Sample IEEE Paper for US Letter Page Size

First Author [#1], Second Author [*2], Third Author [#3]

[#] *First-Third Department, First-Third University*
Address Including Country Name
[1] first.author@first-third.edu
[3] third.author@first-third.edu

[*] *Second Company*
Address Including Country Name
[2] second.author@second.com

***Abstract*—For your paper to be published in the conference proceedings, you must use this document as both an instruction set and as a template into which you can type your own text. If your paper does not conform to the required format, you will be asked to fix it.**

I. Introduction

The frequency of data access at the users' end has been increased by a large number for the past few years. To ensure low latency at the users' end, it is preferable to reside the data as close to the users as possible. Moreover, in present world data privacy has been a great concern and almost all the nations require the data of their citizens not to reside in some place across their national border. Now its a big concern to maintain privacy regulations and keep foreign countries from being able to subpoena data. To mitigate these two issues, almost all the companies have been building their data centers all around the world. As a result, the sparsity of data has been increased by a huge amount during the last few years.

On the other hand, the amount of data generating in present world is quite large. This huge volume has introduced a new term "Big Data". In general "Big Data" is nothing but the large volume of data that can be both structured and unstructured. The excessive use of social media, scientific instruments, portable devices, sensors are the main reason behind the generation of big data. Data analysis, capture, curation, search, share, storage, transfer, visualization, query, updating, information privacy everything associated with big data has been the challenges of the present world. In case of big data the can be both wide and tall that means the dimension of the data can be quite large. As a result available techniques might not be able to perform the data analysis with the provided limited hardware. Therefore, we need approaches that have the property of scalability.

Under these circumstances, we can see that in case of extracting some information from the existing data, we may have to cover a good number of data centres located at different geographical area. This has given the introduction of a new data analysis approach, "Distributed Data Analysis". Therefore we get the idea of two approaches of data analysis.

A. Centralized Approach

The centralized approach to data analysis from distributed data centers is to centralize them first. As shown in Figure 1, this involves a two different steps:

1) **Centralizing step** Data from various data centers are copied into a single data center. It involves recreation of data of all data centers in one location.
2) **Analysis Step** Process of extracting the necessary information from the centralized data takes place in that single data center. Here traditional intra data center technology is sufficient for the analysis purpose.

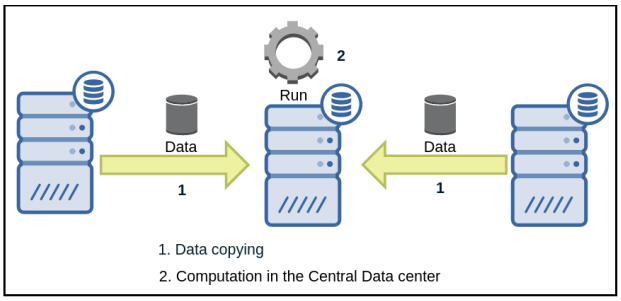

Fig. 1. Centralized Approach

This centralized approach is predominant in most practical settings. There are mainly two reasons behind its popularity.

1) There are lots of frameworks that have already been established for centralized learning approach. That is why centralizing the data is the easiest way to reuse existing data analysis frameworks [1], [2], [3]
2) Learning algorithms are highly communication intensive. Thus it is assumed that they will note be properly responsive to cross data center execution.

For these reasons, this centralized approach is consistent with reports on the infrastructures of other large organizations, such as Facebook [4], Twitter [5], and LinkedIn [6].

However the centralized approach has two shortcomings.

1) While making multiple copy of data at the central data center, it consumes a good amounts of inter data

center bandwidth. Since inter data center bandwidth is expensive, it is not easy to increase it according to the necessity [7], [8], [9], [10].

2) While creating copy of data, it may be a case that data is crossing national borders. However, in current workd data sovereignty is a growing concern that might create a big limitation in this aspect [11], [12].

B. Distributed Approach

In the distributed approach, raw data is kept in their corresponding data centers. Every data center does a portion of the execution that is only on the data of that data center. The final analysis takes pales by passing small amount of information among the data centers.

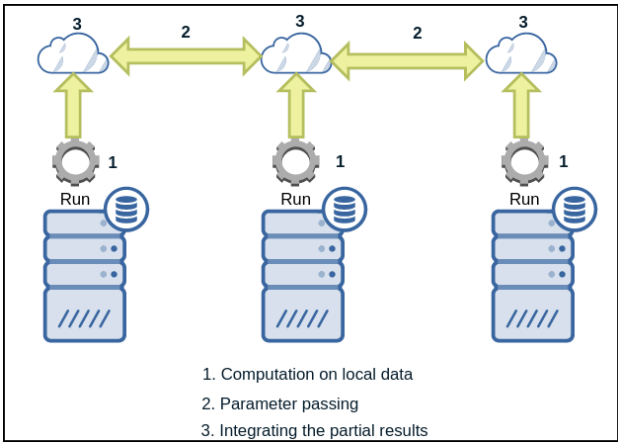

Fig. 2. Distributed Approach

So according to Figure 2, we can see this approach includes three steps:

1) **Local Computation** Whenever the command of starting of any learning process is issued every data center start a partial computation on its own data. They create necessary high level information on data that will be needed in the final computation.
2) **Parameter Passing** Data centers communicate among themselves and share valuable information.
3) **Final Computation** By integrating the partial results data centers make the final computational model.

In this way distributed solutions can achieve much lower cross data centers bandwidth utilization, and thus substantially lower cost for large-scale analysis tasks. As the current world has got a concern about 'Big Data' and 'Data Sovereignty', the distributed approach seems to be more efficient.

II. PCA as Data Analytic Approach

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components [13]. The number of principal components is less than or equal to the smaller of the number of original variables or the number of observations. This transformation is defined in such a way that the first principal component has the largest possible variance and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. As we can use PCA for dimensionality reduction, it can be used as a preprocessing step in many machine learning algorithms that do not perform well with high-dimensional data. Therefore we can easily realize that PCA is an excellent tool for big data analysis.

There are several ways to perform PCA. Eigen Value Decomposition (EVD) of covariance matrix and Singular Value Decomposition (SVD) are two basic and most common ways [14]. These methods usually perform better on small datasets on a single machine. But in distributed settings and for big data, they introduce a new set of difficulties; they do not scale well to high dimensional data and are inefficient in terms of computation and communication cost [15]. To overcome these difficulties, two other methods, Stochastic SVD (SSVD) [16] and Probabilistic PCA (PPCA) [17] are used in practice. But unlike the other methods to calculate principal component (i.e. EVD of covariance matrix, SVD, SSVD), PPCA has two important advantages. It has the ability of handling missing data and calculating probabilistic model to generate synthesized data [18]. These features are best suited for big data as missing values are prevalent in this case and a probabilistic model of the data will be more effective for analytical purposes.

