Proceedings of the 2015 International Conference on Industrial Engineering and Operations Management
Dubai, United Arab Emirates (UAE), March 3 – 5, 2015

A massively parallel processing for the Multiple Linear Regression

Moufida Rehab Adjout[1]

[1] Laboratory LIPN-UMR 7030-CNRS, Paris 13University,
Av. J. B. Clement 93430 Villetaneuse, France
{rehab, boufares }@ lipn.univ-paris13.fr

***Abstract*— The amount of data generated by traditional business activities, has resulted data warehouses with a size up to petabytes. The ability to analyze this torrent of data will become the basis of competition and growth for individual firms by ever-narrower segmentation of customers, improvement of decision-making and unearth valuable insights that would otherwise remain hidden. For this purpose, the large size of data to be processed requires the use of high-performance analytical systems running on distributed environments. Because the data is so big it affects the types of algorithms we are willing to consider. Then standard analytics algorithms need to be adapted to take advantage of cloud computing models which provide scalability and flexibility.**

This work illustrates an implementation of a parallel version of the multiple linear regression. It can extract coefficients from large amounts of data, based on MapReduce Framework with large scale.

Parallel processing of multiple linear regression will be based on the QR decomposition and the ordinary least squares method.

Keywords: Data mining, Predictive analysis, Multiple linear regression, Big Data, Hadoop, MapReduce, Cloud Computing.

1. INTRODUCTION

Processing large volumes of data is at the heart of the emerging issues in research. This issue is especially important for learning supervised systems. They must build their models on large databases collected automatically cost and high speed which requires a large capacity evolutionary computation.

The continued collection of such data is meaningful only if the analysis and the decision can be made at short notice.

The objective is no longer only to understand phenomena through the data, but to optimize the treatment to get results in the time window where the analyzed information remains relevant. Although many supervised learning algorithms seem to be effective [8] (For example: the multiple linear regression, neural networks, naive Bayesian), which offer rich functionality for data analysis, but on limited amounts of data and operate entirely in main memory [].Thus, the massive data require two challenges in supervised learning. First, the massive data set commit memory and computational load for the most complex learning systems. It is therefore impossible to load this mass of data in memory. Second, the response time becomes unacceptable for analysis and learning of this volume of data due to, for example, competitive factors. Thus, the adaptation of these algorithms is essential to meet these challenges [8] [12]. In this work, our contribution is to show that the adaptation of classical learning algorithms of data is possible to provide a response to the phenomenon of BIG DATA. We focused particularly on the adaptation of the multiple linear regression with massive data processing. Multiple linear regression, is among the most powerful and mathematically mature method in data analysis [1]. Its principle treatment is focused on a central approach, where the computation is done on a set of data stored in a single machine. With an increasing volume of data, the transition to the scalability of the algorithm is indispensable. Multiple linear regression proves unsuited to the scalability of the data processed [4]. The use of parallel distributed computing with MapReduce paradigm seems like a natural solution to this problem. Recently the MapReduce technique has gained popularity in the scientific community because of its effectiveness in parallel processing. MapReduce is a programming model that enables distributed algorithms in parallel on clusters of machines with varied [5] features. The structure of the paper is organized as follows. The next section presents the basic concepts related to multiple linear regression. Then, we present the work that implements the MapReduce with QR decomposition (QR decomposition is one of the steps for the resolution of the method of multiple linear regression). We will present subsequently our scalable approach, based on the QR decomposition and the ordinary least squares method in an environment MapReduce. This first work, show an approach to parallelization of classical learning algorithms that can meet the challenges of BIG DATA. In our future work, we will build on this experience to extend this principle to other existing learning algorithms. Thus, it is possible through the performance measures and quality prediction of the new version of each algorithm, assess its relevance in the BIG DATA.

