International Journal of Distributed and Cloud Computing
7 (1) June 2019, 14-24
http://www.publishingindia.com/ijdcc

Cloud Optimized Eclat Growth (COEG) Using Fuzzy Logic

Priya Vaithiyanathan[1*] and Murugan S.[2]
[1]Assistant Professor, Dept. of Computer Science, Nehru Memorial College, Bharathidasan University, Tiruchirappalli, Tamil Nadu, India. Email: vpriyakarthik2014@gmail.com
[2]Associate Professor, Dept. of Computer Science, Nehru Memorial College, Bharathidasan University, Tiruchirappalli, Tamil Nadu, India. Email: murugan_nmc@hotmail.com
*Corresponding Author

Abstract: Eclat is one of the best data mining algorithms in practice. The basic Eclat algorithm has some limitations and an improved version of Eclat was introduced in the name of Eclat-Growth to overcome some of the limitations. Cloud Optimized Eclat Growth (COEG) method is proposed in this work in which a new Eclat based data mining approach is introduced that overcomes the disadvantages of existing methods to work well with real-time cloud environments. New concepts of localized cloud Eclat data processing by cloud offloading method introduced in this work. The localized cloud data processing procedure reduces the process of repeated global database scanning to improve processing speed and to reduce the memory consumption in a single machine. Fuzzy logic based multidimensional quantitative itemsets table is used to improve the accuracy. Since the proposed method uses localized cloud data processing, multidimensional liked lists are used which improves the item, transaction and pattern-based search capabilities. COEG is prepared with the intension to maximize the accuracy, precision, recall, intra-cluster density and to minimize the processing time and memory utilization.

Keywords: Accuracy, Cloud data processing, Eclat, Fuzzy logic, Optimization.

I. Introduction

Data Science is used for gathering knowledge from raw data. This emerging technology is used to facilitate dulcet experience to the information age society. Nowadays data are from everywhere and stored in data warehouses. The accessing of data is also spread all over the world. There the entire computational technology is moving towards the latest cloud technology. Many computational resources are maintained through this technology and users get the computational resources like hardware, software and data from the cloud platform as a service. Users tend to pay based on the usage instead of purchasing a bundle of required software along with undesirable software package. That is the conventional software are available as a bundle to the user in which the user has to pay for the software which he does not use even for a single time.

For Example, Microsoft provides Office as a package with frequently used applications like Word, Excel, Power Point along with rarely used Outlook. But if a user needs and purchases the software in a cloud platform, he can get them individually for a required period. Cloud's Platform as a Service (PaaS) and Software as a Service (SaaS) are much adorable by the user and a lurking market place for the vendors.

Many existing Data mining Procedures such as Apriori, FP-Tree and Eclat are having conventional mechanism to fetch and process the data. They collect the data and store in a single system for repeated database scanning to perform the mining process. The lack of disseminated processing method requires more computational resources like processing time, computational memory usage and power. The support count is also followed in a centralized manner but the real-world requirements are with dynamic nature.

Proposed COEG is designed with the cloud computation offloading adroitness, which reduces the compulsion of executing the entire data mining process in a single machine. Performing entire data mining process in a single machine requires more powerful system to operate. By offloading the data mining process in an aggregation of suitable machines, the data mining process can be performed more quickly even the algorithm is complicated. The overloading is annulled by the process distribution which is performed using Task Dismantling Strategy (TDS) of the proposed system.

Determining worthy machines for aggregation to perform datamining in a cloud environment is one of the vital as well as challenging process because cloud is an environment with heterogeneous nodes. This determination process is also treated well in ECOG with the help of Machine Index Table (MIT).

II. Related Works

Data mining is getting popular in recent years after the terminology 'Bigdata' is introduced. Due to the deep impact of data science, a number of algorithms are proposed to achieve the betterment of the data mining process. Apriori [1], Classification and Regression Tree (CART [2]), Expectation Maximization (EM) [3] [4], Naive Bayes Classifier [5], Equivalence Class Transformation (Eclat) [6], Eclat Optimized (ECLAT-Opt), Bi-Eclat, Eclat growth [7] and many more methods are used to perform data mining. From the above list, the most suitable data mining algorithms Eclat, Eclat-Opt, Bi-Eclat [8] and Eclat growth are much notable procedures for bigdata digital age. These procedures are selected for the comparison with the proposed methods to evaluate the performance of COEG.

A. Eclat

Data are stored in memory in two basic formats, they are horizontal and vertical format. In horizontal data format records are stored in Transaction ID (TID)-Item Set. It is also represented as TID: Itemset. In this representation TID is used as the unique identifier. Itemset refers a number of items involved in the transaction. In Vertical format data are stored as Item-TID set. It is also represented as Item: TID set.

Let T_1, $T_2 \ldots T_m$ are the transactions and I_1, $I_2 \ldots I_n$ are itemset of the database. Let $T_1 = \{I_1, I_2, I_3\}$, $T_2 = \{I_1, I_3, I_4\}$ and $T_3 = \{I_2, I_4, I_5\}$. The horizontal and vertical representations are illustrated in Fig. 1.

TID	Itemset
1	1,2,3
2	1,3,4
3	2,4,5

Horizantal Format

Items	TID Set
1	1,2
2	1,3
3	1,2
4	2,3

Vertical Format

Fig. 1: Data Representation Formats

These two types of data representations have their own merits and demerits. Eclat follows vertical data format in which Items are treated as unique fields and transactions are listed in the column field entries. Database scanning is required to get the degree of support and Eclat effectively reduces the database scanning time by selecting vertical data representation system. Lattice Theory equivalence classes and intersection are involved in Eclat.

The Eclat procedure is given below in simple steps:

Step 1. Initialize $i = 1$

Step 2. Do

Find all frequent i-itemsets by scanning the database

Find all candidate i +1-temsets from frequent -itemsets list

Increment i by value 1

Step 3. Repeat Step 2 until no candidate itemset is generated for i-1-itemsets

Eclat algorithm has join operation referred by the symbol ⋈, this process is similar to union operation (∪) with a limitation that two elements from the beginning of the involved lists should be equal to perform the join operation. For example let $T_1 = \{I_1, I_2, I_3\}$, $T_2 = \{I_1, I_2, I_4\}$ and $T_3 = \{I_2, I_3, I_4\}$. Then $T_1 \bowtie T_2$ is allowed since the first two elements of these two lists are I_1, I_2 whereas $T_1 \bowtie T_3$ is not permitted. If $T_{12} = T_1 \bowtie T_2$, then the result will be $T_{12} = \{I_1, I_2, I_3, I_4\}$. This ⋈ operation is used to compute candidate k + 1-itemsets. The searching process in Eclat is simplified by splitting the search space into multiple nonoverlapping subsets using equivalence classes. The itemsets can be classified in the same class only if they have the same prefix. The candidate itemset generation can be performed only in the same class. This property of equivalence classes improves the efficiency of Eclat procedure and it reduce the memory occupation.