Despite of being an excellent tool of data analysis, we lack in proper approach of PPCA in big data analysis. We saw a decent approach of making scalable PCA or *sPCA* in a distributed environment [15]. It was designed for data analysis on large datasets on distributed commodity clusters. But unfortunately this approach is not capable of handling datasets with very large dimension, we can say data that is at the same time tall and wide. Moreover data sovereignty is not ensured. There was no method of performing PCA on big data located at different geographic location which involves independent computation at different clusters and accumulation of intermediate results.

In this paper, we explore these two issues in depth. We will propose a solution which is capable of handling *Tall and Wide* big data and at the same time can perform on geographically distributed datasets where there is no provision of passing raw data across the data center.

The rest of this paper is organized as follows. In Section II, we discuss the background of PPCA showing its valuable advantages as well as some limitations. In Section III, we analyze already established approach *sPCA*. We present our proposed implementation of PPCA on geographically distributed *Tall and Wide* big data in Section IV. We justify our contribution in Section V. Section VI presents some of the properties of our approach and Section VII presents our experimental evaluation. Finally, Section VIII concludes the paper.

III. Our Focus

In current world, the rate of generation of data is quite large. We live in an age of Big Data and Internet of Things (loT). The term Big Data is not really an overstatement. Numerous web services share and collect data of huge scale, typically in terabytes range, and the data includes various aspects of users, for example, clicks, visits, likes, shares, comments, reviews, ratings, tweets, photos, videos etc. Apart from these web services, big data is also generated by countless sensors all around the globe to gather valuable information about respective fields. For example, closed circuit cameras recording is a good source of big data. In a nutshell, every electronic device, Internet services, embedded system and sensor the globe produce and transmit data and big data is being generated by these multiple sources at an alarming velocity, volume and variety.

For researchers, however, big data brings new sets of challenges like how to efficiently and effectively handle big data in commodity computers, how to apply conventional algorithms, how to properly manage big data in distributed setting etc. Specially, in field of machine leaning and Pattern recognition, big data poses a threat, because the conventional algorithms were designed solely to fulfill its purpose where entire dataset can fit in memory. Distributed setting was not taken into consideration. In nutshell, to extract any meaningful insight from this big data, we need powerful tools to analyze it, elastic frameworks to cope up with its sheer volume and most importantly we have to rethink and redesign existing algorithms which are most suitable for smaller sized data-set and do not take advantage of clusters.

In this chapter we will discuss the present issues that we are going to take into consideration in our research. We will indicate the challenges we targeted and in later chapters we will give our proposals in order to minimize these challenges in data analysis.

A. Tall and Wide Data

In general, the width of data means the number of features that each data sample is comprised of. In present world, this features count is increasing at large scale. The feature count or width of data sample can also be named as "Dimension of Data". One of the most effective use of PCA is dimensionality reduction. Many machine learning algorithms are not capable of handling data with large dimension. Therefore, we can use PCA to reduce the width of data then fit them in some kind of machine learning model. This present some kind of new challenge. The present PCA algorithms are not that much capable of handle very wide data that is tall also. Therefore before making the data suitable for machine learning models by dimensionality reduction, the PPCA algorithm has to deal with the curse of tall and wide data. The present implementation of *sPCA* done good number of optimization to make efficient use of memory. Still it is not capable of handling tall and wide data. The memory overflow occurs while computing the intermediate results.

B. The Blessings and Curses of Geo Distributed Data

The data generation at different geographic location has made the data geographically distributed. Data is being generated mainly at the location close to the end user. There are several advantages of such geographically distributed data such as (1) Assurance of faster data access to the end user (2) Protection of national privacy and data sovereignty (3) Sparsity of data center of the same company (4) Advanced development in business sector.

Despite of having some advantages, geographical distribution of data posses some kind of challenges in data analytic. It creates a scenario where there will be no global view of data during any kind of data analysis. This demands a new technology in data the field of data analysis. Moreover in case of breakdown of any particular data center, the data analysis technique has to be able to handle missing data. Data clusters located at different location might not have the same kind of configuration i.e. the clusters might be heterogeneous.

Taking these challenges, the data analysis algorithm needs to be capable of performing partial analysis on the tall and wide data at every individual data clusters and creates some kind of intermediate results and then needs to accumulate them in an time efficient approach to produce the final results.

IV. Background

We will discuss some background of PPCA according to [25]. We can obtain a probabilistic formulation of PCA by introducing a Gaussian latent i.e. unobserved variable mode. This kind of model is highly related to statistical factor analysis. A latent variable model seeks to relate a D-dimensional observation vector $\boldsymbol{y}$ to a corresponding d-dimensional vector of latent variables $\boldsymbol{x}$ where the relationship is linear:

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{\mu} + \boldsymbol{\epsilon} \tag{1}$$

Where:

- $\boldsymbol{y}$ is a D dimensional observed data vector
- $\boldsymbol{x}$ is a d dimensional latent variable
- $\boldsymbol{W}$ is a $D \times d$ transformation matrix
- the columns of $\boldsymbol{W}$ are the principal components
- $\boldsymbol{\mu}$ is the dimension wise mean vector of $\boldsymbol{y}$. This parameter allows the data to have non zero mean
- $\boldsymbol{\epsilon}$ is a D-dimensional zero-mean noise variable
- Here $\boldsymbol{x}$, $\boldsymbol{\epsilon}$, $\boldsymbol{y}$ are normal distributed i.e. Gaussian distributed

$$\begin{aligned}\boldsymbol{x} &\sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}) \\ \boldsymbol{\epsilon} &\sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\sigma^2 I}) \\ \boldsymbol{y} &\sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{W W}^T + \boldsymbol{\sigma^2})\end{aligned}$$

  where $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{u}, \sum)$ denotes the Normal distribution with $\boldsymbol{u}$ mean and $\sum$ covariance matrix.

The motivation is that, with $d < D$, the latent variables will offer a more parsimonious explanation of the dependencies between the observations. The model parameters may thus be

determined by maximum-likelihood, As there is no closed-form analytic solution for finding $\boldsymbol{W}$ and $\boldsymbol{\sigma}^2$, their values must be obtained via an iterative procedure.

