Scalable Clustering using MapReduce Programming Model

Abhishek Santra[1], Anurag Agarwal[2]

ABSTRACT

The aim is to implement a clustering algorithm, which will run in a distributed computing environment for which, a multi-node Hadoop cluster providing support for the Hadoop Distributed File System and the MapReduce Programming Model has been set up.

In this paper, Exclusive and Complete Clustering (ExCC), a grid based algorithm, is implemented by scheduling consecutive MapReduce Jobs, for massive data sets. An optimal cluster parameter setting of four datanodes with 64 MB block size is obtained upon performing experiments to know the functional characteristics of ExCC in the distributed environment under different parameter settings.

KEYWORDS

Grid based, Incremental, Exclusive, Complete and Scalable Clustering; Distributed Environment; Hadoop Cluster; Hadoop Distributed File System; MapReduce Jobs

1 INTRODUCTION

Today a tremendous amount of data is being produced at great speeds from a large number of sources, and there are complex new data types to deal with, from sensory data to share market feeds in real time [3]. Web analysis andscientific exploration need large scale data processing including querying and analyzing [6].

In the recent past, clustering, has gained popularity because of its unsupervised nature and has found applications in information retrieval, medicine, business, etc. [7]. With this motivation, in order to handle the streaming data, an implementation of the ExCC algorithm (Section 2) is given in distributed environment using MR model (Section 3), which has been analyzed under different parameter setting using Forest CoverType and KDD Cup data (Section 4).

2 EXCLUSIVE AND COMPLETE CLUSTERING (ExCC)

The ExCC (Exclusive and Complete Clustering) Algorithm provides exclusive, complete and natural shaped clusters for mixed data, in a streaming environment. It is a grid based stream clustering algorithm where the bounded range of each dimension is divided into number of equi-width intervals called the granularity (g) of that dimension [2]. The Online phase processes the incoming data points and keeps the summarized statistics of the points received so far known as *synopsis*. The clustering is performed on the synopsis in the Offline phase whenever the number of points received in the synopsis, since last clustering, becomes equal to batchsize (pre-specified) [2]. The main steps of the algorithm are described in the following sections [4]:

2.1 SYNOPSIS MAINTENANCE

For each data point, the signature generated corresponds to a unique cell of a d-dimensional grid where d is the number of dimensions. For thei^{th}dimension, val_i is the value and the signature is $sig_i \in \{0, g_i - 1\}$ where g_i is the granularity and $1 \leq i \leq d$ [2]. If the ith dimension is of numeric type such that, Max_i be the upper limit and Min_i be the lower limit of the range of values, then

$$sig_i = \left\lfloor \frac{val_i * g_i}{Max_i - Min_i} \right\rfloor \quad (1)$$

If the ith dimension is of categorical type, then each of the possible values uniquely corresponds to a value $\epsilon \{0, g_i - 1\}$, which determines sig_i.

For each cell, the timestamp of the first point (t_1),timestamp of the last point (t_N), and the number of points, are also maintained, taking into consideration the *previous synopsis* (Section 2.2).

2.2 REMOVING NON-RECENT CELLS

Before clustering, the grid is pruned of cells, which have not been populated for a long period of time, as it is believed that no more point will come in the corresponding cell due to evolving nature of the data. Staleness of a cell is determined using its data speed.

Let C be a cell in the grid with count N and $t_{current}$ be the current time. Then for uniform speed of the data points, the *inter arrival time*, aat^C, is given as:

$$aat^C = \frac{t_N - t_1 + 1}{N} \quad (2)$$

The cell is pruned if and only if,

$$\frac{t_{current} - t_N}{batchsize} \geq aat^C \quad (3)$$

Copyright@IJICCT, Vol I, Issue II(July – Dec2013) : ISSN 2347-7202

[1,2]Department of Computer Science, University of Delhi, Delhi
Email: [1] abhishek.santra@gmail.com, [2]anuragagarwal90@gmail.com

This journal is cited as : **JIMS 8i-Int'l J. of Inf. Comm. & Computing Technology(IJICCT)**

Apart from forwarding the new synopsis containing recent cells to the next step, it is also used during the processing of next batch, as *previous synopsis* (Section 2.1).

