NOR(Kavosh_Convert)
          For j=1 to NOC(Kavosh_Convert)
              If (NaOC(TableName, j)!=TargetField)
                    MHB_Key=Concatenate(Key, RVT(Kavosh,i,j))    ;
          Next j;
        Insert into Kavosh_Mapper ([Rule_Key],[TargetFieldCount],[ TargetFieldSum]) values (Key, 1, RVT(Kavosh,i,
        TargetField));
        Next i;
        EXEC (Select [Rule_Key],Sum([TargetFieldCount]),Sum([ TargetFieldSum]) From Kavosh_Mapper group by
        [Rule_Key],Kavosh_Rules);

        Output <k'1,v'1,v'2> → <Kavosh_Mapper.[Rule_Key], Kavosh_Mapper.[TargetFieldCount],
        Kavosh_Mapper[TargetFieldSum]
}
```

In the reduce phase, the EXEC() function is repeated to obtain the final *intermediate results* layer. The following pseudocode shows the Reducer phase:

```
Reducer()
{
        Input <k1,v1,v2> → <Kavosh_Mapper.[Rule_Key], Kavosh_Mapper.[TargetFieldCount], Kavosh_Mapper.[
        TargetFieldSum]>

        EXEC (Select [Rule_Key],Sum([TargetFieldCount]),Sum([ TargetFieldSum]) From Kavosh_Rules group by
        [Rule_Key], Kavosh_Distinct_Rules);

        Output <k'1,v'1,v'2> → < Kavosh_Distinct_Rules.[Rule_Key], Kavosh_Distinct_Rules.[TargetFieldCount],
        Kavosh_Distinct_Rules.[ TargetFieldSum]>

}
```

Table 2 Sample input information

City	Sex	Income	New service	Mapper
Tehran	Male	High	Yes	1
Tehran	Female	Low	Yes	
Yazd	Male	High	No	
Yazd	Male	High	Yes	2
Tehran	Female	Low	No	
Yazd	Male	High	No	

Table 3 Binominal format

Tehran	Sex	High	New service	Mapper
1	1	1	1	1
1	0	0	1	
0	1	1	0	
0	1	1	1	2
1	0	0	0	
0	1	1	0	

After the extraction of all the rules with a maximum length in the *intermediate results* layer (Fig. 4), a mapping field is added to these rules. Each rule is converted to $2^n - 1$ rules, where n is the number of input data item fields. The new rule value is equal to the concatenation of the mapping field and the result of the logical *And* between the mapping field and the rule value.

```
Map_Field_Adder()
{
Mapper_Count=2^Input_Field_Count -1
CT("[MHB_Key],[TargetFieldCount],[ TargetFieldSum]","Kavosh_RuleGenerator");
For (i=1  to Mapper_Count)
            MHB_Key=Concatination(I2B(i), (Concatination(I2B(i) & RVT(Kavosh_Distinct_Rules, i,"Key") ));
             Kavosh_RuleGenerator.Insert(MHB_Key, RVT(Kavosh_Distinct_Rules,i, "TargetFieldCount") ,
RVT(Kavosh_Distinct_Rules, i,"TargetFieldSum"));
  Next i
}

Rule_Sum_And_Count_Calculator()
{
        EXEC (Select MHB_Key,Sum([TargetFieldCount]),Sum([ TargetFieldSum]) From Kavosh_RuleGenerator group
        by MHB_Key,Kavosh_RG1);
}

Confidence_And_Support_Calculator()
{
TotalCount=(Sum(TargertFieldCount) from Kavosh_Distinct_Rules) * (2^Input_Field_Count -1);
EXEC (Select MHB_Key,[TargetFieldCount]/TotalCount as Support,[TargetFieldSum]/TargetFieldCount as Confidence
From Kavosh _RG1 group by [MHB_Key],Kavosh_RG2);
}

Apply_Filter(Min_Confidence, Min_Support)
{
   EXEC (Select MHB_Key,Confidence,Support from Kavosh_RG2 where Confidence>Min_Confidence and
Support>Min_Support, Kavosh_Final_Result);
}
```

Suppose the input data are as described in Table 2, where *New Service* is the target field and the data are divided between two Mappers.

