Cloud KSVD: Implementation and analysis

Team XXL

Abstract— **This is a report that introduces what we have done in the project.The first section is the general introduction of our motivation and goals.The second section provides a background knowledge introducing the EMR.The third section is our contribution in this project.The forth section introduces the KSVD and CloudKSVD algorithms, including the implementation in Spark. The fifth section is the experiment evaluation. The six th section concludes the theoretical acknowledge and the experimental result about the local KSVD and cloud KSVD. The last section introduces our future work.**

I. INTRODUCTION

The sparse representation of a signal or a given dataset using dictionary learning algorithms has gained a lot of attention in recent decades. The original signals could be decomposed by sparse linear combinations of a series of prototype signal-atoms with using an overcomplete dictionary which contains prototype signal-atoms. The sparse representation is very popular in many field, such as image denoising [1], image inpainting [2], even in the pattern or face recognition [3]. However, even above contributions for applying the sparse representation could give well performance, they all assume that original signals are stored in the centralized location. So we are interested in how to extend a dictionary learning algorithm for sparse representation could apply on the big and distributed data.

In our project, we have implemented a collaborative dictionary algorithm, cloud K-SVD, for big, distributed data. To finish this project, our challenges are scale out the centralized K-SVD to collaborative compute the dictionary according several sites and the sites can not communicate raw data among themselves for security concern and reduce the communication cost. To realize our goals, we implement the cloud K-SVD on AWS and analyze the performance with local K-SVD.

II. BACKGROUND

Before going to our project, we would like to briefly introduce the Amazon Web Service(AWS) that we used which is the ElascticMapReduce (EMR). Compared with EC2, another well-known AWS, EMR is a collection of EC2 instances with Hadoop (and optionally Hive and/or Pig) installed and configured on them. For us, it really saves a lot of time since we are all beginners to AWS. The only thing that bothers us is that the EMR cost is a little bit higher than EC2. Running spark application on EMR is also easier than EC2. With the help of the button 'add step', we can easily submit the jar file which are uploaded earlier in S3 along with appropriate arguments. The only thing we need to concern about is the deploy model. The EMR provides 2 ways of deploy mode, client and cluster.When running in cluster mode, the driver runs on ApplicationMaster, the component that submits requests to the ResourceManager according to the resources needed by the application. When running in client mode, the driver runs outside ApplicationMaster, in the spark-submit script process from the machine used to submit the application. A simplified and high-level diagram of the application submission process is shown below.

Submitting applications in client mode is advantageous when you are debugging and wish to see the output of your application. For applications in production, the best practice is to run the application in cluster mode. This mode offers you a guarantee that the driver is always available during application execution. However, if you do use client mode and you submit applications from outside your EMR cluster (such as locally, on a laptop),the driver is running outside your EMR cluster and there will be higher latency for driver-executor communication.

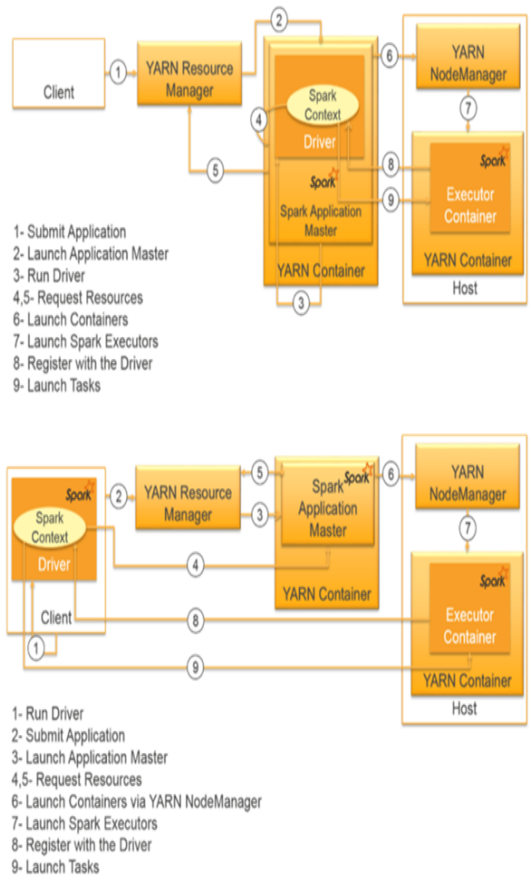

Fig. 1. igh-level diagram of the application submission process

III. Contribution

In our project, we implemented the local KSVD and cloud KSVD algorithm on EMR, both in spark. We did comparison about the performance of the two models and tried to find the bottleneck of the cloud KSVD on EMR. With our experiment results, we have given us analyzes. Besides, we also try to find the superior running model for input data according to their sizes.

