Principal Component Analysis for Distributed Data Sets with Updating

Zheng-Jian Bai[1⋆], Raymond H. Chan[1], and Franklin T. Luk[2]

[1] Department of Mathematics, Chinese University of Hong Kong, Shatin, NT, Hong Kong, China
zjbai, rchan@math.cuhk.edu.hk
[2] Department of Computer Science, Rensselaer Polytechnic Institute, Troy, New York 12180, USA
luk@cs.rpi.edu

Abstract. Identifying the patterns of large data sets is a key requirement in data mining. A powerful technique for this purpose is the principal component analysis (PCA). PCA-based clustering algorithms are effective when the data sets are found in the same location. In applications where the large data sets are physically far apart, moving huge amounts of data to a single location can become an impractical, or even impossible, task. A way around this problem was proposed in [11], where truncated singular value decompositions (SVDs) are computed locally and used to reduce the communication costs. Unfortunately, truncated SVDs introduce local approximation errors that could add up and would adversely affect the accuracy of the final PCA. In this paper, we introduce a new method to compute the PCA without incurring local approximation errors. In addition, we consider the situation of updating the PCA when new data arrive at the various locations.

1 Introduction

Effective clustering of large data sets is a major objective in data mining. Principal component analysis (PCA) [4, 5, 10] offers a popular statistical technique to analyze multivariate data by constructing a concise data representation using the dominant eigenvectors of the data covariance matrix. PCA and PCA-based clustering methods play an important role in various applications such as knowledge discovery from databases [2], disease classification [9], and remote sensing [8]; for more applications, see [7] and the references therein.

PCA is effective for high-dimensional data analysis when the data sets are collocated. However, in present-day applications, the large data sets could be distributed over a network of distant sites, and PCA-based algorithms may no longer be applicable since these distributed data sets are often too large to send to a single location. There is a growing interest in this topic of distributed data sets and here are some relevant works in the literature: the interaction of huge data sets and the limits of computational feasibility in Wegman [13], parallel methods for spectral decomposition of nonsymmetric matrix on distributed memory processors in Bai *et al.* [1], an efficient out-of-core SVD algorithm in Rabani *et al.* [12], an algorithm for data distributed by blocks of columns in Kargupta *et al.* [7], and a method for massive data sets distributed by blocks of rows in Qu *et al.* [11].

In this paper, we consider the problem described in Qu *et al.* [11], where the authors use truncated singular value decompositions (SVDs) in the distributed locations to reduce communications costs. Their approach is very effective when the local data matrices have *low* ranks and can be accurately approximated via a truncated SVD (note that the savings may be nonexistent when the data matrices have high ranks). In addition, the small local approximation errors may add up substantially when the number of locations is large. We will present a new algorithm for computing a global PCA of distributed data sets. In contrast to Qu's approach [11], our method introduces no local approximation errors. At the central processor, Qu's approach works with the approximate covariance matrix while we work directly with the data matrix. Our technique will likely require less communication as well. Suppose that there is an $n_i \times p$ matrix of rank m_i at the ith local site for $i = 1, \ldots, s$. While Qu's approach [11] requires $O(p \sum_{i=1}^{s} m_i)$ communication, our procedure uses $O(p^2 \lceil \log_2 s \rceil)$ communication. When s is large, it is probable that $p \sum_{i=1}^{s} m_i > p^2 \lceil \log_2 s \rceil$ as $p \ll n$. We also consider the important problem of updating, for new data do arise all the time (for example,

⋆ Current Address: Department of Mathematics, National University of Singapore, 2 Science Drive 2, Singapore 117543 (matbzj@nus.edu.sg).

medical information and banking transactions), and we develop a procedure for constructing a global PCA for distributed data sets with data updating, by suitably combining the PCAs of past data and the local PCAs of new data.

This paper is organized as follows. Section 2 contains a brief review of the basic concepts. In Section 3 we present an algorithm for computing the global PCA of distributed data sets, and we include a numerical example to illustrate the advantages of our method. In Section 4 we develop a technique for computing the global PCA of distributed data sets with updating. Load balancing for communications and computation is discussed in Section 5, and Section 6 concludes the paper.

Remark 1. Throughout this paper, for simplicity, we assume that there is one processor at each location and so we will use the two words *location* and *processor* interchangeably.

2 Principal Component Analysis

Let X be an n-by-p data matrix, where rows denote the observations and columns denote the features with $n \gg p$. The data covariance matrix S is given by

$$nS = X^T(I - \tfrac{1}{n}\mathbf{e}_n\mathbf{e}_n^T)X, \tag{1}$$

where

$$\mathbf{e}_\ell \equiv (1, 1, \ldots, 1)^T$$

denotes a vector of length ℓ. The PCA of X is given by an eigenvalue decomposition [3] of nS:

$$nS = V\Sigma^2V^T, \tag{2}$$

where

$$\Sigma^2 = \mathrm{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_p^2),$$

with

$$\sigma_1^2 \geq \sigma_2^2 \geq \cdots \geq \sigma_p^2,$$

and V is an orthogonal matrix. Let

$$J \equiv I - \tfrac{1}{n}\mathbf{e}_n\mathbf{e}_n^T.$$

As the matrix J is symmetric and idempotent, we may therefore compute a singular value decomposition (SVD) [3] of the column-centered data matrix JX:

$$(I - \tfrac{1}{n}\mathbf{e}_n\mathbf{e}_n^T)X = U\Sigma V^T. \tag{3}$$

Therefore, it is not necessary to form the covariance matrix S explicitly. We save work and improve accuracy by working directly with the data matrix X. The matrices Σ and V we get in (3) are exactly the matrices we need in (2).