Table 4 Mapper results

Decimal rule	Target field	Rule count	Rule sum	Mapper
7	1	1	1	1
4	1	1	1	
3	0	1	0	
3	0	1	0	2
4	1	1	1	
3	0	1	0	

Table 5 Reducer results

Rule	Rule count	Rule sum
7	1	1
4	2	2
3	3	0

Table 6 Mapping field addition stage

Mapping field	Rule	MHB_Key (Binary)	MHB_Key (Decimal)	Rule count	Rule sum
111	111	111111	63	1	1
111	100	111100	60	2	2
111	011	111011	59	3	0
110	111	110110	54	1	1
110	100	110100	52	2	2
110	011	110010	50	3	0
101	111	101101	45	1	1
101	100	101100	44	2	2
101	011	101001	41	3	0
100	111	100100	36	1	1
100	100	100100	36	2	2
100	011	100000	32	3	0
011	111	011011	27	1	1
011	100	011000	24	2	2
011	011	011011	27	3	0
010	111	010010	18	1	1
010	100	010000	16	2	2
010	011	010010	18	3	0
001	111	001001	9	1	1
001	100	001000	8	2	2
001	011	001001	9	3	0

Table 3 shows the data after conversion to a binominal format. Table 4 shows the Mapper results. The rules are now combined based on their keys in the Reducer layer. As shown in Table 5, the rules are combined in this phase, and their rule count and rule summation values are recalculated. Then, the mapping fields are added to the

Table 7 Rule sum and count recalculation

MHB_Key (Binary)	MHB_Key (Decimal)	Rule count	Rule sum
111111	63	1	1
111100	60	2	2
111011	59	3	0
110110	54	1	1
110100	52	2	2
110010	50	3	0
101101	45	1	1
101100	44	2	2
101001	41	3	0
100100	36	3	3
100000	32	3	0
011011	27	4	1
011000	24	2	2
010010	18	4	1
010000	16	2	2
001001	9	4	1
001000	8	2	2

Table 8 Support and confidence results

Rule	Support (%)	Confidence (one) (%)
63	2	100
60	5	100
59	7	0
54	2	100
52	5	100
50	7	0
45	2	100
44	5	100
41	7	0
36	7	100
32	7	0
27	10	20
24	5	100
18	10	20
16	5	100
9	10	20
8	5	100

rules. The mapping fields are sorted to prevent future shuffling among the Mappers. As the input item, the field count is equal to three, and seven mapping fields $(2^3 - 1)$ must be added to the rules. Table 6 shows this stage.

The rule sum and count are recalculated, and the same key rules are combined, as depicted in Table 7. In the next stage, *confidence* and *support* are calculated. Table 8 shows the results of this stage. As mentioned earlier, confidence values must be calculated for two values: zero and one. Finally, the Reduce stage of the rule generation

Table 9 Reducer results

Rule	Support (%)	Confidence (one) (%)
59	7	0
50	7	0
41	7	0
36	7	100
32	7	0

layer filters rules according to thresholds for *support* and *confidence*. If *support* is greater than 5% or *confidence* is greater than 95%, the Reducer results are as shown in Table 9. As shown in Table 9, some rows have *confidence* equal to 0%. These rows have *confidence* equal to 100% for a zero value. In other words, for rule filtering with *confidence* greater than 95%, it is necessary to consider *confidence* lower than 5% (100 − 95%). Kavosh can be summarized as follows:

- The input data are converted to a binominal format.
- The converted data are distributed among the Mappers.
- The results of the Mappers are sent to the Reducer layer.
- Mapping fields are added to the results of the *Reducer* layer, and MHB_Keys are generated.
- The Reducer results are sent to the *Rule generation* layer.
- *Confidence* and *support* parameters are applied to the extracted rules to create the final results.

Evaluation

The proposed method is evaluated on the TPC-DS dataset and a real-world dataset. "The TPC Benchmark DS (TPC-DS) is a decision support benchmark that models several generally applicable aspects of a decision support system, including queries and data maintenance. The benchmark provides a representative evaluation of performance as a general purpose decision support system. A benchmark result measures query response time in single user mode, query throughput in multi user mode and data maintenance performance for a given hardware, operating system, and data processing system configuration under a controlled, complex, multi-user decision support workload. The purpose of TPC benchmarks is to provide relevant, objective performance data to industry users. TPC-DS Version 2 enables emerging technologies, such as Big Data systems, to execute the benchmark" (http://www.tpc.org/tpcds).

The proposed method can use DBMS on each node independently. PostgreSQL is used as DBMS on each node. PostgreSQL is a powerful, open-source object-relational database system that uses and extends the SQL language, combined with many features that safely store and scale the most complicated data workloads (https://www.postgresql.org/about/).

To achieve higher performance, an In-Memory database is used. Redis is an open-source (BSD licensed), in-memory data structure store, which is used as a database, cache and message broker (https://redis.io/).