The use of the isotropic Gaussian noise model $\mathcal{N}(\mathbf{0}, \boldsymbol{\sigma^2 I})$ for in conjunction with equation (1) implies that the $\boldsymbol{x}$-conditional probability distribution over $\boldsymbol{y}$-space is given by:

$$\boldsymbol{y}|\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{\mu}, \boldsymbol{\sigma}^2 \boldsymbol{I}) \tag{2}$$

With the marginal distribution over the latent variables also Gaussian and conventionally defined by $\boldsymbol{x} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, the marginal distribution for the observed data $\boldsymbol{y}$ is readily obtained by integrating out the latent variables and is likewise Gaussian:

$$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{C}) \tag{3}$$

where the observation covariance model is specified by $\boldsymbol{C} = \boldsymbol{W}\boldsymbol{W}^T + \boldsymbol{\sigma^2}$. given N observations $\{\boldsymbol{y_n}\}_1^N$ as the input data, the log likelihood of data is given by:

$$\begin{aligned}\mathcal{L}(\{\boldsymbol{y_r}\}_1^N) &= \sum_{n=1}^{N} \ln p(\boldsymbol{n_r}) \\ &= -\frac{1}{N}\{D * \ln(2\pi) + \ln|\boldsymbol{C}| + tr(\boldsymbol{C}^{-1} * \boldsymbol{S})\}\end{aligned} \tag{4}$$

where $\boldsymbol{S}$ is the sample covariance matrix of data $\boldsymbol{\{y_r\}}$ given by $\boldsymbol{S} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{y_r} - \boldsymbol{\mu})(\boldsymbol{y_r} - \boldsymbol{\mu})^T$ and $tr(\boldsymbol{M})$ is the trace of matrix $\boldsymbol{M}$

According to [25], the likelihood of $\boldsymbol{W}$ and $\boldsymbol{\sigma}$ is maximized when:

$$\boldsymbol{W}_{ML} = \boldsymbol{U}_d\sqrt{\boldsymbol{\Lambda}_d - \sigma^2 \boldsymbol{I}}\boldsymbol{R} \tag{5}$$

It may also be shown that for $\boldsymbol{W} = \boldsymbol{W}_{ML}$, the maximum-likelihood estimator for σ^2 is given by:

$$\sigma^2_{ML} = \frac{1}{D-d}\sum_{i=d+1}^{D}\lambda_i \tag{6}$$

A. An EM Algorithm for Probabilistic PCA

In the EM approach to maximising the likelihood for PPCA, we consider the latent variables $\{\boldsymbol{x}_n\}$ to be 'missing' data and the 'complete' data to comprise the observations together with these latent variables. The corresponding complete-data log-likelihood is then:

$$\mathcal{L}_C = \sum_{n=1}^{N} \ln p(\boldsymbol{y}_n), \boldsymbol{x}_n) \tag{7}$$

In order to make the EM algorithm look simple we define:

$$\begin{aligned}\boldsymbol{E}_n &= \langle \boldsymbol{x}_n \rangle \\ \boldsymbol{F}_n &= \langle \boldsymbol{x}_n \boldsymbol{x}_n^T \rangle \\ \boldsymbol{Y}_m &= (\boldsymbol{y}_n - \boldsymbol{\mu}) \\ Frob(\boldsymbol{Y}_m) &= ||\boldsymbol{y}_n - \boldsymbol{\mu}||^2\end{aligned}$$

Therefore, we can write the EM steps with simplified notations:

$$\textbf{E-step:} \quad \boldsymbol{E} = (\boldsymbol{Y}_m)\boldsymbol{W}\boldsymbol{M}^{-1} \tag{8}$$

$$\textbf{E-step:} \quad \boldsymbol{F} = \boldsymbol{E}^T\boldsymbol{E} + \sigma^2\boldsymbol{M}^{-1} \tag{9}$$

$$\textbf{M-step:} \quad \widetilde{\boldsymbol{W}} = (\boldsymbol{Y}_m^T\boldsymbol{E})\boldsymbol{F}^{-1} \tag{10}$$

$$\textbf{M-step:} \quad \sigma^2 = \frac{1}{ND}\left[Frob(\boldsymbol{Y}_m) - 2\sum_{n=1}^{N}\boldsymbol{E}_n^T\widetilde{\boldsymbol{W}}\boldsymbol{Y}_{mn} + tr\left(\boldsymbol{F}\widetilde{\boldsymbol{W}}^T\widetilde{\boldsymbol{W}}\right)\right] \tag{11}$$

Iterating Equation (8), (9), (10), (11) upto convergence, we can make a good approximation of both $\boldsymbol{W}$ and $\boldsymbol{\sigma^2}$. The basic algorithm of PPCA is given below:

Algorithm 1 Basic PPCA

```
W = normrnd(D, d)
ss = normrnd(1, 1)
Y_m = columnMean(Y)
Y_c = Y − Ym
while (!Stop_Condition) do
    M = W^T * W + ss * I
    X = Y_c * W * M^?1
    XtX = X^T * X + ss * M^−1
    YtX = Y_c^T * X
    W = YtX× invert(XtX)
    ss2 = trace(XtX * W^T * W)
    ss3 = Σ_{n=1}^{N} X_n * W^T * Y_cn
    ss = (||Yc||_F^2 + ss2 − 2 * ss3) × N^−1 × D^−1
end while
```

V. Recent Implementation of *sPCA*

[15] presented a scalable implementation of Probabilistic PCA for distributed platforms such as MapReduce and Spark.

A. Special Features

The implementation of *sPCA* has a good number of features:

1) Mean Propagation to Leverage Sparsity: The first optimization we propose is the mean propagation idea, which preserves and utilizes the sparsity of the input matrix $\boldsymbol{Y}$. PPCA requires the input matrix to be mean-centered, meaning that the mean vector $\boldsymbol{\mu}$ must be subtracted from each row of the original matrix $\boldsymbol{Y}$. Large matrices, however, are mostly sparse, with many zero elements. Sparse matrices can achieve a small disk and memory footprint by storing only non-zero elements, and performing operations only over non-zero elements. Subtracting the non-zero mean from the matrix would make many elements non-zero, so the advantage of sparsity is lost.

To avoid the problems of subtracting the mean, they keep the original matrix $\boldsymbol{Y}$ and the mean $\boldsymbol{\mu}$ in two separate data structures. they did not subtract the mean $\boldsymbol{\mu}$ from $\boldsymbol{Y}$. Rather,

they propagate the mean throughout the different matrix operations.