2.3 SELECTING DENSE CELLS

After the pruning of stale cells, the sparse cells are pruned as they represent noise at the time of current clustering. Let N_1 be the sum of the number of points in all the cells obtained after pruning, L be the total number of recent cells, *G* be the average of all dimensions' granularity values and μ be the average number of points in the cell, i.e. $\mu = \frac{N1}{L}$. Then a cell C is dense if and only if the number of points in the cell (N) is greater than the *cell density threshold (Ψ)*, i.e.,

$$N > \Psi \qquad where, \Psi = \frac{\mu}{\ln(G+d)} \tag{4}$$

2.4 CLUSTERING

Connected Component Analysis (CCA) is used for clustering the recent dense cells by identifying adjacent regions/cells. If two cells have one edge or one corner in common, then they are adjacent and form a cluster.

Let $CS_1(sig_1, \ldots, sig_d)$ and $CS_2(sig'_1, \ldots, sig'_d)$ be two cells with respective cell signatures. CS_1 and CS_2 are connected if,

$$\forall_{i=1}^{c}\left(|sig_i - sig'_i| = 0 \bigvee |sig_i - sig'_i| = 1\right) \bigwedge \frac{1}{b}\sum_{i=1}^{b} \delta(sig_i, sig'_i) \leq \varepsilon \tag{5}$$

where,

c is the number of numeric attributes,
b is the number of categorical attributes
$such\ that, d = c + b$

$$\delta = 0, if\ sig_i = sig'_i$$
$$\delta = 1, if\ sig_i \neq sig'_i$$

ε *is the distance threshold parameter,* ϵ [0,1], *defined by the user*

Only significant clusters are reported. A cluster is said to be significant if the number of points in the cluster (N_c) is greater than the *cluster density threshold function* thus defined as:

$$N^c > \left\lceil \frac{\Psi' * (\ln G + \ln d)}{2 * \ln G * \ln d} \right\rceil \tag{6}$$

where Ψ' is the current cell density threshold based on average number of points per dense cell (Refer Eq. (4)).

3 IMPLEMENTING ExCC USING MAPREDUCE PROGRAMMING MODEL

The ExCC algorithm is modified to support incremental, batch processing of data using the MapReduce Programming Model as online processing of data stream is not feasible in a distributed environment as considerable time and cost is spent on fetching the resources for processing it.

In our MR implementation, the input data file is split into a number of batches. Each batch, having *batchsize* number of data points, contains the configuration and arrival time of the data points. In each iteration, the *significant clusters* and a *synopsis* containing recent cells, to be used during the processing of the next batch, are output. After all batches have been processed, we get the details of all the significant clusters i.e. clusters that constantly evolved over all the batches. The overall flow of this implementation is given in Figure 1.