B. Optimized Eclat (Eclat-Opt)

Eclat-opt is developed by Feng *et al.* in the year 2013. This procedure added the facility to existing Eclat to use double layer hash table and partition group of Itemsets. This modified procedure helps to snip the candidate 3-itemsets to reduce processing time of generating candidate itemsets by reducing the search space. It also increases the speed of intersection calculation process. Eclat-opt has the ability to itemset connections and intersections. Equivalence classes are assorted and separated based in the suffix to remove the connection judgements of itemsets. A lost threshold value is introduced to the transaction-id to improve the speed of intersection process.

C. Bi-Eclat

Bi-Eclat has three prime process, they are creating a tid-list in which each frequent item is arranged in descending order, generating candidate itemsets by constructing equivalence class lattice and performing data mining through pruning the candidate itemset.

Creating a tid-list process in performed after the horizontal to vertical conversion of input data. This process is performed on the fly that is the data are converted as they arrive. The tid-list is scanned repeatedly with the incremented value of support count of the item. The 1-frequet itemsets sorted in descending order after calculating support and comparison with minimum support.

The candidate itemsets generation process starts with a set of class atoms arranged in ascending order based on their supports. The set of atoms are used to generate an itemset by performing the union process of the subsets. The intersection process is applied on the tid-lists of atoms to calculate the support of the atom subsets.

Infrequent items are discarded from the assorted frequent itemsets. Similarly, the process of removing itemset redundancy is also performed in the frequent itemsets. These processes are

performed without much effort because the frequent itemsets are already arranged in ascending order based on their support.

Bi-Eclat pseudo code is given below:

Procedure Find_Frequent_1-itemsets:

For all atoms $A_i \in S$ do

For all transactions $T_j \in A_i$ do

|tid-list(A_i)| = |tid-list(A_i)| + 1

End for

For all $A_i \in S$ and |*tid-list*(A_i)| ≥ min_sup do

Sort A_i in descending order based on support

End for

$$S = \mathrm{S} \cup \{R\}; T_1 = T_1 \cup \{A_i\};$$

End for

This procedure reads vertical tid-list database as input and calculates 1-frequent itemsets arranged in descending order based on the support.

Procedure Candidate_Frequent_Itemsets:

For all atoms $A_i \in S$ do

$T_i = \phi$

For all atoms $A_j \in S$ with $sup(A_j) > sup(A_i)$ do

$R = A_i \cup A_j$;

tid-list (R)= tid-list(A_i) ∩ tid-list (A_j);

if(|tid-list (R)| ≥ min_sup

$s = S \cup \{R\}$; $T_i = T_i \cup \{R\}$;

End if

End for

While $T_i \neq \phi$ do store candidate_frequent_itemsets(T_i)

End for

This procedure is used to find frequent itemsets by reading atom set which is arranged in ascending order based on the support.

D. Eclat-Growth

Eclat-Growth algorithm is proposed by Zhiyong Ma *et al.* in 2016. Eclat-Growth is designed based on increased search strategy. This procedure uses increased two-dimensional pattern tree and the TID-sets in vertical data format table added to the pattern tree rows. The existing pattern tree itemsets are combined with newly added itemsets to generate frequent itemsets. Breadth-first-search approach is used to perform the combining process. The frequent 2-itemsets and candidate itemsets are generated in the beginning. Then higher order candidate and frequent itemsets until entire frequent itemset of the pattern table are combined with the newly added itemset. The breadth-first-search procedure is used to clip the candidate itemsets easily and redundant candidate itemsets are clipped. Eclat-growth adopts Boolean Array setting and retrieval by indexes of transactions) instead of intersecting the TID-sets to calculate the support degree of two itemsets. This alternative process is also used to reduce the computational complexity.

The Eclat-Growth algorithm is given below:

Let D = Database; MS = Min_Sup; VM = Vertical Matrix; PT = Two-dimensional Pattern Tree; FIs= Frequent Itemsets

VM = CreateVMfromDatabase(D)

Initialize PT with ∅ values

For I = 1 to length(VM) do

If(length(VM[i].TID_sets) ≥ MS then

AddItemSetToPatternTree(VM[i],PT,MS)

End for

FIs = GetAllFrequentItemsetsFromPatternTree(PT)

This procedure gets Database and Minimum Support as inputs and generates Frequent Itemsets. The increased two-dimensional pattern tree contains nodes and a layer pointer array. A node is initialized as TreeNode = {SET itemset, SET TID-set, PTR PointersToFathers, PTR PointersToChildren, PTR HorizontalPointer, BOOL IsValid}. A frequent itemset is stored in every TreeNode. The field 'PointersToFathers' refers the pointers of the parent nodes and the filed 'PointersToChildren' refers the pointers of the succeeding nodes of a particular node. The 'HorizontalPointer' is used to link the same length frequent itemsets. The filed 'IsValid' is used as a flag that denotes TRUE if the node can be combined with another node and FALSE in case that node is not combinable with any other node. A pattern tree can contain many layers in which each layer can hold same length frequent itemsets. The layer pointer Array elements points to the first node of all layers. It is used to find any frequent 1-itemsets by locating the first element of the layer pointer array, similarly second element refers the frequent 2-itemsets and the same procedure is flowed till the end of the array.

The increased two-dimensional pattern tree is initialized with null father node for all frequent 1-itemsets. The lengths of the layer pointers are initialized with the value 0. Then, every TID_ set of frequent 1-itemsets from the vertical data format table are assigned as the nodes of first layer in the pattern tree. Let the newly added item set be I_n which contains the transactions of TID-set(I_n), then the process of adding new itemset to increased two-dimensional pattern tree is as follows.