2) Minimizing Intermediate Data: Intermediate data can slow down the distributed execution of any PCA algorithm, because it needs to be transferred to other nodes for processing to continue. Their second optimization is job consolidation, which means merging multiple distributed jobs into one in order to reduce the communication between these jobs.

3) Efficient Matrix Multiplication: PPCA requires many matrix multiplications, which are expensive operations in a distributed setting. To appreciate the techniques that *sPCA* employs to overcome the inefficiency of matrix multiplication, they briefly explain different possible implementations of this operation.

4) Efficient Frobenius Norm Computation: The PPCA algorithm requires computing the Frobenius norm of the mean-centered input matrix. To solve this problem, they design an algorithm which does not even require creating the dense vector. Many machine learning algorithms compute various norms of matrices. The proposed method for optimizing the computation of the Frobenius norm can be extended to other matrix norms using similar ideas. Thus, this simple optimization can benefit several other machine learning algorithms.

B. Limitations

Despite of having some wonderful features *sPCA* has some limitations also. We can discuss these limitations from two aspects.

1) The approach is not applicable for high dimensional data or we can say tall and wide big data. Though the approach has done some efficient use of memory by reducing the generation of intermediate data, it still runs out of memory while running on big data whose vertical dimension is in the range of millions. We failed to run *sPCA* on dataset of dimension $10M \times 5M$ in our set-up cluster where we have machines with $64GB$ of RAM.

2) There was no implementation on geographically distributed big data. In current world where data is distributed across national border to ensure fast access and national data sovereignty, it is necessary to be capable of doing data analysis on such geo-distributed data. However, *sPCA* did not indicate a process of generating some kind of partial result and later accumulate them to produce the final analytic results.

VI. Our Proposed Approach

As discussed before, we are interested in presenting an approach that will resolve the memory limitation problem and hence give an systematic way to run *PPCA* on geographically distributed data.

A. Handling Tall and Wide Big Data

Our first concern was to make commodity computers capable of performing PPCA on such big data which has very large sample count as well as very high dimension i.e. very large feature count. In this respect we will follow the approach from [15] with some necessary improvements. To have a distributed settings we will take advantage of cluster computing. In case of very wide data the size of principal subspace W is going to be very big. Therefore, we have to treat the principal subspace W as big data also. Performing PPCA in the conventional approach on tall and wide big data will result in the overflow of memory. Therefore, we have to make some kind of improvements to make the approach capable of handling tall and wide big data in case of performing PPCA overcoming the memory scarcity. The steps of performing PPCA in a single cluster is as follows:

*1) **Step 1 - Partition of Principal Subspace** W:*
For data with the dimension $N \times D$ we will get a principal subspace of dimension $D \times d$ where d denotes the number of principal components we want. Working with full horizontal D dimension of W will result in memory overflow. Therefore, we will make n horizontal partition of W and create $W_1, W_2, \ldots, W_n$. As a result each stage of creating W will consists of n smaller stages of creating W_i.

*2) **Step 2 - Data Partition**:*
As we are partitioning the horizontal D dimension of principal subspace W, we are indirectly make vertical partition of our data of $N \times D$. At each iteration of generating W, we have to work on a specific horizontal segment of it. Therefore, we have to work with only that segment of dimension of our data. In that sense we can say that we are creating vertical partition of our main data matrix. It might be a case that data is pretty large that even a distributed setting with multiple machines might not be capable of performing the PPCA task by keeping the full data in memory. In such case our approach can be memory efficient by loading only a single segment of data at each iteration. This will help in memory intense tasks.

*3) **Step 3 - Expectation Step and Omission of Noise Model**:*
From Algoritm IV-A we can see that computation of $\boldsymbol{\sigma^2}$ involves X which is going to a $N \times d$ matrix. Therefore, passing such big data will result in communication bottleneck. For simplicity of computation and minimizing the inter data cluster communication, we will omit the noise calculation.

On the other hand, in order to take advantage from segmented computation, we have to make some changes in the Expectation and Maximization steps of the main **EM Algorithm** of PPCA. Omitting the noise model and from Equation (8), (9) and Algorithm IV-A, the E-Steps are:

$$M = W^T * W \tag{12}$$

$$X = Y_c * W * M^{-1} \tag{13}$$

$$XtX = X^T * X \tag{14}$$

$$YtX = Y_c^T * X \tag{15}$$

In our approach M is not going to be produces at the start of the process as ot depends on W which will be segmented in our system. Therefore we are going to produce multiple

number of M's and accumulate them to produce the final M. The steps equivalent to Equation 12:

$$M_i = W_i^T * W_i \tag{16}$$

$$M = \sum_{i=1}^{n} M_i \tag{17}$$

Similar thing will happen for producing X of Equation 13:

$$X_i = Y_{ci} * W_i \tag{18}$$

$$X = \sum_{i=1}^{n} X_i \tag{19}$$

Steps to produce XtX:

$$\begin{aligned} X &= X * M^{-1} \\ XtX &= X^T * X \\ &= (X * M^{-1})^T * (X * M^{-1}) \\ &= {M^{-1}}^T * X^T * X * M^{-1} \\ &= (M^{-1})^T * (X^T * X) * (M^{-1}) \end{aligned}$$

Similarly we will produce YtX

$$\begin{aligned} X &= X * M^{-1} \\ (YtX)_i &= Y_{ci}^T * X \\ &= (Y_{ci}^T * X) * M^{-1} \end{aligned}$$

*4) **Step 4 - Maximization Step**:*
As we have mentioned earlier we are omitting the calculation of variance $\boldsymbol{\sigma^2}$. Therefore, our maximization step gets limited to the only computation of new W. As we are generating W in a segmented approach, at iteration i we are going to generate W_i as follows:

$$Wi = (YtX)_i * XtX^{-1}$$

Here $(YtX)_i$ is the YtX generated from Y for the range of dimension corresponding to i.