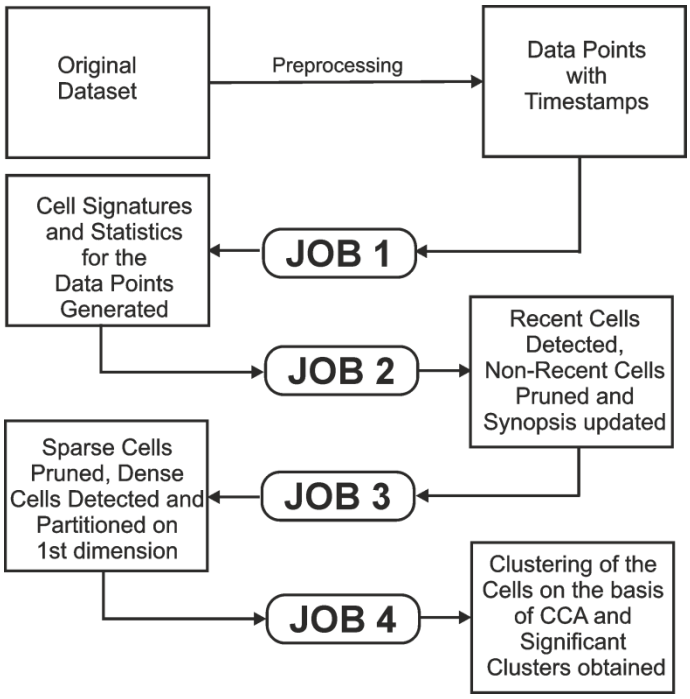

Figure 1: Overall Flow of Implementation

The following four consecutive MR Jobs are scheduled for the implementation.

3.1 JOB 1: CELL SIGNATURE AND STATISTICS GENERATION JOB

Job 1 generates cell signatures for each data point in the current batch (Section 2.1). For each cell C in the current batch, the *CellStatistics* calculated are t'_1 (timestamp of the first data point), t'_2 (timestamp of the last data point) and N' (number of data points).

3.1.1 MAP PHASE

Copyright@IJICCT, Vol I, Issue II(July – Dec2013): ISSN 2347-7202

The cell signatures are generated using configuration file which contains specifications like type, granularity and range for each dimension.

i. The input <key, value> pair is of the form, <*timestamp, CoordVector*>. *CoordVector* corresponds to [$val_0, val_1, \ldots, val_{d-1}$].
ii. The intermediate <key, value> pair is of the form, <*CellSignature, timestamp*>. *CellSignature*corresponds to [$sig_0, sig_1, \ldots, sig_{d-1}$].

3.1.2 REDUCE PHASE

The statistics for each cell obtained from the current batch are calculated.

i. The input <key, value-list> pair is of the form, <*CellSignature,* {list of *timestamp*s}>.
ii. The output <key, value> pair is of the form, <*CellSignature, CellStatistics*>.

3.2 JOB 2: SYNOPSIS MAINTENANCE AND RECENT CELL DETECTION JOB

Job 2 detects the recent cells and updates synopsis, taking into consideration the previous synopsis (Section 2.2).

3.2.1 MAP PHASE

The previous synopsis and the file containing cells of the current batch are merged and output to generate intermediate <key, value> pairs of the form, <*CellSignature, CellStatistics*>.

3.2.2 REDUCE PHASE

Using Eq. (3), recent cells are obtained. These are passed on to the next job for detection of dense cells and are also used in the next iteration as previous synopsis.

i. The input <key, value-list> pair is of the form, <*CellSignature,* {list of*CellStatistic*s}>.
ii. For every cell C (with unique *CellSignature*), the overall *CellStatistics* updated are t_1(timestamp of the first data point), t_N(timestamp of the last data point) and *N* (Number of data points).
iii. The recent cells, detected as intermediate <key, value> pairs, are of the form, <*CellSignature, CellStatistics*>.

3.3 JOB 3: DENSE RECENT CELL DETECTION AND PARTITIONING JOB

Job 3 identifies dense cells (Section 2.3) and then partitions them on the basis of adjacency with respect to the first dimension.

3.3.1 MAP PHASE

The dense cells are identified (using Eq. (4)) and processed so that the framework sorts them on the 1st dimension and directs them to a single reducer.

i. The input <key, value> pair is of the form, <*CellSignature, CellStatistics*>.
ii. For each dense cell detected, the intermediate <key, value>pair is of the form, <*FirstDim, SignandStat*>, where *FirstDim*is the firstdimension of the *CellSignature*, and *SignandStat*is the concatenation of *CellSignature* and *CellStatistics*.

3.3.2 REDUCE PHASE

The cells are partitioned on the basis of adjacency with respect to the first dimension.

i. The shuffle/sort phase sorts the dense cells on *FirstDim* and directs them to a single reducer, which leads to the input <key, value-list> pair of the form, <*FirstDim,* {list of*SignandStat*s}>.
ii. The cells adjacent with respect to the first dimension are assigned to the same partition. The output <key, value> pair is of the form, <*PartitionNumber, SignandStat*>.

