Abstract: Cloud Optimized Eclat Growth (COEG) method is a new Eclat based data mining approach to work well with real-time cloud environments. COEG used new concepts of localized cloud Eclat data processing by cloud offloading method. 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. The COEG method uses localized cloud data processing, multidimensional liked lists are used which improves the item, transaction and pattern-based search capabilities. “Multi-core Processing Cloud Eclat Growth” (MPCEG) is intended to create a smooth interface for Eclat growth algorithm to run in multi-core processor-based cloud computing environments. MPCEG blending multi-core high performance Central Processing Units (CPUs) and Graphics Processing Units (GPUs). MPCEG used new procedures for Cloud Parallel Processing, GPU Utilization, Annihilation of floating point arithmetic errors by fixed point replacement in GPUs and Hierarchical offloading aggregation. The performance of these two algorithms is compared based on the Time efficiency. After the comparison, we conclude that MPCEG algorithm is the fastest algorithm, it takes less time to generate the frequent item-sets as compared to other algorithm, that is, COEG algorithm.

Index Terms: Cloud Offloading, Cloud parallel processing, Data Mining, Fixed point arithmetic, Graphics Processing Units, Hierarchical offloading, Multi-core processing

1 INTRODUCTION

Now-a-days 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. 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. The COEG [32] 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. Determining worthy machines for aggregation to perform data mining 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 COEG with the help of Machine Index Table (MIT). The revolution in communication systems made an emergent increase in the usage mobile computing devices. The increased number of mobile computing devices causes a notable raise in the swarming digital data quantity. Therefore

the conventional desktop based data processing is fading now and the emerging cloud computing environments were spread in a very impressive manner. Present computers are also powered by multicore processors along with huge memory and storage capacity. Initially GPUs are introduced to process image transformations and rendering. The CPUs are offloading their excessive image processing tasks to the GPUs to improve the user experience by reducing the response time. But advanced GPUs are good enough to perform massive mathematical calculations swiftly by their thousands of powerful energy-efficient parallel micro-core architecture [2]. This property of GPU is very useful for modern machine learning and big data analysis. The cloud platform already taking advantages of these GPUs and present virtual machines are ascending with CPUs combined with GPUs. Companies such as Amazon and Google are providing cost effective parallel cloud services. Amazon Light sail, EC2 [3], Microsoft Azure and Cloud4C are using multicore CPU-GPU combination servers to provide Virtual Machines for their clients. Conventional data mining algorithms are designed to run in single core processor-based computer systems. Though the performances of conventional data mining algorithms are improving with high speed processors, the performance can be significantly improved by modifying them. The proposed method is created to develop the tune-up utilities for Eclat Growth algorithm with some enhancements. Using GPUs in data mining has a vulnerability of GPU Floating point Paranoia [4][5] – a precision problem in real numbers. The proposed system is aspired also to eliminate this problem by introducing fixed point arithmetic in Eclat Growth GPU processing. The primary idea of this work is to use the Eclat Growth data mining procedure in powerful combination of Multi-core CPUs, Massive micro-core GPUs and cloud computing environment.

2 ALGORITHM DESCRIPTION

2.1 Cloud Optimized Eclat Growth

Cloud optimized Eclat Growth (COEG) [32] is contrived to achieve high accuracy in cloud based large scale data mining
Eclat Growth method and execution site is a common mechanism used to offload the data mining process in the cloud environment. The performance 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). TDS works by estimating the overall computational resource availability of the different classes in the cloud. The cloud node task allocation procedure of TDS is given below.