Build a new node *Node*(I_n) for itemset I_n. The member elements of *Node*(I_n) are *Node*(I_n).Itemset = I_n, *Node*(I_n).TID_set (I_n) = TID_set, *Node*(I_n).PointersToFathers = ∅, *Node*(I_n). PointersToChildren = ∅ and *Node*(I_n).IsValid = TRUE. The horizontal pointer of the first layer's last node is assigned to point *Node*(I_n). Candidate 2-itemsets are generated as $\forall x$

= 1 to n,$Node(I_n) = Node(I_n) \cup Node(I_{x-1})$. Then the support degree of candidate 2-itemsets are calculated. If it is found that a candidate 2-itemset is frequent then a new node for that candidate 2-itemset will be constructed with the itemset member element value with the combination of I_n and I_{x-1}. The TID_set element of new node will be calculated as TID_set(I_{x-1}) ∩ TID_set(I_n). The PointToFathers element is assigned to point Node(I_{x-1}) and Node(I_n). Then the HorizontalPointer of the second layer's last node is assigned to point the new node. If the candidate 2-itemset is not frequent, then the IsValid flag of all children will be assigned as FALSE. Similar process is performed to complete the increased two-dimensional pattern tree until all candidate itemsets are generated.

The process of adding new itemset data to pattern tree is performed as follows,

Let VM be the Vertical Matrix, PT be two-dimensional Pattern Tree and MS be the Minimum Support

Let tmpNode, newNode and newCombineNode are of TreeNode type

```
For i = 1 to length(LayerPointers) do
   If i = 1 then
newNode = AddNewNodeToTree(VM[i])
tmpNode = LayerPointers[i]
whiletmpNode ≠ ∅ do
iftmpNode.IsValid = TRUE then
      newCombineNode = CombineNewPattern(newNode, tmpNode)
      if Support(newCombineNode) ≥ MS then
      AddNodeToTree(newCombineNode)
      else
      SetAllChildrenFalse(newCombineNode)
      end if
else
      tmp.Node.IsValid = TRUE
end if
tmpNode = tmpNode.HorizontalPointer
      end while
      end for
```

This procedure is used to get Vertical Matrix and Pattern Tree as inputs to generate a new Pattern Tree as the output.

III. Cloud Optimized Eclat Growth

Cloud Optimized Eclat Growth (COEG) is contrived to achieve high accuracy in cloud based large scale data mining process. COEG adds two additional modules with generic Eclat procedure to make the procedure more suitable in cloud environment. The inaugurated modules are Cloud offloaded depth first search & tree branch aggregation and Fuzzy logic based multidimensional table of quantitative item fields.

A. Cloud Offloaded Depth First Search and Tree Branch Aggregation

The major tasks of this module are Depth first search dismantling for parallel execution, Cloud offloading and Tree branch aggregation.

i) Depth First Search Dismantling for Parallel Execution

The efficiency of parallel execution (cloud offloading) depends on the dismantling procedure. A legacy dismantling procedure is framed for Cloud Optimized Eclat Growth method and named as Task Dismantling Strategy (TDS). The usable members of the cloud architecture are represented as execution sites or nodes. A node with highest computational resources in the cloud is selected as the root node to distribute and accumulate the process. The Stack architecture is used to ensemble the processed data to construct the final tree data. The input data is loaded into a stack in the root execution site.

The cloud architecture is heterogeneous which allows different category of nodes can be a member of the cloud. The nodes are classified based on their computational resources and arranged in the order of most powerful category to least power category and numbered from 1 to m, where m is the number of classifications. The collection of execution nodes of the cloud is represented as $E = \{\{\varepsilon 1_1, \varepsilon 1_2 \ldots \varepsilon 1_{n1}\}, \{\varepsilon 2_1, \varepsilon 2_2 \ldots \varepsilon 2 n_2\}, \ldots \{\varepsilon m_1, \varepsilon m_2 \ldots \varepsilon m_{nn}\}\}$, where $\forall \varepsilon 1_i$ is the eligible execution node of category 1, $\forall \varepsilon 2_i$ is the execution node of category 2 and n_1, $n_2 \ldots n_n$ are the number of execution nodes for category $\varepsilon 1, \varepsilon 2, \ldots \varepsilon m$ respectively. The COEG node classification is illustrated in Fig. 2.

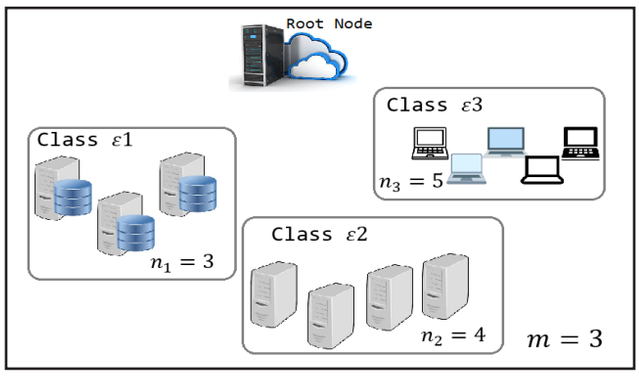

Fig. 2: COEG Cloud Node Configuration

TDS works by estimating the overall computational resource availability of the different classes in the cloud. Each category is pinged for initial computational power ρ estimation. The computational powers are computed in such a way that $\rho_x = \sum_{i=1}^{n} \rho(\varepsilon x_i)$ where $\rho(\varepsilon x_i)$ is the computational power of i^{th}

node of x^{th} execution site category. The depth threshold Δ is used to limit the cloud offloading up to a particular tree branch. The depth threshold is calculated using the equation $\Delta = \eta / m$, where η is number of DFS tasks and m is the number of execution site categories.

The computational cost of the input data tree is determined using two estimation methods α and β. The cost to reach i^{th} node from the root node is defined using α as $\alpha(i)$. The cost to reach the nearest node from i^{th} node is defined as $\beta(i)$. The total computational cost for i^{th} node is calculated as $\alpha(i) + \beta(i)$. This process iteratively continues to estimate the branch-wise cost estimation. Each iteration is assigned as the work availability in the work set $\{\omega_1, \omega_2 \ldots \omega_\eta\}$ where the work availability flag ω will be TRUE if any of the member is not equal to ϕ or ω will be FALSE if $\sum_{i=1}^{n} \omega_i = 0$, where η refers the number of available DFS tasks.