B. Flow Graph and IO Operations

Figure VI-B shows the flow of the algorithm of handling tall and wide big data in a single cluster. The algorithm will start from a random state. It will randomly generate initial segments $W_1, W_2, \ldots, W_n$. Then save them in File System. At each round the algorithm will load a particular segment W_i and with the corresponding dimensions of data Y_i it will generate M_i and X_i. after n iterations complete X and M will be generated. Then using X, M^{-1} and ${M^{-1}}^T$, it will generate XtX. At the same time using Y_i, X and M^{-1}, it will generate particular $(YtX)_i$. Then using XtX and $(YtX)_i$, the algorithm will generate new W_i and store it in File System and then call the accumulation process to start for that particaulr W_i. VI-B is the Algorithm to handle tall and wide biog data in a single cluster while

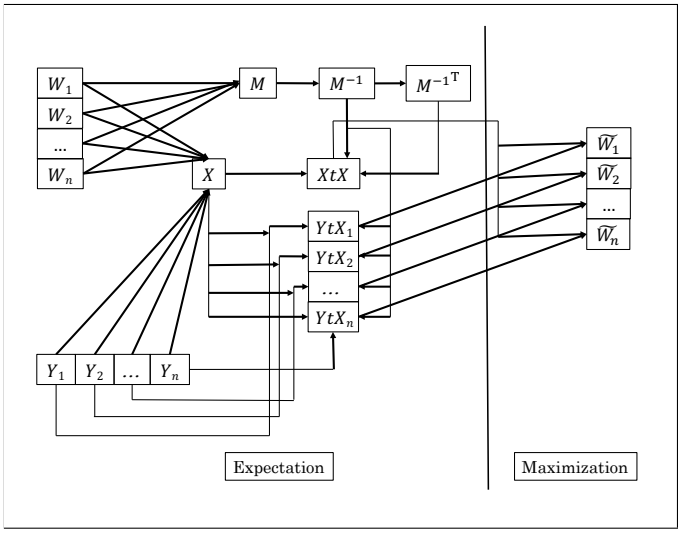

Fig. 3. Flow Graph of Algorithm VI-B

C. Accumulation of partial results from geographically distributed clusters

As we mentioned earlier our data will be geographically distributed. That means we are not going to have any global view of data and no raw data passing is allowed to preserve national data sovereignty. Therefore, the partially computed results have to accumulated to produce the final result. We had the intention to minimize (a) the volume of data to be transmitted throughout the whole process and (b) accumulate the partial results in a way that will ensure the minimum accumulation time. The former was handled in the previous subsection. In The later part we are going to be use some graph theory properties to achieve what we desire.

At each iteration of PPCA we are generating a segment of our principal subspace W. An arbitrary segment can can be denoted as W_i. Whenever one such segment is newly generated, each data cluster will call the **Accumulation** procedure from Algorithm VI-C4. The full accumulation process that we are proposing can be presented as follows:

*1) **Step 1: Generating The Initial Graph**:*
The data clusters that are geographically distributed are connected by high or low bandwidth connection based on their location. According to [26] and [27]:

- Clusters that are located at the same region generally have connection of bandwidth in the range above $100gbps$
- Clusters that are located at nearby regions generally have connection of bandwidth in the range of 10-$100gbps$
- Clusters that are located at different and distanced regions generally have connection of bandwidth in the range of 1-$10gbps$

At the start of the full PPCA process, every data cluster will check the bandwidth of the connections of every other clusters it is connected with and generate a graph. The full process will results in a complete graph if there is an interconnection between every two data clusters. In this graph every node v_i

Algorithm 2 PPCA on Tall and Wide Big Data

createMyInfo()
$myInfo$ = *readFile*($myInfo$)
Let $Range[1.....partitionCount + 1]$ be an array
for i from 1 to $partitionCount$ **do**
    $Range[i] = (i-1) \times D \div partitionCount$
end for
for i from 1 to $partitionCount$ **do**
    $start$ = $Range[i]$
    end = $Range[i+1]$
    W_i =*GenerateRandomW*($start, end, d$)
    saveWInStorage(W_i)
end for
$Stop\_Condition$ = $False$
while (!($Stop\_Condition$)) **do**
    $M = null$
    $X = null$
    for i from 1 to $partitionCount$ **do**
        $start$ = $Range[i]$
        end = $Range[i+1]$
        W_i = *loadWFromStorage*($start, end$)
        $M_{new} = W_i^T \times W_i$
        $M = M + M_{new}$
        $X_m = Y_m \times W_i$
        SegementedXJob($X, Y, X_m, W_i, start, end$)
    end for
    $invM$ =*invert*(M)
    $YtX = null$
    $XtX = null$
    for i from 1 to $partitionCount$ **do**
        s = $Range[i]$
        e = $Range[i+1]$
        generateYtXandXtX($YtX, XtX, X, Y, Y_m, i, s, e$)
        $YtX = YtX \times invM$
        $XtX = invM^T \times XtX \times invM$
        $Wi = YtX \times$ *invert*(XtX)
        startAccumulation($myInfo, i$)
    end for
end while

will represent a data cluster DC_i while edge $e_{ij}(cost)$ between nodes v_i and v_j denotes that DC_i and DC_j are connected at a b/w using which certain amount of data needs $cost$ amount of time to transmit among the two vertices. The Figure VI-C1 shows such a generated graph with 10 data clusters.

*2) **Step 2: Generating MST**:*
Using the graph generated in *step 1*, every node will generate a *Minimum Spanning Tree* to make a connected component with the least cost using time required to transmit data as the cost of the tree edges. Figure VI-C2 shows the MST generated from graph from Figure VI-C1.

*3) **Step 3: Sub Tree Generation for Parallel Accumulation**:*
We are going to generate two sub trees from the MST generated in *Step 2* according to Algorithm VI-C3. We will

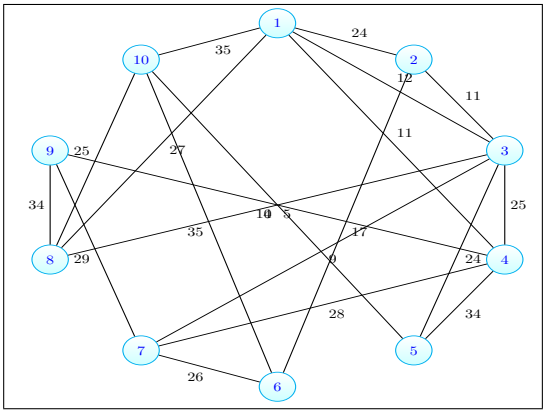

Fig. 4. Generated Graph G According to Step-1

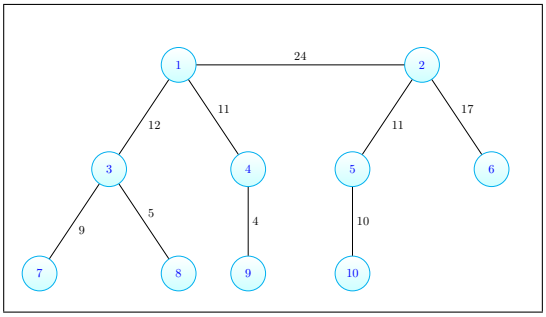

Fig. 5. Generated MST According to Step-1 from Graph G

intended to choose the maximum time consuming edge e_m and make a sub tree rooted at each of the end vertices of that edge. This will be done to make the process to use the maximum time consuming link only once while accumulating data. What we are trying to achieve is that all the data cluster nodes at each of the sub trees now can perform the accumulation task in parallel. Accumulation will be done in bottom up direction. Therefore two accumulated results will available at the two root data nodes. At that point a single exchange will results in having the final result at each of the root node of the two sub trees.