3.4 JOB 4: CLUSTERING JOB

Job 4 performs the clustering on the dense recent cells using CCA and performs the statistical analysis of the significant clusters (Section 2.4).

3.4.1 MAP PHASE

An identity mapper is used where the <key, value> pair is of the form, <*PartitionNumber, SignandStat*>.

3.4.2 REDUCE PHASE

i. The input <key, value-list> pair is of the form, <*PartitionNumber,* {list of *SignandStat*s}>.
ii. We consider, TDR to be total dense recent cells

Algorithm 1 Algorithm used for clustering based on CCA

Require: Input: The Cell Signatures (CS) and corresponding Cluster Labels (CL)

1: **initialize**i and j to 1.
2: **repeat**
3: **if** the ith cell has not been assigned a cluster label (CL_i), **then** assign it a new label.
4: **repeat**
5: **if** the jth cell has not been assigned a cluster label (CL_j) **then**,
6: **if**CS_i and CS_j are adjacent, **then**$CL_j = CL_i$
7: **until** j ≤ TDR
8: **until**i ≤ TDR

iii. For each cluster CL, *ClusterStatistics* generated are N_c (total number of cells), N_{dp} (total number of data points), t_{first} (timestamp of the first data point) and t_{last} (timestamp of the last data point).

iv. The output <key, value> pair is of the form, <*ClusterLabel*, *ClusterStatistics*> for the significant clusters (Refer Eq. (6)).

After all the batches of data points have been processed, the statistics of the significant clusters, which evolved over the iterations, is obtained. Figure 2 depicts the execution of this implementation using an example.

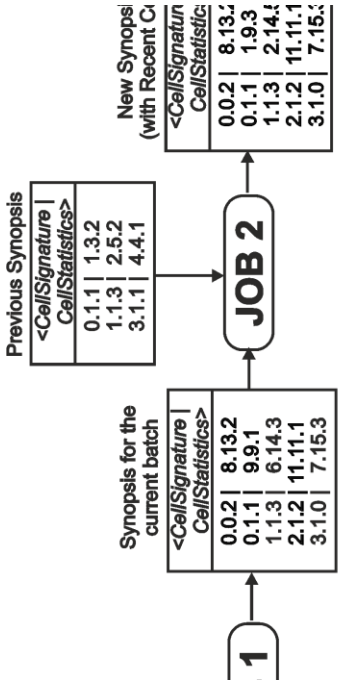

Figure 2: An Example running ExCC algorithm as a MapReduce Program

4 EXPERIMENTAL ANALYSIS

To perform the experiments, a four node Hadoop Cluster of ACER desktops (2GB RAM; 0-100Mbps bandwidth) configured with Apache Hadoop 1.0.4 is set up [5]. KDD-CUP'99 Network Intrusion Detection and Forest CoverType are the data sets used [1]. The complete description of these datasets is given in Table 1.

Dataset	Size (MB)	Number of Instances	Number of Attributes	Number of Numeric Attributes	Number of Categorical Attributes
Forest CoverType	76	5,81,012	54	10	44
KDD-CUP' 99 (Only Numeric Attributes)	604.9	48,98,431	34	34	0
KDD-CUP' 99 (Full Data)	708.8	48,98,431	41	34	7

Table 1: Dataset Description

Initially, the entire dataset is taken as a single batch to obtain best parameter setting w.r.t number of datanodes and block size. The results obtained upon execution are given in Table 2.

Dataset	Data Points	Cell Signatures	Dense Recent Cells	Clustered Data Points	Clust-ers	Signi-ficant Clust-ers	Optimal Execution Time
Forest CoverType	5,81,012	1,24,055	52,813	4,88,095	11	7	55min 2sec
KDD-CUP' 99 (Only Numeric Attributes)	48,98,431	46,591	2,983	47,50,474	213	49	3min 21sec
KDD-CUP' 99 (Full Data)	48,98,431	57,713	4,148	47,29,064	229	58	3min 49sec

Table 2: Dataset Statistics

4.1 VARIATION WITH NUMBER OF DATANODES

The performance of the implementation is analysed with respect to the number of datanodes taking 64MB as the block size. For the two data sets, the best execution times for the MR implementation are obtained with *four datanodes*, as shown in Figure 3 and Figure 4.