One application of the PCA is to reduce the dimensions of the given data matrix X. To do so, let $\tilde{V}$ denote the first m columns of V, corresponding to the m largest eigenvalues of nS. The m principal components of X is given by the n-by-m matrix

$$\tilde{X} = (I - \tfrac{1}{n}\mathbf{e}_n\mathbf{e}_n^T)X\tilde{V}.$$

It is an optimal m-dimensional approximation of $(I - \frac{1}{n}\mathbf{e}_n\mathbf{e}_n^T)X$ in the least squares sense [6]. The ratio η_m, given by

$$\eta_m \equiv (\sum_{i=1}^{m} \sigma_i^2)/(\sum_{i=1}^{p} \sigma_i^2),$$

reflects the total variance of $\tilde{X}$ in the original data. If $\eta_m \approx 1$ for some $m \ll p$, the n-by-p transformed data matrix JX can be well represented by the much smaller n-by-m matrix $\tilde{X}$, which forms the crux of the approach described in Qu *et al.* [11].

3 Distributed PCA without Updating

We start with the case of no updating. The global data matrix X is distributed among s locations:

$$X = \begin{pmatrix} X_0 \\ X_1 \\ \vdots \\ X_{s-1} \end{pmatrix},$$

where X_i is an n_i-by-p matrix, and resides at Processor i, for $0 \leq i < s$. So,

$$n = \sum_{i=0}^{s-1} n_i$$

gives the number of rows in X. Let S be the covariance data matrix corresponding to X as given in (1). If we are to form S explicitly, then we have to move X_i across the processors, and the communication cost will be $O(np)$. In [11], Qu *et al.* compute the local PCA for each X_i using the SVD. They then send m_i, where $m_i < p$, singular vectors to the central processor where an approximation of S is assembled, and its PCA is computed. The communication cost of the method is thus $O(p\sum_{i=0}^{s-1} m_i)$. A drawback is that the local SVD will introduce approximation errors. In the following, we give a method of finding PCA of X exactly using the QR decomposition. For simplicity, we assume that $s = 2^\ell$ and that the global PCA is computed in location 0 (i.e., Processor 0).

Algorithm 1:

- *At Processor i, for* $0 \leq i < s$*:* Compute the column means of X_i, i.e.,

  $$\bar{\mathbf{x}}_i^T = \frac{1}{n_i}\mathbf{e}_{n_i}^T X_i.$$

  Form the column-centered data matrix

  $$\bar{X}_i = (I - \tfrac{1}{n_i}\mathbf{e}_{n_i}\mathbf{e}_{n_i}^T)X_i = X_i - \mathbf{e}_{n_i}\bar{\mathbf{x}}_i^T. \tag{4}$$

  Then compute its QR decomposition [3]:

  $$\bar{X}_i = Q_i^{(0)} R_i^{(0)}, \tag{5}$$

  where $R_i^{(0)}$ are upper triangular p-by-p matrices. Send n_i and $\bar{\mathbf{x}}_i^T$ to Processor 0. If $i \geq s/2$, send $R_i^{(0)}$ to Processor $(i - s/2)$. There is no need to send any $Q_i^{(0)}$.
- *At Processor i, for* $0 \leq i < s/2$*:* Compute the QR decomposition of $R_i^{(0)}$ and $R_{i+s/2}^{(0)}$ by using Givens' rotations:

  $$\begin{pmatrix} R_i^{(0)} \\ R_{i+s/2}^{(0)} \end{pmatrix} = Q_i^{(1)} R_i^{(1)}, \tag{6}$$

  where $R_i^{(1)}$ are p-by-p upper triangular matrices. If $i \geq s/4$, send $R_i^{(1)}$ to Processor $(i - s/4)$. Again, there is no need to send any $Q_i^{(1)}$.
- *Continue until we reach Processor 0 after* $\ell = \lceil \log_2 s \rceil$ *steps.*
- *At Processor* 0*:* Compute the QR decomposition of $R_0^{(\ell-1)}$ and $R_1^{(\ell-1)}$ by using Givens' rotations:

  $$\begin{pmatrix} R_0^{(\ell-1)} \\ R_1^{(\ell-1)} \end{pmatrix} = Q_0^{(\ell)} R_0^{(\ell)}, \tag{7}$$

where $R_0^{(\ell)}$ is an p-by-p upper triangular matrix. Form the following $(s+p)$-by-p upper-trapezoidal matrix and compute its QR decomposition by Householder's reflections:

$$\begin{pmatrix} \sqrt{n_0}(\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}) \\ \sqrt{n_1}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}) \\ \vdots \\ \sqrt{n_{s-1}}(\bar{\mathbf{x}}_{s-1} - \bar{\mathbf{x}}) \\ R_0^{(\ell)} \end{pmatrix} = QR. \tag{8}$$

Here, R is an p-by-p upper triangular matrix and

$$\bar{\mathbf{x}} \equiv \frac{1}{n}\sum_{i=0}^{s-1} n_i \bar{\mathbf{x}}_i$$

gives the column mean of X. The PCA of S can now be obtained by computing the SVD of R:

$$R = U\Sigma V^T. \tag{9}$$

Remark 2. Algorithm 1 works for an arbitrary $s > 0$ if we replace s by s_+, where $s_+ := 2^{\lceil \log_2 s \rceil}$. For $s_+ > s$, the matrices $\{X_i\}_{i=s+1}^{s_+}$ are empty.