TPC-DS

To implement the Kavosh method, Hadoop 2.7.3 is used. Ubuntu 16.04 is installed on each node. Parts of Hadoop were modified for Kavosh implementation. As described above, there are three layers of nodes in Kavosh: The first layer consists of the Mapper nodes, the second layer consists of the Reducer node, and the third layer is the rule generation layer. The DataNodes of Hadoop are used for these three layers. A MetaNode is added to the Hadoop NameNode to maintain the type information of each group of nodes. Because of the Kavosh data format and the independence of each node, it is possible to use a database management system for each data node. Therefore, PostgreSQL 9.6.1 is used for the Mapper nodes. A Kavosh converter exists on the Mapper nodes to convert the input data format into the Kavosh format. To achieve faster computation speed, an in-memory database is used for the Reducer node and the rule generation nodes. Redis 3.2.5 is used as an in-memory database on these nodes with AOF disk persistence for the Reducer and disk persistence is disabled for the rule generation nodes. Table 10 shows DBMS for the various node types.

Figure 2 shows Kavosh architecture using Hadoop.

For data generation, TPC-DS_Tools_v2.8.0 was used. Four tables (Customer_demographics, Store_sales, Web_sales, and Item) were used for association rule mining. Figures 3 and 4 show the ER-Diagrams.

A database view is created over four mentioned tables. This view contains customer information and items that each customer bought. Figures 5, 6, 7 and 8 show the table structures.

The Kavosh method extracts market basket analysis (MBA) rules for items on various sell channels. The frequent items with a maximum length of twenty are extracted. In other words, twenty related items are extracted. The scale factor is 100 TB and SF = 100,000. Table 11 shows the number of rows in each data table.

Tables 12 and 13 show the node specifications for the Kavosh method. A total of 86 nodes are used for evaluation. Seventy nodes were used as the Mapper nodes, one node was used as the Reducer node and fifteen nodes were used as the rule generation nodes. The input data must be converted to the Kavosh format and it requires 6740 s for the Kavosh converter to convert the input data to the Kavosh format. Each Mapper node simultaneously converts approximately 1.5 TB of data.

Table 10 DBMS for the various node types

Node type	DBMS
Mapper	PostgreSQL
Reducer	Redis
Rule generation	Redis

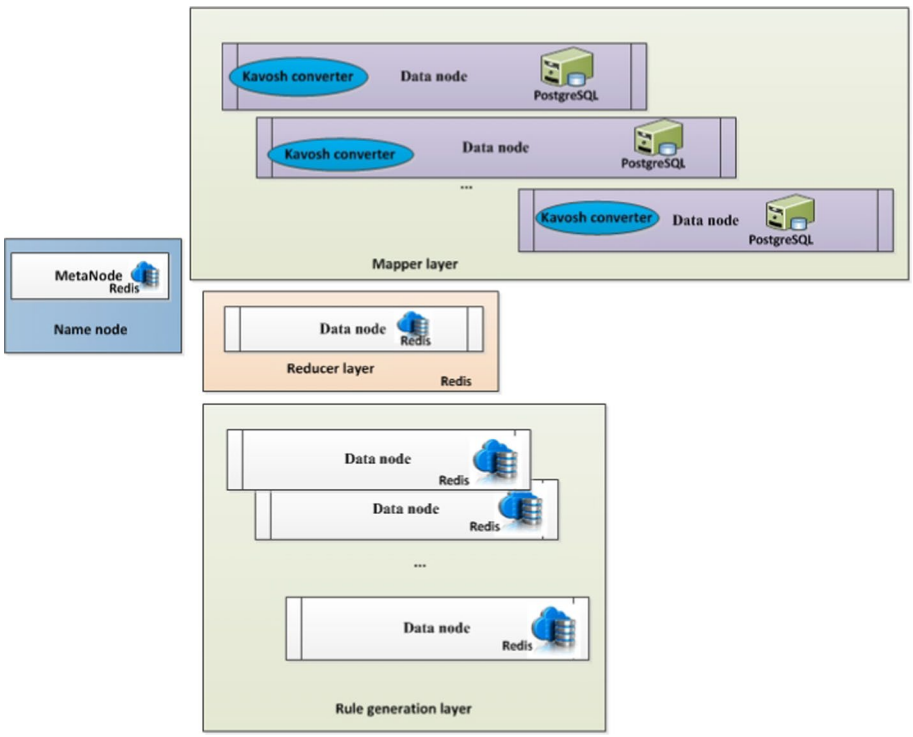

Fig. 2 Using Hadoop to implement Kavosh

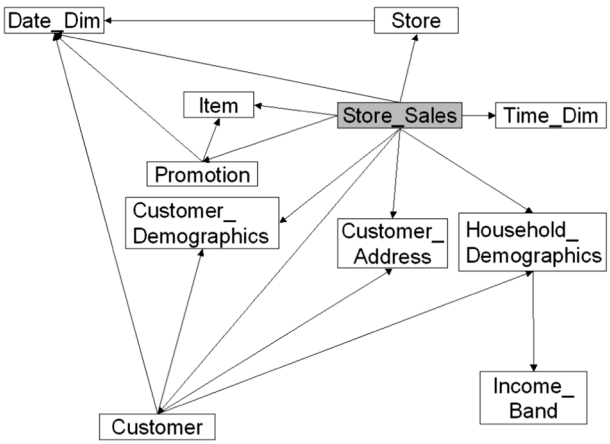

Fig. 3 Store sales diagram

The Kavosh method is compared with FiDoop [10] and Sequence-Growth [20]. Table 14 shows the node specifications for the FiDoop and Sequence-Growth methods.

