An Iterative Divide-and-Merge-Based Approach for Solving Large-Scale Least Squares Problems

Chi-Yuan Yeh, Yu-Ting Peng, and Shie-Jue Lee, *Member*, *IEEE*

Abstract—Singular value decomposition (SVD) is a popular decomposition method for solving least squares estimation (LSE) problems. However, for large data sets, applying SVD directly on the coefficient matrix is very time consuming and memory demanding in obtaining least squares solutions. In this paper, we propose an iterative divide-and-merge-based estimator for solving large-scale LSE problems. Iteratively, the LSE problem to be solved is processed and transformed to equivalent but smaller LSE problems. In each iteration, the input matrices are subdivided into a set of small submatrices. The submatrices are decomposed by SVD, respectively, and the results are merged, and the resulting matrices become the input of the next iteration. The process is iterated until the resulting matrices are small enough which can then be solved directly and efficiently by SVD. The number of iterations required is determined dynamically according to the size of the input data set. As a result, the requirements in time and space for finding least squares solutions are greatly improved. Furthermore, the decomposition and merging of the submatrices in each iteration can be independently done in parallel. The idea can be easily implemented in MapReduce and experimental results show that the proposed approach can solve large-scale LSE problems effectively.

Index Terms—Linear system, matrix decomposition, error minimization, least squares solution, large-scale data set, batch SVD, MapReduce

1 Introduction

Multiple linear regression analysis is a technique for modeling the relationship between a dependent (output) variable and one or more independent (input) variables [1], [2]. It has been widely used for prediction and forecasting in various fields, including engineering, physical sciences, chemical sciences, biological sciences, social sciences, economics, and management [3], [4]. Usually, the resulting system of linear equations may not be consistent and an exact solution of the system does not exist. In such situations, one looks for a least squares solution instead which is an approximate solution minimizing the sum of squares of errors [5].

Many decomposition-based methods have been proposed for solving the least squares problem [6], [7], [8], [9], [10], [11]. Björck and Yuan [8] proposed three algorithms to find linearly independent rows of the matrix by LU factorization. The Cholesky decomposition is roughly twice as efficient as the LU decomposition [12]. The QR decomposition [6] can provide a more accurate solution than the Cholesky decomposition, but may take more time in computation. If a matrix is rank deficient, there are infinitely many solutions to the least squares problem. In this case, singular value decomposition (SVD) can be used [6]. Foster [10] proposed an algorithm using a sequence of QR decompositions, indicating that the proposed algorithm is slightly worse than the truncated SVD method. Giraud et al. [13] proposed a posteriori reorthogonalization technique, and claimed that the solution obtained is close to the desired one produced by SVD. Lee and Ouyang [9] proposed a recursive SVD-based least squares estimator which considers one row of the matrix at a time.

In many application domains, e.g., astronomy, bio-information engineering, web mining, and data mining, a large-scale training data set may contain hundreds of thousands of data patterns which correspond to a linear system consisting of hundreds of thousands of linear equations. Finding the least squares solution efficiently in this case is a big challenge. Many parallel or iterative algorithms have been proposed [14], [15], [16], [17], [18], [19], [20], [21], [22], [23]. However, they focus on how to speed up the computation of SVD. Further computation is required for deriving the solutions. For example, Vogel and Wade [15] proposed two separate approaches to compute partial SVD for very large ill-conditioned matrices. The first approach is based on subspace iteration, while the second approach is based on the Lanczos method. Zamiri-Jafarian and Gulak [18] proposed an iterative algorithm to estimate the SVD of a multiple-input multiple-output channel from the received signal. The proposed algorithm is based on the constrained minimum mean-square error criterion. Konda and Nakamura [21] proposed a parallel divide-and-conquer algorithm for bidiagonal SVD to reduce the execution time of SVD. Bečka and Vajteršic [16] proposed a parallel two-side block-Jacobi algorithm to reduce the execution time of SVD. Wei and Lin [23] introduced the block Lanczos technique to replace the original exact SVD in each iteration

• *The authors are with the Department of Electrical Engineering, National Sun Yat-Sen University, Kaohsiung 80424, Taiwan. E-mail: {yuan, yuting}@water.ee.nsysu.edu.tw, leesj@mail.ee.nsysu.edu.tw.*

Manuscript received 13 Nov. 2011; revised 6 May 2012; accepted 7 May 2012; published online 22 May 2012.
Recommended for acceptance by K. Li.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2011-11-0830.
Digital Object Identifier no. 10.1109/TPDS.2012.161.

by solving it approximately. They also proposed to utilize the subspace obtained in the previous iteration to start the block Lanczos procedure.

For large data sets, applying SVD directly on the coefficient matrix to finding optimal solutions in large-scale LSE problems is usually both time and space demanding. In this paper, we propose a least squares estimator based on an iterative divide-and-merge scheme to overcome these difficulties. In each iteration, the input matrices are subdivided into a set of submatrices. The submatrices are decomposed by SVD and the results are merged, and the resulting matrices become the input of the next iteration. The process is iterated until the resulting matrices are small enough which can then be solved directly and efficiently by SVD. The number of iterations required is determined dynamically according to the size of the input data set. As a result, the requirements in time and space for finding least squares solutions are greatly improved. Furthermore, the decomposition and merging of the submatrices in each iteration can be independently done in parallel. The novelty of our proposed method is that we do not deal with the computation of SVD of the large coefficient matrix itself. Instead, we decompose the LSE problem to be solved into a collection of small LSE problems in each iteration. So, we only need to do the computation of SVD on small matrices. The correctness of the method is proved. A full complexity analysis is also given. The idea can be easily implemented in MapReduce on the Hadoop distributed platform which can run a job in parallel on a collection of computers [24], [25], [26], [27], [28]. Experimental results demonstrate the effectiveness of the proposed approach.

The rest of this paper is organized as follows: Section 2 introduces the least squares problem. Section 3 presents a brief description about the batch SVD-based least squares estimator. Section 4 describes the proposed iterative divide-and-merge approach. Section 5 describes the implementation using MapReduce on the Hadoop distributed platform. Experimental results are presented in Section 6. Finally, a conclusion is given in Section 7.

2 Least Squares Problem

Let $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$ be a set of N training data patterns where $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,m}] \in \mathbb{R}^m$ and $y_i \in \mathbb{R}$, $i = 1, \ldots, N$. A multiple linear regression model fit to these points is described as

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m, \tag{1}$$

where $\beta_0, \beta_1, \ldots, \beta_m$ are regression coefficients. Substituting the data patterns into (1) yields the following system of N linear equations:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta}, \tag{2}$$

where

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & \ldots & x_{1,m} \\ 1 & x_{2,1} & \ldots & x_{2,m} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N,1} & \ldots & x_{N,m} \end{bmatrix}, \tag{3}$$

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \ \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_m \end{bmatrix} \tag{4}$$

with the sizes $N\times n$, $N\times 1$, and $n\times 1$, respectively, and $n = m + 1$. The least squares problem is to find a vector $\boldsymbol{\beta}$ that minimizes $\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|$, i.e.,

$$\min_{\boldsymbol{\beta}} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\| \tag{5}$$

with respect to the euclidean inner product on $\mathbb{R}^N$. The vector $\boldsymbol{\beta}$ satisfying (5) is called a least squares solution of (2).

3 Batch SVD-Based Estimator

The least squares solution of (2) can be obtained by applying singular value decomposition [6] directly as follows: First, $\mathbf{X}$ is decomposed as the multiplication of three matrices

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T, \tag{6}$$

where $\mathbf{U}$ is an $N\times N$ orthogonal matrix, $\mathbf{V}$ is an $n\times n$ orthogonal matrix, and Σ is an $N\times n$ diagonal matrix with $(\Sigma)_{11} \geq (\Sigma)_{22} \geq \cdots \geq (\Sigma)_{rr} > 0$, $r \leq n$, and all the other entries being 0. Note that $(\Sigma)_{ii}^2$, $1 \leq i \leq n$, are eigenvalues of $\mathbf{X}^T\mathbf{X}$ or $\mathbf{X}\mathbf{X}^T$. The corresponding eigenvectors of $\mathbf{X}\mathbf{X}^T$ form the columns of $\mathbf{U}$, and the corresponding eigenvectors of $\mathbf{X}^T\mathbf{X}$ form the columns of $\mathbf{V}$. Let

$$\Sigma = \begin{bmatrix} \Sigma' & \mathbf{0}_{r\times(n-r)} \\ \mathbf{0}_{(N-r)\times r} & \mathbf{0}_{(N-r)\times(n-r)} \end{bmatrix}, \tag{7}$$

where Σ' is the $r \times r$ principal submatrix of Σ. Also, let

$$\mathbf{U} = [\,\mathbf{U}' \quad \mathbf{U}''\,], \ \mathbf{V}^T = \begin{bmatrix} \mathbf{V}'^T \\ \mathbf{V}''^T \end{bmatrix}, \tag{8}$$

where $\mathbf{U}'$ is an $N\times r$ matrix, $\mathbf{U}''$ is an $N\times(N-r)$ matrix, $\mathbf{V}'$ is an $n\times r$ matrix, and $\mathbf{V}''$ is an $n\times(n-r)$ matrix. Then, solving (5) is equivalent to solving the following problem:

$$\min_{\boldsymbol{\beta}} \|\mathbf{U}'^T\mathbf{Y} - \Sigma'\mathbf{V}'^T\boldsymbol{\beta}\|, \tag{9}$$

whose solution is

$$\boldsymbol{\beta}^* = \left(\Sigma'\mathbf{V}'^T\right)^+\mathbf{U}'^T\mathbf{Y}, \tag{10}$$

where $(\Sigma'\mathbf{V}'^T)^+$ is the pseudoinverse of $\Sigma'\mathbf{V}'^T$. Note that

$$\begin{aligned} (\Sigma'\mathbf{V}'^T)^+ &= (\mathbf{V}'^T)^T(\mathbf{V}'^T(\mathbf{V}'^T)^T)^{-1}(\Sigma'^T\Sigma')^{-1}\Sigma'^T \\ &= \mathbf{V}'(\mathbf{V}'^T\mathbf{V}')^{-1}\Sigma'^{-1}(\Sigma'^T)^{-1}\Sigma'^T \\ &= \mathbf{V}'\Sigma'^{-1}. \end{aligned} \tag{11}$$