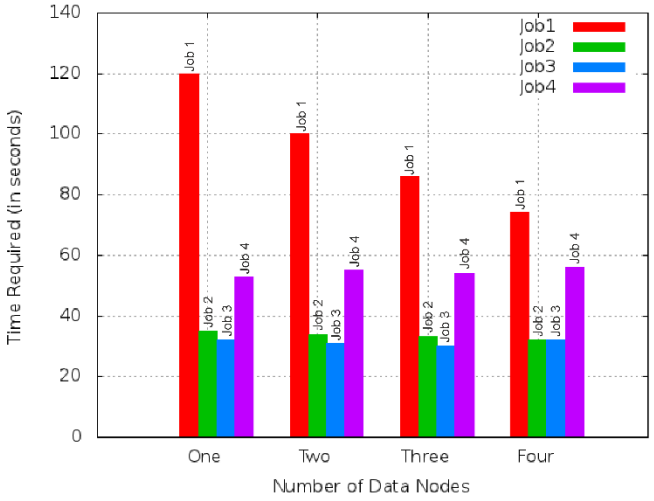

Figure 3: Variation with number of DataNodes for KDD-CUP' 99 Dataset (only numeric attributes)

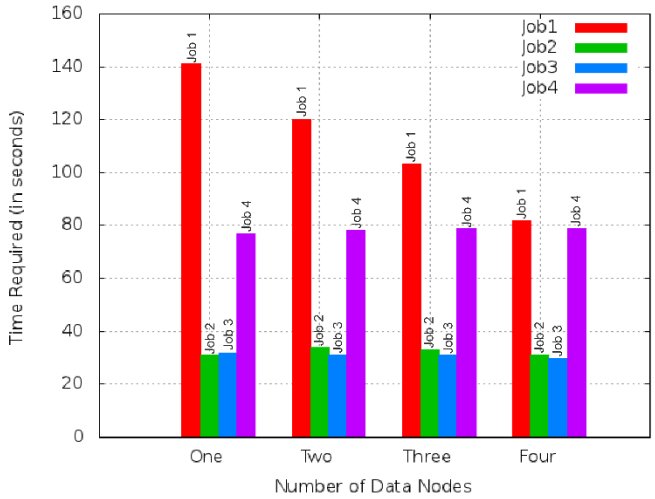

Figure 4: Variation with number of DataNodes for KDD-CUP' 99 Dataset (Full Data)

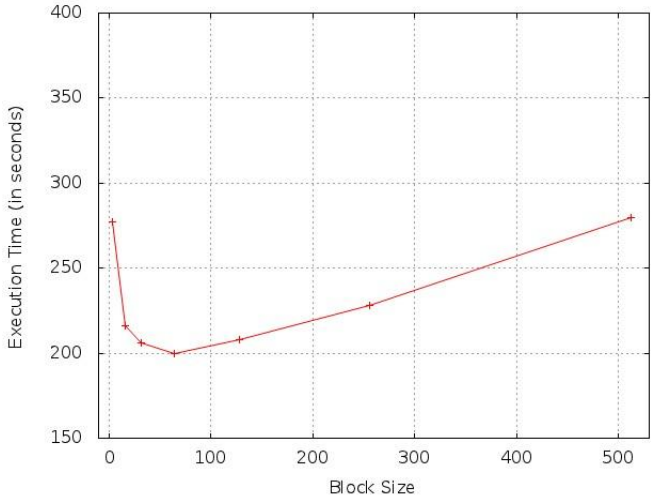

Figure 5: Variation with Block Size (in MB) for KDD-CUP' 99 Dataset (only numeric attributes)

Upon generation of cells after Job1, the size of the input to the subsequent jobs decreases and it goes below the block size leading to underutilization of cluster. Thus, variation in the number of datanodes effects Job 1, only, for which, on increasing the number of datanodes, more number of map tasks are executed in parallel and execution time decreases.