Whenever an execution site is free, it will be assigned to perform a part of DFS where multiple executions of the same process with different parts of the tree are processed simultaneously. After completion of the DFS for allocated data portion, the execution site will search for further data branch of the tree which is not yet taken by another execution site.

The cloud node task allocation procedure of TDS is given below:

Step 1. Let stack[i] be the data memory of i^{th} execution site
Step 2. Let ω be the work availability flag
Step 3. While (ω = 1)
    If (stack[i]=∅)
        GetWork();
        While(stack[i] ∅)
            DFS(stack[i]);
        End While
    End if
    UpdateWorkAvailability(ω)
End while

The procedure GetWork is defined as follow,

Step 1. Let be the number of available works
Step 2. Let f be the busy flag with initial value 0
Step 3. For (j = 1 to η AND f ≠ 1) do
    If (ω_j = ∅) then
        Load(stack[i], stack[j])
        Set f = 1
    End if
Step 4. If (f = 0) set ω = FALSE

ii) Cloud Offloading

Cloud offloading [9] [10] is a common mechanism used to maximize the resource utilization and minimizing node overloading. The offloading process in employed in two situations. The first situation is when the consented task is too huge to perform in a single computer and the second situation is when the device has very little computational resource to handle the task. In the case of data mining the first criteria is betided. Since data mining is considered as one of the most wanted chore of the modern society and the enormous size of the big data, obviously cloud offloading is recommended.

Markov's Decision Process (MDP) is used to offload the data mining process in the cloud environment. The performance rating is calculated using the following equations.

The execution time in Cloud process Tc is calculated as $T_c = T_{ci} + T_{cp} + T_{co}$ where T_{ci} is the process distribution time, T_{cp} is the measured processing time and T_{co} is the result accumulation time.

The execution time of Local process T_l is equal to the value of T_{lp}, which is the processing time measured in the local system. The process distribution and accumulation times are not applicable in local processing.

The cloud offloading process will be stated as beneficial based on the condition $T_c < T_l$. A threshold value φ is calculated and tracked throughout the offloading process to find weather the offloading is performed in the right track. The threshold value φ is calculated using the Bayesian average based on the formula $\frac{|c| \times m + \sum_{i=1}^{|c|} \varphi_i}{|c| + m}$ where $|c|$ is the dataset size, m is the previous mean value and φ_i is the previous threshold value. This decision-making procedure makes the result of "Is Local Execution better than Delay" process into dynamic.

The efficiency of cloud offloading can be calculated using the equation $\mu \equiv \lim_{n \to m} \frac{1}{n} \sum_{i=1}^{n} (\delta_i - \alpha_i)$ where μ is the mean job response time, δ_i is the process completion time and α_i is the process accepted time of execution site εx_i here x refers the node class.

The cloud offloading decision process flow is given in Fig. 3.

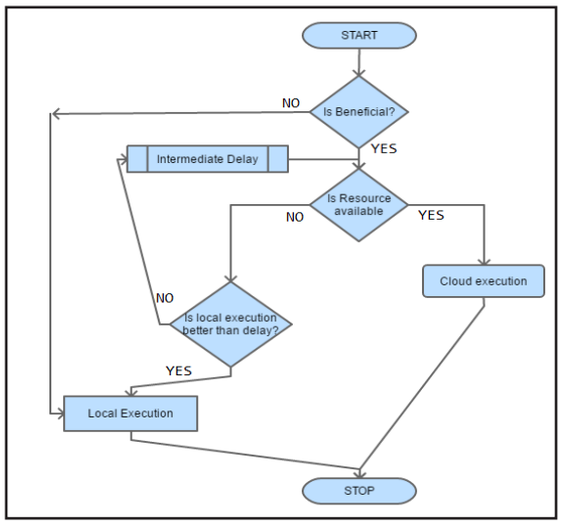

Fig. 3: Cloud offloading Decision Process Flow

iii) Tree Branch Aggregation

This is the processed data collection phase which is responsible for accumulating the works performed by cloud offloading procedure. This module collects the stack[i] data from all execution sites and store them in the root execution node. This information is stored in a table named as Machine Index Table that represents a tree architecture. The datamining process starts immediately after completion of this phase. An example cloud offloading and tree branch aggregation is illustrated in Fig. 4.

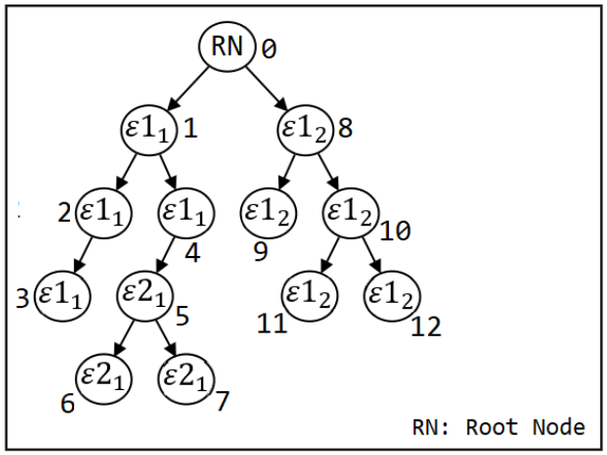

Fig. 4: Cloud Offloading and Tree Aggregation Example

In Fig. 4, the number 0 refers the root node and 1 to 12 refers the dataset work sets ($\omega_1 \ldots \omega_\eta$) given in the hierarchy where $\eta = 12$. Each execution εx_y can handle any number of work sets based on its computational capability and availability in the cloud. In this example, execution site $\varepsilon 1_1$ handles the work sets ω_1, ω_2, ω_3, ω_4, execution site $\varepsilon 1_2$ handles ω_8, ω_9, ω_{10}, ω_{11}, ω_{12} and execution site $\varepsilon 2_1$ handles ω_5, ω_6, ω_7. Since the proposed procedure uses depth first search, work sets are ordered in DFS order and cloud offloading procedure follows breadth first search order because the execution sites are classified based on the computational capability. As per the example given here, execution sites belong to the category-1 accomplish more number of work sets whereas execution site $\varepsilon 2_1$ accomplishes less number of work sets. The root node collects all processed data and construct the entire tree structure after the tree aggregation process.