Lemma 1. *The covariance matrix S as defined in (1) is given by:*

$$nS = {R_0^{(\ell)}}^T R_0^{(\ell)} + \sum_{i=0}^{s-1} n_i(\bar{\mathbf{x}}_i - \bar{\mathbf{x}})(\bar{\mathbf{x}}_i - \bar{\mathbf{x}})^T = R^T R. \tag{10}$$

In particular, the PCA of S is given by the Σ and V computed in (9).

Proof. The last equality in (10) follow from (8). To prove the first equality, we note that

$$\bar{\mathbf{x}}^T = \tfrac{1}{n}\mathbf{e}_n^T X.$$

Hence

$$(I - \tfrac{1}{n}\mathbf{e}_n\mathbf{e}_n^T)X = (I - \tfrac{1}{n}\mathbf{e}_n\mathbf{e}_n^T)\begin{pmatrix} X_0 \\ \vdots \\ X_{s-1} \end{pmatrix} = \begin{pmatrix} X_0 - \mathbf{e}_{n_0}\bar{\mathbf{x}}^T \\ \vdots \\ X_{s-1} - \mathbf{e}_{n_{s-1}}\bar{\mathbf{x}}^T \end{pmatrix}$$

$$= \begin{pmatrix} X_0 - \mathbf{e}_{n_0}\bar{\mathbf{x}}_0^T + \mathbf{e}_{n_0}(\bar{\mathbf{x}}_0 - \bar{\mathbf{x}})^T \\ \vdots \\ X_{s-1} - \mathbf{e}_{n_{s-1}}\bar{\mathbf{x}}_1^T + \mathbf{e}_{n_{s-1}}(\bar{\mathbf{x}}_{s-1} - \bar{\mathbf{x}})^T \end{pmatrix} = \begin{pmatrix} \bar{X}_0 + \mathbf{e}_{n_0}(\bar{\mathbf{x}}_0 - \bar{\mathbf{x}})^T \\ \vdots \\ \bar{X}_{s-1} + \mathbf{e}_{n_{s-1}}(\bar{\mathbf{x}}_{s-1} - \bar{\mathbf{x}})^T \end{pmatrix},$$

where the last equality follows from the definition in (4). By (4), we see that the column sums of $\bar{X}_i$ are all zero, i.e.,

$$\mathbf{e}_{n_i}^T \bar{X}_i = \mathbf{0},$$

for $0 \le i < s$. Hence

$$nS = \begin{pmatrix} \bar{X}_0^T + (\bar{\mathbf{x}}_0 - \bar{\mathbf{x}})\mathbf{e}_{n_0}^T \mid \cdots \mid \bar{X}_{s-1}^T + (\bar{\mathbf{x}}_{s-1} - \bar{\mathbf{x}})\mathbf{e}_{n_{s-1}}^T \end{pmatrix} \begin{pmatrix} \bar{X}_0 + \mathbf{e}_{n_0}(\bar{\mathbf{x}}_0 - \bar{\mathbf{x}})^T \\ \vdots \\ \bar{X}_{s-1} + \mathbf{e}_{n_{s-1}}(\bar{\mathbf{x}}_{s-1} - \bar{\mathbf{x}})^T \end{pmatrix}$$

$$= \sum_{i=0}^{s-1} \bar{X}_i^T \bar{X}_i + \sum_{i=0}^{s-1} n_i(\bar{\mathbf{x}}_i - \bar{\mathbf{x}})(\bar{\mathbf{x}}_i - \bar{\mathbf{x}})^T. \tag{11}$$

Using (5)–(7), we have

$$\sum_{i=0}^{s-1} \bar{X}_i^T \bar{X}_i = \sum_{i=0}^{s-1} {R_i^{(0)}}^T R_i^{(0)} = \sum_{i=0}^{s/2-1} {R_i^{(1)}}^T R_i^{(1)} = \cdots = \sum_{i=0}^{1} {R_i^{(\ell-1)}}^T R_i^{(\ell-1)} = {R_0^{(\ell)}}^T R_0^{(\ell)}.$$

Put this in (11) and we get (10).

To get the first m principal components of X, we broadcast $\bar{\mathbf{x}}$ and $\tilde{V}$ (the first m columns of V) to every processor. Then the m principal components of X are given by the matrix $\tilde{X}$:

$$\tilde{X} = (I - \tfrac{1}{n}\mathbf{e}_n\mathbf{e}_n^T)X\tilde{V} = (X - \mathbf{e}_n\bar{\mathbf{x}}^T)\tilde{V}. \tag{12}$$

In particular, at Processor i, we have the n_i-by-m approximation $\tilde{X}_i$:

$$\tilde{X}_i = (X_i - \mathbf{e}_{n_i}\bar{\mathbf{x}}^T)\tilde{V},$$

for $0 \le i < s$. Regarding the communication costs, note that there are $\lceil \log_2 s \rceil$ steps in the algorithm. In step j, we need to move a total number (=$s/2^j$) of p-by-p upper triangular matrices $R_i^{(j)}$. Hence the communication cost is $O(p^2\lceil \log_2 s \rceil)$. We state once more that the PCA (i.e., Σ and V) we obtain is exact.

We ran some numerical experiments on synthetic data using **MATLAB 7.0.1**. They simulated the scenario of distributed data sets to assess computational accuracy and communication costs. Execution times are not provided since they are not meaningful in simulations (cf. [11]).