IV. Algorithms overview

As we mentioned before, a given signal $y \in R^n$ can be represented as a sparse linear combination of atoms in an overcomplete learned dictionary matrix $D \in R^{n*k}$ which contains k atoms. With K-SVD we can approximate represent signal $y = Dx$ satisfying $||y - Dx|| < \varepsilon$

A. K-SVD for dictionary learning

K-SVD is a iteration dictionary learning algorithm between sparse coding of the examples based on the current dictionary phase and dictionary updating phase of updating the dictionary atoms to better fit the data for creating a dictionary to sparse represent a given signal. The update of the dictionary columns is combined with an update of the sparse representations, thereby accelerating convergence. The K-SVD algorithm is flexible and can work with any pursuit method. The flow of K-SVD is showed in the following Figure 2.

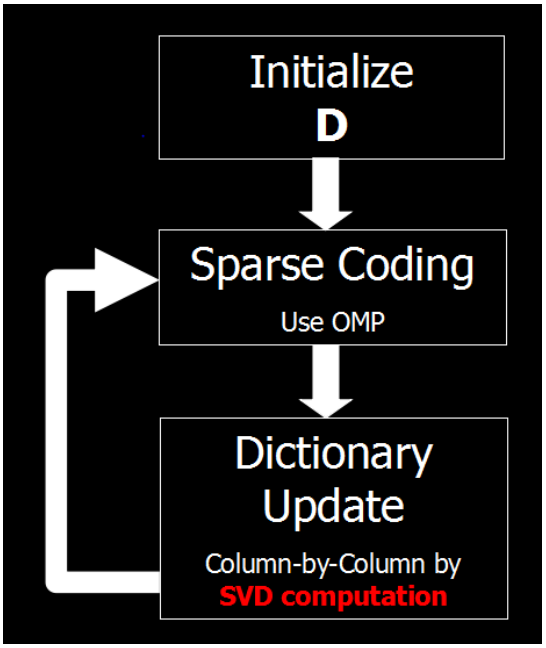

Fig. 2. Overview of K-SVD

K-SVD starts with initializing a dictionary matrix D(always using random value), then move to sparse coding. In the sparse coding phase of iteration t, we compute the representation coefficients vector x based on the exist dictionary D and given signal y to satisfy the following equation:

$$\forall s, x_s^{(t)} = \arg\min_{x \in R^n} \|y_s - D^{(t-1)}x\|_2^2$$

,

where y_s denotes the s^{th}column of Y. Then we go to next step, dictionary updating. In this phase, we want to update dictionary D with above x to minimize the $||Y - DX||_2^2$. This step involves iterating through the K atoms of D^{t-1}

and individually updating the k^{th} column of D, keep other columns constant, as follows:

$$d_k^{(t)} = \arg\min_{d\in\mathbb{R}^n} \left\| \left(Y - \sum_{j=1}^{k-1} d_j^{(t)} x_{j,T}^{(t)} - \sum_{j=k+1}^{K} d_j^{(t-1)} x_{j,T}^{(t)}\right) - d\, x_{k,T}^{(t)} \right\|_F^2 = \arg\min_{d\in\mathbb{R}^n} \left\| E_k^{(t)} - d\, x_{k,T}^{(t)} \right\|_F^2 .$$

To find the most sparsest coefficient X, we just keep nonzero element in coefficient representation X and correspondent $d_k^{(t)}$. Now, according to Eckart-Young Theorem, we can update $d_k^{(t)}$ with the u_1 by using SVD to decompose the reduced error matrix $E_k^{(t)}$ to $\sigma_1 u_1 v_1^T$. After that we repeat this work for K column to get the new overcomplete dictionary matrix D. Then we repeat this whole work alternatively between sparse coding and dictionary updating stage until the difference of $Y - DX$ under the threshold.

B. Cloud K-SVD for dictionary updating

Now, we come to introduce the algorithm used in our project, we have implemented Cloud K-SVD algorithm to realize our target. Cloud K-SVD is a collaborative dictionary updating algorithm among several sites. The main thought of Cloud K-SVD are basically as same as traditional K-SVD, except for dictionary updating stage. We will introduce what we have done to solve the project challenge, that each site only store a part of data, in later section. Now, we introduce the Orthogonal Matching Pursuit (OMP) algorithm, which we used in sparse coding stage to compute the coefficient representation X.