2. MULTIPLE LINEAR REGRESSION

2.1. Definition of multiple linear regression

"Multiple regression [1] is a classical statistical model that is used when one is faced with a dependent variable called "explain" or "endogenous", which seeks to understand the logic of distribution based on a set of independent variables called "explanatory" or "exogenous" reflecting observable

phenomena. It is possible to estimate this relationship statistically, from a series of observations, through the introduction of a random term related (stochastic term), to take into account the estimation errors"

2.2. General form of the multiple linear regression

This equation specifies how the dependent variable for a given observation y_k is connected to the explanatory variables x_{Ki}:

$$y_k = \beta_0 + \beta_1 x_{k1} + \beta_2 x_{k2} + \ldots + \beta_n x_{kn}\,, k := 1, \ldots, m$$

Where:

- y_k (k = 1,2, .., m) is the dependent or explained randomness variable;
- x_{ki} (i = 1,2, .., n) are the independent or explanatory variables measured without error (not random);
- $\beta_0, \beta_1, \beta_2, .., \beta_n$ are the parameters of the model;

Objective is to find $\beta_0, \beta_1, \beta_2 \ldots \beta_n$ so that the sum of squared errors is the smallest (minimum). The goal is to find the influence degree $\beta_0, \beta_1, \beta_2 \ldots \beta_n$ of these factors (x_{ki}) on the variable (y_k), where **m:** number of observations and **n:** number of variables,

2.3. Estimation of model parameters

2.3.1. The model

The matrix formulation of the multiple linear regression to calculate the coefficients as follows [3]:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{nm} \end{pmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_m \end{bmatrix}$$

We note:

$$\boldsymbol{y = \beta X} \quad \text{Where} \quad : \quad \boldsymbol{\beta = (X^T X)^{-1} X^T y} \qquad (\mathbf{1})$$

Where **m:** number of observations, **n:** number of variables, $\mathbf{X^T}$: Transposed matrix **X.**

2.3.2. The method of ordinary least squares

2.3.3. QR decomposition

QR decomposition [2] (also called QR factorization) is one of the most common decomposition matrices in scientific computing to solve the problems of ordinary least squares. The QR decomposition of a matrix X is a decomposition of the latter into an orthogonal matrix Q and an upper triangular matrix R. QR decomposition is a decomposition of X such that:

$$X = QR$$

With the orthogonal matrix Q (such as Q^T Q = I). If X is invertible, then this factorization is unique [6].

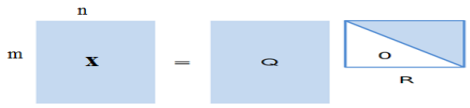


Fig. 1: The QR decomposition

2.3.4. Model estimation

To determine the model coefficients, the method is to simplify the calculation by decomposing the data matrix X into two matrices Q and R obtained with the QR decomposition, and thus:

In $\boldsymbol{\beta = (X^T X)^{-1} X^T y}$ $(\mathbf{1})$, replacing **X** by **QR**:

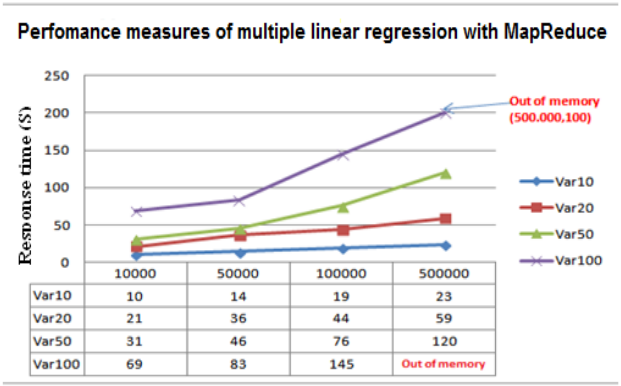
$$\boldsymbol{\beta = (R^T Q^T Q R)^{-1} R^T Q^T Y =}$$
$$\boldsymbol{(R^T R)^{-1} R^T Q^T Y = R^{-1} (R^T)^{-1} R^T Q^T Y = R^{-1} Q^T Y}$$

Are obtained:

$$\boldsymbol{\beta = R^{-1} Q^T Y} \qquad (\mathbf{2})$$

2.3.5. The boundaries of the multiple linear regression

Despite the contribution of multiple linear regression in predictive analytics, it presents a major effort with single processor's calculation prevents its scalability. Thus, it limits its use on small and medium data sets. In order to overcome these limitations, experiments were made by varying the number of observations (m) and the number of explanatory variables (n). The results show that a limit of m = 500,000 and n = 100 is reached quickly with single processor (used machine with 4 GB of RAM and a processor of 2.67 GHz). Using a more powerful server will certainly analyze a large volume of data but it will always be limited by its computational power. Indeed, if we double the power of the machine, we can't necessary double the volume. (Example: for 8GB RAM, it can not necessarily analyze a volume of m = 1.000.000, n = 200). This prevents having a robust and scalable system according to the load calculation input. The figure below can give an idea about the maximum volume of data supported by the multiple linear regression.