Example (Synthetic data). The data X are generated as follows (cf. [11]). Let

$$X = GE^T + N,$$

where the n-by-d data matrix G is a d-dimensional Gaussian data, i.e., its entries are identical, independently distributed (*iid*) as $\mathcal{N}(0,1)$ (normal distribution with mean 0 and variance 1), E is a p-by-d matrix with 1's on the diagonal and zeros elsewhere, and N is a p-dimensional Gaussian noise whose entries are *iid* as $\mathcal{N}(0,\sigma^2)$. We partition the data X among s processors evenly. If the modulus r after n divided by s is not zero, let the first r processors contain $\lfloor n/s \rfloor + 1$ observations.

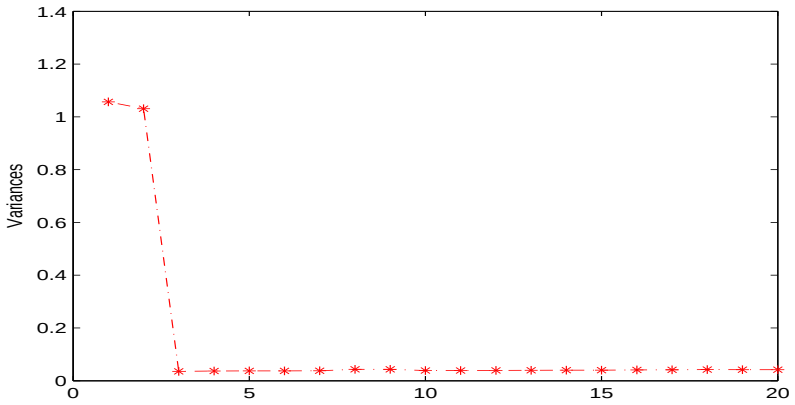

Fig. 1. Eigenvalue distribution of covariance matrix for synthetic data

We took $n = 6,000$, $p = 20$, $d = 2$, and $\sigma = 0.2$, and we set the local and global PC selection thresholds to be $\sqrt{0.8}$ and 0.8, respectively. To further characterize the data X, we plot the eigenvalue distribution of the theoretical covariance matrix with these parameters in Figure 1. Ten simulations were run using the distributed principal component algorithm (DPCA) proposed by Qu *et al.* [11] (Method a) and our algorithm (Method b) for various values of s. In Table 1, we report the means and standard deviations (*s*d) of the quantities given as follows.

$$T_{ae} = \frac{T_a}{T_e}, \quad T_{be} = \frac{T_b}{T_e} \quad \text{and} \quad T_{ba} = \frac{T_{be}}{T_{ae}},$$

where $T_a = (\sum_{i=0}^{s-1} m_i)(p+1) + s(p+3)$ with m_i being the number of PCs selected from the ith processor, $T_b = \frac{1}{2}p(p+1)\ell + s(p+1)$ and $T_e = np$. The quantities T_{ae} and T_{be} provide the ratios of the communication

costs.

$$d_a = \frac{\|(I - n^{-1}\mathbf{e}_n\mathbf{e}_n^T)(\hat{X} - X)\|_2}{\|(I - n^{-1}\mathbf{e}_n\mathbf{e}_n^T)X\|_2},$$

$$d_b = \frac{\|(I - n^{-1}\mathbf{e}_n\mathbf{e}_n^T)(\bar{X} - X)\|_2}{\|(I - n^{-1}\mathbf{e}_n\mathbf{e}_n^T)X\|_2}$$

and

$$d_{ba} = \frac{d_b}{d_a},$$

where $\hat{X}$ is the dimension reduced data obtained by the DPCA [11] and $\bar{X} = \tilde{X}\tilde{V}^T$. Here, $\tilde{X}$ is defined in (12) where m is the number of global PCs which is obtained based on the global PC selection threshold. d_a and d_b are the relative error between the original data X and the data approximated by Methods a and b, respectively. From Table 1 and Figures 2 and 3, we see that our method behaves better than the DPCA in terms of both communication costs and data approximation errors.

	s	1	4	8	16	32	64	128
T_{ae}	mean	.0025	.0092	.0182	.0348	.0653	.1193	.2065
	sd	.0000	.0001	.0001	.0004	.0008	.0011	.0017
T_{be}	mean	.0002	.0042	.0067	.0098	.0143	.0217	.0347
	sd	0	.0000	.0000	.0000	.0000	0	.0000
T_{ba}	mean	.0709	.4556	.3645	.2814	.2197	.1819	.1678
	sd	0	.0041	.0025	.0036	.0028	.0016	.0014
d_a	mean	.2054	.2030	.2025	.2042	.2043	.2049	.2044
	sd	.0017	.0025	.0018	.0024	.0021	.0016	.0015
d_b	mean	.1993	.1976	.1979	.1988	.1988	.1992	.1988
	sd	.0014	.0018	.0014	.0021	.0021	.0012	.0016
d_{ba}	mean	.9703	.9737	.9778	.9738	.9731	.9726	.9725
	sd	.0042	.0063	.0061	.0066	.0033	.0042	.0034