The OMP algorithm is a greedy algorithm to find the sparse representation of a given signal based on a specific exist dictionary. The OMP attempts to find the best atom in each iteration to reduce the representation error until the error could be stood by our application. This achieved by selection of that atom from the dictionary which has the largest absolute projection on the error vector.This essentially implies that we select that atom, which adds the maximum information and hence maximally reduces the error in reconstruction. The OMP algorithm is showed in following algorithm 1.

Algorithm 1 Orthogonal Matching Pursuit

Input: Original signal Y, sparsity threshold T, error tolerance E,

Dictionary matrix D

1: is initialized with Y

2: while k < T and $r^k < E$ do

3: Select the atom which has maximal projection on the residual

4: Update $X_k = \arg\min_{X_k} ||y - DX_k||_2$

5: Update the residual $r^k = Y - Y^k$

From the above process, we can find that the OMP is easy to implement and provides a satisfactory stable result. An indepth analysis of stable recovery by greedy methods in the presence of noise can be found in [4].

In the dictionary updating stage of Cloud K-SVD, we want to compute a locally dictionary based on their own raw data. Each site start dictionary learning stage with a random initialized D. Then we update k^{th} column in D with the distributed power method to compute dominant eigenvector of $M^{(t)} = E_{K,R}^{(t)} {E_{K,R}^{(t)}}^T$, since it is equal to u_1 of applying SVD on the error matrix. To compute eigenvector, we need using consensus averaging [6] method in the power method iteration [5]. We choose consensus averaging to collaborative update dictionary because it can reduce the information which each need to communicate with others. To represent the topology of the network, we use a matrix W, in our project, we initialize W with $1/n$, where n is the amount of total sites. We choose consensus averaging to collaborative update dictionary because it can reduce the communication cost, since the computation of sparse coding phase and main part of distributed power method can be done in local. By this methodology, we don't need share raw data among sites neither. The algorithm whole workflow is shown in Figure 3 and the distributed power method within consensus averaging are introduced in algorithm 2.

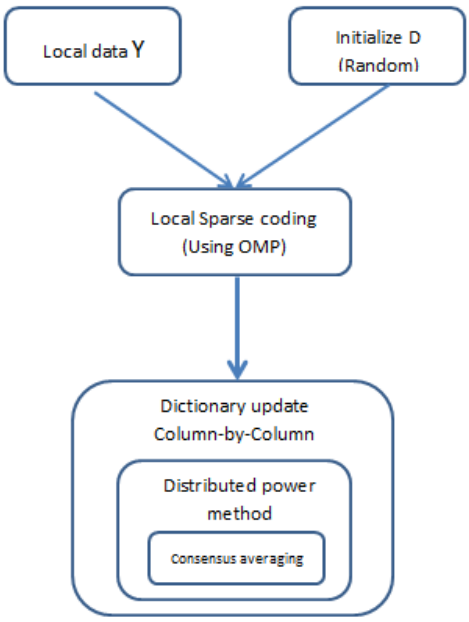

Fig. 3. Workflow of Cloud K-SVD

Algorithm 2 Distributed Power and Consensus Averaging

Input: Iterations of distributed power method t_p, and consensus averaging t_c 1: While $t_1 = 0$ to t_p do

2: $z_i^{(0)} = M_i d_i^{t_i - 1}$

3: While $t_2 = 0$ to t_c do (consensus averaging)