	10000	50000	100000	500000
Var10	10	14	19	23
Var20	21	36	44	59
Var50	31	46	76	120
Var100	69	83	145	Out of memory

Fig. 2: Measurements and performance tests of the rise in the scale for the multiple linear regression

To overcome these limitations and adapt the multiple linear regression for this huge amount of data, we present a new computational approach. It will be massively parallel to absorb this mass and increase the performance.

3. *Related Work*

One of the first items that explicitly discusses the QR decomposition with MapReduce architecture was written by Constantine and Gleich [2].They present an algorithm for calculating the QR decomposition of matrices for large sizes (up to m = 500 million lines, n = 100 columns) with MapReduce architecture [2]. Their method of QR decomposition with MapReduce provides only a quick way to calculate R. To calculate Q, they used $Q = XR^{-1}$ proved later that numerically unstable calculation and Q cannot be at the final truly orthogonal [3].

A second approch proposed for the matrix Q is described below [2].We will present their Limits with an alternative to have a final Q with QR decomposition approach. However, the method of decomposition to compute Q and R factors follows a series of treatments that consumes a lot of memory. The goal is to distribute these operations in parallel architectures.

To understand the proposed methodology, let us take an example: Considering the matrix X [5], where X is a large matrix (m> 500.000, n> 100), X is divided into small blocks X_i with size (m_i, n_i). The decomposition is done in two steps:

$$\mathbf{X_{(m,n)}} = \begin{bmatrix} \mathbf{X_1} \\ \mathbf{X_2} \\ \mathbf{X_3} \\ \mathbf{X_4} \end{bmatrix} \qquad where: \ X_{i\,(m_i,n_i)}$$

Where: $m_i << m$ and $n_i << n$

- ✓ **The first step**: represents the "map" function that calculates the local QR decomposition for each X_i.

$$\mathbf{X} = \begin{pmatrix} \mathbf{X_1} \\ \hline \mathbf{X_2} \\ \hline \mathbf{X_3} \\ \hline \mathbf{X_4} \end{pmatrix} = \begin{pmatrix} Q_1R_1 \\ \hline Q_2R_2 \\ \hline Q_3R_3 \\ \hline Q_4R_4 \end{pmatrix} \qquad \text{Where: } X_i = Q_i R_i;$$

- ✓ **The second step**: represents the "Reduce" function. It consists to build with the intermediate matrices R_i, an input matrix to the "Reduce" function, and a single QR decomposition is calculated for this time:

$$\begin{pmatrix} R_1 \\ \hline R_2 \\ \hline R_3 \\ \hline R_4 \end{pmatrix} = \begin{pmatrix} Q_1^2 \\ Q_2^2 \\ Q_3^2 \\ Q_4^2 \end{pmatrix} R$$

In the end, the result of the QR decomposition is:

- R: the final triangular matrix R.
- Q: The two matrices multiplication of the two previous steps gives the final matrix Q.

$$Q = \begin{pmatrix} Q_1 & & & \\ & Q_2 & & \\ & & Q_3 & \\ & & & Q_4 \end{pmatrix} \begin{pmatrix} Q_1^2 \\ Q_2^2 \\ Q_3^2 \\ Q_4^2 \end{pmatrix} = \begin{pmatrix} Q_1Q_1^2 \\ Q_2Q_2^2 \\ Q_3Q_3^2 \\ Q_4Q_4^2 \end{pmatrix}$$

$$\text{Where:} \begin{cases} \text{A with: } m_A = m_i * \text{nbBloc} \,;\, n_A = n_i * \text{nbBloc} \\ \text{B with: } m_B = m_i * \text{nbBloc};\ n_B = n_i \end{cases}$$

We have implemented and tested the method presented in [2]. This allowed us to highlight the limitations of this approach. Indeed, given the size of the new matrix A (m_i * nbBloc , n_i * nbBloc). It was found that multiplies A by B with certain size causes memory saturation and prevents the calculation of the final Q. The multiplication of A by B did not benefit to parallelism. So the idea is to also parallelize the multiplication for each pair (Q_i, Q'_i) and multiply them in a "map" task to accelerate and streamline the process of calculation. This will overcome the constraint of limited memory and the impossibility calculation from a certain size (depending on the power of the machine).

4. *Calculation the coefficients β_i of multiple linear regression*

The calculation of the multiple linear regression using the QR decomposition and the method of ordinary least squares (OLS) [7] with MapReduce will be on two iterations to optimize the process and reduce the computational load. The algorithm takes as parameters the Block size to be used to decompose the matrix X and distribute it on several tasks of "map".

Thus the block number will be generated: $nbBloc = \frac{m}{Bloc\ size}$

- ✓ **First iteration**

The first iteration takes as input the matrix of observations X (m, n) and decomposes it on several matrices X_i (Bloc size, n). The results matrices Q_i (Bloc size, n) and R_i (n, n) are associated with the key «Key_i » (i: equal to nbBloc) and sent them to "reduce".

Thus, the "reduce" will receive two types of matrices Q_i and R_i:

- Each pair (Key_i, Q_i) is stored in the output file of iteration 1.
- Each R_i will be used to construct the matrix R_{temp} (n * nbBloc, n) by superposing the matrices R_i. At the conclusion of treatment, the QR decomposition is applied to the matrix R_{temp}. Note the result Q' and R_{final}:
  - R_{final} will be saved in the output file of the iteration with the key "R".
  - Q is decomposed into several Q'_i matrices. Each Q'_i is associated with the key «Key_i » (i: equal to nbBloc) and stored them in the output file of the iteration.

Thus at the end of the first iteration, the output file contains:

- Couples (Key_i, Q_i)
- Couples (Key_i, Q'_i)
- Couples (« R », R_{final}).

This treatment highlights the fact that each "map" task load into memory at maximum a matrix with size (Bloc size, n) which significantly reduces the risk of "out of memory". Similarly the task "reduce" will process a maximum an array at a time with size $(n * nbBloc, n)$. Thus, the choice of the number of blocks must be considered depending on the size of the machines in the cluster. The increase in computing power will be naturally by adding new machines to the cluster to take advantage of the MapReduce's parallelism.

✓ **Second iteration**

The second iteration takes as input the result of the first iteration and the vector y. In the step of "map", the vector y is decomposed into several vector y_i (Bloc size) and sent to reduce with "Key_i ". The task "reduce" will execute the input data according to the associated key:

- If the key is "R" then R_{final} is saved and will be used in the calculation of β at the end of treatment
- Otherwise we will need necessarily both matrices Q_i , Q'_i and y_i vector to be used in the calculation of the vector V_i as follows:

  Q: Multiply (Q_i, Q'_i)
  Q^T: Transpose (Q)
  V_i: Multiply (Q^T, y_i)

At the end of treatment, the V_i vectors are added to have the final vector V.

$$\beta := \text{Solve}(R_{final}, \sum Vi[\,])$$

In this iteration, the task "reduce" will process a multiplication of two matrices with size (Bloc size, n) and (n, n) which limits the necessary computing power.
The following section presents the algorithm with a diagram that illustrates our proposed approach.

4.1. Algorithm

Iteration 1:

Input : matrix X

For all row of X Do
  X_i :=Blockfactorize(X,BlockSize)

  For all block X_i of X
    *(Q_i, R_i) = **Map1(X_i)***
    Emit(key_i, Qi)
    Emit(key_r, Ri)
  End for

*(Q'_i, R_{final}) = **Reduce1(Key_r, [R_1,R_2,..,R_i])***
  With i:= 1,…,nbBlocks

  For all block Q'_i of Q' Do
    Emit(key_i, Q'_i)
  End For
  Emit("R", R_{final})
End For

Function Map1(X_i)
  Input : block X_i of matrix X
Begin
  (Q_i, R_i):= QRfactorize(X_i) ;
  Output : Q_i , R_i
End

Function Reduce1(Key_r, [R_1,R_2,..,R_i])
  Input : R_{temp} = Matrix [R_1, R2,.., R_i]
Begin
  (Q', R_{final}) := QRfactorize(R_{temp}) ;
    For all row of *Q'* Do
      Q'_i := Decompose (Q', BlockSize);
    End For
  Output : Q'_i ; *With i:=1...nbBlocks*
    R_{final}
End

Iteration 2:

Input : ListQi:= List [key$_i$, (Q_i , Q'_i)]
Vector y
Matrix R_{final}

For all row of y Do
  y_i := ***Blockfactorize(y, tailleBlock)***
End For

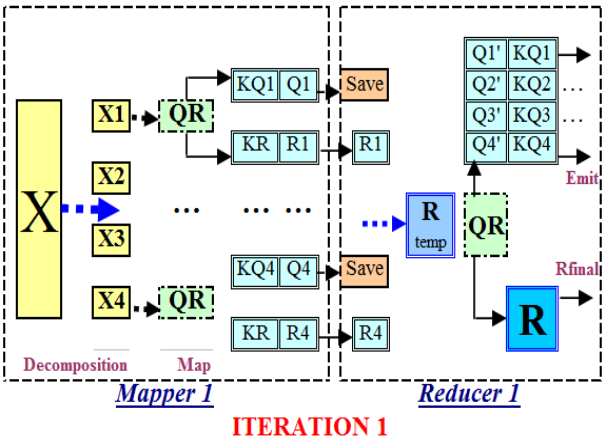
y_i := **Map2(y_i)**
With i:= 1,…,nbBlocks

For all block *i*
  $V_i[\,] =$ ***Reduce2***(key_i , ***List*** [Q_i , Q'_i, y_i])
End For
  $\boldsymbol{\beta}\,[\,] := Solve(R_{final}, \sum Vi[\,])$

Function Map2(y_i)
Input : y_i
Begin
  Emit(key_i, y_i) ;
End

Function Reduce2 (key$_i$, **List** [Q_i , Q'_i, y_i])
Input : List [Q_i , Q_i, y_i]
Begin
  Q := Multiply (Q_i, Q'_i)
  Q^T :=*Transpose (Q)*
  V_i = Multiply (Q^T, y_i)
  Output: V_i[ ] ;
End

4.2. Schematic description of the approach



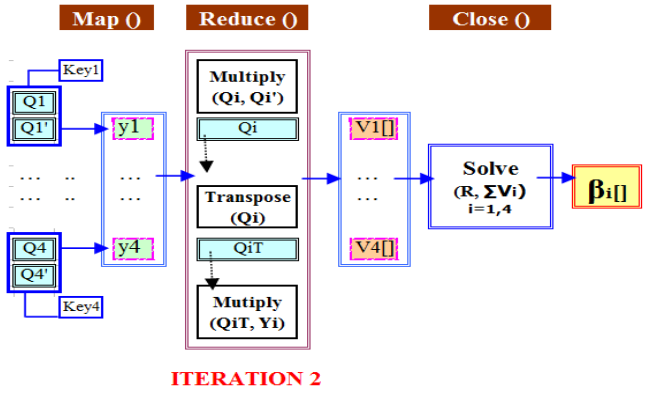


Fig. 3: The approach of multiple linear regression with MapReduce

5. *Experimentation*

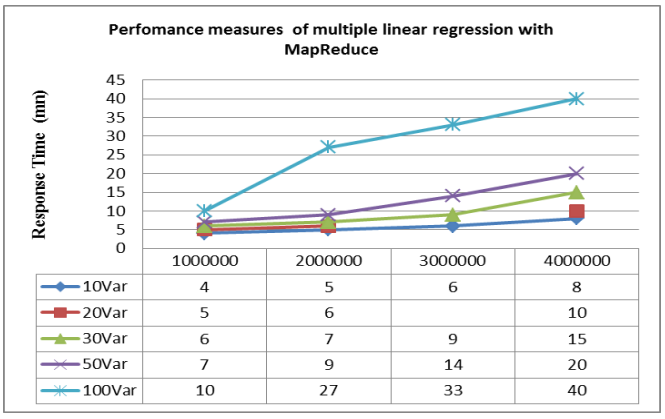
To test our approach, we used a java implementation for the new algorithm on Hadoop Version 2.4.2. We tested the program on a cluster Amazon Web Services (AWS) using Elastic Map Reduce (EMR) service.

5.1. Description cluster

The cluster implemented on AWS is composed of:

- ✓ A small machine type EC2 plays the role of "master".
- ✓ Two small machine type EC2 (1.7 GB with a single Core), which play the role of "slave".
- ✓ The version of Hadoop cluster installed is version 2.4.2 (latest version installed).
- ✓ Using Elastic MapReduce service (EMR) to start a Hadoop cluster with the number of machines desired.

The performance measures are described in the following diagram:



	1000000	2000000	3000000	4000000
10Var	4	5	6	8
20Var	5	6		10
30Var	6	7	9	15
50Var	7	9	14	20
100Var	10	27	33	40

Fig. 4: Performance measures of multiple linear regression with MapReduce.

6. *Discussion*

The choice of a low power cluster in this experiment will demonstrate the robustness of the algorithm and its ability to handle large volumes of data regardless the size of the machines in the cluster. This is possible by distributing the calculation on multiple tasks (machines) "map" and the optimization of the algorithm throughout the steps work only with matrices of limited size. Indeed, this algorithm takes as input a matrix (m, n) will not be charged fully in the memory which avoids any risk of "out of memory". The performance measurements show that the limit of memory (out of memory) is not achieved even with machines 1.7GB of RAM. This new version has exceed the volume of 4 million observations with 100 variables (it was limited to 500,000 observations traditional algorithm).

7. *Conclusion*

In this work, we have illustrated the implementation of a new parallel algorithm for multiple linear regression able to extract its coefficients with massive data. With a distributed algorithm based on the MapReduce paradigm, we managed to increase processing performance by avoiding the memory limits. This mechanism providing scalability and therefore need to increase computing capacity in contrast to the multiple linear regression in not distributed environment. The experiments carried out on an increasing volume confirmed these assumptions. Further improvements are possible by adding a third iteration, allowing you to run multiple "reduce" tasks instead of one made in the second iteration. Also, a semantic evaluation of β_i calculated is needed to judge the relevance of the regression when the number of observations becomes large. With this deluge of information, adapting the methods of data mining in Big Data presents a significant challenge to follow. Indeed, as the volume of data increases, the algorithms used to exploit these data must keep pace, otherwise the information potential of the data may be lost.

REFERENCES

[1] S.Tufféry (2010). Data Mining et statistique décisionnelle, Éditions Technip, nouvelle édition revue et enrichie, pp 10-16.

[2] P.Constantine, D.Gleich (2011). Tall and skinny QR factorizations in mapreduce architectures. Proceedings of the second international workshop on MapReduce and its applications, pp 43-50.

[3] N.Marz, J.Warren. Big Data: Principles and best practices of scalable realtime data systems. Manning Publications, 2013.

[4] A.Benson, F.Gleich, J.Demmel (2013). Direct QR factorizations for tall-and-skinny matrices in MapReduce architectures pp 44-45.

[5] A.Rajaraman, J.Leskovec, D.Ullman (2010). Mining of Massive Datasets, pp 4-7.

[6] J.Dean, S. Ghemawat. MapReduce: Simplified data processing on large clusters. In Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI2004), pp 137–150.

[7] L.Breiman, J.Friedma (1985).Estimating Optimal Transformations for Multiple Regression and Correlation. Journal of the American Statistical Association, pp 580-581.

[8] P.Foster, A.Stine (2004), Variable selection in data mining: building a predictive model for bankruptcy," Journal of the American Statistical Association, pp 303-313.

[9] A. Bifet, E. Frank (2010). Sentiment knowledge discovery in Twitter streaming data. In Proc 13th International Conference on Discovery Science, Canberra, Australia, pp 1–15.

[10] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer (2010). MOA: Massive Online Analysis http://moa.cms.waikato.ac.nz/. Journal of Machine Learning Research (JMLR).

[11] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, J. M. Hellerstein (2010). Graphlab: A new parallel framework for machine learning. In Conference on Uncertainty in Artificial Intelligence (UAI), Catalina Island, California.

[12] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. ANF: a fast and scalable tool for data mining in massive graphs. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada, pp 81–90, 2002.

[13] D.boyd, K.Crawford (2012). Critical Questions for Big Data. Information, Communication and Society, pp 662-679.

[14] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C.Guestrin, J. M.Hellerstein (2010).Graphlab: A new parallel framework for machine learning. In Conference on Uncertainty in Articial Intelligence (UAI), Catalina Island, California.

[15] C. Parker (2012). Unexpected challenges in large scale machine learning. In Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, BigMine '12, pp 1-6, New York,NY, USA.