The total CPU, RAM and HDD used for Kavosh and other methods are equal.

To evaluate the three methods, frequent items for MBA of web sales and store sales are extracted. The association rule parameters are shown in Table 15.

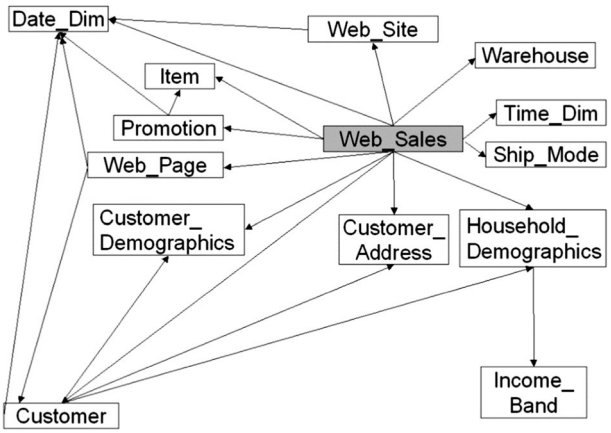

Fig. 4 Web sales diagram

Column	Datatype	NULLs	Primary Key	Foreign Key
ss_sold_date_sk	identifier			d_date_sk
ss_sold_time_sk	identifier			t_time_sk
ss_item_sk (1)	identifier	N	Y	i_item_sk
ss_customer_sk	identifier			c_customer_sk
ss_cdemo_sk	identifier			cd_demo_sk
ss_hdemo_sk	identifier			hd_demo_sk
ss_addr_sk	identifier			ca_address_sk
ss_store_sk	identifier			s_store_sk
ss_promo_sk	identifier			p_promo_sk

Fig. 5 Store_sales

Column	Datatype	NULLs	Primary Key	Foreign Key
ws_sold_date_sk	identifier			d_date_sk
ws_sold_time_sk	identifier			t_time_sk
ws_ship_date_sk	identifier			d_date_sk
ws_item_sk (1)	identifier	N	Y	i_item_sk
ws_bill_customer_sk	identifier			c_customer_sk
ws_bill_cdemo_sk	identifier			cd_demo_sk
ws_bill_hdemo_sk	identifier			hd_demo_sk
ws_bill_addr_sk	identifier			ca_address_sk
ws_ship_customer_sk	identifier			c_customer_sk
ws_ship_cdemo_sk	identifier			cd_demo_sk
ws_ship_hdemo_sk	identifier			hd_demo_sk
ws_ship_addr_sk	identifier			ca_address_sk
ws_web_page_sk	identifier			wp_web_page_sk
ws_web_site_sk	identifier			web_site_sk
ws_ship_mode_sk	identifier			sm_ship_mode_sk
ws_warehouse_sk	identifier			w_warehouse_sk
ws_promo_sk	identifier			p_promo_sk

Fig. 6 web_sales

They are compared with one another in three dimensions: execution time, data compression and load balancing.

Execution time

The method proposed by FiDoop requires three Map-Reduce cycles to generate the final results. The execution time for each cycle is almost the same, and in each cycle, a full database scan is required. Sequence-Growth requires twenty Map-Reduce cycles

Column	Datatype	NULLs	Primary Key	Foreign Key
i_item_sk	identifier	N	Y	
i_item_id (B)	char(16)	N		
i_rec_start_date	date			
i_rec_end_date	date			
i_item_desc	varchar(200)			
i_current_price	decimal(7,2)			
i_wholesale_cost	decimal(7,2)			
i_brand_id	integer			
i_brand	char(50)			
i_class_id	integer			
i_class	char(50)			
i_category_id	integer			
i_category	char(50)			
i_manufact_id	integer			
i_manufact	char(50)			
i_size	char(20)			
i_formulation	char(20)			
i_color	char(20)			
i_units	char(10)			
i_container	char(10)			
i_manager_id	integer			
i_product_name	char(50)			

Fig. 7 Items

Column	Datatype	NULLs	Primary Key	Foreign Key
cd_demo_sk	identifier	N	Y	
cd_gender	char(1)			
cd_marital_status	char(1)			
cd_education_status	char(20)			
cd_purchase_estimate	integer			
cd_credit_rating	char(10)			
cd_dep_count	integer			
cd_dep_employed_count	integer			
cd_dep_college_count	integer			

Fig. 8 Customer_Demographics

Table 11 Number of rows in each data table

Table name	Number of rows
Items	502,000
web_sales	71,999,670,164
Store_sales	287,997,818,084
Customer_Demographics	1,920,800

Table 12 Mapper node specifications

CPU	**Intel Core i7-3770 Quad-Core Processor 3.4 GHz**
HDD	2 TB
RAM	64 GB

Table 13 Reducer and rule generation node specifications

CPU	**Intel Core i7-3770 Quad-Core Processor 3.4 GHz**
HDD	500 GB
RAM	512 GB

Table 14 Nodes for FiDoop and Sequence-Growth

CPU	Intel Core i7-3770 Quad-Core Processor 3.4 GHz
HDD	1.5 TB
RAM	150 GB

Table 15 Association rule parameters

Parameter name	Parameter value (%)
Confidence	90
Support	0.05

Table 16 Execution details

Method name	Map (s)	Reduce (s)	Rule generation (s)
FiDoop	1265	1194	0
Sequence-Growth	3500	2890	0
Kavosh	288	265	1159

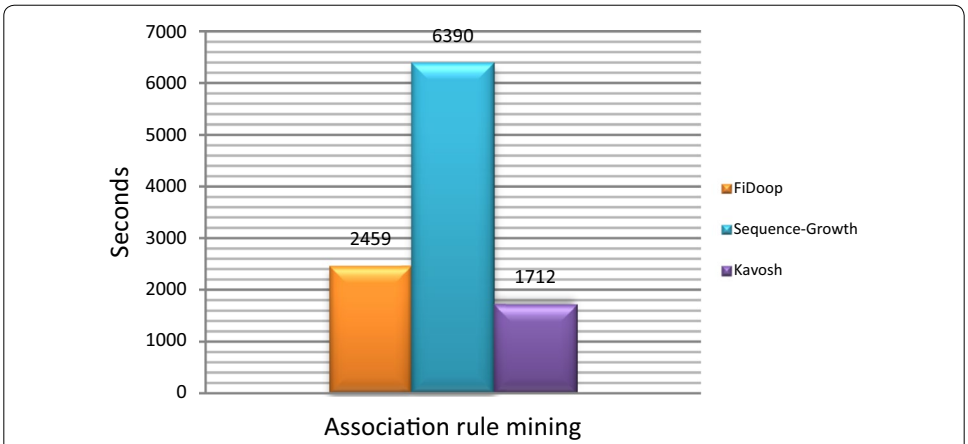

Fig. 9 Association rule mining execution time

because it generates each frequent item in a Map-Reduce cycle. However, the execution time decreases each cycle in comparison with the previous cycle because the number of frequent items decreases. Nevertheless, Kavosh requires a Map-Reduce cycle on compressed data and a rule generation phase. According to the data format, all nodes can complete their processes independently. In addition, Kavosh uses an in-memory database in the Reducer and rule generation nodes, which causes a reduction in I/O time. According to the above description, Table 16 shows the execution time for each method.

Figure 9 shows the total execution time for each method.

Table 17 Memory usage

Method name	Used table data volume (TB)
FiDoop	39.9
Sequence-Growth	70
Kavosh	70

Table 18 Hadoop configuration

Method	Balance_Factor (s)
FiDoop	800
Sequence-Growth	3230
Kavosh	15

Compression

In this part of the evaluation, the Kavosh data volume is compared with those of FiDoop and Sequence-Growth. In FiDoop and Sequence-Growth, no compression is performed on the input data, whereas the data are compressed in Kavosh due to the conversion of the input data to the Kavosh data format. The compression rates are different: if fields have fewer discrete values, the compression rate reaches ninety percent, but Kavosh compresses data by an average of approximately 57%. The data compression in the Kavosh data format is due to the conversion of string and number values to the bit data type. In the Kavosh method, no data replication or collocation is required for performance issues. In addition, data compression helps Kavosh retrieve more data in the data retrieval phase. Table 17 shows the memory usage for different methods.

Load balancing

In this section, the load balancing of various methods is investigated. To calculate the load balance, the length of time during which a processor has a CPU usage of over eighty percent while other CPUs have CPU usages of under thirty percent is calculated. In this paper, this measure is called Balance_Factor. Using this definition, the results in Table 18 were obtained.

Real traffic data of a mobile operator

In this section, the proposed method is evaluated using real traffic data of a mobile value-added service (VAS) provider. The evaluation data include more than 60,000,000 subscribers, together with their services. The subscribers used 3000 services and all the subscribers' activities were considered in the evaluation. The mobile operator needs accurate information about their customers' favourites to be able to provide appropriate VAS suggestions. Due to the high volume of information, existing association rule algorithms would take a long time to execute and, therefore, are not suitable. In this evaluation, high-dimensional data items were used. If all the combinations of rules were applied, $2^{3000}-1$ mapping fields would have to be added to each rule that is generated

Table 19 Input data format

MobileNumber	ServiceID
09121450111	1041
09121450111	58
09121450111	971
09121450111	119
09123895004	971
09191005069	113
…	…

in the *rule generation* layer. Such a large number of rules is neither necessary nor calculable. Based on additional information obtained about the services, a maximum of ten services is required for recommending VAS. Thus, rules with a maximum length of ten must be generated.

The input data format is as shown in Table 19. The first column is the subscriber mobile number and the second column is the service's ID.

We converted Table 18 to a binominal format. Finally, a <Key,Value> pair is created. The Key is the mobile number, and the value is a bit stream with a length of 2999 (Number of VAS services − 1 (Target Field)). Kavosh was deployed on the hardware nodes with the specifications detailed in "TPC-DS" section.

In the first step, the customer service information table is converted to the Kavosh format. In this phase, one service is considered as a target field. The relationships between this service and other services are extracted as rules. The customer service information table is distributed over 70 nodes. The conversion time is 1200s. Fifty servers are used for the Mapper, one server is used for the Reducer and sixty servers are used for the rule generation layer.

Table 20 presents the parameters that are used for the association rule algorithm.

Table 21 shows execution details for the three methods.

Table 20 Association rule parameters

Parameter name	Parameter value (%)
Confidence	85
Support	0.1

Table 21 Execution details

Method name	Map (s)	Reduce (s)	Rule generation (s)
FiDoop	206	199	0
Sequence-Growth	779	750	0
Kavosh	75	71	119

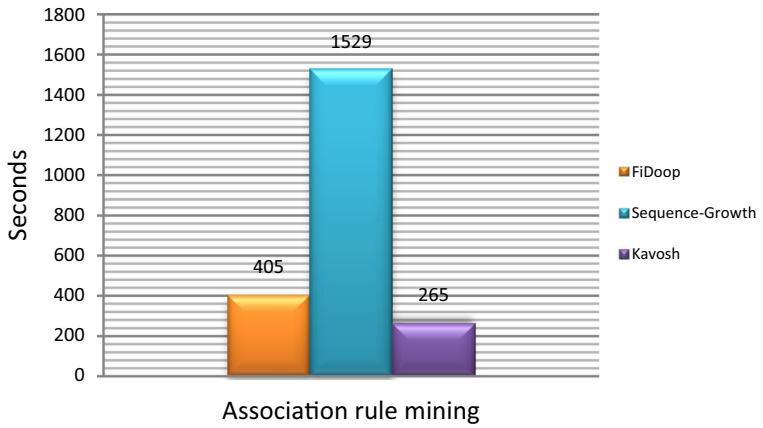

Fig. 10 Association rule mining execution time

Table 22 Memory usage

Method name	Table data volume used (TB)
FiDoop	19
Sequence-Growth	40
Kavosh	40

Table 23 Load balancing

Method	Balance_ Factor (s)
FiDoop	683
Sequence-Growth	2800
Kavosh	17

Figure 10 shows the total execution time for each method.

Compression

Table 22 shows the memory usage for the three methods.

Load balancing

Table 23 shows the values of Balance_Factor for the three methods

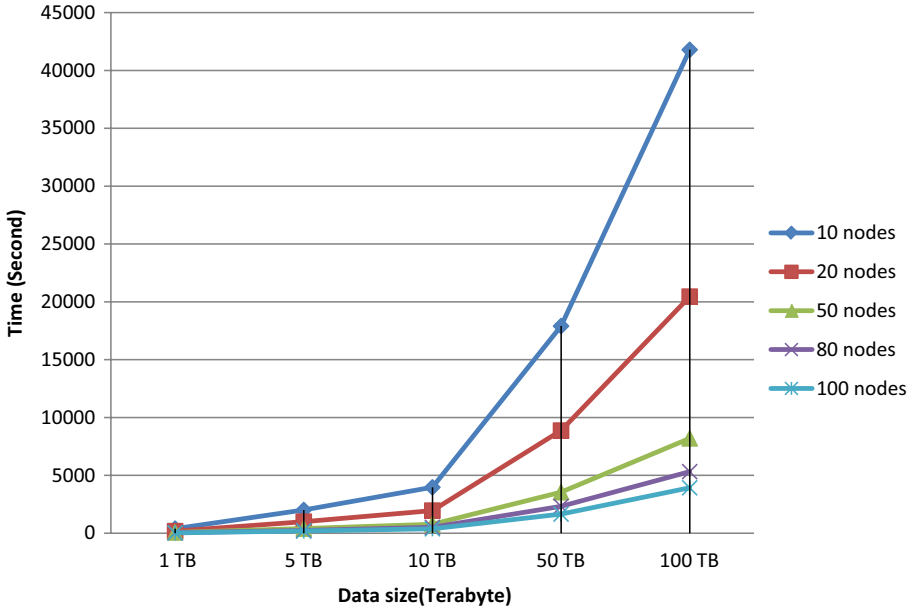

Fig. 11 Kavosh scalability

Scalability

In this section Kavosh scalability is evaluated. Different data sizes(TPC-DS) and various number of nodes are investigated and the results are depicted as Fig. 11. The results show that by adding more nodes and scale out data over nodes less time is required for data processing.