Therefore, the least squares solution of (2) is

$$\boldsymbol{\beta}^* = \mathbf{V}'\Sigma'^{-1}\mathbf{U}'^T\mathbf{Y}. \tag{12}$$

4 Iterative Divide-and-Merge Estimator

When the number of training patterns, N, is large in an application, the batch SVD-based method presented previously is very time consuming and memory demanding in obtaining the optimal solution. We propose an iterative

divide-and-merge SVD-based least squares estimator for overcoming these difficulties. We do not deal with the computation of SVD of the matrix $\mathbf{X}$ itself. Instead, we divide the LSE problem to be solved into a collection of small LSE problems in each iteration, and we only need to do the computation of SVD on small matrices. As a result, the requirements in time and space for finding least squares solutions are greatly improved. The proposed estimator consists of a number of iterations, beginning with iteration 1. For iteration j, $\mathbf{X}(j)$ and $\mathbf{Y}(j)$ are the input matrices, and $\mathbf{X}(j+1)$ and $\mathbf{Y}(j+1)$ are the output matrices. Initially, we set $j=1$, $N(1)=N$, $\mathbf{X}(1)=\mathbf{X}$, and $\mathbf{Y}(1)=\mathbf{Y}$. In each iteration, three steps are performed. In step 1, $\mathbf{X}(j)$ is divided into submatrices. In step 2, each submatrix is decomposed into three matrices by SVD. In step 3, two smaller matrices, $\mathbf{X}(j+1)$ and $\mathbf{Y}(j+1)$, are obtained. The process is iterated until the resulting matrices are small enough which can then be solved directly and efficiently by SVD. Details are described below. Also, an illustrating example is given in Section 1 of the supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.161.

4.1 Problem Reduction

Given $\mathbf{X}(j)$ and $\mathbf{Y}(j)$ as the input, in iteration j, we divide $\mathbf{X}(j)$ into s_j submatrices such that $\mathbf{X}(j)=\bigcup_{i=1}^{s_j}\mathbf{X}_{j,i}$ where $\mathbf{X}_{j,i}$ is an $N_{j,i}\times n$ matrix, $N_{j,i}>0$, $N(j)$ is the number of rows in $\mathbf{X}(j)$, and $N(j)=\sum_{i=1}^{s_j}N_{j,i}$. Note that $N_{j,i}\geq n$. Usually, we set $s_j=\lceil\frac{N(j)}{L}\rceil$ where L is a user-specified constant greater than n, and $N_{j,1}=N_{j,2}=\cdots=N_{j,s_j-1}=L$ and $N_{j,s_j}=N(j)-(s_j-1)L$. Similarly, we divide $\mathbf{Y}(j)$ into s_j submatrices, i.e., $\mathbf{Y}(j)=\bigcup_{i=1}^{s_j}\mathbf{Y}_{j,i}$. Then, we have

$$\min_{\boldsymbol{\beta}}\|\mathbf{Y}-\mathbf{X}\boldsymbol{\beta}\|=\min_{\boldsymbol{\beta}}\left\|\begin{bmatrix}\mathbf{Y}_{j,1}\\ \mathbf{Y}_{j,2}\\ \vdots\\ \mathbf{Y}_{j,s_j}\end{bmatrix}-\begin{bmatrix}\mathbf{X}_{j,1}\\ \mathbf{X}_{j,2}\\ \vdots\\ \mathbf{X}_{j,s_j}\end{bmatrix}\boldsymbol{\beta}\right\|. \tag{13}$$

By applying SVD, we have

$$\mathbf{X}_{j,i}=\mathbf{U}_{j,i}\boldsymbol{\Sigma}_{j,i}\mathbf{V}_{j,i}^T, \tag{14}$$

where $\mathbf{U}_{j,i}$ is an $N_{j,i}\times N_{j,i}$ orthogonal matrix, $\mathbf{V}_{j,i}$ is an $n\times n$ orthogonal matrix, and $\boldsymbol{\Sigma}_{j,i}$ is an $N_{j,i}\times n$ diagonal matrix with $r_{j,i}$ nonzero diagonal entries and

$$\boldsymbol{\Sigma}_{j,i}=\begin{bmatrix}\boldsymbol{\Sigma}'_{j,i} & \mathbf{0}_{r_{j,i}\times(n-r_{j,i})}\\ \mathbf{0}_{(N_{j,i}-r_{j,i})\times r_{j,i}} & \mathbf{0}_{(N_{j,i}-r_{j,i})\times(n-r_{j,i})}\end{bmatrix}, \tag{15}$$

where $\boldsymbol{\Sigma}'_{j,i}$ is the $r_{j,i}\times r_{j,i}$ principal submatrix of $\boldsymbol{\Sigma}_{j,i}$. Also, let

$$\mathbf{U}_{j,i}=\begin{bmatrix}\mathbf{U}'_{j,i} & \mathbf{U}''_{j,i}\end{bmatrix},\ \mathbf{V}_{j,i}^T=\begin{bmatrix}\mathbf{V}'^T_{j,i}\\ \mathbf{V}''^T_{j,i}\end{bmatrix}, \tag{16}$$

where $\mathbf{U}'_{j,i}$ is an $N_{j,i}\times r_{j,i}$ matrix, $\mathbf{U}''_{j,i}$ is an $N_{j,i}\times(N_{j,i}-r_{j,i})$ matrix, $\mathbf{V}'_{j,i}$ is an $n\times r_{j,i}$ matrix, and $\mathbf{V}''_{j,i}$ is an $n\times(n-r_{j,i})$ matrix. Let

$$\mathbf{Y}(j+1)=\begin{bmatrix}\mathbf{Y}_1(j)\\ \mathbf{Y}_2(j)\\ \vdots\\ \mathbf{Y}_{s_j}(j)\end{bmatrix}=\begin{bmatrix}\mathbf{U}'^T_{j,1}\mathbf{Y}_{j,1}\\ \mathbf{U}'^T_{j,2}\mathbf{Y}_{j,2}\\ \vdots\\ \mathbf{U}'^T_{j,s_j}\mathbf{Y}_{j,s_j}\end{bmatrix}, \tag{17}$$

$$\mathbf{X}(j+1)=\begin{bmatrix}\mathbf{X}_1(j)\\ \mathbf{X}_2(j)\\ \vdots\\ \mathbf{X}_{s_j}(j)\end{bmatrix}=\begin{bmatrix}\boldsymbol{\Sigma}'_{j,1}\mathbf{V}'^T_{j,1}\\ \boldsymbol{\Sigma}'_{j,2}\mathbf{V}'^T_{j,2}\\ \vdots\\ \boldsymbol{\Sigma}'_{j,s_j}\mathbf{V}'^T_{j,s_j}\end{bmatrix}, \tag{18}$$

where $\mathbf{Y}_i(j)$ is an $r_{j,i}\times 1$ matrix, $\mathbf{Y}(j+1)$ is an $N(j+1)\times 1$ matrix, $\mathbf{X}_i(j)$ is an $r_{j,i}\times n$ matrix, $\mathbf{X}(j+1)$ is an $N(j+1)\times n$ matrix, and $N(j+1)=\sum_{i=1}^{s_j}r_{j,i}$. Then, the optimization problem

$$\min_{\boldsymbol{\beta}}\|\mathbf{Y}(j)-\mathbf{X}(j)\boldsymbol{\beta}\| \tag{19}$$

is equivalent to the following optimization problem:

$$\min_{\boldsymbol{\beta}}\|\mathbf{Y}(j+1)-\mathbf{X}(j+1)\boldsymbol{\beta}\|. \tag{20}$$

The solution of (20) is identical to that of (19). The equivalence between (19) and (20) is shown below.

Lemma 1. *Let $\mathbf{U}$ be an orthogonal matrix. Suppose $\mathbf{B}$ is another matrix and $\mathbf{U}^T\mathbf{B}$ is defined. Then, we have $\|\mathbf{U}^T\mathbf{B}\|=\|\mathbf{B}\|$.*

Proof. Since $\mathbf{U}$ is orthogonal, $\mathbf{U}\mathbf{U}^T=\mathbf{U}^T\mathbf{U}=\mathbf{I}$. Therefore, we have

$$\begin{aligned}\|\mathbf{U}^T\mathbf{B}\|&=\sqrt{trace((\mathbf{U}^T\mathbf{B})^T(\mathbf{U}^T\mathbf{B}))}\\ &=\sqrt{trace(\mathbf{B}^T\mathbf{B})}\\ &=\|\mathbf{B}\|.\end{aligned} \tag{21}$$

□

Lemma 2. *Let $\mathbf{U}(j)$ be defined as*

$$\mathbf{U}(j)=\begin{bmatrix}\mathbf{U}_{j,1} & \cdots & \mathbf{0}\\ \vdots & \ddots & \vdots\\ \mathbf{0} & \cdots & \mathbf{U}_{j,s_j}\end{bmatrix}, \tag{21}$$

where $\mathbf{U}_{j,i}$, $1\leq i\leq s_j$, is obtained by (14). Then, $\mathbf{U}(j)$ is an orthogonal matrix.