From this sub tree generation with using the node information, every data node will learn the information of its child nodes and parent nodes. During accumulation it will accumulate results from its child data nodes and notify the completion of it accumulation to its parent node.

What we keep in mind that this subtree formulation method might create some kind of unbalanced subtree. Figure VI-C3 shows such an unbalanced sub tree where $subtree1$ has a total cost of 51 and $subtree1$ has a total cost of 21.

To overcome such condition, we will then make a trade off between choosing the highest time consuming link and the balanced cost of the sub trees. To make the sub tree costs more or less balanced we will try to take the next maximum time consuming link and so on.

*4) **Step 4: Redistribution of Final Accumulated Result**:*
Whenever each of the root nodes completes the accumulation

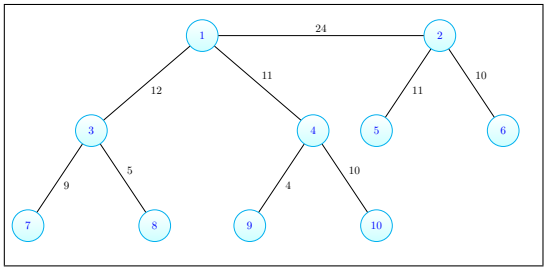

Fig. 6. Unbalanced Sub Tree

Algorithm 3 createSubTree

Input: V_m, Vertex Set of MST
Input: E_m, Edge Set of MST
Output: $\{subTree1, subTree2\}$, Two Subtrees
sort E_m according to increasing b/w
$done$=$Fasle$
while !$done$ **do**
  e = *nextMinimumBWEdge*(E_m)
  $v1$ = $e.startVertex$
  $v2$ = $e.endVertex$
  $subTree1$ = *subTree*($v1$)
  $subTree2$ = *subTree*($v2$)
  $cost1$ = *cost*($subTree1$)
  $cost2$ = *cost*($subTree2$)
  if (*max*($cost1, cost2$)/*min*($cost1, cost2$)≤ 1.5) **then**
    $done = True$
  end if
end while
return $\{subTree1, subTree2\}$

of its own sub tree, it will wait for the other root to complete. After that it will access the partially accumulated result from the other root node and complete the full computation. At this stage it will distribute the final result by pushing it through the links connecting to its neighbours. As there will be dedicated links, the result can be sent to each of the neighbours at the same time. Whenever ant node in a sub tree receives the final result it will also redistribute it to its child nodes in the tree configuration. The job will end with the leaf data nodes receiving the final accumulated result.

The Algorithm VI-C4 is for accumulating the partial result. It will be called from every data nodes (leaf or non leaf) and according to the node information generated by Algorithm VI-C4 will communicate with other nodes in the communication tree.

VII. Contribution

VIII. Properties of our Approach

In this section, we analyze the computational and communication complexity of our proposed algorithm and derive some theoretical properties.

Algorithm 4 startAccumulation

Input: $myInfo$
Input: $indexOfW$
W = *loadW*($indexOfW$)
for (each $childNode$ in $myInfo.child()$) **do**
  if ($childNode.WisReady(indexOfW)$) **then**
    W_{child} = *getWFromNode*($childNode$)
    W = W+W_{child}
  end if
end for
notifyParent($myInfo.parent$)
if ($myInfo.rootNode$) **then**
  Wait Until Other Root Node Data is Ready
  $W_{otherRoot}$ = *getWFromNode*($myInfo.otherRootNode$)
  W = W+$W_{otherRoot}$
  Announce $doneW(indexOfW)$ as $True$
  for (each $childNode$ in $myInfo.child()$) **do**
    transmit(W)
    transmit($doneW(indexOfW)$)
  end for
else
  while (!$doneW(indexOfW)$) **do**
    wait()
  end while
  for (each $childNode$ in $myInfo.child()$) **do**
    transmit(W)
    transmit($doneW(indexOfW)$)
  end for
end if

Algorithm 5 createMyInfo

Input: V : Set of Clusters as Vertices
Input: E : Set of Weighted Edges; b/w as Link Weights
Input: $G = \{V, E\}$ Clusters' Distribution Graph
$MST(V_m, E_m)$ = *createMST*(G)
$\{subTree1, subTree2\}$ = **createSubTree**(V_m, E_m)
$myInfo$ = *createMyInfo*($subTree1, subTree2, myID$)
saveMyInfo($myInfo$)

A. Computational Complexity

We will discuss the computational complexity for Algorithm VI-B that do the task of computing principal subspace W pf the input data Y. For computational complexity analysis, we will consider a spark-like [28] distributed setting where in a data cluster each of the data nodes has a portion of full data and intermediate results are stored in-core. Specifically if there are c data nodes in a single cluster and each of them have approximately $N_i \times D$ data matrix Y_i, then we can write:

$$Y = \sum_{i=1}^{c} Y_i$$

We assume the partition count of W to be p. if every segment of W is of size $D_i \times d$ where d is number of principal

components to be computed. Then we can write:

$$W = \sum_{i=1}^{p} W_i$$

Now our Algorithm VI-B has three different parts.

From line 7 to 12 a **For Loop** is generating initial random W_i's. Here the running time is $\mathcal{O}(pD_id)$

From line 17 to 24 another **For Loop** is there to compute X of Equation 16 and 17. Here tasks of line 20-24 will have a time complexity of $\mathcal{O}(D_id + d^2 + d + d + N_iD_id)$. That makes the aggregated runtime of this for loop $\mathcal{O}(pN_iD_id)$.

From line 29 to 37 another **For Loop** is generating XtX and YtX, fining at generating new W_i. Therefore line 32-35 will have a time complexity of $\mathcal{O}(N_id^2 + D_iN_id + D_id^2 + d^4 + D_id^2)$. That makes the aggregated runtime of this for loop $\mathcal{O}(pN_iD_id)$.