Step 1. Let stack[i] be the data memory of ith execution site
Step 2. Let ω be the work availability flag
Step 3. While (ω = 1)
    If (stack[i] ≠ 0)
    GetWork();
    While(stack[i] ≠ 0)
    DFS(stack[i]);
    End While
    End if
End while

The procedure GetWork is defined as follow.
Step 1. Let n be the number of available works
Step 2. Let f be the busy flag with initial value 0
Step 3. For (j = 1 to n AND f ≠ 1) do
    If (ωj = 0) then
    Load (stack[i], stack[j])
    Set f = 1
    End if
End if
Step 4. If (f = 0) set ω = FALSE

2.1.2 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 $T_c$ is calculated as $T_c = T_{cl} + T_{cd} + T_{co}$, where $T_{cl}$ is the process distribution time, $T_{cd}$ 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 $T_{l1}$, 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 $\phi$ is calculated and tracked throughout the offloading process to find weather the offloading is performed in the right track. The threshold value $\phi$ is calculated using the Bayesian average based on the formula $\phi = \frac{\sum_{i=1}^{m} \phi_i}{m}$ where $m$ the dataset size is, $\phi$ the previous mean value and $\phi_i$ 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 $\mu = \frac{\sum_{i=1}^{n} (\delta_i - \alpha_i)}{n}$ where $\mu$ is the mean job response time, $\delta_i$ is the process completion time and $\alpha_i$ is the process accepted time of execution site $x$ here $x$ refers the node class.

2.1.1.3 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 Figure 1.

**Fig 1. Cloud offloading and Tree aggregation example**

In Figure 1, the number 0 refers the root node and 1 to 12 refers the dataset work sets ($\omega_1$ ... $\omega_n$) given in the hierarchy where $\omega_1 = 12$. Each execution $x$ can handle any number of work sets based on its computational capability and availability in the cloud. In this example, execution site $e_1$ handles the work sets $\omega_1$, $\omega_2$, $\omega_3$, $\omega_4$, execution site $e_2$ handles $\omega_5$, $\omega_6$, $\omega_7$, $\omega_8$, and execution site $e_3$ handles $\omega_9$, $\omega_{10}$, $\omega_{11}$, $\omega_{12}$. 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 e2 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

\[ T_e = \sum_{i=1}^{n} T_e[f_i] + T_{cl} + T_{c0}, \]

where \( T_{e[f_i]} \) is the processing time of \( f_i \)\textsuperscript{th} execution site \( e \), \( T_{cl} \) is the TDS processing time otherwise called as process distribution time and \( T_{c0} \) is the processing time of tree branch aggregation time which is known as result accumulation time.

### 2.1.2 Fuzzy logic based Multidimensional Table of quantitative item fields (FMDT)

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. 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 \( T \) be \( \{q_1, q_2, q_3, \ldots, q_n\} \). This crisp set is converted into fuzzy set with the following characteristics

\[
\begin{align*}
X_{\mu}(q_1) &= \{\forall y, y_1 \leq y \land y_2 \geq y \} \\
X_{\mu}(q_2) &= \{\forall y, y_1 \geq y \land y_2 \leq y \} \\
X_{\mu}(q_3) &= \{\forall y, y_1 \leq y_2 \land y_3 \geq y \} \\
X_{\mu}(q_n) &= \{\forall y, y_1 \geq y_2 \land y \geq y \}
\end{align*}
\]

Where \( \Delta = q_n - q_1 \) and \( y \) is the number of quantity variations (count) in the transaction set \( T \) 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.

### 2.2 Multi-core Processing Cloud Eclat

The main modules of the MPCEG [33] methods are

- GPU Eclat-growth (G-Eclat-growth),
- Floating-point to Fixed-point Arithmetic Conversion (FL2FP)
- Hierarchical Offloading - Aggregation (HOA)

#### 2.2.1 G-Eclat-growth

The G-Eclat-growth is derived from the standard Eclat algorithm for GPU based parallel cloud optimization given below as Algorithm

#### Algorithm 1: G-Eclat-Growth

For \( x = 1 \) to \( \rho \) do

\[
\text{VM}_x = \text{CreateVMfromDatabase}(D)
\]

Initialize \( PT_x \) with 0 values

For \( i = 1 \) to \( \text{length}(	ext{VM}_x) \) do

Parallel\_GPU\_Execute(\( PT_x \)) Begin

If(\( \text{length}(	ext{VM}_x[i]) \leq \text{MS} \)) then

AddItemSetToPatternTree(\( VM_x[i], PT_x \), MS)

End for

End for

\( VM = \text{Merge} [\text{VM}_1, \text{VM}_2, \ldots, \text{VM}_\rho] \)