Proof. Since every $\mathbf{U}_{j,i}$ is orthogonal, $\mathbf{U}(j)$ is also orthogonal. □

Theorem 1. *The two optimization problems (19) and (20) are equivalent.*

Proof. Note that

$$\begin{aligned}&\min_{\boldsymbol{\beta}}\|\mathbf{Y}(j)-\mathbf{X}(j)\boldsymbol{\beta}\|\\ &\quad=\min_{\boldsymbol{\beta}}\left\|\begin{bmatrix}\mathbf{Y}_{j,1}\\ \mathbf{Y}_{j,2}\\ \vdots\\ \mathbf{Y}_{j,s_j}\end{bmatrix}-\begin{bmatrix}\mathbf{U}_{j,1}\boldsymbol{\Sigma}_{j,1}\mathbf{V}_{j,1}^T\\ \mathbf{U}_{j,2}\boldsymbol{\Sigma}_{j,2}\mathbf{V}_{j,2}^T\\ \vdots\\ \mathbf{U}_{j,s_j}\boldsymbol{\Sigma}_{j,s_j}\mathbf{V}_{j,s_j}^T\end{bmatrix}\boldsymbol{\beta}\right\|.\end{aligned} \tag{22}$$

Let $\mathbf{U}(j)$ be the matrix given in (21) and

$$\boldsymbol{\Sigma}(j)=\begin{bmatrix}\boldsymbol{\Sigma}_{j,1} & \cdots & \mathbf{0}\\ \vdots & \ddots & \vdots\\ \mathbf{0} & \cdots & \boldsymbol{\Sigma}_{j,s_j}\end{bmatrix}. \tag{23}$$

Equation (22) can be rewritten as

$$\min_{\boldsymbol{\beta}} \left\| \begin{bmatrix} \mathbf{Y}_{j,1} \\ \mathbf{Y}_{j,2} \\ \vdots \\ \mathbf{Y}_{j,s_j} \end{bmatrix} - \mathbf{U}(j)\boldsymbol{\Sigma}(j) \begin{bmatrix} \mathbf{V}_{j,1}^T \\ \vdots \\ \mathbf{V}_{j,s_j}^T \end{bmatrix} \boldsymbol{\beta} \right\|. \tag{24}$$

By Lemmas 1 and 2, (24) can be expressed as

$$\begin{aligned} &\min_{\boldsymbol{\beta}} \left\| \begin{bmatrix} \mathbf{Y}_{j,1} \\ \mathbf{Y}_{j,2} \\ \vdots \\ \mathbf{Y}_{j,s_j} \end{bmatrix} - \mathbf{U}(j)\boldsymbol{\Sigma}(j) \begin{bmatrix} \mathbf{V}_{j,1}^T \\ \vdots \\ \mathbf{V}_{j,s_j}^T \end{bmatrix} \boldsymbol{\beta} \right\| \\ &= \min_{\boldsymbol{\beta}} \left\| \mathbf{U}(j)^T \left(\begin{bmatrix} \mathbf{Y}_{j,1} \\ \mathbf{Y}_{j,2} \\ \vdots \\ \mathbf{Y}_{j,s_j} \end{bmatrix} - \mathbf{U}(j)\boldsymbol{\Sigma}(j) \begin{bmatrix} \mathbf{V}_{j,1}^T \\ \vdots \\ \mathbf{V}_{j,s_j}^T \end{bmatrix} \boldsymbol{\beta} \right) \right\| \\ &= \min_{\boldsymbol{\beta}} \left\| \mathbf{U}(j)^T \begin{bmatrix} \mathbf{Y}_{j,1} \\ \mathbf{Y}_{j,2} \\ \vdots \\ \mathbf{Y}_{j,s_j} \end{bmatrix} - \boldsymbol{\Sigma}(j) \begin{bmatrix} \mathbf{V}_{j,1}^T \\ \mathbf{V}_{j,2}^T \\ \vdots \\ \mathbf{V}_{j,s_j}^T \end{bmatrix} \boldsymbol{\beta} \right\| \\ &= \min_{\boldsymbol{\beta}} \left\| \begin{bmatrix} \mathbf{U}_{j,1}^T\mathbf{Y}_{j,1} - \boldsymbol{\Sigma}_{j,1}\mathbf{V}_{j,1}^T\boldsymbol{\beta} \\ \mathbf{U}_{j,2}^T\mathbf{Y}_{j,2} - \boldsymbol{\Sigma}_{j,2}\mathbf{V}_{j,2}^T\boldsymbol{\beta} \\ \vdots \\ \mathbf{U}_{j,s_j}^T\mathbf{Y}_{j,s_j} - \boldsymbol{\Sigma}_{j,s_j}\mathbf{V}_{j,s_j}^T\boldsymbol{\beta} \end{bmatrix} \right\|. \end{aligned} \tag{25}$$

Let $\boldsymbol{\Sigma}_{j,i}$, $\mathbf{U}_{j,i}$, and $\mathbf{V}_{j,i}^T$ be represented as in (15) and (16), respectively. Then, (25) can be rewritten as

$$\begin{aligned} &\min_{\boldsymbol{\beta}} \left\| \begin{bmatrix} \begin{bmatrix} \mathbf{U}_{j,1}'^T \\ \mathbf{U}_{j,1}''^T \end{bmatrix} \mathbf{Y}_{j,1} - \begin{bmatrix} \boldsymbol{\Sigma}'_{j,1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}_{j,1}'^T \\ \mathbf{V}_{j,1}''^T \end{bmatrix} \boldsymbol{\beta} \\ \begin{bmatrix} \mathbf{U}_{j,2}'^T \\ \mathbf{U}_{j,2}''^T \end{bmatrix} \mathbf{Y}_{j,2} - \begin{bmatrix} \boldsymbol{\Sigma}'_{j,2} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}_{j,2}'^T \\ \mathbf{V}_{j,2}''^T \end{bmatrix} \boldsymbol{\beta} \\ \vdots \\ \begin{bmatrix} \mathbf{U}_{j,s_j}'^T \\ \mathbf{U}_{j,s_j}''^T \end{bmatrix} \mathbf{Y}_{j,s_j} - \begin{bmatrix} \boldsymbol{\Sigma}'_{j,s_j} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}_{j,s_j}'^T \\ \mathbf{V}_{j,s_j}''^T \end{bmatrix} \boldsymbol{\beta} \end{bmatrix} \right\| \\ &= \min_{\boldsymbol{\beta}} \left\| \begin{bmatrix} \begin{bmatrix} \mathbf{U}_{j,1}'^T\mathbf{Y}_{j,1} \\ \mathbf{U}_{j,1}''^T\mathbf{Y}_{j,1} \end{bmatrix} - \begin{bmatrix} \boldsymbol{\Sigma}'_{j,1}\mathbf{V}_{j,1}'^T\boldsymbol{\beta} \\ \mathbf{0} \end{bmatrix} \\ \begin{bmatrix} \mathbf{U}_{j,2}'^T\mathbf{Y}_{j,2} \\ \mathbf{U}_{j,2}''^T\mathbf{Y}_{j,2} \end{bmatrix} - \begin{bmatrix} \boldsymbol{\Sigma}'_{j,2}\mathbf{V}_{j,2}'^T\boldsymbol{\beta} \\ \mathbf{0} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \mathbf{U}_{j,s_j}'^T\mathbf{Y}_{j,s_j} \\ \mathbf{U}_{j,s_j}''^T\mathbf{Y}_{j,s_j} \end{bmatrix} - \begin{bmatrix} \boldsymbol{\Sigma}'_{j,s_j}\mathbf{V}_{j,s_j}'^T\boldsymbol{\beta} \\ \mathbf{0} \end{bmatrix} \end{bmatrix} \right\| \\ &= \min_{\boldsymbol{\beta}} \left\{ \left\| \begin{bmatrix} \mathbf{U}_{j,1}'^T\mathbf{Y}_{j,1} - \boldsymbol{\Sigma}'_{j,1}\mathbf{V}_{j,1}'^T\boldsymbol{\beta} \\ \mathbf{U}_{j,2}'^T\mathbf{Y}_{j,2} - \boldsymbol{\Sigma}'_{j,2}\mathbf{V}_{j,2}'^T\boldsymbol{\beta} \\ \vdots \\ \mathbf{U}_{j,s_j}'^T\mathbf{Y}_{j,s_j} - \boldsymbol{\Sigma}'_{j,s_j}\mathbf{V}_{j,s_j}'^T\boldsymbol{\beta} \end{bmatrix} \right\| + C \right\}, \end{aligned} \tag{26}$$

where C is related to $\mathbf{U}_{j,1}''^T\mathbf{Y}_{j,1}$, $\mathbf{U}_{j,2}''^T\mathbf{Y}_{j,2}, \ldots$, and $\mathbf{U}_{j,s_j}''^T\mathbf{Y}_{j,s_j}$. Note that C has nothing to do with $\boldsymbol{\beta}$ and can be dropped. Therefore, (26) is equivalent to

$$\begin{aligned} &\min_{\boldsymbol{\beta}} \left\| \begin{bmatrix} \mathbf{U}_{j,1}'^T\mathbf{Y}_{j,1} - \boldsymbol{\Sigma}'_{j,1}\mathbf{V}_{j,1}'^T\boldsymbol{\beta} \\ \mathbf{U}_{j,2}'^T\mathbf{Y}_{j,2} - \boldsymbol{\Sigma}'_{j,2}\mathbf{V}_{j,2}'^T\boldsymbol{\beta} \\ \vdots \\ \mathbf{U}_{j,s_j}'^T\mathbf{Y}_{j,s_j} - \boldsymbol{\Sigma}'_{j,s_j}\mathbf{V}_{j,s_j}'^T\boldsymbol{\beta} \end{bmatrix} \right\| \\ &= \min_{\boldsymbol{\beta}} \left\| \begin{bmatrix} \mathbf{U}_{j,1}'^T\mathbf{Y}_{j,1} \\ \mathbf{U}_{j,2}'^T\mathbf{Y}_{j,2} \\ \vdots \\ \mathbf{U}_{j,s_j}'^T\mathbf{Y}_{j,s_j} \end{bmatrix} - \begin{bmatrix} \boldsymbol{\Sigma}'_{j,1}\mathbf{V}_{j,1}'^T \\ \boldsymbol{\Sigma}'_{j,2}\mathbf{V}_{j,2}'^T \\ \vdots \\ \boldsymbol{\Sigma}'_{j,s_j}\mathbf{V}_{j,s_j}'^T \end{bmatrix} \boldsymbol{\beta} \right\| \\ &= \min_{\boldsymbol{\beta}} \|\mathbf{Y}(j+1) - \mathbf{X}(j+1)\boldsymbol{\beta}\|. \end{aligned} \tag{27}$$