Table 1. Numerical results for synthetic data

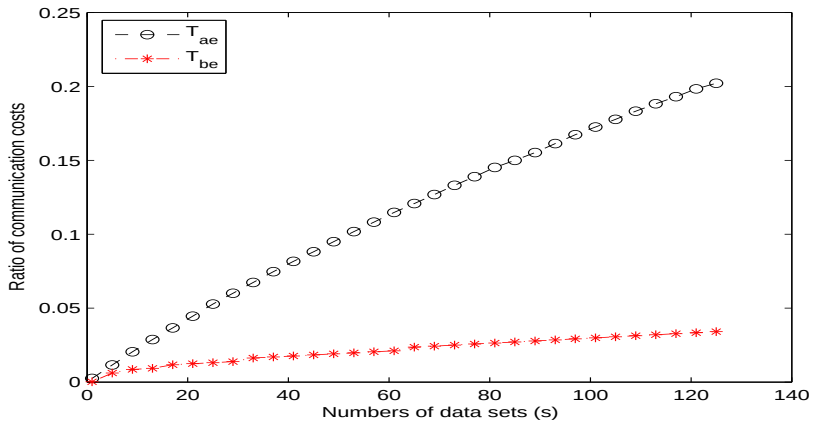

Fig. 2. Comparison of communication costs for synthetic data

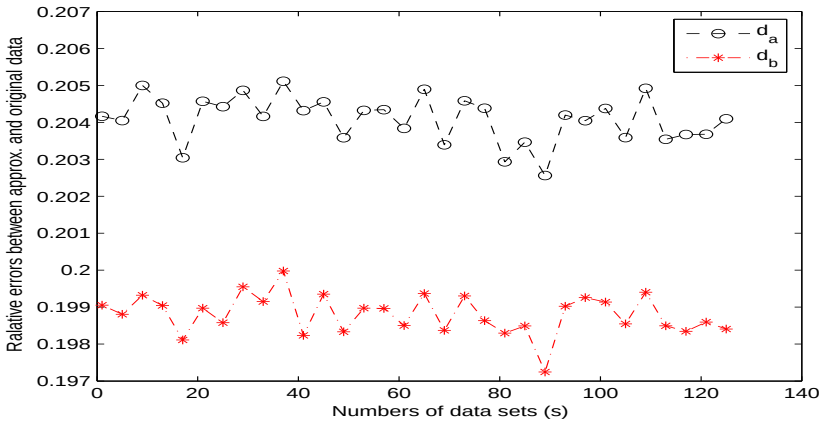

Fig. 3. Comparison of data approximate error for synthetic data

4 Distributed PCA with Updating

In this section, we develop an algorithm for computing the PCA when new data arise in the s locations. We assume that the initial time $t_0 = 0$. In our algorithm, we use a global synchronization to keep track of the updating and the evaluation of the PCA for the global data matrix. More precisely, we fix the time instants $t_1, t_2, \ldots$, when updating is stopped and the evaluation of PCA for the global data commences.

In particular, let $X_i^{(k)}$ denote the block of data of size $n_i^{(k)}$-by-p added to Processor i between $t_{k-1} < t \leq t_k$ where $k \geq 1$. The updated matrix for this time interval is denoted by

$$X_{n^{(k)}} = \begin{pmatrix} X_0^{(k)} \\ X_1^{(k)} \\ \vdots \\ X_{s-1}^{(k)} \end{pmatrix},$$

where $n^{(k)} = \sum_{i=0}^{s-1} n_i^{(k)}$ is the number of rows in $X_{n^{(k)}}$. We will use $X_{n^{(0)}}$ to denote the data matrix already present at the processors at time $t_0 = 0$. For $k \geq 0$, let

$$\bar{\mathbf{x}}_{n^{(k)}}^T = \tfrac{1}{n^{(k)}} \mathbf{e}_{n^{(k)}}^T X_{n^{(k)}} \tag{13}$$

denote the column means of $X_{n^{(k)}}$. The p-by-p covariance matrix S_k corresponding to $X_{n^{(k)}}$ is given by

$$n^{(k)} S_k = X_{n^{(k)}}^T (I - \tfrac{1}{n^{(k)}} \mathbf{e}_{n^{(k)}} \mathbf{e}_{n^{(k)}}^T) X_{n^{(k)}}. \tag{14}$$

We note that for each $k \geq 0$, the PCA of S_k can be obtained by Algorithm 1 in Section 3.

We want to compute the PCA of the global data matrix collected from $t_0 = 0$ up to t_k, $k \geq 0$. Let $\mathbb{X}_{g(k)}$ denote this global data matrix

$$\mathbb{X}_{g(k)} = \begin{pmatrix} X_{n^{(0)}} \\ X_{n^{(1)}} \\ \vdots \\ X_{n^{(k)}} \end{pmatrix}, \tag{15}$$

where $g(k) = \sum_{j=0}^{k} n^{(j)}$ is the number of rows in $\mathbb{X}_{g(k)}$. We emphasize that the data blocks $\{X_i^{(j)}\}_{i=0}^{s-1}$ always reside on their respective processors and will not be moved. The p-by-p covariance matrix $\mathbb{S}_k$ corresponding to $\mathbb{X}_{g(k)}$ is given by

$$g(k) \mathbb{S}_k = \mathbb{X}_{g(k)}^T (I - \tfrac{1}{g(k)} \mathbf{e}_{g(k)} \mathbf{e}_{g(k)}^T) \mathbb{X}_{g(k)}. \tag{16}$$

We now show that $\mathbb{S}_k$ can be obtained from the covariance matrices S_j of $\{X_{n^{(j)}}\}_{j=0}^{k}$.

Theorem 1. *For any positive integer k,*

$$g(k)\mathbb{S}_k = \sum_{j=0}^{k} n^{(j)} S_j + \sum_{j=1}^{k} \frac{g(j-1)n^{(j)}}{g(j)} \left(\bar{\mathbf{x}}_{g(j-1)} - \bar{\mathbf{x}}_{n^{(j)}}\right)\left(\bar{\mathbf{x}}_{g(j-1)} - \bar{\mathbf{x}}_{n^{(j)}}\right)^T, \tag{17}$$

where S_j and $\bar{\mathbf{x}}_{n^{(j)}}$ are given by (14) and (13) respectively, and

$$\bar{\mathbf{x}}_{g(k)}^T = \tfrac{1}{g(k)} \mathbf{e}_{g(k)}^T \mathbb{X}_{g(k)}, \tag{18}$$

is the column mean of $\mathbb{X}_{g(k)}$, which can be obtained by $\bar{\mathbf{x}}_{g(k)} = \frac{1}{g(k)} \sum_{j=0}^{k} n^{(j)} \bar{\mathbf{x}}_{n^{(j)}}$.