\( PT = \text{Merge} [PT_1, PT_2, \ldots, PT_\rho] \)

\( FI = \text{GetAllFrequentItemsetsFromPatternTree}(PT) \)

#### 2.2.2 Floating-Point to Fixed-Point Arithmetic Conversion

GPUs are known for their rapid parallel execution speed where accuracy in floating point is compromised. This phenomenon is known as GPU floating paranoa. To eliminate the precision problems in GPU, A fixed-point to floating-point arithmetic interface is introduced in this work. IEEE 754 standard based Floating points are used in the latest CPUs and GPUs. IEEE 754 standard floating-point numbers are having three components. They are the sign, exponent and mantissa which are common for both 32-bit and 64-bit architectures. The number of bits used to represent the components differs between 32-bit and 64-bit representations. The sign component is assigned with a single bit memory that can hold either 0 or 1 to represent positive sign and negative sign respectively. The exponent component in single precision gets 8 bits memory with the bias value of 127 and in double precision gets 11 bits memory with the bias value of 1023. The Mantissa component is used to represent the precision bits of a number. The latest optimization of IEEE 754 standard is available with base 2. Special Values such as NaN (Not a Number), \( +\infty \) (Positive Infinity) and \( -\infty \) (Negative Infinity) are handled substantially in IEEE 754 standard. The operations with special values are listed in Table 1.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Process</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>( \pm \infty \times \pm \infty )</td>
<td>( \pm \infty )</td>
</tr>
<tr>
<td>3</td>
<td>( \pm \times \pm )</td>
<td>( \pm )</td>
</tr>
<tr>
<td>4</td>
<td>( \pm \text{finite} \times \pm \text{finite} )</td>
<td>( \pm )</td>
</tr>
<tr>
<td>5</td>
<td>( 0 + 0 )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>6</td>
<td>( 0 - 0 )</td>
<td>( -\infty )</td>
</tr>
<tr>
<td>7</td>
<td>( \infty - \infty )</td>
<td>NaN</td>
</tr>
<tr>
<td>8</td>
<td>( \infty + \infty )</td>
<td>NaN</td>
</tr>
<tr>
<td>9</td>
<td>( 1 \times 0 )</td>
<td>NaN</td>
</tr>
<tr>
<td>10</td>
<td>( \infty \times 0 )</td>
<td>NaN</td>
</tr>
<tr>
<td>11</td>
<td>NaN==NaN</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
The normalizing procedures used in GPUs cause accuracy issues, thus all the floating-point number are converted into fixed-point number in MPCEG. The gainsays in this conversion are signed-unsigned handling, memory overflow and Invalid results. The signed-unsigned problem occurs when the result of an operation is negative and the host variable is unsigned type. Memory overflow occurs while processing huge operands. The invalid results refer conditional such as divide-by-zero. MPCEG typecasts all data types into signed one to overcome the signed-unsigned handling issues. Memory overflow is prevented by checking the size of the input operands before fed them to process and discarding huge size data from calculations to trigger the overflow indication message and to save processing time. The memory overflow condition uncommon in could environments due to the sufficient memory resources. The invalid results are handled in the same way as IEEE 754 standard. Pointers are used to preserve the precision of input operands. The decimal places are neutralized using these pointers before converting them into fixed-point numbers. Same pointers are used to place the decimal point in the results based on the operation performed. A floating-point operation and its equivalent fixed-point operation are explained in Figure 2.

Algorithm 2: MPCEG Floating-point to Fixed-point conversion

Let \( y \) be the maximum permitted data size
Let \( \delta_a \) and \( \delta_b \) are the decimal location pointers of operand \( a \) and \( b \)
Let \( n \) be the number available cores in GPU
Let \( O \) be the operator
For \( x = 1 \) to \( y \) do

\[
\begin{align*}
\text{Parallel_GPU_Execute}(n) & \begin{array}{l}
\delta_a = \text{GetDecimalPosition}(a) \\
\delta_b = \text{GetDecimalPosition}(b) \\
\text{Normalize}(a, b, \max(\delta_a, \delta_b)) \\
c = \text{ApplyBatch}(a, b, O) \\
\text{Parallel_GPU_Execution} \\
\text{FixDecimal}(c, \delta_a, \delta_b, O)
\end{array} \\
\end{align*}
\]
End for
This algorithm is applied in parallel to the GPU cores.