This completes the proof. □

4.2 Algorithm

As stated earlier, our proposed estimator consists of a number of iterations. $\mathbf{X}(j)$ and $\mathbf{Y}(j)$ are the input matrices to iteration j, and $\mathbf{X}(j+1)$ and $\mathbf{Y}(j+1)$ are the output matrices from iteration j. Initially, we set $j = 1$, $\mathbf{X}(1) = \mathbf{X}$, $N(1) = N$, and $\mathbf{Y}(1) = \mathbf{Y}$. In iteration j, $\mathbf{X}(j)$ is divided into s_j submatrices, $\mathbf{X}_{j,1}$, $\mathbf{X}_{j,2}, \ldots$, and $\mathbf{X}_{j,s_j}$. Each submatrix $\mathbf{X}_{j,i}$, $1 \le i \le s_j$, is decomposed into three matrices by (14). The output matrices $\mathbf{Y}(j+1)$ and $\mathbf{X}(j+1)$ are then obtained by (17) and (18), respectively, and $N(j+1) = \sum_{i=1}^{s_j} r_{j,i}$. The above process is carried out in iterations 1, 2, etc. Suppose that after $z-1$ iterations, $z \ge 1$, are done, $N(z)$ is satisfactorily small, i.e., $N(z) \le L$. Then, by applying Theorem 1 iteratively, it is clear to see that solving (5) is identical to solving the following problem:

$$\min_{\boldsymbol{\beta}} \|\mathbf{Y}(z) - \mathbf{X}(z)\boldsymbol{\beta}\|. \tag{28}$$

Now, we solve (28) by SVD directly and have

$$\boldsymbol{\beta}^* = \mathbf{V}(z)'\boldsymbol{\Sigma}(z)'^{-1}\mathbf{U}(z)'^T\mathbf{Y}(z), \tag{29}$$

which is the desired least squares solution to (2). The procedure can be summarized in Algorithm 1.

Algorithm 1. The proposed iterative divide-and-merge SVD-based least square estimator

Input: $\mathbf{X}$, $\mathbf{Y}$, N, n, and L.

Output: Regression coefficients $\boldsymbol{\beta}^*$.

1: Set $\mathbf{X}(1) = \mathbf{X}$, $\mathbf{Y}(1) = \mathbf{Y}$, and $N(1) = N$.
2: $j = 1$;
3: **repeat**
4: // Step 1
5: Calculate the number of submatrices, $s_j = \left\lceil \frac{N(j)}{L} \right\rceil$;
6: Divide $\mathbf{X}(j)$ into s_j submatrices, such that $\mathbf{X}(j) = \bigcup_{i=1}^{s_j} \mathbf{X}_{j,i}$;
7: Divide $\mathbf{Y}(j)$ into s_j submatrices, such that $\mathbf{Y}(j) = \bigcup_{i=1}^{s_j} \mathbf{Y}_{j,i}$;
8: // Step 2
9: **for** $i = 1$ to s_j **do**
10: Decompose $\mathbf{X}_{j,i}$ into three matrices $\mathbf{U}_{j,i}$, $\boldsymbol{\Sigma}_{j,i}$, and $\mathbf{V}_{j,i}$ by (14);

11: **end for**
12: // Step 3
13: Obtain $\mathbf{Y}(j+1)$ by Eq.(17);
14: Obtain $\mathbf{X}(j+1)$ by Eq.(18);
15: $N(j+1) = \sum_{i=1}^{s_j} r_{j,i}$;
16: $j = j+1$;
17: **until** $N(j) \le L$;
18: Decompose $\mathbf{X}(j)$ by Eq.(6) and obtain three matrices $\mathbf{U}(j)'$, $\mathbf{\Sigma}(j)'$, and $\mathbf{V}(j)'$;
19: Calculate the optimal solution β^* by (29).

4.3 Sequential Complexity Analysis

We give a complexity comparison between the batch SVD-based estimator and our proposed estimator. We consider space complexity first. The batch SVD-based estimator has to decompose the matrix $\mathbf{X}$. The largest matrix it has to store in the memory is $\mathbf{U}$ which is an $N \times N$ matrix. Therefore, the space complexity for the batch SVD-based estimator is $O(N^2)$. For our proposed estimator, we decompose one smaller matrix, $\mathbf{X}_{j,i}$, each time. The largest matrix to be stored is $\mathbf{U}_{j,i}$ which is an $L \times L$ matrix. Therefore, the space complexity for our proposed estimator is $O(L^2)$.

Next, we consider time complexity. For the batch SVD-based estimator, the decomposition time requirement by (6) is $O(4N^2n + 8Nn^2 + 9n^3)$ [2]. In addition, the time requirement for dealing with (12) is $O(nr^2 + Nr + nr) \approx O(n^3 + Nn + n^2)$. Thus, the time complexity for the batch SVD-based estimator is

$$\begin{aligned} &O(4N^2n + 8Nn^2 + 9n^3) + O(n^3 + Nn + n^2) \\ &\quad \approx O(N^2n). \end{aligned} \tag{30}$$

For our proposed estimator, we have to consider the time requirements of all the operations performed in each iteration. In iteration 1, we have

$$s_1 = \left\lceil \frac{N(1)}{L} \right\rceil \approx \frac{N}{L}. \tag{31}$$

The decomposition time requirement for iteration 1 is

$$s_1\big(4L^2n + 8Ln^2 + 9n^3\big). \tag{32}$$

In iteration 2, we have

$$\begin{aligned} s_2 &= \left\lceil \frac{N(2)}{L} \right\rceil = \left\lceil \frac{\sum_{i=1}^{s_1} r_{j,i}}{L} \right\rceil \\ &\le \left\lceil \frac{s_1 n}{L} \right\rceil \approx \frac{N}{L}\Big(\frac{n}{L}\Big). \end{aligned} \tag{33}$$

The decomposition time requirement for iteration 2 is

$$s_2\big(4L^2n + 8Ln^2 + 9n^3\big). \tag{34}$$

Similarly, in iteration j, we have

$$s_j \approx \frac{N}{L}\Big(\frac{n}{L}\Big)^{j-1}. \tag{35}$$

The decomposition time requirement for iteration j is

$$s_j\big(4L^2n + 8Ln^2 + 9n^3\big). \tag{36}$$

Therefore, the decomposition time requirement for iteration 1 through iteration $z-1$ and for $\mathbf{X}(z)$ of (28) is

$$\begin{aligned} &\big(4L^2n + 8Ln^2 + 9n^3\big)\left(1 + \sum_{j=1}^{z-1} s_j\right) \\ &\quad \approx \big(4L^2n + 8Ln^2 + 9n^3\big)\left(1 + \sum_{j=1}^{z-1} \frac{N}{L}\Big(\frac{n}{L}\Big)^{j-1}\right) \\ &\quad = \big(4L^2n + 8Ln^2 + 9n^3\big)\left(1 + \frac{N}{L}\frac{1 - \big(\frac{n}{L}\big)^{z-1}}{1 - \frac{n}{L}}\right) \\ &\quad \approx \big(4L^2n + 8Ln^2 + 9n^3\big)\left(1 + \frac{N}{L-n}\right). \end{aligned} \tag{37}$$

Similarly, the time requirement for dealing with (17) and (18) for iteration 1 through iteration $z-1$ is

$$\begin{aligned} &\big(Ln + n^3\big)\sum_{j=1}^{z-1} s_j \\ &\quad \approx \big(Ln + n^3\big)\left(\frac{N}{L-n}\right). \end{aligned} \tag{38}$$

Finally, we have to deal with (29) whose time complexity is

$$O(nr^2 + Lr + nr) \approx O(n^3 + Ln). \tag{39}$$

Thus, the time complexity of our proposed estimator is the sum of (37), (38), and (39), i.e.,

$$\big(4L^2n + 8Ln^2 + 9n^3 + Ln + n^3\big)\left(1 + \frac{N}{L-n}\right), \tag{40}$$

which is approximate to

$$O(NLn) \tag{41}$$

for $N \gg L > n$.

5 Parallel Implementation

Apparently, the tasks of decomposing the submatrices by SVD in each iteration are independent. They can be done in parallel. Assume that there are many enough machines such that lines 9∼11 in iteration j of Algorithm 1 can be done by s_j machines running in parallel. Therefore, the decomposition of $\mathbf{X}_{j,i}$ is handled by machine i, $1 \le i \le s_j$. The major gain is for (37). Note that z iterations are required in our algorithm. We can estimate the upper bound of z as follows: From (35), we can see that

$$s_z = 1 \le \frac{N}{L}\Big(\frac{n}{L}\Big)^{z-1}, \tag{42}$$

which becomes

$$\log\frac{N}{L} + (z-1)\log\Big(\frac{n}{L}\Big) \ge 0. \tag{43}$$

After rearrangement, we have

$$z \le \frac{\log N - \log L}{\log L - \log n} + 1, \tag{44}$$

leading to

$$z \le \left\lceil \frac{\log N - \log n}{\log L - \log n} \right\rceil, \tag{45}$$

which becomes

$$z \approx \log_L N \tag{46}$$

for $N \gg L > n$.