Therefore, if **While Loop** of line 14 runs for r rounds then the total run time is going to be $\mathcal{O}(2rpN_iD_id)$.

B. Communication Complexity

The approach is highly communication intensive. At the end of generating each new segment of principal subspace W each data cluster will call Algorithm VI-C4 as a background process. This will make a data cluster to accumulate from its child nodes and give a notification to its parent. Each accumulation demands the transmission of a $D_i \times d$ matrix where D_i varies with the number of partition is done on W. As partition information i.e. D_i is available at the start of the process the communication tree generation can be done based on the available bandwidth between any two data clusters and the time required to transmit data of that size. For simplicity we assume the availability of dedicate bandwidth between any two data clusters.Under these assumptions, the time needed to make the accumulated result of a subtree available at the root node of that subtree is equal to the maximum aggregated time of any path from root to leaf node. If such paths are P_1 and P_2 of $subTree1$ and $subTree2$ respectively and maximum time consuming link time is max_time, then the time of communication will be

$$max\_time + 2 * max(time(P_1), time(P_2))$$

The 2 is added as accumulated data will be redistributed in the same paths. In Figure 7 the maximum time consumtioon link has a time of 27 units the gray shaded paths has the maximum time consumption of 46 and 31 units. Therefore the communication time of accumulating one segement of W will be

$$27 + 2 * max(46, 31) = 119 \ units$$

IX. Implementation in Spark Cluster System

We implemented our proposed system on Spark [28] cluster computing system. Spark gives us various types of storage system. Spark provides resilient distributed datasets (RDDs) and parallel operations on these datasets. Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark [29].

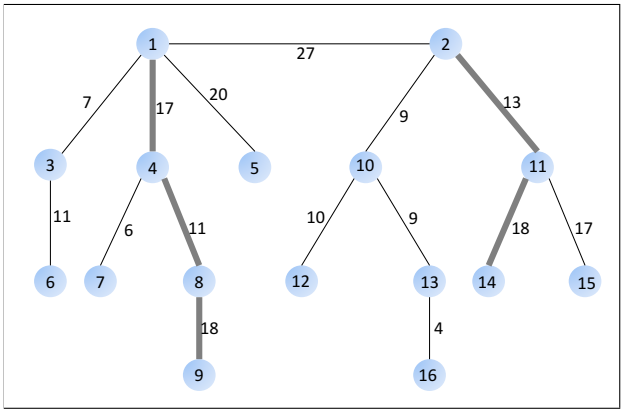

Fig. 7. Communication Complexity

It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. From user level the persistanc of RDD can be controlled. It can be cached in memory or store on disk to be used later through IO operations. Moreover the user can control its partitioning process like partition by key.

In case of small data that can be kept in memory while computing, we did the in-memory computations by making the input matrix Y persistent in the memory of the machines of our spark cluster. Therefore we could perform faster distributed operations on our data. On the other hand, for clusters with small count of nodes i.e. small amount of memory, we have to make IO operations by keeping the data on disks and load a specific segment of it while needed. The Algorithms IX and IX are based on the Spark Programming.

Algorithm 6 SegmentedXJob($X, Y, X_m, W, start, end$)

1: Y_nX = Y.zip(X)
2: X = Y_nX.map{$(Y_nX)_i \Rightarrow$
3: Y_i = $(Y_nX)_i$*.arg0().range*($start, end$)
4: X_i = $(Y_nX)_i$*.arg1()*
5: dotRes = $Y_i \times W$
6: X_i = X_i + dotRes - $(X_m)_i$
7: }

X. Evaluation

In this section we will present our experimental result that we found while implementing our approach in *Spark* Cluster computing system.

A. Cluster Setup

We built *Spark* clusters in our *Undergraduate Thesis Lab*. To evaluate the performance of the method to handle *Tall and Wide Big Data*, we created clusters that was composed of three machines with one master node and two slave nodes. Each of the machines had $64GB$ of memory and quad core processor of $3.6GHz$. That makes our cluster with 12 cores of processor to work with. We installed *Apache Spark 2.0* with *Hadoop*

Algorithm 7 SegmentedXJob($YtX, XtX, X, Y, Y_m, i, s, e$)

```
1: Y_nX = Y.zip(X)
2: if (i == 1) then                    ▷ Iteration for First Segment of W
3:     YtXSum = spark.accumultor(newMatrix(D, d))
4:     XtXSum = spark.accumultor(newMatrix(D, d))
5:     Y_nX.map{(Y_nX)_i ⇒
6:         Y_i = (Y_nX)_i.arg0().range(s, e)
7:         X_i = (Y_nX)_i.arg1()
8:         (YtX)_i = Y_i^T × X_i - Y_m^T × X_i
9:         (XtX)_i = X_i^T × X_i
10:        YtXSum.add((YtX)_i)
11:        XtXSum.add((XtX)_i)
12:    }
13:    YtX = YtXSum.value()
14:    XtX = XtXSum.value()
15: else
16:    YtXSum = spark.accumultor(newMatrix(D, d))
17:    Y_nX.map{(Y_nX)_i ⇒
18:        Y_i = (Y_nX)_i.arg0().range(s, e)
19:        X_i = (Y_nX)_i.arg1()
20:        (YtX)_i = Y_i^T × X_i - Y_m^T × X_i
21:        YtXSum.add((YtX)_i)
22:    }
23:    YtX = YtXSum.value()
24: end if
```

2.7 for *HDFS*. For simplicity we assume to have homogenius data clusters ehere each data cluster has the same hardware configuration.

On the other hand, to test the accumulation process, we created 8 spark clusters each of which contained only one slave node. as we did not have that amount of machines in a single cluster, we used small dimensional data to test this part of the method.

B. Data Sets

We use two real datasets. These data sets contains data of different types and of different sizes. The dimensionalities are also different. The datasets are:

- **Amazon Product Ratings**
  This data set is from *Amazon* web sites that includes the user ratings on various products items. In the raw data, it contains three columns: 1) unique User ID, 2) unique Product ID, 3) Product Rating. To make it easily fit in our system we gave each of the User ID and Product ID a number. There was 21 $millions$ user id's and 9 $millions$ product id's. That makes it $21M \times 9M$ sparse data matrix. In the matrix if cell (x, y) have a number z, that means user x has given a rating z to the product y. The raw data was of size $4.73GB$
- **Twitter Follower Relationship**
  This data set is of *Twitter* that includes the follower information of every user ID. The raw data has two columns: two User IDs. The first User ID is followed by the second one. The maximum user Id is 61578414. That makes it a data set where every data sample has a dimension of $61M$ while it has $61M$ rows that makes it a data set of $61M \times 61M$. a 1 in cell (x, y) indicates that user y follows user x while a 0 indicates the negatively. The raw data was of size $26.17GB$

C. Performance Metrics

We did not have much opportunity to compare our method with others as there are no other algorithm that can handle geo-distributed tall and wide big data. Though the *sPCA* [15] showed some kind of implementation of PPCA in a distributed cluster, it also fails to run on very large dimensional data. Moreover, it has no method to accumulate partial data from different geographic locations.