The overall execution time is measured in cloud using the equation $T_\alpha = \forall j = 1 \rightarrow n, \forall k = 1 \rightarrow m$: $\max(T_{cp}(\varepsilon k_j)) + T_{ci} + T_{co}$, where $T_{cp}(\varepsilon k_j)$ is the processing time of j^{th} execution site ε of k^{th} classification, T_{ci} is the TDS processing time otherwise called as process distribution time and T_{co} is the processing time of tree branch aggregation time which is known as result accumulation time.

B. Fuzzy Logic Based Multidimensional Table of Quantitative Item Fields (FMDT)

Modern data mining procedures has to perform more tasks than the conventional data mining procedures which operate based on Boolean values. Because modern data mining procedures has to deal with multiple tasks as customer profiling, customer requirement identification, target marketing, cross market analysis, finding purchasing patterns, finance planning, asset evaluation, resource planning and identifying frauds using anomaly detection.

Finding the relationships between transaction items is alone not enough to perform the above-mentioned tasks. Thus, here in this proposed work, Fuzzy logic based multidimensional table of quantitative item fields' procedure is introduced. This procedure is used to extract data by finding the relationships between transaction items as well as it keeps track on different quantities of the items involved in the transactions. For example, a transaction history taken from a car tyre garage given in Table I as per the conventional way, Table II represents the same information in Eclat's vertical format and the same transactions are presented as Table III in the way of fuzzy logic based multidimensional table of quantitative item fields.

Table I: Conventional Representation of Car Tyre Garage Transactions

Transaction	Item Sets
1	T
2	T, WA, WB
3	T, WA
4	T, WA, WB

Table II: Vertical Format Representation of a Car Tyre Garage Transactions

Item Set	Transactions
T	1,2,3,4
WA	2,3,4
WB	2,4

Table III: FMDT Representation of a Car Tyre Garage Transactions

Transaction	Number of Tyres	Item Sets
1	1	T
2	4	T, WA, WB
3	2	T, WA
4	4	T, WA, WB

Data description of Table I, II and III: T: Tyre, WA: Wheel Alignment, WB: Wheel Balance. Here Wheel Alignment and Wheel Balance are not actual physical items, but they are the services provided by garages and treated as items based on the bill receipts. In Table II, the relationship between the items T, WA and WB are look like in complete random probability. But in the case of Table III, it is easily predicted that if the quantity of item T is more than 1, then there is a higher probability for occurrence of item WA. Similarly, if the quantity of item T is

greater than 2, then there is a higher probability for occurrences of items WA and WB together. This quantity-based association extraction is a vital part of modern data mining procedures to cover today's real-world situation. The visualization of FMDT Table III illustrated in Fig. 5.

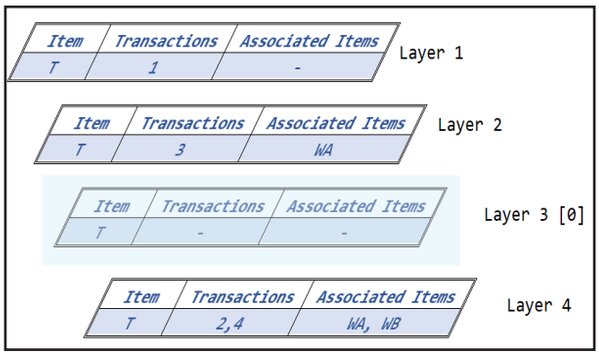

Fig. 5: FMDT Visualization

In the above illustration, each table dimension is given in separate layer for easy representation. The Layer number refers the quantity of the item involved. For example, Layer 1 shows that the item T in single quantity is occurred in transaction 1. Even though T took place in transactions 2, 3 and 4, they are not stored in the first layer because the quantity variation of T. Similarly, Layer 2 shows that the item T in two quantities took place in transaction 3 and this quantitative item is associated with another item WA. Since there is no transaction with 3 number of item T, Layer 3 is kept as the hidden layers with ∅ value hence not stored in memory. Layer 4 shows that the item T in quantity 4 occurred in transactions 2 & 4 and the associated items are WA and WB.

If less number of quantity variation involved in the transactions, then all the individual quantities are stored in FMDT as separate layers. When the quantity raises to more than hundred numbers, handling that much of individual layered tables will be more resource consuming process. The quantities of itemsets are random and there is no standard formula available to predict them. Therefore, the layer classification is performed in COEG with the help of fuzzy logic. The itemset quantities are taken as the input for the fuzzification process. The interference system determines the quantity range allocated for dimensional layers.

For example, let the quantities of item X in a transaction set ψ are $\{q_1, q_2, q_3, q_y\}$. This crisp set is converted into fuzzy set with the following characteristics:

$$\chi_\psi(q_1) = \begin{cases} 1 \; if \; q_1 \neq 0 \; and \; q_1 \leq \dfrac{\Delta}{y} \\ 0 \; otherwise \end{cases}$$

$$\chi_\psi(q_2) = \begin{cases} 1 \; if \; q_2 > \dfrac{\Delta}{y} \; and \; q_2 \leq \dfrac{2\Delta}{y} \\ 0 \; otherwise \end{cases}$$

$$\chi_\psi(q_3) = \begin{cases} 1 \; if \; q_3 > \dfrac{\Delta}{y/2} \; and \; q_3 \leq \dfrac{4\Delta}{y} \\ 0 \; otherwise \end{cases}$$

$$\chi_\psi(q_y) = \begin{cases} 1 \; if \; q_y > \dfrac{2^{y-1}\Delta}{y} \\ 0 \; otherwise \end{cases}$$

Where $\Delta = q_y - q_1$ and y is the number of quantity variations (count) in the transaction set ψ of item X.

The defuzzification results are loaded into the memory for FMDT process. This fuzzy enabled methodology makes it possible to handle large quantities of different items in the transactions.

IV. Experimental Setup

The experimental setup is created based on the HP ProLiant DL160 Gen9 Intel Xeon E5-2620v4 12 core cloud server [11]. This server is equipped with Intel Xeon 2.1 GHz 8 core processor with 20MB L1 cache memory, 16GB DDR4-2400 RAM in 16 memory slots operates with 240V 1.8 KW power. To evaluate the existing methods and proposed method, this server is leased for 6 months along with its dedicated cloud services.