Conclusion

In this paper, the Kavosh method is proposed. Kavosh is a Map-Reduce-based association rule mining method that can manage an immense amount of data. This method converts input data into the Kavosh format, which is a unified and compressed format. Using the Kavosh data format, data can be distributed over nodes, and nodes do not require additional data from other nodes. Kavosh also compresses input data and facilitates data management. The main advantage of the proposed method is the omission of iterations for extracting rules. This is a substantial achievement because writing intermediate results to a disk and retrieving them for the next iteration is the most time-consuming portion of association rule mining. Another important achievement of the proposed method is the balanced process for each node. Because of the unified data format, each node is allocated the same number of rows in the homogenous nodes; thus, the process is equally divided among the nodes. By modifying Hadoop, the proposed method was implemented. Kavosh was compared with prominent association rule mining methods and achieved a faster execution time.

Kavosh is not suitable for high-dimensional data because creating all the input field combinations requires many nodes in the data generation layer. However, Kavosh is completely usable for normal data (not high-dimensional data) and can generate rules of a predefined length if other rule lengths are not required.

Data format unification can be applied to problems in other fields. This method can be used in data warehouse like Aras and Atrak methods [22, 23], graph processing [24], integrating multidimensional data sources [25] and specific problems like finding patient similarity [26]. For future works, this method can also be used for interactive query processing, online data mining and stream processing.

Authors' contributions
All authors contributed equally. All authors read and approved the final manuscript.

Acknowledgements
None.

Ethics approval and consent to participate and Consent for publication
In submitting an article to any of the journals published by SpringerOpen I certify that: (1) I am authorized by my co-authors to enter into these arrangements. (2) I warrant, on behalf of myself and my co-authors, that: (i) the article is original, has not been formally published in any other peer-reviewed journal, is not under consideration by any other journal and does not infringe any existing copyright or any other third party rights; (ii) I am/we are the sole author(s) of the article and have full authority to enter into this agreement and in granting rights to Springer are not in breach of any other obligation; (iii) the article contains nothing that is unlawful, libellous, or which would, if published, constitute a breach of contract or of confidence or of commitment given to secrecy; (iv)I/we have taken due care to ensure the integrity of the article. To my/our—and currently accepted scientific—knowledge all statements contained in it purporting to be facts are true and any formula or instruction contained in the article will not, if followed accurately, cause any injury, illness or damage to the user. (3) I, and all co-authors, agree that the article, if editorially accepted for publication, shall be licensed under the Creative Commons Attribution License 4.0. If the law requires that the article be published in the public domain, I/we will notify Springer at the time of submission, and in such cases the article shall be released under the Creative Commons 1.0 Public Domain Dedication waiver. For the avoidance of doubt it is stated that "Introduction" and "Related works" sections of this license agreement shall apply and prevail regardless of whether the article is published under Creative Commons Attribution License 4.0 or the Creative Commons 1.0 Public Domain Dedication waiver. (4) I, and all co-authors, agree that, if the article is editorially accepted for publication in Chemistry Central Journal, Chemical and Biological Technologies in Agriculture, Geochemical Transactions, Heritage Science, Journal of Cheminformatics, or Sustainable Chemical Processes, data included in the article shall be made available under the Creative Commons 1.0 Public Domain Dedication waiver, unless otherwise stated. For the avoidance of doubt it is stated that "Introduction", "Related works" and "Methods" sections of this license agreement shall apply and prevail.

Competing interests
The authors declare that they have no competing interests.

Availability of data and materials
http://www.tpc.org/TPC_Documents_Current_Versions/download_programs/tools-download-request.asp?bm_type=TPC-DS&bm_vers=2.7.0&mode=CURRENT-ONLY.

Funding
None.

Publisher's Note

Received: 22 March 2018 Accepted: 18 June 2018
Published online: 14 July 2018

References

1. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Commun ACM. 2008;51(1):107–13.
2. Zaki MJ. Parallel and distributed association mining: a survey. IEEE Concurr. 1999;7(4):14–25.
3. Pramudiono I, Kitsuregawa M. FP-tax: tree structure based generalized association rule mining. In: Proceedings of the 9th ACM SIGMOD workshop on research issues in data mining and knowledge discovery. New York: ACM; 2004. p. 60–3.