Proof. We use induction to prove the lemma. For $k = 0$, the equation (17) is obviously true. Let us assume that it is also true for $k-1$. We first write

$$\begin{aligned} I - \tfrac{1}{g(k)} \mathbf{e}_{g(k)} \mathbf{e}_{g(k)}^T &= \begin{pmatrix} I - \frac{1}{g(k-1)} \mathbf{e}_{g(k-1)} \mathbf{e}_{g(k-1)}^T & 0 \\ 0 & I - \frac{1}{n^{(k)}} \mathbf{e}_{n^{(k)}} \mathbf{e}_{n^{(k)}}^T \end{pmatrix} \\ &\quad + \begin{pmatrix} [\frac{1}{g(k-1)} - \frac{1}{g(k)}] \mathbf{e}_{g(k-1)} \mathbf{e}_{g(k-1)}^T & -\frac{1}{g(k)} \mathbf{e}_{g(k-1)} \mathbf{e}_{n^{(k)}}^T \\ -\frac{1}{g(k)} \mathbf{e}_{n^{(k)}} \mathbf{e}_{g(k-1)}^T & [\frac{1}{n^{(k)}} - \frac{1}{g(k)}] \mathbf{e}_{n^{(k)}} \mathbf{e}_{n^{(k)}}^T \end{pmatrix} \\ &\equiv E_k + F_k. \end{aligned}$$

For E_k, using (14) and (16), we have

$$\mathbb{X}_{g(k)}^T E_k \mathbb{X}_{g(k)} = \begin{pmatrix} \mathbb{X}_{g(k-1)} \\ X_{n^{(k)}} \end{pmatrix}^T E_k \begin{pmatrix} \mathbb{X}_{g(k-1)} \\ X_{n^{(k)}} \end{pmatrix} = g(k-1)\mathbb{S}_{k-1} + n^{(k)} S_k. \tag{19}$$

For F_k, using (13) and (18), we have

$$\begin{aligned} &\mathbb{X}_{g(k)}^T F_k \mathbb{X}_{g(k)} \\ &= \frac{1}{g(k)} \begin{pmatrix} \mathbb{X}_{g(k-1)} \\ X_{n^{(k)}} \end{pmatrix}^T \begin{pmatrix} \frac{n^{(k)}}{g(k-1)} \mathbf{e}_{g(k-1)} \mathbf{e}_{g(k-1)}^T & -\mathbf{e}_{g(k-1)} \mathbf{e}_{n^{(k)}}^T \\ -\mathbf{e}_{n^{(k)}} \mathbf{e}_{g(k-1)}^T & \frac{g(k-1)}{n^{(k)}} \mathbf{e}_{n^{(k)}} \mathbf{e}_{n^{(k)}}^T \end{pmatrix} \begin{pmatrix} \mathbb{X}_{g(k-1)} \\ X_{n^{(k)}} \end{pmatrix} \\ &= \frac{1}{g(k)} \begin{pmatrix} \mathbb{X}_{g(k-1)} \\ X_{n^{(k)}} \end{pmatrix}^T \begin{pmatrix} n^{(k)} \mathbf{e}_{g(k-1)} \bar{\mathbf{x}}_{g(k-1)}^T - n^{(k)} \mathbf{e}_{g(k-1)} \bar{\mathbf{x}}_{n^{(k)}}^T \\ -g(k-1) \mathbf{e}_{n^{(k)}} \bar{\mathbf{x}}_{g(k-1)}^T + g(k-1) \mathbf{e}_{n^{(k)}} \bar{\mathbf{x}}_{n^{(k)}}^T \end{pmatrix} \\ &= \frac{g(k-1)n^{(k)}}{g(k)} \left\{ \bar{\mathbf{x}}_{g(k-1)} \bar{\mathbf{x}}_{g(k-1)}^T - \bar{\mathbf{x}}_{g(k-1)} \bar{\mathbf{x}}_{n^{(k)}}^T - \bar{\mathbf{x}}_{n^{(k)}} \bar{\mathbf{x}}_{g(k-1)}^T + \bar{\mathbf{x}}_{n^{(k)}} \bar{\mathbf{x}}_{n^{(k)}}^T \right\} \\ &= \frac{g(k-1)n^{(k)}}{g(k)} (\bar{\mathbf{x}}_{g(k-1)} - \bar{\mathbf{x}}_{n^{(k)}})(\bar{\mathbf{x}}_{g(k-1)} - \bar{\mathbf{x}}_{n^{(k)}})^T. \end{aligned} \tag{20}$$

Adding (19) and (20), and invoking the induction hypothesis, we get (17).