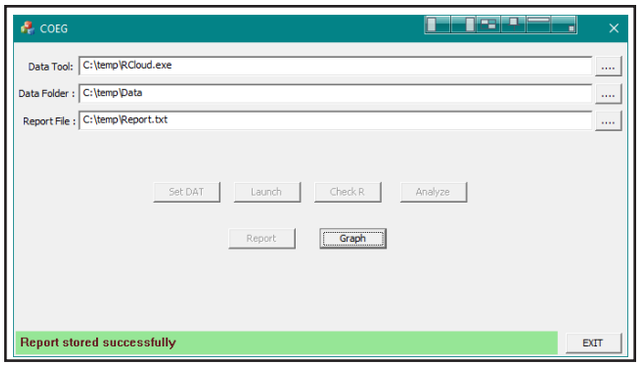

Fig. 6: User Interface

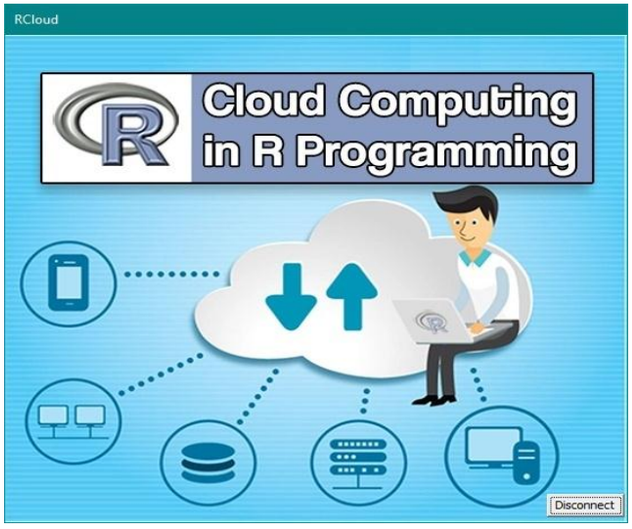

Fig. 7: R-Cloud Connectivity Tool

Since the existing methods are implemented in different programming languages, a standard Common Language

Runtime (CLR) [12] environment is created to measure the performance metrics of the procedures. Visual Studio 2013 IDE [13] [14] is used to create the CLR and Visual C++ programming language is used to write scripts to utilize the cloud services. A Legacy User Interface is designed to upload data and to distribute them in cloud environment. The data analytical tool R is accessed in the Cloud server through the legacy user interface. All algorithms are triggered to run in the server one by one and their performances are logged for generating the report file and comparison graphs. The user interface is showed Fig. 6. The R-Cloud [15] is a community of github social coding provides the R-Cloud connectivity tool which is given in Fig. 7.

V. Results and Analysis

The benchmark datasets [16] [17] T10I4D100K, Kosarak, T40I10D100K, Mushroom, Chess and Accidents are used to evaluate the performance of the different procedures discussed in this work. Parameters like Processing Time, Accuracy, Precision and Recall and memory consumption are measured in the cloud server to prepare a detailed report and comparison graphs. The records from the above-mentioned datasets are treated as 100% data given for the methods in comparison. Total number records in the datasets accumulation is 7707525. The metrics are measured after processing 1541505 number of records and the results are tabulated for every 20% of data.

A. Accuracy

Accuracy is the main objective of data mining. A data mining procedure is considered as a best method if it provides more accuracy. In this case, Eclat-Opt secures the least average accuracy score of 81.89% whereas the proposed COEG method secures the highest accuracy score average of 90.77%. The very close result in accuracy of 87.63% is achieved by the Eclat-Growth procedure. The measured values are tabulated in Table IV and the comparison graph is given in Fig. 8.

Table IV: Accuracy (%)

Data (%)	Eclat	Eclat-Opt	Bi-Eclat	Eclat-Growth	COEG
20	83.09	80.84	86.76	88.01	90.28
40	83.33	81.79	86.09	88.39	91.69
60	83.56	82.26	87.63	88.33	91.54
80	83.18	82.11	86.53	88.96	89.38
100	82.51	82.49	87.15	87.63	90.95

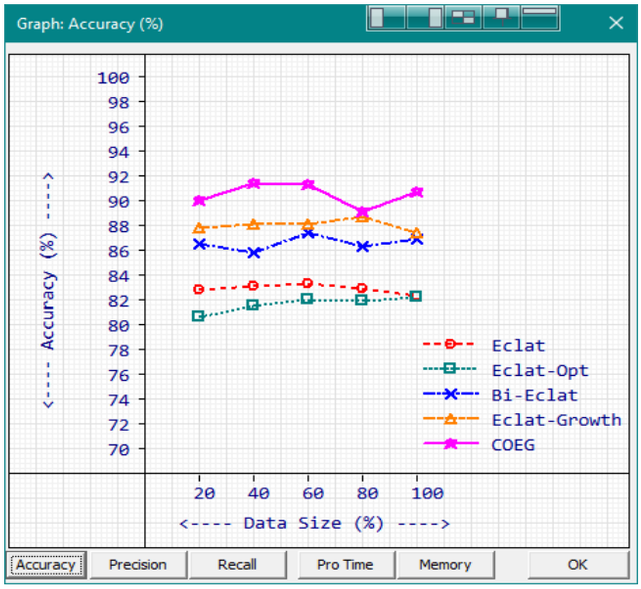

Fig. 8: Accuracy Graph

B. Precision

Precision is also called as positive predicted value, which refers the specificity of a particular datamining procedure. The higher values of prediction refer the higher quality of a datamining procedure. Prediction is represented in terms of percent (%) in general. Based on the execution results of existing and proposed methods, the value of precision is calculated and given in Table V. The methods Eclat, Eclat-Opt, Bi-Eclat, Eclat-Growth and COEG achieved the precision value averages of 83.06%, 82.19%, 86.67%, 88.61% and 90.82% respectively. From the above values, it is observed that the COEG procedure achieved highest precision values. The comparison graph for precision values is given in Fig. 9.