Also, in Job 4 the clustering technique is such thateach cell is compared against the remaining cells for adjacency. So, in general, more is the number of dense recent cells, more is the number of comparisons and thus, more is the execution time.

4.2 VARIATION WITH BLOCK SIZE

The performance of the implementation is analysed with respectto the data block size, in a cluster of four datanodes. For the two data sets, the best execution times are obtained for the MR implementation with an *optimal block size of 64 MB*, as shown in Figure 5 and Figure 6.

From Figure 5 and Figure 6, it is evident that there is a threshold/optimal block size, beyond which, on either side the execution time increases. On decreasing the threshold, the total waiting time of map tasks becomes a dominant factor, and on increasing the threshold, the speed of the block transfer becomes a dominant factor due to bandwidth hindrance.

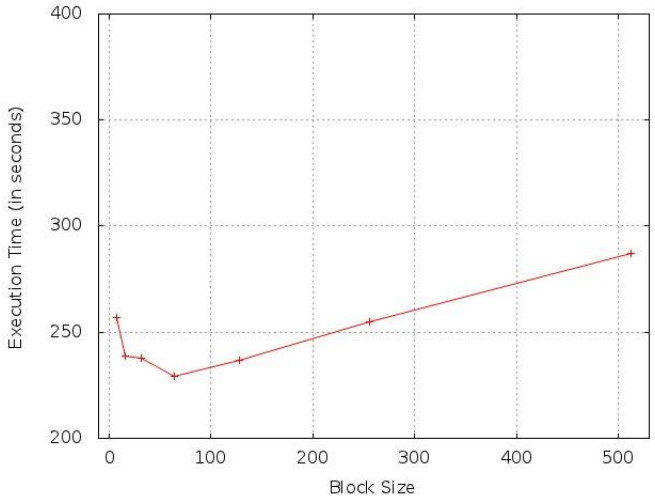

Figure 5: Variation with Block Size (in MB) for KDD-CUP' 99 Dataset (Full Data)

4.3 COMPARISON OF MR PROGRAM WITH JAVA PROGRAM

The ExCC algorithm was executed as an MR program on a cluster and as a sequential Java program on a single machine. The statistics obtained from both the implementations matched to the statistics stated in Table 2, except the Execution Time. For each dataset, the execution times obtained for the Java Program were *significantly* larger than the execution times for the MR implementation.

For finer settings i.e. larger values of granularity (g) of the attributes, the Java program reports an exception of *OutOfMemoryError* as the synopsis obtained becomes larger than the available memory. But, such a scenario is not encountered in case of the MR implementation, because the synopsis gets distributed over the cluster.

5 CONCLUSION AND FUTURE WORK

An MR implementation of ExCC on a multi-node Hadoop cluster is explained. Experiments have been performed to know the functional characteristics of ExCC in distributed environment under different parameter setting. The optimal parameter setting is a cluster of four datanodes with 64 MB block size.

In future, our MR implementation can be compared with similar implementations done on other platforms.

REFERENCES

[1] Uci machine learning repository.

[2] BHATNAGAR, V., KAUR, S., AND CHAKRAVARTHY, S. Clustering data streams using grid-based synopsis. KAIS (2013), 1–26.

[3] KARENA, C. Top 6 big data issues, 2012.

[4] KAUR, S., KHANNA, D., GUPTA, T., AND BHATNAGAR, V. Scalable clustering using pact programming model. In ICDM Workshops'12 (2012), pp. 424–430.

[5] NOLL, M. Running hadoop on ubuntulinux (multi-node cluster).

[6] S., C., AND S.W., S. Map-reduce meets wider varieties of applications.

[7] TAN P.N., S. M., AND V., K. Introduction to Data Mining. Pearson Education Inc., 2006.