4. Oruganti S, Ding Q, Tabrizi N. Exploring Hadoop as a platform for distributed association rule mining. In: Future computing 2013 the fifth international conference on future computational technologies and applications; 2013. p. 62–7.
5. Kovacs F, Illés J. Frequent itemset mining on hadoop. In: 2013 IEEE 9th international conference on computational cybernetics (ICCC). New York: IEEE; 2013. p. 241–5.
6. Li N, Zeng L, He Q, Shi Z. Parallel implementation of apriori algorithm based on mapreduce. In: 2012 13th ACIS international conference on software engineering, artificial intelligence, networking and parallel and distributed computing (SNPD). New York: IEEE; 2012. p. 236–41.
7. Yang XY, Liu Z, Fu Y. MapReduce as a programming model for association rules algorithm on Hadoop. In: 2010 3rd international conference on information sciences and interaction sciences (ICIS). New York: IEEE; 2010. p. 99–102.

8. Li L, Zhang M. The strategy of mining association rule based on cloud computing. In: 2011 international conference on business computing and global informatization (BCGIN). New York: IEEE; 2011. p. 475–8.
9. Lin MY, Lee PY, Hsueh SC. Apriori-based frequent itemset mining algorithms on MapReduce. In: Proceedings of the 6th international conference on ubiquitous information management and communication. New York: ACM; 2012. p. 76.
10. Xun Y, Zhang J, Qin X. Fidoop: parallel mining of frequent itemsets using mapreduce. IEEE Trans Syst Man Cybern Syst. 2016;46(3):313–25.
11. Barkhordari M, Niamanesh M. ScadiBino: an effective MapReduce-based association rule mining method. In: Proceedings of the sixteenth international conference on electronic commerce. New York: ACM; 2014. p. 1.
12. Yu KM, Zhou J, Hong TP, Zhou JL. A load-balanced distributed parallel mining algorithm. Expert Syst Appl. 2010;37(3):2459–64.
13. Li H, Wang Y, Zhang D, Zhang M, Chang EY. Pfp: parallel fp-growth for query recommendation. In: Proceedings of the 2008 ACM conference on recommender systems. New York: ACM; 2008. p. 107–14.
14. Bechini A, Marcelloni F, Segatori A. A MapReduce solution for associative classification of big data. Inf Sci. 2016;332:33–55.
15. Yang L, Shi Z, Xu LD, Liang F, Kirsh I. DH-TRIE frequent pattern mining on Hadoop using JPA. In: 2011 IEEE international conference on granular computing (GrC). New York: IEEE; 2011. p. 875–8.
16. Tlili R, Slimani Y. A novel data partitioning approach for association rule mining on grids. Int J Grid Distributed Comput. 2012;5(4):1–20.
17. Riondato M, DeBrabant JA, Fonseca R, Upfal E. PARMA: a parallel randomized algorithm for approximate association rules mining in MapReduce. In: Proceedings of the 21st ACM international conference on Information and knowledge management. New York: ACM; 2012. p. 85-94.
18. Yu KM, Zhou J. Parallel TID-based frequent pattern mining algorithm on a PC Cluster and grid computing system. Expert Syst Appl. 2010;37(3):2486–94.
19. Moens S, Aksehirli E, Goethals B. Frequent itemset mining for big data. In: 2013 IEEE international conference on Big Data. New York: IEEE; 2013. p. 111–8.
20. Liang YH, Wu SY. Sequence-growth: A scalable and effective frequent itemset mining algorithm for big data based on MapReduce framework. In: 2015 IEEE international congress on Big Data (BigData Congress). New York: IEEE; 2015. p. 393–400.
21. Chaudhary S, Sharma A, Singh R, Kumar P. Lexicographic logical multi-hashing for frequent itemset mining. In: 2015 international conference on computing, communication and automation (ICCCA). New York: IEEE; 2015. p. 563–8.
22. Barkhordari M, Niamanesh M. Atrak: a MapReduce-based data warehouse for big data. J Supercomput. 2017;73:4596–610.
23. Barkhordari M, Niamanesh M. Aras: a method with uniform distributed dataset to solve data warehouse problems for big data. Int J Distributed Syst Technol (IJDST). 2017;8(2):47–60.
24. Barkhordari M, Niamanesh M. ScaDiGraph: a MapReduce-based method for solving graph problems. J Inform Sci Eng. 2017;33(1):143–58.
25. Barkhordari M, Niamanesh M. Arvand: a method to integrate multidimensional data sources into big data analytic structures. J Inf Sci Eng. 2018;34(2):505–18.
26. Barkhordari M, Niamanesh M. ScaDiPaSi: an effective scalable and distributable MapReduce-based method to find patient similarity on huge healthcare networks. Big Data Res. 2015;2(1):19–27.