Table V: Precision (%)

Data (%)	Eclat	Eclat-Opt	Bi-Eclat	Eclat-Growth	COEG
20	83.48	81.51	86.54	88.57	90.61
40	82.67	81.63	85.27	88.23	91.88
60	83.23	82.37	87.84	87.98	91.13
80	83.75	82.87	85.99	89.12	89.24
100	82.17	82.61	87.72	89.16	91.28

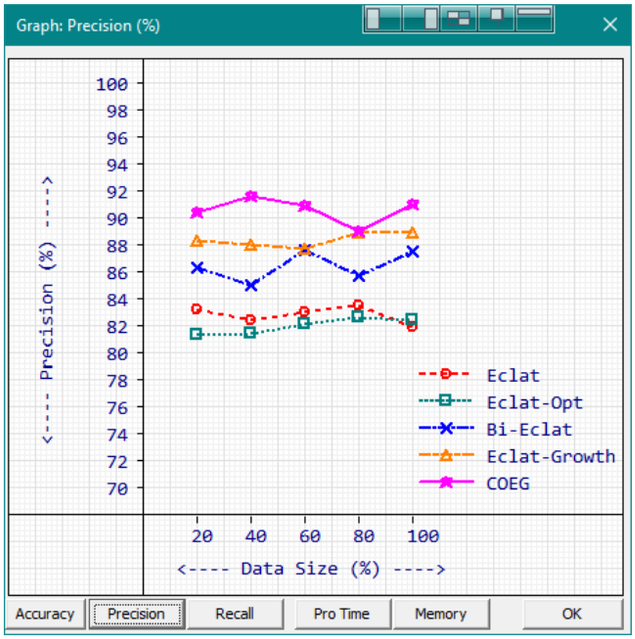

Fig. 9: Precision Graph

C. Recall

The term Recall and Sensitivity are interchangeable in datamining. This refers the unit of relevant instances successfully retrieved from the overall relevant documents. A good datamining procedure should have higher sensitivity towards the input data. Based on the implementation results, it is observed that the Recall averages for methods Eclat, Eclat-Opt, Bi-Eclat, Eclat-Growth and COEG are 83.19%, 81.71%, 86.85%, 88.01% and 90.68% in order. COEG method secured the highest recall value average of 90.68%. The Eclat-Growth procedure takes the second position with recall value average of 88.01%. The recall values are calculated using the measured True Positive, False Positive, True Negative and False Negative values observed from the implementation. The Recall values of datamining procedures are given in Table VI. The same is plotted as the graph for ease of understanding given in Fig. 10.

TABLE VI: RECALL (%)

Data (%)	Eclat	Eclat-Opt	Bi-Eclat	Eclat-Growth	COEG
20	82.83	80.43	86.91	87.6	90.01
40	83.78	81.9	86.69	88.5	91.53
60	83.77	82.19	87.47	88.6	91.88
80	82.81	81.63	86.94	88.84	89.5
100	82.74	82.42	86.73	86.51	90.68

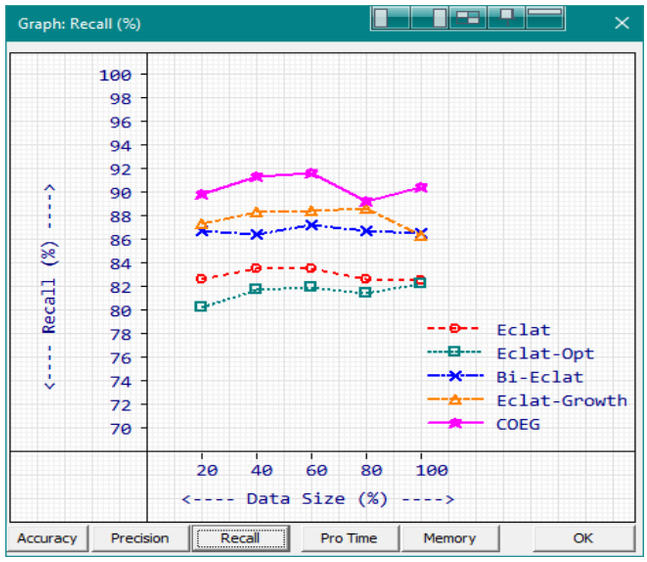

Fig. 10: Recall Values Comparison Graph

D. Processing Time

Processing time is one of the vital argument in datamining procedures. The execution time of existing and proposed methods are measured here in millisecond units for higher precision. The entire transaction records are split into 20% data chunks and the processing time is measured after every data chunk is processed. Therefore, 5 processing times are noted for every datamining procedure discussed here. The observed processing times are produced in Table VII.

TABLE VII: PROCESSING TIME (MS)

Data (%)	Eclat	Eclat-Opt	Bi-Eclat	Eclat-Growth	COEG
20	17269	16171	19217	15390	11387
40	34556	32530	38530	31133	23100
60	51825	48614	57985	46170	34262
80	69361	64686	76921	61778	45643
100	86626	80926	96426	77195	57143

Based on the results, it is observed that the proposed method COEG accomplished the mining task in 57143 mS which is lesser than the processing time 77195 mS of Eclat-Growth. COEG completed the datamining process first in all 5 different stages of the entire mining process. The comparison graph for processing time is given in Fig. 11.

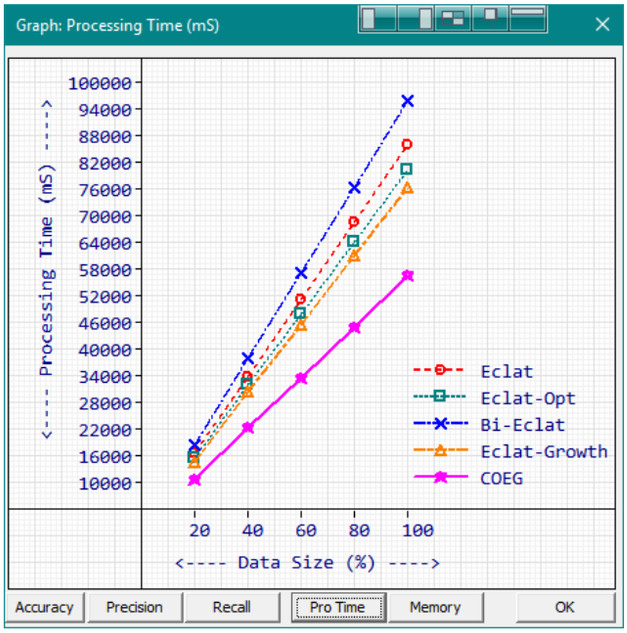

Fig. 11: Processing Time (mS)

E. Memory

Memory is one of the important elements in computational resources. Every computational task occupies some amount of memory in Random Access Memory (RAM) which is considered as primary memory of a computer system. Adequate free memory in RAM is required to run a computer system optimally. Overloading the RAM will cause virtual memory utilization in which the process depends on the secondary memory. While comparing primary memory, secondary memory accessing is very slow, thus it leads to prolonged processing time. Therefore, it is important that not to overload a single system to the core to get the better performance. Proposed COEG uses a well-designed offloading mechanism splits the datamining process into several small tasks and distributes them to the appropriate execution site. The memory utilization is measured throughout the execution of datamining procedures and the result is given in Table VIII.