4: $Z_i^{t_2} = \sum_{j \in N_i} w_{i,j} z_j^{t_2 - 1}$

5: $v_i^{(t_1)} = z_i^{(t_c)} / [W_1^{t_c}]_i$

6: update dictionary by normalizing $v_i^{(t_1)}$

7: end while

V. Experiment Evaluation

The experiments are implemented on both local laptop and the ElasticMapReduce(EMR). Compared with EC2, another service of AmazonWebSeivice(AWS), EMR is much easier to work with, especially for the beginners. Generally speaking, the EMR is a set of EC2 instances with Hadoop installed on them. The best thing about the EMR is that the configuration part is easy, almost set ready to go. Users could launch an application by a single click on 'add step' in the EMR to launch a spark application. After a step is completed, the task history could be clearly monitored from the EMR , even failed, a detailed task log could be available, which is helpful for the users to debug or modify. There are two ways to submit an application on EMR, one is client model and the other one is cluster model. In our project, we use the cluster model since it could offer a guarantee that the driver is available to the application executors since the driver is located on the master. We choose spark as our test framework, since spark allow us to use scala write our application, and we can use the breeze library to do the matrix computations. With the RDD mechanism of spark, we can cache our input data, since it will be used throughout the whole work flow and it could auto partition the raw data. These advantages allow us to focus on the algorithm itself. According to the experiments, We have evaluated the Cloud K-SVD performance in 3 aspects as shown below.

A. Run time VS Number of Slaves

At first, we assumed that with more slaves getting involved in a task, the total run time would be decreased. However, our experiment result contradicts this assumption. In this case, each slave has 8 cores. Figure 4 shows the result of the experiment.

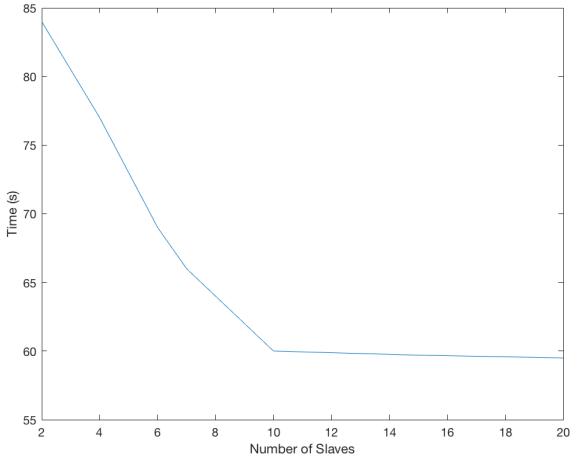

Fig. 4. Running time VS number of slaves

In the first experiment, the input data is a randomly generated matrix with 10 columns and multiple rows, in

other words, the more rows, the bigger the input data. As the Figures 3 shows, the running time is decreasing as more and more slaves getting involved. However, it reaches a bottleneck when the number of slaves reaches 10. After that, having more slaves does not help improving the performance. The reason behind this scenario is that: the input data has only 10 column, and each slave could take care of 1 column. Thus, with more slaves adding to the master, the newly added slaves would not have tasks distributed by the driver or have very few. As a result, the overall performance is not improved with the these slaves beyond 10.

B. Local VS CLoud

The gold of the second experiment is to compare the performance of local KSVD and cloud KSVD. It is implemented on the local laptop and on the EMR cluster. The laptop is Macbook Pro, 2.7Hz Intel Core i5, 8GB Memory, and the cluster consist of 1 master and 10 slaves with 8 cores each. The node type is m1.large, cluster region is us-east-1. The result is shown as Figure 5.

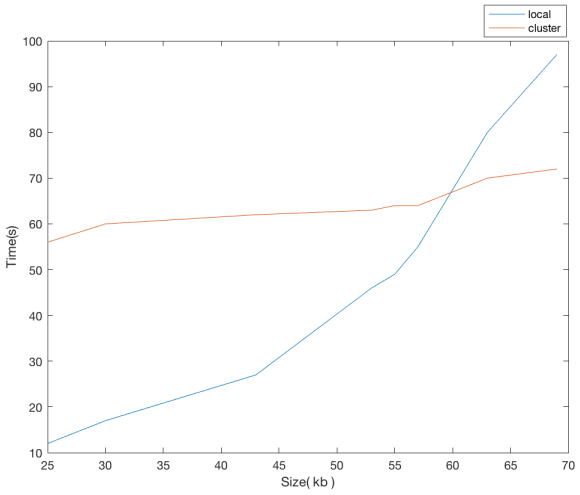

Fig. 5. Local KSVD VS Cloud KSVD

According to the result, the local and cloud KSVD has their own advantages. For local KSVD, the running time increases dramatically with the increase of the input data size. At the very beginning , the running time is neglectable compared with cloud KSVD. On the other hand, the running time of cloud KSVD is not sensitive to the data size increase. Though the runnting time of Cloud KSVD is 6 time longer than the local on , but after the input size reaching 60kb, the cloud KSVD becomes superior to the local KSVD. Especially for our case, when the input data size reaches 70kb, the laptop can not come up with a result due to lack of memory.