5.1 Parallel Complexity Analysis

We adopt the parallel computation model BSP [29] to analyze our method. A BSP program is executed as a sequence of supersteps. The cost of a superstep is determined as the sum of three terms: the cost of the longest running local computation, the cost of global communication between the machines, and the cost of the barrier synchronization at the end of the superstep [30]. The cost of one superstep for s_j machines in iteration j is

$$\max_{i=1}^{s_j}(w_i) + \max_{i=1}^{s_j}(h_i g) + m, \tag{47}$$

where w_i is the time requirement for dealing with (14), (17), and (18), h_i is the number of messages sent or received by machine i, g is the communication throughput ratio (also called "bandwidth inefficiency" or "gap"), and m is the communication latency (also called "synchronization periodicity"). Note that the time complexity for (14) is $O(4L^2n + 8Ln^2 + 9n^3)$, and that for (17) and (18) is $O(Ln + n^3)$. Since each machine deals with the same sizes of matrices, we have $w_1 = w_2 = \cdots = w_{s_j}$. Therefore,

$$\max_{i=1}^{s_j}(w_i) = O\big(4L^2n + 8Ln^2 + 10n^3 + Ln\big). \tag{48}$$

Also, machine i needs to send out $\mathbf{Y}_i(j)$ and $\mathbf{X}_i(j)$ in iteration j, involving $n + n^2$ elements, and receive $\mathbf{Y}_{j+1,i}$ and $\mathbf{X}_{j+1,i}$ in iteration $j+1$, involving $L + Ln$ elements. Therefore,

$$h_1 = h_2 = \cdots = O(n + n^2 + L + Ln). \tag{49}$$

Assume that g and m are constants. The cost of iteration j is

$$\begin{aligned} &O(4L^2n + 8Ln^2 + 10n^3 + Ln) \\ &\quad + g \times O(n + n^2 + L + Ln) + m \approx O(L^2n) \end{aligned} \tag{50}$$

for $L > n$. Therefore, the total complexity of our parallel method is

$$z \times O(L^2n) \tag{51}$$

due to z iterations.

We compare the complexity of our method with that of the parallel method proposed by Bečka and Vajteršic [16] which used the two-sided block-Jacobi method for the computation of the SVD. The complexity of this method is

$$z \times \left[8c_1\frac{N^3}{L^3} + 16c_2\frac{N^3}{L^2} + O\left(\frac{2N^2}{L} + \frac{L^3}{8}\right)\right], \tag{52}$$

where c_1 and c_2 are constants [16]. For $N > L$, (52) is approximated as

$$z \times O\left(\frac{N^3}{L^2}\right). \tag{53}$$

Apparently, if $L^2n < \frac{N^3}{L^2}$, (51) is smaller than (53) and our method can run more efficiently than Bečka's method.

5.2 Dynamic Adaptation

As the LSE problem is highly regular and productivity is not an issue, our proposed approach should fit well to any parallel programming paradigm, e.g., GPU [17], [19], [20], [28] or PLAPACK [31], [32]. Since we have built up a Hadoop distributed platform [24] in our laboratory and MapReduce is a parallel programming technique supported by Hadoop, we implemented the parallel version of our proposed approach in MapReduce on this platform. For easy reference, a brief introduction to MapReduce is given in Section 2 of the supplementary file, available online.

The MapReduce framework has two phases, map and reduce. A job may contain both map and reduce phases, or it may contain only map phase. In the map phase, several map tasks which perform the Map function independently can operate simultaneously. In the reduce phase, several reduce tasks which perform the Reduce function independently can operate simultaneously. One obvious MapReduce implementation of our algorithm is described as follows: The main program treats each iteration as one job. In the job for iteration j, there are s_j input key/value pairs in the map phase each of which computes one row of $\mathbf{Y}(j+1)$ and $\mathbf{X}(j+1)$ in (17) and (18), respectively, and there is one input key/value pair in the reduce phase which collects the results from the map tasks and forms $\mathbf{Y}(j+1)$ and $\mathbf{X}(j+1)$. Note that a map task contains an input key/value pair or several input key/value pairs. Therefore, a total of $z-1$ jobs are executed. Finally, the optimal solution $\boldsymbol{\beta}^*$ is obtained by (29) in the main program. This implementation looks simple. However, it is not efficient. The reduce task in each job does not perform significant work and a lot of data movement occurs during the switch of map and reduce phases.

We develop a more efficient implementation. We estimate s_j and s_{j+1}. If $s_{j+1} > 1$, we perform iteration j in the map phase and iteration $j+1$ in the subsequent reduce phase. Therefore, two iterations are bundled in one job to reduce the amount of data movement as stated earlier. In this case, s_j input key/value pairs for the Map function and s_{j+1} input key/value pairs for the Reduce function are involved in this job. A map/reduce task contains an input key/value pair or several input key/value pairs. If $s_{j+1} = 1$, we perform iteration j in the map phase in which s_j key/value pairs for the Map function are involved. In this case, no reduce phase is required and only one iteration is performed in this job. Dynamic adaptation is developed to control the number of jobs according to the user-specified parameter L, the maximum number of rows in the submatrices for SVD in each iteration. The flowchart of the MapReduce implementation for our algorithm is shown in Section 3 of the supplementary file, available online. Note that when the stopping criterion, $N(j) \le L$, is reached, the iteration is stopped and the optimal solution $\boldsymbol{\beta}^*$ is obtained by (29). A detailed description of the main MapReduce program is given in Algorithm 2. The Map function is defined as follows:

$$\text{Map} : (k_1, [\mathbf{X}_{j,i}, \mathbf{Y}_{j,i}]) \rightarrow [(k_2, [\mathbf{X}_i(j), \mathbf{Y}_i(j)])], \tag{54}$$

where $\mathbf{X}_i(j) = \boldsymbol{\Sigma}'_{j,i}\mathbf{V}'^T_{j,i}$ and $\mathbf{Y}_i(j) = \mathbf{U}'^T_{j,i}\mathbf{Y}_{j,i}$ as shown in (17) and (18). The input key k_1 is set to $1, 2, \ldots,$ or s_j, and the intermediate key k_2 is set to $1, 2, \ldots,$ or s_{j+1}. Let $q = \lceil \frac{s_j}{s_{j+1}} \rceil$. Then, the outputs of q map functions will have the same k_2 key. We apply SVD to the input matrix $\mathbf{X}_{j,i}$ and then

produce $\mathbf{X}_i(j)$ and $\mathbf{Y}_i(j)$ as output. A detailed description of the Map function is given in Algorithm 3.

Algorithm 2. The main MapReduce program for the distributed least squares estimator
Input: $\mathbf{X}$, $\mathbf{Y}$, N, n, and L ($L > 2n$).
Output: Regression coefficients $\boldsymbol{\beta}^*$.
1: Set $\mathbf{X}(1) = \mathbf{X}$, $\mathbf{Y}(1) = \mathbf{Y}$, and $N(1) = N$.
2: $j = 1$.
3: **while** ($N(j) \leq L$) **do**
4: Calculate the number of mappers, $s_j = \lceil \frac{N(j)}{L} \rceil$;
5: Estimate the number of reducers, $s_{j+1} = \lceil \frac{s_j \times n}{L} \rceil$;
6: Divide $\mathbf{X}(j)$ and $\mathbf{Y}(j)$ into s_j submatrices;
7: **if** $s_{j+1} = 1$ **then**
8: // Run job
9: Call the Map function;
10: $j = j + 1$;
11: **else**
12: // Run job
13: Call the Map function;
14: Call the Reduce function;
15: $j = j + 2$;
16: **end if**
17: Form $\mathbf{Y}(j)$ by Eq.(17);
18: Form $\mathbf{X}(j)$ by Eq.(18);
19: **end while**
20: Decompose $\mathbf{X}(j)$ by Eq.(6) and obtain three matrices $\mathbf{U}(j)'$, $\boldsymbol{\Sigma}(j)'$, and $\mathbf{V}(j)'$;
21: Calculate the optimal solution $\boldsymbol{\beta}^*$ by Eq.(29).

Algorithm 3. The Map function
Input: $k_1 = 1, 2, \ldots,$ or s_j, $v_1 = [\mathbf{X}_{j,i}, \mathbf{Y}_{j,i}]$.
Output: $k_2 = 1, 2, \ldots,$ or s_{j+1}, $v_2 = [\mathbf{X}_i(j), \mathbf{Y}_i(j)]$.
1: Decompose $\mathbf{X}_{j,i}$ into three matrices $\mathbf{U}_{j,i}$, $\boldsymbol{\Sigma}_{j,i}$, and $\mathbf{V}_{j,i}$ by Eq.(6);
2: Obtain $\mathbf{Y}_i(j) = \mathbf{U}_{j,i}^{'T}\mathbf{Y}_{j,i}$;
3: Obtain $\mathbf{X}_i(j) = \boldsymbol{\Sigma}_{j,i}^{'}\mathbf{V}_{j,i}^{'T}$.

The Reduce function is defined as follows:

$$\begin{aligned}\text{Reduce}: \quad & (k_2, [\mathbf{X}_i(j), \mathbf{Y}_i(j)]) \rightarrow \\ & [(k_3, [\mathbf{X}_i(j+1), \mathbf{Y}_i(j+1)])].\end{aligned} \tag{55}$$

We aggregate all the outputs from the Map function of distributed machines associated with the same intermediate key and obtain $\mathbf{X}_{j+1,i}$ and $\mathbf{Y}_{j+1,i}$ where $i = 1, 2, \ldots, s_{j+1}$. Then, we apply SVD to each matrix $\mathbf{X}_{j+1,i}$ and produce $\mathbf{X}_i(j+1)$ and $\mathbf{Y}_i(j+1)$ as output. Note that the output key k_3 is arbitrarily set to 1. A detailed description of the Reduce function is given in Algorithm 4.

Algorithm 4. The Reduce function
Input: $k_2 = 1, 2, \ldots,$ or s_{j+1}, $v_2 = [\mathbf{X}_i(j), \mathbf{Y}_i(j)]$.
Output: $k_3 = 1$, $v_3 = [\mathbf{X}_i(j+1), \mathbf{Y}_i(j+1)]$.
1: $\mathbf{X}_{j+1,i} = \bigcup_{\forall i \text{ with same } k_2} \mathbf{X}_i(j)$;
2: $\mathbf{Y}_{j+1,i} = \bigcup_{\forall i \text{ with same } k_2} \mathbf{Y}_i(j)$;
3: Decompose $\mathbf{X}_{j+1,i}$ into three matrices $\mathbf{U}_{j+1,i}$, $\boldsymbol{\Sigma}_{j+1,i}$, and $\mathbf{V}_{j+1,i}$ by Eq.(6);
4: Obtain $\mathbf{Y}_i(j+1) = \mathbf{U}_{j+1,i}^{'T}\mathbf{Y}_{j+1,i}$;
5: Obtain $\mathbf{X}_i(j+1) = \boldsymbol{\Sigma}_{j+1,i}^{'}\mathbf{V}_{j+1,i}^{'T}$.

TABLE 1
The Three Data Sets for Experiments

dataset	DS-I	DS-VII	DS-III
N	5,000,000	5,000,000	5,000,000
n	11	51	101
size	1020.2 MB	4.6 GB	9.1 GB

6 Experimental Results

In this section, we show experimental results to demonstrate the efficiency of our proposed iterative divide-and-merge-based least squares estimator and the MapReduce implementation for it. For convenience, we abbreviate the conventional batch SVD-based least squares estimator [6] as BSVD-LSE, the recursive SVD-based least squares estimator [9] as RSVD-LSE, the iterative divide-and-merge-based least squares estimator as IDMSVD-LSE, and the MapReduce implementation of the iterative divide-and-merge-based least squares estimator as MRDSVD-LSE.

The implementation of RSVD-LSE is briefly described as follows: In the first iteration, the first n data patterns form an $n \times n$ matrix and SVD is applied. In the second iteration, the results obtained from the first iteration and the $(n+1)$th data pattern form an $(n+1) \times n$ matrix and SVD is applied. In the third iteration, the results obtained from the second iteration and the $(n+2)$th data pattern form an $(n+1) \times n$ matrix and SVD is applied. The above process continues until the Nth data pattern is considered. Note that in each iteration, except the first one, a matrix of size $(n+1) \times n$ is considered. Therefore, the memory demand is usually not an issue and the problem of out-of-memory is very unlikely to happen.

Note that BSVD-LSE, RSVD-LSE, and IDMSVD-LSE run on a single machine, while MRDSVD-LSE runs on a cluster of machines. Experiments with three synthetical data sets are conducted. The unit of execution time is second (sec). Detailed information about the three data sets is shown in Table 1, in which "N" means the number of training patterns, "n" means the number of input variables plus one, "MB" means Megabyte, and "GB" means Gigabyte. In the following, we use three SUN Fire X4150 servers with Intel(R) Xeon(TM) CPU X5420 2.50 GHz to conduct the experiments. Each server contains eight cores and 16 GB of RAM. The operating system is CentOS 5.3. The programming language used is JAVA 1.6. Besides, JAMA, a basic linear algebra package for Java, is used to perform matrix inversion, SVD decomposition, etc.

6.1 Experiment I

First, we compare BSVD-LSE, RSVD-LSE, and IDMSVD-LSE on the efficiency of running the three data sets. A comparison on execution time is listed in Table 2, in which "L" means the maximum size of submatrices, "z" means the number of iterations, and the dash "$-$" means "Out of memory" error encountered without completion.Clearly, IDMSVD-LSE runs much faster than the other two methods for all the three data sets. For example, IDMSVD-LSE takes 114.439 seconds for the DS-I data set, while BSVD-LSE takes 290.943 seconds and RSVD-LSE takes 3,901.923 seconds. For DS-III, BSVD-LSE fails to get a solution. RSVD-LSE takes

TABLE 2
A Comparison on Execution Time for Experiment I

method	dataset		
	DS-I	DS-II	DS-III
BSVD-LSE	290.943	3901.923	–
RSVD-LSE	531.733	16771.796	120285.483
IDMSVD-LSE ($L = 2000$)	114.439 ($z = 3$)	766.542 ($z = 4$)	2365.221 ($z = 4$)

12,0285.483 seconds to have a solution. However, IDMSVD-LSE takes only 2,365.221 seconds to get a solution.

We compare the memory consumption among BSVD-LSE, RSVD-LSE, and IDMSVD-LSE. BSVD-LSE decomposes an $N \times n$ matrix and obtains an $N \times N$ matrix. Its memory consumption is $O(N^2)$. RSVD-LSE decomposes an $(n+1) \times n$ matrix and obtains an $(n+1) \times (n+1)$ in each iteration. Its memory consumption is $O(n^2)$. IDMSVD-LSE decomposes an $L \times n$ matrix and obtains an $L \times L$ matrix in each iteration. Its memory consumption is $O(L^2)$. The memory consumption for each data set by each method is listed in Table 3. For example, BSVD-LSE uses 14.75 GB for DS-II and cannot work with DS-III due to an out-of-memory error. RSVD-LSE uses 0.10 GB for DS-II and 0.13 GB for DS-III. IDMSVD-LSE uses 0.49 GB for DS-II and 0.75 GB for DS-III using $L = 2{,}000$.

6.2 Experiment II

In this experiment, we compare IDMSVD-LSE and MRDSVD-LSE on the efficiency of running the three data sets. First, we set $L = 2{,}000$ for IDMSVD-LSE and MRDSVD-LSE. A comparison on execution time is listed in Table 4, in which "P" means the number of machines for running MRDSVD-LSE. Due to the processing mechanism of MapReduce, there is overhead for both map and reduce tasks for MRDSVD-LSE. Some setup overhead is required for node assignment before the map task is performed, and another overhead is required for shuffling and sorting before the reduce task is performed. MapReduce uses, in default, two cores in each machine. So, MRDSVD-LSE can run two tasks at one time in a machine. IDMSVD-LSE, however, can only run its job in one core no matter how many machines are available. For P machines, suppose IDMSVD-LSE takes t_e seconds for a data set. MRDSVD-LSE takes about $t_o + t_e/(2P)$ seconds where t_o is the overhead and $t_e/(2P)$ is the time required for map and reduce tasks. For the data set DS-I, the dimension n is small and t_e is small too. In the case of $P = 1$, t_o is slightly larger than $t_e/2$. Therefore, MRDSVD-LSE runs slower than IDMSVD-LSE.

TABLE 3
A Comparison on Memory Consumption, in GB, for Experiment I

method	dataset		
	DS-I	DS-II	DS-III
BSVD-LSE	3.80	14.75	–
RSVD-LSE	0.08	0.10	0.13
IDMSVD-LSE ($L = 2000$)	0.42 ($z = 3$)	0.49 ($z = 4$)	0.75 ($z = 4$)

TABLE 4
A Comparison on Execution Time with $L = 2{,}000$ for Experiment II

method	dataset		
	DS-I	DS-II	DS-III
IDMSVD-LSE	114.439	766.542	2365.221
MRDSVD-LSE ($P = 1$)	116.890	732.996	2190.796
MRDSVD-LSE ($P = 2$)	67.894	404.432	1200.162
MRDSVD-LSE ($P = 3$)	54.610	276.628	832.348

In the case of $P = 2$ and $P = 3$, $t_o + t_e/4$ and $t_o + t_e/6$ are smaller than t_e. Therefore, MRDSVD-LSE runs faster than IDMSVD-LSE for these two cases. For the data sets DS-II and DS-III, the dimension n is large and t_e is large too. The overhead t_o is smaller than $t_e/(2P)$ in all the three cases $P = 1$, $P = 2$, and $P = 3$. Therefore, MRDSVD-LSE runs faster than IDMSVD-LSE in each case for DS-II and DS-III.

From Table 4, we can see that MRDSVD-LSE runs slower than IDMSVD-LSE for DS-I in the case of $P = 1$. MRDSVD-LSE takes 116.890 seconds and IDMSVD-LSE takes 114.439 seconds. However, for DS-II and DS-III, MRDSVD-LSE runs faster than IDMSVD-LSE when $P = 1$. For example, MRDSVD-LSE takes 732.996 seconds for the DS-II data set, while IDMSVD-LSE takes 766.542 seconds. When P increases, more tasks can be done in parallel in more machines and the advantages of MRDSVD-LSE become more significant. Apparently, MRDSVD-LSE with $P = 2$ or $P = 3$ runs much faster than IDMSVD-LSE for all the three data sets, since more machines can share map/reduce tasks. For example, IDMSVD-LSE takes 114.439 seconds for the DS-I data set, while MRDSVD-LSE takes 67.984 seconds and 54.610 seconds in the case of $P = 2$ and $P = 3$, respectively. For DS-III, IDMSVD-LSE takes 2,365.221 seconds to get a solution. However, MRDSVD-LSE takes 1,200.162 seconds and 832.348 seconds in the case of $P = 2$ and $P = 3$, respectively.

We compare the memory consumption between IDMSVD-LSE and MRDSVD-LSE, and the results are listed in Table 5. MRDSVD-LSE decomposes an $L \times n$ matrix and obtains an $L \times L$ matrix in each iteration in each core. Its memory consumption is $O(L^2)$ in each core. However, the memory consumed depends on the number of machines used and the degree of concurrency of the

TABLE 5
A Comparison on Memory Consumption, in GB, with $L = 2{,}000$ for Experiment II

method	dataset		
	DS-I	DS-II	DS-III
IDMSVD-LSE	0.42	0.49	0.75
MRDSVD-LSE ($P = 1$)	0.70	1.06	1.74
MRDSVD-LSE ($P = 2$)	0.75	1.26	1.93
MRDSVD-LSE ($P = 3$)	0.81	1.46	2.21

TABLE 6
A Comparison on Execution Time with Different Map Tasks for Experiment II

parameter			dataset		
P	M	T	DS-I	DS-II	DS-III
1	2	2	116.890	732.996	2190.796
1	4	4	68.936	405.571	1261.450
1	6	6	63.906	334.337	1002.473
1	8	8	53.799	298.655	893.384
2	2	4	67.894	404.432	1200.162
2	4	8	49.044	243.467	724.590
2	6	12	47.872	207.606	596.672
2	8	16	41.002	183.715	545.943
3	2	6	54.610	276.628	832.348
3	4	12	44.060	189.071	541.676
3	6	18	33.004	148.955	467.314
3	8	24	32.033	141.619	438.531

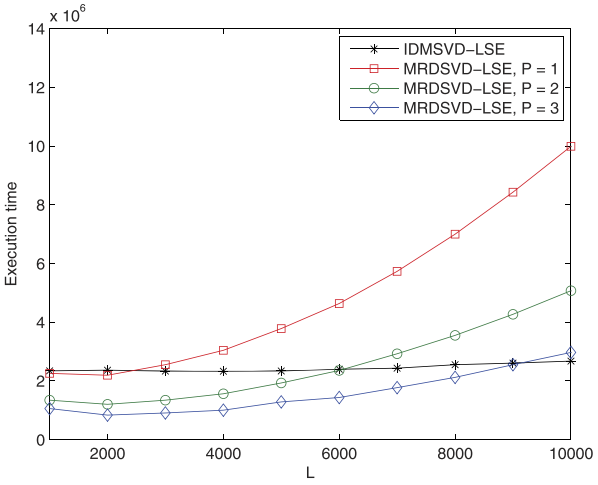

Fig. 1. The results obtained with different settings of L for Experiment III.

map and reduce tasks. As mentioned, two map tasks are executed in two cores of a machine. If the two map tasks perform SVD at the same time, the memory consumption would be about double the volume used in IDMSVD-LSE. Otherwise, MRDSVD-LSE would use less than double. For example, MRDSVD-LSE uses 0.70 GB which is less than double that used by IDMSVD-LSE for DS-I in the case of $P = 1$, since DS-I is small and the two map tasks in the machine are less likely to perform SVD concurrently all the time. For DS-II and DS-III, MRDSVD-LSE uses more memory since these two data sets are larger and the two map tasks in the machine are more likely to perform SVD simultaneously. Besides, some extra memory is required for resource management by MapReduce. The required extra memory increases when the number of machines increases. Therefore, we can see that MRDSVD-LSE uses more than double the memory used by IDMSVD-LSE. For example, MRDSVD-LSE uses 2.21 GB for DS-III for the case of $P = 3$, which is about three times that used by IDMSVD-LSE.

Next, we compare the performance of MRDSVD-LSE on different settings of maximum number of map tasks, M, per machine. That is, M is the maximum number of map tasks each machine can execute simultaneously. A comparison on execution time is listed in Table 6, in which "P" means the number of machines, "M" means the maximum number of map tasks per machine, and "T" means the total number of map tasks executed simultaneously. For example, for the case of $P = 2$ and $M = 4$, we have $T = 2 \times 4 = 8$ map tasks executed simultaneously for MRDSVD-LSE. Clearly, using the same P, large M runs much faster than small M for all the three data sets. For example, when $P = 1$, $M = 8$ takes 53.799 seconds for the DS-I data set, while $M = 2$ takes 116.890 seconds, $M = 4$ takes 68.936 seconds, and $M = 6$ takes 63.906 seconds. When $P = 1$, $M = 8$ takes 893.384 seconds for the DS-III data set, while $M = 2$ takes 2,190.796 seconds, $M = 4$ takes 1,261.450 seconds, and $M = 6$ takes 1,002.473 seconds. However, execution time is not inversely proportional to the total number of map tasks, T. For example, $T = 6$ ($P = 3$ and $M = 2$) takes 54.060 seconds for the DS-I data set, while $T = 12$ ($P = 3$ and $M = 4$) takes 44.060 seconds and $T = 24$ ($P = 3$ and $M = 8$) takes 32.033 seconds. Again, $T = 6$ ($P = 3$ and $M = 2$) takes 832.348 seconds for the DS-III data set, while $T = 12$ ($P = 3$ and $M = 4$) takes 541.676 seconds and $T = 24$ ($P = 3$ and $M = 8$) takes 438.531 seconds. Moreover, the same total number of map tasks does not mean close execution time. However, in the case of being the same total number of map tasks, large P runs much faster than small P. For example, when $T = 8$, ($P = 2$ and $M = 4$) takes 49.044 seconds for the DS-I data set, while ($P = 1$ and $M = 8$) takes 53.799 seconds. For the DS-III data set, when $T = 8$, ($P = 2$ and $M = 4$) takes 724.590 seconds while ($P = 1$ and $M = 8$) takes 893.384 seconds.

6.3 Experiment III

In this experiment, we look into the effect of different settings of L on the performance of IDMSVD-LSE and MRDSVD-LSE. Note that the default value, $M = 2$, is used as the maximum number of map tasks per machine for MRDSVD-LSE in this experiment. Fig. 1 shows the results obtained by IDMSVD-LSE and MRDSVD-LSE for the data set DS-III with different settings of L. From this figure, we can see that the curve of IDMSVD-LSE, marked with "$*$", is flat. That is, the execution time for the data set does not vary significantly for IDMSVD-LSE with different settings of L. However, variations in execution time for MRDSVD-LSE with different settings of L are large. When L increases, the overhead of each map task becomes increasingly significant. Also, we can see that the curve of MRDSVD-LSE with $P = 1$, marked with "□," is steep, while the curves of MRDSVD-LSE with $P = 2$ and $P = 3$ are relatively smooth. Note that MRDSVD-LSE runs better with smaller L values; namely, a large number of mappers is preferable. Table 7 lists the values of some snapshots in Fig. 1. The results obtained by IDMSVD-LSE and MRDSVD-LSE for the other two data sets, DS-I and DS-II, with different settings of L are presented in Section 4 of the supplementary file, available online.

L is $N_{j,i}$ in Section 4.1. As stated in Section 4.1, $N_{j,i}$ is required to be greater than or equal to n in order to reasonably divide the matrix $\mathbf{X}(j)$ into s_j submatrices in iteration j. Therefore, $L \geq n$. When L is small, each

TABLE 7
Values of Some Snapshots Taken from Fig. 1

	L		
method	2000	5000	8000
IDMSVD-LSE	2365.221	2337.380	2551.516
MRDSVD-LSE ($P = 1$)	2190.796	3783.358	6997.365
MRDSVD-LSE ($P = 2$)	1200.162	1930.378	3549.741
MRDSVD-LSE ($P = 3$)	832.348	1281.630	2117.587

submatrix is small and the SVD decomposition for each submatrix can be done very quickly in each iteration. However, the number of submatrices in each iteration is large and the number of iterations is also large. As a result, MRDSVD-LSE may run slower. When L is large, the number of submatrices in each iteration is small and the number of iterations is also small. However, each submatrix is large and the SVD decomposition for each submatrix takes more time in each iteration. As a result, MRDSVD-LSE may also run slower. Finding the best L for each case is difficult. Hadoop provides a framework which automatically takes care of interprocess coordination, distributed counters, failure detection, automatic restart of failed process, internode communication, sorting, shuffling, etc. These mechanisms are complicated and it is hard to figure out the time Hadoop spends on each of these mechanisms. Choosing L to balance the map cost, the iteration depth, and the cache performance is an interesting topic. Currently, we can only say that L between 1,000 and 5,000 is a good choice by observations from the experiments we have done.

7 Conclusion

SVD is usually applied to finding optimal solutions in LSE problems expressed as $\mathbf{Y} = \mathbf{X}\beta$, especially when $N \gg n$. Most of the existing methods focus on speeding up the computation of SVD on the matrix $\mathbf{X}$ using parallel or distributed algorithms. The size of the largest matrix involved is $N \times N$. In the case of huge $\mathbf{X}$, finding the solutions can be both time and space demanding. Our proposed method, IDMSVD-LSE, is aimed to overcome these difficulties. The novelty of our proposed method is that we do not deal with the computation of SVD on the matrix $\mathbf{X}$ itself. Instead, we divide the LSE problem to be solved into a collection of small LSE problems in each iteration. So, we only need to do the computation of SVD on small matrices. The size of the largest matrix involved is $L \times L$. As a result, the requirements in time and space for finding least squares solutions are greatly improved. Furthermore, the decomposition and merging of the submatrices in each iteration can be independently done in parallel, leading naturally to a parallel or distributed implementation of the algorithm.