TABLE VIII: MEMORY (KB)

Data (%)	Eclat	Eclat-Opt	Bi-Eclat	Eclat-Growth	COEG
20	421	388	402	368	294
40	849	782	805	742	591
60	1272	1168	1209	1108	893
80	1687	1558	1612	1480	1185
100	2114	1947	2022	1845	1475

Based on the experiment results, it is observed that the proposed method uses the minimum memory of 294 kB for handling 1541505 number of transaction data and it consumes 1475 kB of memory to handle 7707525 number of transaction data. The nearest performing method is the Eclat growth with the memory consumption of 368 kB to 1845 kB.

The memory utilization is plotted as graph for comparison and presented as Fig. 12.

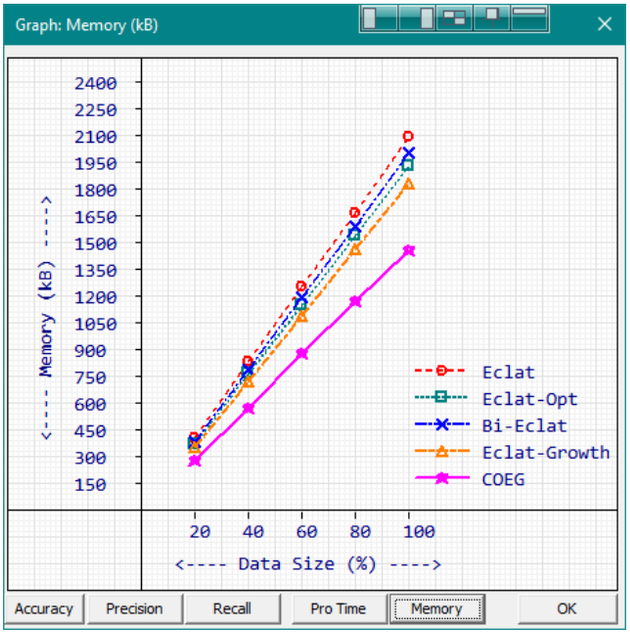

Fig. 12: Memory Consumption (kB)

VI. CONCLUSION

An optimized datamining procedure is proposed as "Cloud Optimized Eclat Growth using Fuzzy logic" in this work to perform better in the cloud-based environment. The electronic gadgets with computational capability are increasing rapidly. The modern network architecture makes it possible to relish the ultimate cloud computing environment by connecting almost all the communicable devices in a cost-effective way – which is projected as the future of computing. Based on the implementation results in a dedicated cloud environment, it is learned that the proposed COEG method performs the datamining process more efficiently in the modern computational environment.

REFERENCES

[1] M. A. Khan, S. K. Pradhan, and H. Fatima, "An efficient technique for Apriori algorithm in medical data mining," *Innovations in Computer Science and Engineering*, Springer, pp. 187-195, 2019.

[2] L. Pietruczuk, L. Rutkowski, M. Jaworski, and P. Duda, "How to adjust an ensemble size in stream data mining?," *Information Sciences*, Elsevier, vol. 381, pp. 46-54, March 2017.

[3] M. Nilashia, O. bin Ibrahima, N. Ithnin, and N. H. Sarmin, "A multi-criteria collaborative filtering recom-

mender system for the tourism domain using Expectation Maximization (EM) and PCA–ANFIS," *Electronic Commerce Research and Applications*, Elsevier, vol. 14, no. 6, pp. 542-562, 2015.

[4] Y. G. Jung, M. S. Kang, and J. Heo, "Clustering performance comparison using K-means and expectation maximization algorithms," *Biotechnology & Biotechnological Equipment*, vol. 28, no. supp1, pp. 44-48, 2014.

[5] P. Bermejo, J. A. Gámez, and J. M. Puerta, "Speeding up incremental wrapper feature subset selection with Naive Bayes classifier," *Knowledge-Based Systems*, Elsevier, vol. 55, pp. 140-147, January 2014.

[6] J. Heaton, "Comparing dataset characteristics that favor the Apriori, Eclat or FP-Growth frequent itemset mining algorithms," *SoutheastCon*, IEEE, 2016.

[7] Z. Ma, J. Yang, T. Zhang, and F. Liu, "An improved Eclat algorithm for mining association rules based on increased search strategy," *International Journal of Database Theory and Application (IJDTA),* 2016.

[8] M. Karanjikar, and S. V. Kedar, "Secure association rule mining using Bi-Eclat algorithm on vertically partitioned databases," *2017 International Conference on Intelligent Sustainable Systems (ICISS),* IEEE, 2017.

[9] Z. Jiang, and S. Mao, "Energy delay tradeoff in cloud offloading for multi-core mobile devices," *Emerging Cloud-Based Wireless Communications and Networks*, IEEE, 2015.

[10] R. Aldmour, S. Yousef, M. Yaghi, S. Tapaswi, K. K. Pattanaik, and M. Cole, "New cloud offloading algorithm for better energy consumption and process time," *International Journal of System Assurance Engineering and Management*, Springer, vol. 8, no. supp2, pp. 730-733, November 2017.

[11] https://serverental.com/product/hpe-proliant-dl160-gen9-server-sale/

[12] https://docs.microsoft.com/en-us/dotnet/standard/clr

[13] M. Gousset, M. Hinshelwood, B. A. Randell, B. Keller, and M. Woodward, *Professional Application Lifecycle Management with Visual Studio 2013*, Wiley Publications.

[14] S. Amann, S. Proksch, S. Nadi, and M. Mezini, "A study of visual studio usage in practice," *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, IEEE, 2016.

[15] https://rcloud.social/index.html

[16] G. Nguyen, T. Le, B. Vo, and B. Le, "A new approach for mining top-rank-k erasable itemsets," *Asian Conference on Intelligent Information and Database Systems*, Springer, 2014.

[17] J. Yang, C. Yang, and Y. Wei, "Frequent pattern mining algorithm for uncertain data streams based on sliding window," *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC),* IEEE, 2016.