For each update matrix $X_{n^{(j)}}$, by Algorithm 1 and (10), its covariance matrix S_j is given by

$$n^{(j)} S_j = R_{n^{(j)}}^T R_{n^{(j)}},$$

where $R_{n^{(j)}}$ is p-by-p upper triangular. Hence by (17),

$$g(k)\mathbb{S}_k = \sum_{j=0}^{k} R_{n^{(j)}}^T R_{n^{(j)}} + \sum_{j=1}^{k} \frac{g(j-1)n^{(j)}}{g(j)} (\bar{\mathbf{x}}_{g(j-1)} - \bar{\mathbf{x}}_{n^{(j)}})(\bar{\mathbf{x}}_{g(j-1)} - \bar{\mathbf{x}}_{n^{(j)}})^T. \tag{21}$$

Let $\mathbb{R}_{g(0)} = R_{n^{(0)}}$. Using Householder's reflections, we can recursively obtain the QR decomposition of the following $(2p+1)$-by-p matrix:

$$\begin{pmatrix} \mathbb{R}_{g(k-1)} \\ \sqrt{\frac{g(k-1)n^{(k)}}{g(k)}} (\bar{\mathbf{x}}_{g(k-1)} - \bar{\mathbf{x}}_{n^{(k)}}) \\ R_{n^{(k)}} \end{pmatrix} = \mathbb{Q}_{g(k)} \mathbb{R}_{g(k)}, \tag{22}$$

where $k \geq 1$ and $\mathbb{R}_{g(k)}$ is an p-by-p upper triangular matrix. It is easy to check from (21) that

$$g(k)\mathbb{S}_k = \mathbb{R}_{g(k)}^T \mathbb{R}_{g(k)}.$$

Hence the PCA of $\mathbb{S}_k$ can be obtained by computing the SVD of $\mathbb{R}_{g(k)}$:

$$\mathbb{R}_{g(k)} = \mathbb{U}\Sigma\mathbb{V}^T,$$

where Σ and $\mathbb{V}$ are p-by-p matrices. To get the first m principal components of the global data matrix $\mathbb{X}_{g(k)}$, we broadcast $\bar{\mathbf{x}}_{g(k)}$ and $\tilde{\mathbb{V}}$ (the first m columns of $\mathbb{V}$) to every processor. Then the m principal components of $\mathbb{X}_{g(k)}$ are given by the matrix $\tilde{\mathbb{X}}_{g(k)}$:

$$\tilde{\mathbb{X}}_{g(k)} = (I - \tfrac{1}{g(k)}\mathbf{e}_{g(k)}\mathbf{e}_{g(k)}^T)\mathbb{X}_{g(k)}\tilde{\mathbb{V}} = (\mathbb{X}_{g(k)} - \mathbf{e}_{g(k)}\bar{\mathbf{x}}_{g(k)}^T)\tilde{\mathbb{V}}.$$

From (21), we see that the PCA of the global data matrix $\mathbb{X}_{g(k)}$ at time t_k can be obtained from the R factors $R_{n^{(j)}}$ of the updated matrices S_j, for $j = 0, \ldots, k$. These R factors can be computed in turn by Algorithm 1 as in (8). Once these factors are computed, they can be assembled at a particular processor to form $\mathbb{R}_{g(k)}$ as in (22) and then the PCA of $\mathbb{S}_k$ can be computed. One potential problem is that it may create bottlenecks at certain processors if the assembling are not scheduled correctly.

5 Load Balancing

In this section, we give a procedure such that the loads among the processors will be balanced provided that the size of the data blocks are more or less the same on each processor. For notational simplicity, we will denote the set of all R factors of $\bar{X}_{n^{(k)}}$ by $\{R_{n^{(k)}}^{(0)}\}$ (see (4) and (5)), and the subsequent set of R factors of $\{R_{n^{(k)}}^{(i)}\}$ by $\{R_{n^{(k)}}^{(i+1)}\}$ (see (6) and (7)). We illustrate the main idea with $s = 8$. Figure 4 gives the flowchart of our algorithm when $s = 8$, i.e. $\ell = \log_2 s = 3$. In the figure, each time interval $(t_{j-1}, t_j]$ is divided into two phases: the computation phase where the QR decomposition are done, and the communication phase where the R factors are moved across the processors.

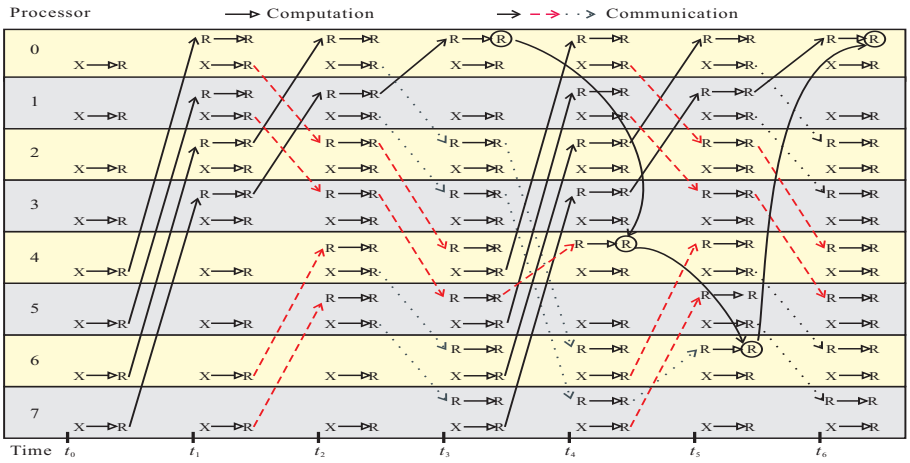

Fig. 4. Flowchart for the procedure when $s = 8$

For example, in $(t_0, t_1]$, we first compute all the R factors $\{R_{n^{(0)}}^{(0)}\}$ of $\bar{X}_{n^{(0)}}$ using Algorithm 1 (marked in the figure by $X \rightarrow R$). There are 8 of them. Then during the communication phase, half of them will be sent to Processor i, $i < s/2 = 4$, according to Algorithm 1 (marked by the solid arrows in the figure). Then in $(t_1, t_2]$, we compute all the R factors $\{R_{n^{(1)}}^{(0)}\}$ of $\bar{X}_{n^{(1)}}$ (marked by $X \rightarrow R$), and the R factors $\{R_{n^{(0)}}^{(1)}\}$ of $\{R_{n^{(0)}}^{(0)}\}$

(marked by $R \rightarrow\!\triangleright R$). Half of these R factors will be moved during the communication phase. However, in order to achieve load balancing, the factors $\{R^{(0)}_{n^{(1)}}\}$ should not be moved according to Algorithm 1 again, but according to the figure, i.e. to Processors 2, 3, 4, and 5 (marked by dashed arrows in the figure).

Continuing in this manner, we see that the R factor $R_{n^{(0)}}$ of the covariance matrix S_0 will be formed at Processor 0. (Recall that $R_{n^{(0)}} = \mathbb{R}_{g(0)}$ and $S_0 = \mathbb{S}_0$.) Also $R_{n^{(1)}}$ and $R_{n^{(2)}}$ will be formed at Processors 4 and 6 respectively (see the marked circles). Once $R_{n^{(k)}}$ are formed, they can be combined with previously obtained $\mathbb{R}_{g(k-1)}$ to form $\mathbb{R}_{g(k)}$ by using (22), provided that $\mathbb{R}_{g(k-1)}$ are sent there from the previous time-step (marked by curve arrows in the figure).

In this procedure, we assume that once $R_{n^{(k)}}$ is formed at time step $t_{k+\ell}$, it will be merged with $\mathbb{R}_{g(k-1)}$ to form $\mathbb{R}_{g(k)}$, see the circled-R in Figure 4. However, one can also send all these $R_{n^{(k)}}$ to a central processor, where all the $\mathbb{R}_{g(k)}$ are formed. The nice thing about this alternate approach is that if for some reasons, $R_{n^{(k)}}$ arrive to the central processor before $R_{n^{(j)}}$, for some $j < k$, then we can still form the $\mathbb{R}_{g(k)}$ at the central processor without waiting for $R_{n^{(j)}}$. Of course, $\mathbb{R}_{g(k)}$ so formed is the R factors of $\mathbb{X}_{g(k)}$ without the update block $X_{n^{(j)}}$, i.e. it is equivalent to $X_{n^{(j)}} = O$ in (15). When $R_{n^{(j)}}$ arrives at a later time, we can do the updating of $\mathbb{R}_{g(k)}$ first, and then include the contribution of $X_{n^{(j)}}$.

6 Conclusions

In this paper, we propose a new algorithm for finding the global PCA of distributed data sets. Our method works directly with the data matrices and has a communications requirement of only $O(p^2\lceil \log_2 s \rceil)$, (i.e., independent of n, the number of observations, which is very large). As compared against the DPCA algorithm [11], our algorithm introduces no local PCA approximation errors. We also consider data updating, and we present a method for computing the PCA for the new extended data sets after new data are added.

References

1. Z. Bai, J. Demmel, J. Dongarra, A. Petitet, H. Robinson, and K. Stanley, The Spectral Decomposition of Nonsymmetric Matrices on Distributed Memory Parallel Computers, *SIAM J. Sci. Comput.*, 18(5): 1446–1461, 1997.
2. D. Boley, Principal Direction Divisive Partitioning, *Data Min. Knowl. Discov.*, 2(4): 325–344, 1998.
3. G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, 3rd ed., 1996.
4. H. Hotelling, Analysis of a Complex of Statistical Variables into Principal Components, *J. Educ. Psych.*, 24 (): 417–441, 498–520, 1933.
5. J. E. Jackson, *User's Guide to Principal Components*, Wiley, New York, 1991.
6. I. T. Jolliffe, *Principal Component Analysis*, Springer-Verlag, 1986.
7. H. Kargupta, W. Y. Huang, K. Sivakumar, and E. Johnson, Distributed Clustering Using Collective Principal Component Analysis, *Knowl. Inf. Syst.*, 3(4): 422–448, 2001.
8. J. B. Lee, A. S. Woodyatt, and M. Berman, Enhancement of High Spectral Resolution Remote Sending Data by a Noise-Adjusted Principal Component Transform, *IEEE Trans. Geosci. Remote Sensing*, 28(3):295-304, May 1990.
9. R. H. Lilien, H. Farid, and B. R. Donald, Probabilistic Disease Classification of Expression-Dependent Proteomic Data from Mass Spectrometry of Human Serum, *J. Comp. Biol.*, 10 (6): 925–946, 2003.
10. K. Pearson, On Lines and Planes of Closest Fit to Systems of Points in Space, *Phil. Mag.*, 2 (6): 559–572, 1901.
11. Y. M. Qu, G. Ostrouchov, N. Samatova, and A. Geist, *Principal Component Analysis for Dimension Reduction in Massive Distributed Data Sets*, Proceedings to the Second SIAM International Conference on Data Mining, April 2002.
12. E. Rabani and S. Toledo, *Out-of-Core SVD and QR Decompositions*, in Proceedings of the 10th SIAM Conference on Parallel Processing for Scientific Computing, Norfolk, Virginia, March 2001.
13. E. J. Wegman, Huge Data Sets and the Frontiers of Computational Feasibility, *J. Comput. Graph. Statist.*, 4 (4): 281–295, 1995.