1) Partition Count of W: In case of handling Tall and Wide data in a single cluster we made partitions of the principal subspace W. However, one of the limitations of our method is that we could not find a optimum partition count based on the data size and available resources. Rather we gathered some empirical results by giving manual input on the partition count. Table I shows the influence of no of partitions on the total running time and IO operation time of the data matrix of *Amazon Product Rating* Dataset. As at a time, we are working on a single partition of W and also creating a single partition of W, we had to allow IO operations. We had to load and store W_i while working on the range of dimension of our data matrix Y corresponding to the subscript i. Table II shows the same information for data matrix of *Twitter Follower* Dataset.

Memory	Data Size	Partitions	Iterations	Runtime	IO Time
$6GB$	$21M \times 9.8M$	15	7	18.0 Hrs	65 Mins
		13	8	18.5 Hrs	50 Mins
		11	7	14.1 Hrs	44 Mins
		10	Failed		
$8GB$	$21M \times 9.8M$	12	8	9.8 Hrs	61 Mins
		11	7	9.5 Hrs	57 Mins
		10	6	5.5 Hrs	50 Mins
		9	Failed		
$8GB$	$10M \times 7M$	9	8	5.3 Hrs	41 Mins
		8	7	4.1 Hrs	38 Mins
		7	Failed		

TABLE I
CAPTION

Memory	Data Size	Partitions	Iterations	Runtime	IO Time
$8GB$	$61M \times 61M$	22	6	91.1 Hrs	345 Mins
		20	7	108.5 Hrs	397 Mins
		19	Failed		
$12GB$	$61M \times 61M$	20	8	99.3 Hrs	405 Mins
		18	7	87.4 Hrs	317 Mins
		17	Failed		

TABLE II
CAPTION

2) Running Time: We compare our Spark implementation with *sPCA*. It should be noted that

XI. Conclusion

In this paper, we tried to analyze various issues of computing the principal components of an input data matrix which has large number of data samples while each of the samples has very large dimension. We characterized this kind of data as *Tall and Wide Big Data*. In addition the data is distributed at different geographic locations. Our analysis also indicated that the present algorithms for performing *PCA* are not capable of performing the task on such data sets. We presented a new approach and implementation for PCA that is capable of handling big data with very large dimension and can accumulate the partial results from different geographic locations in an efficient way. Our approach is based on the probabilistic PCA (PPCA) algorithm defined in [25]. We implemented this method on Spark platform and showed that it did the job significantly well.

References

[1] Matei Zaharia et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In NSDI, 2012.

[2] Yucheng Low et al. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. PVLDB, 2012.

[3] Mu Li et al. Scaling distributed machine learning with the parameter server. In OSDI, 2014.

[4] Ashish Thusoo et al. Data warehousing and analytics infrastructure at Facebook. SIGMOD, 2010.

[5] George Lee et al. The unified logging infrastructure for data analytics at Twitter. PVLDB, 2012.

[6] Aditya Auradkar et al. Data infrastructure at linkedIn. In ICDE, 2012.

[7] Ariel Rabkin et al. Aggregation and degradation in jetstream: Streaming analytics in the wide area. In NSDI, 2014.

[8] Ashish Vulimiri et al. Global analytics in the face of bandwidth and regulatory constraints. In NSDI, 2015.

[9] Nikolaos Laoutaris et al. Inter-datacenter bulk transfers with netstitcher. In SIGCOMM, 2011.

[10] Albert Greenberg et al. The cost of a cloud: research problems in data center networks. SIGCOMM, 2008.

[11] Martin Rost and Kirsten Bock. Privacy by design and the new protection goals. DuD, January, 2011.

[12] European Commission press release. Commission to pursue role as honest broker in future global negotiations on internet governance. http://europa.eu/rapid/press-releaseIP-14-142en.htm.

[13] Principal component analysis: https://en.wikipedia.org/wiki/Principal_component_analysis.

[14] J. Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*. 2014.

[15] T. Elgamal and M. Hefeeda. Analysis of pca algorithms in distributed environments. *arXiv preprint arXiv:1503.05214*. 2015.

[16] N. P. Halko. Randomized methods for computing low-rank approximations of matrices. Ph.D. dissertation, Boulder, CO, USA, 2012, aAI3507998.

[17] M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology), vol. 61, no. 3, pp. 611?622*. 1999.

[18] S. Roweis. Em algorithms for pca and spca. *Advances in neural information processing systems, pp. 626?632* 1998.

[19] I. Jolliffe. Principal component analysis. 1986. 1986.

[20] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. MLbase: A distributed machine-learning system. In *Proc. Conf. on Innovative Data Systems Research (CIDR), 2013*.

[21] G. Golub and C. E. Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14(5), 1970.

[22] J. Demmel and W. Kahan. Accurate singular values of bidiagonal matrices. *SIAM J. Sci. Stat. Comput*, 11(5), 1990.

[23] V. Hernandez, J. Roman, and A. Tomas. A robust and efficient parallel SVD solver based on restarted Lanczos bidiagonalization. *Electronic Transactions on Numerical Analysis*, 31, 2008.

[24] N. P. Halko. Randomized methods for computing low-rank approximations of matrices. *PhD thesis, University of Colorado*, 2012.

[25] M. E. Tipping and C. M. Bishop. *Mixtures of probabilistic principal component analysers. Neural Computation, 11(2)*, 1999.

[26] Nokia White Paper. *Datacenter interconnect market trends and requirements*. weblink: `https://resources.ext.nokia.com/?cid=181666`

[27] Fujitsu White Paper. *Application Note: Data Center Interconnect*. weblink: `http://www.fujitsu.com/us/Images/Data-Center-Interconnect-app-note.pdf`

[28] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In Proc. *USENIX Conf. on Networked Systems Design and Implementation (NSDI)*, NSDI12. USENIX Association, 2012.

[29] Tutorials Point Tutorial. weblink: `https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm`