FAST PRINCIPAL COMPONENT ANALYSIS FOR HYPERSPECTRAL IMAGING BASED ON CLOUD COMPUTING

Yonglong Li[1], Zebin Wu[1,2], Jie Wei[1], Antonio Plaza[2], Jun Li[2], Zhihui Wei[1]*

[1] School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China;
[2] Department of Technology of Computers and Communications, University of Extremadura, Cáceres E-10003, Spain;
[3] School of Geography and Planning, Sun Yat-sen University, Guangzhou, 510275, China.

ABSTRACT

Principal component analysis (PCA) is an important method for feature extraction of hyperspectral remote sensing image. With the development of hyperspectral sensors, the magnitude of hyperspectral data grows quickly, and it is a challenging task to efficiently reduce the data dimension and compress massive data volumes in hyperspectral imaging. In this paper, a distributed parallel optimization of PCA algorithm (PCA_DP) is presented on cloud computing architecture. The realization of the proposed method using Apache Hadoop and MapReduce model is described and evaluated. The experiments conducted on real hyperspectral images of different sizes, demonstrate significant acceleration factor of PCA_DP. It is efficient for massive hyperspectral data processing.

Index Terms— Principal component analysis, hyperspectral remote sensing, Hadoop, distributed parallel optimization

1. INTRODUCTION

It is important to efficiently reduce the dimensionality of hyperspectral images and extract the primary features from hundreds of high correlated spectral bands [1]. One of the most popular dimensionality reduction methods is principal component analysis (PCA), a statistical procedure that uses orthogonal transformations to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components [2]. Since the neighboring bands in a hyperspectral image are highly correlated, PCA can effectively transform the original data to remove the correlation among the bands [3]. Taking into consideration that the magnitude of hyperspectral data grows quickly along with the development of hyperspectral sensors, it is a challenging task to efficiently reduce the data dimension and compress massive data volumes in hyperspectral imaging.

With the rapid development of high performance computing technology, it is now possible to accelerate hyperspectral image processing algorithms significantly in parallel computer architectures [4,5], among which, cloud computing [6] is a promising technique for distributed parallel optimization and massive data processing. Cloud computing technology has the characteristics of resource pooling, rapid elasticity and measured services, and can provide high-data capacity and high-performance computing at low cost. Hadoop [7] is a widely-used open-source cloud computing platform, which consists of a distributed parallel-computing model MapReduce and a Hadoop distributed file system (HDFS). In this paper, a distributed parallel optimization of PCA algorithm is proposed on Hadoop (PCA_DP), and its efficiency is evaluated in terms of accuracy and parallel execution performance, as compared with a serial PCA implementation on a single central processing unit (CPU).

2. DISTRIBUTED PARALLEL OPTIMIZATION FOR PCA BASED ON HADOOP

Hadoop has powerful computing capacity, huge storage resources and good scalability, which can effectively solve the challenges involved in massive hyperspectral data storage and processing. HDFS can be used to efficiently store high-dimensional hyperspectral data sets. MapReduce can split the original computing task into many small tasks distributed on each node of the cloud computing platform, even with unreliable and weak communication links. By distributing the computational tasks to different data nodes, MapReduce can quickly complete distributed parallel processing tasks for hyperspectral big data. In the following we describe the parallelization framework implemented in Hadoop. Let us denote by $\boldsymbol{X} = [\boldsymbol{X}_{ij}]_{n\times m}$ a hyperspectral image with n pixel vectors, each with m spectral bands. The traditional PCA algorithm first computes the

This work was supported in part by the National Natural Science Foundation of China under Grant No.61471199, 61101194, 11431015, the China Scholarship Fund under Grant No. 201406845012, the Research Fund for the Doctoral Program of Higher Education of China under Grant No. 20113219120024, the Fundamental Research Funds for the Central Universities under Grant No. 30915012204, the Jiangsu Province Six Top Talents project of China under Grant No. WLW-011.
*Corresponding author. Email: Zebin.wu@gmail.com

covariance matrix $\Sigma_{m\times m} = \frac{1}{n-1}(X - \bar{X})^T (X - \bar{X})$. The eigenvalues $\lambda = [\lambda_1, \lambda_2, ...\lambda_m]$, arranged in descending order so that $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_m$, and the corresponding eigenvectors $V = [v_1, v_2, ...v_m]$ can be obtained by the eigen-decomposition of the covariance matrix Σ . Finally, the original data X can be transformed by multiplying it by V . In practice, we usually select k (k < m) eigenvalues instead of V to transform X , which reduces the data redundancy, and the k eigenvalues corresponding to the k eigenvectors are ones comprising more information.

However, the traditional PCA algorithm is not suitable for distributed parallel optimization, due to the strong existing correlation among the pixel vectors when calculating the covariance matrix. Therefore, the calculation of the covariance matrix Σ is first optimized. The calculation of the covariance matrix can be derived as follows:

$$\begin{aligned}\Sigma_{ij} &= \frac{1}{n-1}\sum_{k=1}^{n}(X_{ki} - \bar{X}_i)(X_{kj} - \bar{X}_j) \\ &= \frac{1}{n-1}(\sum_{k=1}^{n} X_{ki} * X_{kj} - n * \bar{X}_i * \bar{X}_j)\end{aligned}, \quad (1)$$

Then,

$$\Sigma = \frac{1}{n-1}(X_{n\times m}^T * X_{n\times m} - n * \bar{X}^T * \bar{X}) . \quad (2)$$

Since n is much larger than the number of spectral bands m, (2) will greatly reduce the amount of computation. However, calculating every entry of Σ has to transpose the hyperspectral data by columns, which is not suitable for distributed parallel optimization and cannot make full use of cache memory to improve computation efficiency. Therefore, (2) is changed into:

$$\Sigma = \frac{1}{n-1}(\sum_{i=1}^{n} X_{i*}^T * X_{i*} - n * \bar{X}^T * \bar{X}), \quad (3)$$

where, $X_{i*} = [X_{i1}, X_{i2}, ..., X_{im}]$ is a pixel vector. Now each pixel vector just needs to be multiplied by itself, and there is no correlation among the pixel vectors when calculating Σ , which is well suited for distributed parallel computing. Moreover, every pixel vector can be sequentially read from the data by rows, which leads to a good locality of the program, and makes the cache memory more efficiently exploited.

After conducting these steps, we realize the distributed parallel optimization on Hadoop which consists of two procedures, e.g. the eigen-decomposition of covariance matrix and the transformation of matrix projection. At the beginning, a HSIInputFormat class is implemented to read the hyperspectral image in pixel-wise fashion. Then we create a job to calculate and perform the calculation of the eigen-decomposition. In the Map function, we multiply the pixel vector by itself. The pixel is read from the corresponding data split X' , and then we accumulate the result. Since the final cumulative result Λ is a symmetrical matrix, we just need to calculate its upper triangular matrix Λ^z , where z denotes that this map task is the z-th map task of the job. We now add up all the pixel vectors of X' by band, then get the sum $S_{1\times m}$. Finally, the Λ^z and $S_{1\times m}$ are sent to the Reduce task. In the Reduce function, we accumulate the upper triangular matrix of the Λ and $S_{1\times m}$ from every Map, respectively, and then calculate Σ . Take into consideration that Σ has at most a few hundred dimension, the eigen-decomposition takes less than 1 second to compute. Thus, the eigen-values λ and corresponding eigen-vectors V can be efficiently computed by JAMA [8] in the Reduce phase. The eigen-decomposition of the covariance matrix of the hyperspectral image based on Hadoop is summarized in Algorithm 1.

Algorithm 1: The eigen-decomposition of covariance matrix on Hadoop

Map ***Input*** $: (X'_{p\times m})$, ***Output*** $: (\Lambda^z, S^z)$

1. $for(i = 1; i <= p; i++)\{$
2. $\quad for(j = 1; j <= m; j++)\{$
3. $\quad\quad S_j += X'_{ij}$
4. $\quad\quad for(k = j; k <= m; k++)\{$
5. $\quad\quad\quad \Lambda^z_{jk} += X'_{ij} * X'_{ik}\}\}\}$
6. ***output***(Λ^t, S^t)

Reduce ***Input*** $: ((\Lambda^1, \Lambda^2, ..., \Lambda^r), (S^1, S^2, ..., S^r))$,
Output $: (V, \lambda)$

1. $\Lambda = sum(\Lambda^1, \Lambda^2, ..., \Lambda^r)$
2. $S = sum(S^1, S^2, ..., S^r)$
3. $\bar{X} = \frac{1}{n} S$
4. $for(j = 1; j <= m; j++)\{$
5. $\quad for(k = j; k <= m; k++)\{$
6. $\quad\quad \Lambda_{kj} = \Lambda_{jk}\}\}$
7. $\Sigma = \frac{1}{n-1}(\Lambda - n * \bar{X}^T * \bar{X})$
8. $[V, \lambda] = Eigendecomposition(\Sigma)$
9. ***output :*** (V, λ)

Finally, another job task is created to transform the original data X . The distributed cache is utilized to copy the eigenvectors V to every Map. Then we multiply the corresponding data split X' by the first 100 eigenvectors of

the V , and the dimensionally reduced image can be synthesized from all of the results of the Map tasks. This job takes full advantage of the locality of the matrix without the Reduce tasks, and further reduces system overhead, eliminates the unnecessary intermediate data, and increases computing performance dramatically.

3. EXPERIMENTAL EVALUATION

To verify the proposed distributed implementation of PCA, experiments were performed on a Hadoop equipped cluster which has 1 NameNode and 8 DataNodes. NameNode is a virtual machine created on the host with an Intel Xeon E5630 CPU at 2.6 GHz with 8 cores by the Vmware vSphere. NameNode is configured with 8 CPUs, 5GB of RAM and 50GB of disk storage. The Datanodes are implemented by virtual machines created based on the virtualization of a 4-blade IBM BladeCenter HX5 with 48 cores by Vmware vSphere. Each DataNode is configured with 6 1.87-GHz CPUs, 6GB of RAM and 17GB of disk storage, has 5 Map slots and 2 Reduce slots. Both NameNode and DataNodes are installed with Ubuntu 12.04, Hadoop 1.2.1 and Java 1.6.45. Moreover, all nodes are connected by a gigabit switch. Fig. 1 graphically illustrates the architecture of the system.

The hyperspectral dataset used in our experiments is the well-known Ariborne Visible Infra-Red Imaging Spectrometer (AVIRIS) Cuprite image with 224 spectral bands. Water absorption bands and bands with low SNR were removed prior to the analysis, including bands 1~3, 105~115, 150~170, 223~224. The dataset consists of 614×512 pixels, 187 bands and a total size of about 112MB. In order to test the performance on a large dataset, we use the Mosaicking function of ENVI software to generate three data sets with different sizes (including 6140×512 with the size of about 1.09GB, 12280×512 with the size of about 2.18GB, 24560×512 with the size of about 4.37GB) by mosaicking the original 112MB dataset.

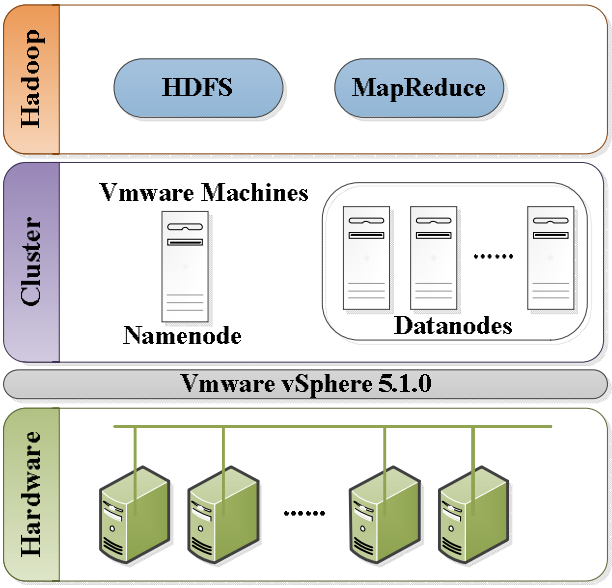


Fig.1. The architecture of the system

TABLE 1 The execution time of PCA_DP and speedups

Computing Mode	Nodes	112MB		1.09GB	
		Time(s)	Speedup(x)	Time(s)	Speedup(x)
Serial	1	301	--	2955	--
PCA_DP on Cluster	1	28	10.75	98	30.15
	2	26	11.58	58	50.95
	4	25	12.04	39	75.77
	8	25	12.04	34	86.91

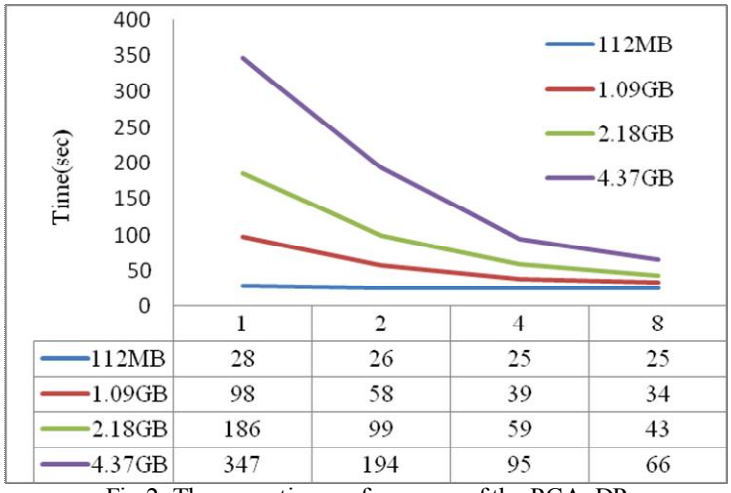
TABLE 2 The first three largest eigen-values comparison of PCA_DP for the dataset of 112MB

Eigen-values	Matlab	PCA_DP
Eigen-value 1	47572539.5857669	47572539.5857669
Eigen-value 2	2894606.45288447	2894606.452884475
Eigen-value 3	1457124.491871871	1457124.491871879

Tables 1 and 2 illustrate the execution performance and accuracy comparison of PCA_DP respectively. It can be seen that the implementation of PCA_DP gets almost the same eigen-values as the Matlab implementation of PCA. Fig. 2 shows the execution performance of PCA_DP for different data sizes, which demonstrates that the speedups increase with the size of the dataset. This is ideal in terms of massive hyperspectral data processing.

4. CONCLUSIONS AND FUTURE WORK

The volume of hyperspectral dataset is becoming increasingly large, it is a challenging task to efficiently reduce the data dimension and compress massive data volumes in hyperspectral imaging. Cloud computing is a promising technique for distributed parallel optimization and massive data processing. In this paper, a distributed parallel optimization of PCA is proposed based on Apache Hadoop. The experimental results comparing with a serial PCA implementation demonstrate the effectiveness of the proposed method. Future work will focus on the distributed parallel optimization for other algorithms of hyperspectral image processing.



	1	2	4	8
112MB	28	26	25	25
1.09GB	98	58	39	34
2.18GB	186	99	59	43
4.37GB	347	194	95	66

Fig.2. The execution performance of the PCA_DP

5. REFERENCES

[1] A. Plaza, J. M. Bioucas-Dias, A. Simic, W. J. Blackwell, "Foreword to the Special Issue on Hyperspectral Image and Signal Processing," IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 5, no.2, pp. 347-353, Apr., 2012.

[2] I.T. Jolliffe. Principal Component Analysis, Series: Springer Series in Statistics, 2nd ed., Springer, NY, 2002.

[3] Rodarmel C, Shan J. Principal component analysis for hyperspectral image classification. Surveying and Land Information Science, 2002, 62(2): 115-122.

[4] C. A. Lee, S. D. Gasster, A. Plaza, et al., "Recent Developments in High Performance Computing for Remote Sensing: A Review," IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 4 , no.3, pp. 508 - 527, 2011.

[5] A. Plaza, Q. Du, Y. Chang, R. L. King. "High Performance Computing for Hyperspectral Remote Sensing," IEEE J. Sel. Topics Signal Process., vol.4, no.3, pp.528-544, Sep. 2011.

[6] Armbrust M, Fox A, Griffith R, et al. A view of cloud computing. Communications of the ACM, 2010, 53(4): 50-58.

[7] White T. Hadoop: The Definitive Guide. 3nd ed. O' Reilly Media, 2012.

[8] H. Joe, M. Cleve, W. Peter. JAMA:A Java Matrix Package. Available: http://math.nist.gov/javanumerics/jama/, 2014.11.29.