Our proposed method can be implemented using any parallel programming paradigm, such as GPU [17], [19], [20], [28] or PLAPACK [31], [32]. Since we have a Hadoop distributed platform [24] in our laboratory and MapReduce is a parallel programming technique supported by Hadoop, we implemented the parallel version of our proposed approach in MapReduce on this platform. In this implementation, we don't apply directly the MapReduce scheme in which one iteration corresponds to one job consisting of a map task and a reduce task where decomposition and merging are performed, respectively. The direct application would cause more splitting and shuffling for data, resulting in a large overhead. To reduce this overhead, we develop a mechanism, called dynamic adaptation. A job of one iteration also consists of a map task and a reduce task. However, the reduce task in one job, except for the last job, would perform the map task of the next job. We also develop a formula to estimate the number of iterations required. The proposed algorithm can determine, according to N and L, whether or not the current job is the last job. Experimental results, obtained by running on three data sets generated synthetically, have shown that the proposed method is effective for solving large-scale LSE problems.

Due to tight budget, we were not able to conduct experiments on some other platforms with more compute nodes, e.g., the Amazon EC2. We are seeking for financial support to add more compute nodes in our Hadoop platform to make it more effective to evaluate the proposed method. We hope things will change soon, and we can then have a bigger Hadoop platform or run on commercial cloud systems.

Acknowledgments

The authors are grateful to the anonymous reviewers for their comments, which were very helpful in improving the quality and presentation of the paper. This work was supported by the National Science Council under the grants NSC-97-2221-E-110-048-MY3 and NSC-98-2221-E-110-052, and by "Aim for the Top University Plan" of the National Sun Yat-Sen University and Ministry of Education.

References

[1] G.H. Golub and C. Reinsch, "Singular Value Decomposition and Least Squares Solutions," *Numerische Mathematik,* vol. 14, no. 5, pp. 403-420, Apr. 1970.
[2] G.H. Golub and C.F.V. Loan, *Matrix Computations,* third ed. The Johns Hopkins Univ. Press, Oct. 1996.
[3] D.C. Montgomery, E.A. Peck, and G.G. Vining, *Introduction to Linear Regression Analysis,* fourth ed. Wiley-Interscience, July 2006.
[4] R.H. Myers, D.C. Montgomery, G.G. Vining, and T.J. Robinson, *Generalized Linear Models: With Applications in Engineering and the Sciences,* second ed. Wiley-Interscience, Mar. 2010.
[5] O. Bretscher, *Linear Algebra with Applications,* third ed. Prentice Hall, July 2004.
[6] Å. Björck, *Numerical Methods for Least Squares Problems,* first ed. SIAM, Dec. 1996.
[7] S.S. Niu, L. Ljung, and Å. Björck, "Decomposition Methods for Solving Least-Squares Parameter Estimation," *IEEE Trans. Signal Processing,* vol. 44, no. 1, pp. 2847-2852, Nov. 1996.
[8] Å. Björck and J.Y. Yuan, "Preconditioners for Least Squares Problems by LU Factorization," *Electronic Trans. Numerical Analysis,* vol. 8, pp. 26-35, Nov. 1999.
[9] S.J. Lee and C.S. Ouyang, "A Neuro-Fuzzy System Modeling with Self-Constructing Rule Generation and Hybrid SVD-Based Learning," *IEEE Trans. Fuzzy Systems,* vol. 11, no. 3, pp. 341-353, June 2003.
[10] L.V. Foster, "Solving Rank-Deficient and Ill-Posed Problems Using UTV and QR Factorizations," *SIAM J. Matrix Analysis and Applications,* vol. 25, no. 2, pp. 582-600, Feb. 2003.
[11] C.B. Moler, *Numerical Computing with Matlab.* Soc. for Industrial Math., Jan. 2004.
[12] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing,* second ed. Cambridge Univ. Press, Oct. 1992.

[13] L. Giraud, S. Gratton, and J. Langou, "A Rank-*k* Update Procedure for Reorthogonalizing the Orthogonal Factor from Modified Gram-Schmidt," *SIAM J. Matrix Analysis and Applications,* vol. 25, no. 4, pp. 1163-1177, Apr. 2004.
[14] R.P. Brent and F.T. Luk, "The Solution of Singular-Value and Symmetric Eigenvalue Problems on Multiprocessor Arrays," *SIAM J. Scientific and Statistical Computation,* vol. 16, no. 1, pp. 69-84, 1985.
[15] C.R. Vogel and J.G. Wade, "Iterative SVD-Based Methods for Ill-Posed Problems," *SIAM J. Scientific Computing,* vol. 15, no. 3, pp. 736-754, May 1994.
[16] G.O.M. Bečka and M. Vajteršic, "Dynamic Ordering for a Parallel Block-Jacobi SVD Algorithm," *Parallel Computing,* vol. 28, pp. 243-262, Feb. 2002.
[17] V. Hari, "Accelerating the SVD Block-Jacobi Method," *Computing,* vol. 75, no. 1, pp. 27-53, Mar. 2005.
[18] H. Zamiri-Jafarian and G. Gulak, "Iterative MIMO Channel SVD Estimation," *Proc. IEEE Int'l Conf. Comm.,* pp. 1157-1161, May 2005.
[19] Y. Yamamoto, T. Fukaya, T. Uneyama, M. Takata, K. Kimura, M. Iwasaki, and Y. Nakamura, "Accelerating the Singular Value Decomposition of Rectangular Matrices with the CSX600 and the Integrable SVD," *Proc. Int'l Conf. Parallel Computing Technologies,* pp. 340-345, 2007.
[20] S. Lahabar and P.J. Narayanan, "Singular Value Decomposition on GPU Using CUDA," *Proc. IEEE Int'l Symp. Parallel and Distributed Processing,* pp. 1-10, 2009.
[21] T. Kondaa and Y. Nakamura, "A New Algorithm for Singular Value Decomposition and Its Parallelization," *Parallel Computing,* vol. 35, no. 6, pp. 331-344, June 2009.
[22] H. Ltaief, J. Kurzak, and J. Dongarra, "Parallel Two-Sided Matrix Reduction to Band Bidiagonal form on Multicore Architectures," *IEEE Trans. Parallel and Distributed Systems,* vol. 21, no. 4, pp. 417-423, Apr. 2010.
[23] S. Wei and Z. Lin, "Accelerating Iterations Involving Eigenvalue or Singular Value Decomposition by Block Lanczos with Warm Start," technical report, Microsoft Corp., Dec. 2010.
[24] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Comm. ACM,* vol. 51, no. 1, pp. 107-113, Jan. 2008.
[25] W. Zhao, H. Ma, and Q. He, "Parallel *k*-Means Clustering Based on MapReduce," *Proc. Int'l Conf. Cloud Computing,* pp. 674-679, 2009.
[26] J. Cohen, "Graph Twiddling in a Mapreduce World," *Computing in Science & Eng.,* vol. 11, no. 4, pp. 29-41, Jan. 2009.
[27] S.J. Matthews and T.L. Williams, "MrsRF: An Efficient MapReduce Algorithm for Analyzing Large Collections of Evolutionary Trees," *BMC Bioinformatics,* vol. 11, no. Suppl. 1, Jan. 2010.
[28] W. Fang, B. He, Q. Luo, and N.K. Govindaraju, "Mars: Accelerating Mapreduce with Graphics Processors," *IEEE Trans. Parallel and Distributed Systems,* vol. 22, no. 4, pp. 608-620, Apr. 2011.
[29] L.G. Valiant, "A Bridging Model for Parallel Computation," *Comm. ACM,* vol. 33, no. 8, pp. 103-111, Aug. 1990.
[30] http://en.wikipedia.org/wiki/Bulk_synchronous_parallel, 2012.
[31] R.A. van de Geijn, *Using PLAPACK Parallel Linear Algebra Package.* MIT Press, June 1997.
[32] G. Baker, J. Gunnels, G. Morrow, B. Riviere, and R.A. van de Geijn, "PLAPACK: High Performance through High Level Abstraction" *Proc. Int'l Conf. Parallel Processing,* 1998.

Chi-Yuan Yeh received the BS and MS degrees in business administration from Shu-Te University, Taiwan, in 2002 and 2004, respectively, and the PhD degree in electrical engineering from National Sun Yat-Sen University, Taiwan, in 2011. His main research interests include Machine Learning, Data Mining, and Soft Computing. He received the Best Paper Award of the International Conference on Machine Learning and Cybernetics, 2008.

Yu-Ting Peng received the BS degree from National Kaohsiung Marine University, Taiwan, in 2009 and the MSEE degree from National Sun Yat-Sen University, Taiwan, in 2011. His main research interests include parallel and distributed computing and cloud computing.

Shie-Jue Lee received the BS and MS degrees in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1977 and 1979, respectively, and the PhD degree in computer science from the University of North Carolina, Chapel Hill, in 1990. He joined the faculty of the Department of Electrical Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan, in 1983, and has been a professor in the Department since 1994. His current research interests include artificial intelligence, machine learning, data mining, information retrieval, and soft computing. He received the Best Paper Award in several international conferences. Among the other awards won by him are the Distinguished Teachers Award of the Ministry of Education, Taiwan, in 1993, the Distinguished Research Award in 1998, the Distinguished Teaching Award in 1993 and 2008, and the Distinguished Mentor Award in 2008, all from National Sun Yat-Sen University. He served as the program chair for several international conferences. He was the director of the Southern Telecommunications Research Center, National Science Council, from 1998 to 1999, the chair of the Department of Electrical Engineering from 2000 to 2003, and the deputy dean of the Academic Affairs from 2008 to 2011. He is now the director of the NSYSU-III Research Center and the vice president of Library and Information Services, National Sun Yat-Sen University. He is a member of the IEEE.