2.2.3 Hierarchical Offloading Aggregation (HOA)
Offloading the job and aggregating the completed tasks is an important phase in Cloud computing. In this work, the work dilution process is designed in to two stages. The first one is dealing with the work allocation between the virtual machines. The second one is thinning of tasks into sub-levels to assign them into CPU and GPU processing units. The CPU is dedicated to do this process. Gathering and accumulating the results from GPU cores is the main task of the CPUs in the Virtual Machines, whereas the GPU units performs the parallel calculations simultaneously in their massive number of cores. The data and control flow in HOA process is explained in Figure 3.

Data are collected for processing in Hybrid Cloud. The first level of hierarchy H1 starts here. Entire work is parsed and offloaded to different Virtual Machines in the server. Parsed jobs are distributed to different Processors cores in hierarchical level H2. Each core of the CPU then splits the jobs into different tasks and allocates them to the GPU cores. The aggregation process is performed in straight reverse process. Processed results are collected from the GPU cores by the CPU cores. Then the Virtual Machine accumulates the data from CPUs. The hybrid cloud server ensembles the entire
process.

3 EXPERIMENTAL SETUP

The experimental setup is created based on the HP ProLiant DL160 Gen9 Intel Xeon E5-2620v4 12 core cloud server [25]. 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. Since the existing methods are implemented in different programming languages, a standard Common Language Runtime (CLR) [26] environment is created to measure the performance metrics of the procedures. Visual Studio 2013 IDE [27][28] 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 Figure 4. The R-Cloud [29] is a community of github social coding provides the R-Cloud connectivity tool which is given in Figure 5.

Fig 4: MPCEG User Interface

![Fig 4: MPCEG User Interface](image)

Fig 5. R-Cloud Connectivity Tool

![Fig 5. R-Cloud Connectivity Tool](image)

4 RESULT AND ANALYSIS

The benchmark datasets [30][31] 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, 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. A total number record 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. 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 2.

Table 2: Processing Time

<table>
<thead>
<tr>
<th>Data (%)</th>
<th>COEG (mS)</th>
<th>MPCEG (mS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>11387</td>
<td>3876</td>
</tr>
<tr>
<td>40</td>
<td>23100</td>
<td>7762</td>
</tr>
<tr>
<td>60</td>
<td>34262</td>
<td>11651</td>
</tr>
<tr>
<td>80</td>
<td>45643</td>
<td>15536</td>
</tr>
<tr>
<td>100</td>
<td>57143</td>
<td>19406</td>
</tr>
</tbody>
</table>

Based on the results, it is observed that the method MPCEG accomplished the mining task in 19406mS which is lesser than the processing time 57143mS of COEG. Completed the data mining process first in all 5 different stages of the entire mining process. The comparison graph for processing time is given in Figure 6.

![Fig 6. Processing Time (mS)](image)

5 CONCLUSION

Based on the implementation results in a dedicated cloud environment, it is learned that the COEG and MPCEG
methods are performs the datamining process more efficiently in the modern computational environment. Multi-core Processing Cloud Eclat Growth (MPCEG) is consume less processing time than COEG.

REFERENCES
[24] Silviu Răileanu, Florin Anton and Theodor Borangiu, "High Availability Cloud Manufacturing System Integrating Distributed MES Agents" in International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing (Service Orientation in Holonic and Multi-Agent Manufacturing), Springer 2017, pp 11-23
[26] https://docs.microsoft.com/en-us/dotnet/standard CLR
[27] Mickey Gousset, Martin Hinshelwood, Brian A.


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


[31]. Junrui Yang, Cai Yang, Yanjun Wei, "Frequent Pattern Mining Algorithm for Uncertain Data Streams Based on Sliding Window", Intelligent Human-Machine Systems and Cybernetics (IHMSC), IEEE 2016