C. Run time VS Iteration

The last experiment in the report is to verify the proportional relationship between run time and iteration times. The result is shown in Table 1. The cluster is consist of 1 master and 10 slaves as it is in the previous sec

Iterations	Local(15kb)	Local(30kb)	Cloud(15kb)	Cloud(15kb)
1	6	17	41	60
2	10	21	44	66
3	17	28	55	78
4	19	38	59	90
5	20	48	66	102

TABLE I

RUN TIME(S) VS ITERATIONS

As mentioned before, there are three iteration types in the KSVD, and for here, the iteration by one time means doing the iteration only 1 time for each type. The result shows that the run time is getting longer and longer as the input data size increases both for local KSVD and cloud KSVD. Figure 6 gives a direct look at the result.

The result is quiet promising, with more iteration times, results in higher run time consumption.

VI. CONCLUSION

Through our theoretical acknowledgment and experimental evaluations, we predict that the bottleneck of the cloud KSVD on EMR cluster exists. In other words, having more slaves working at the same time does not necessarily improves the performance. The reason behind this would be the task itself. From the comparison between local KSVD and cloud KSVD, we find that the local model is more suitable for small data and the cluster model is

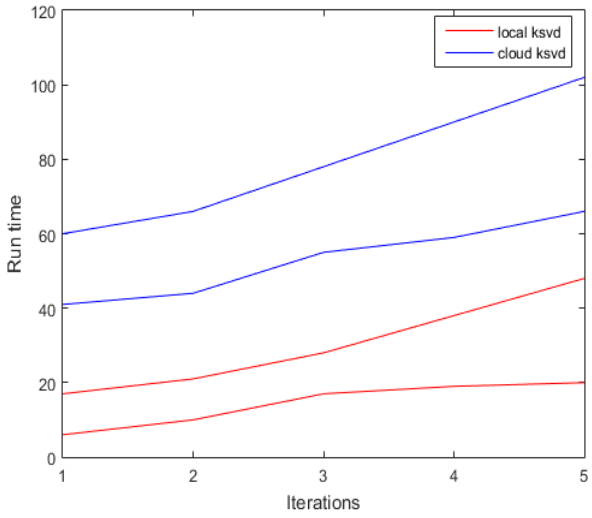

Fig. 6. Run time VS Iterations

good for big data.

We also tried to analyze the detailed influence of a single iteration by arranging various combination of iterations and find the best solution of the best performance with the least time consumption.

Iterations	Run time
1,1,1	$1.5x10^{-14}$
2,1,1	$6.3x10^{-14}$
3,1,1	$7.7x10^{-15}$
1,2,1	$8.7x1010^{-15}$
1,1,2	$9x10^{-15}$
1,3,1	$3x10^{-14}$
1,1,3	$8.2x10^{-15}$

TABLE II

COMBINATION OF ITERATION

However, the result as shown in the Table 2 is not very convincing.It is difficult to tell which type of iteration has the most significant influence on the performance. This result brings us another consideration about the necessity of increasing the iteration times since the performance does not improve dramatically. The reason behind this scenario could be the randomly initialized dictionary matrix.

VII. FUTURE WORK

Even we have implemented Cloud K-SVD on the AWS, we still think that we have a lot of work to do in the future and Cloud K-SVD algorithms can be improved by following aspect. Even our project give a good performance when we tested it on the spark, we think if we can find out a more optimizer partition method, it will give a better performance in efficiency. Another improvement we want to try is finding a better relationships among iteration numbers to make the algorithm performs more better on accuracy but don't waste too much time on the useless stages. For the phase of consensus averaging, in current project, we just defined the constant matrix W, but in the real world, the common case, we may not know the topology of the sites. So we need to make the algorithm to adapt the real case, as the author of this paper mentioned [10]. The last one, we think may be give us surprise, is the process of dictionary initializing. Due to our project initialize the dictionary with the random values, we found that the running time is not stable enough, which means, if you are lucky enough, the algorithm could find a better overcomplete dictionary but only cost fewer time than the bad lucky. So find a more efficient initializing method will make Cloud K-SVD become more stable, which means it can satisfy the real world requirement.

REFERENCES

[1] Marc Lebrun, and Arthur Leclaire, An Implementation and Detailed Analysis of the K-SVD Image Denoising Algorithm,Image Processing On Line, 2 (2012), pp. 96–133

[2] Julien Mairal, Michael Elad,, and Guillermo Sapiro, Sparse Representation for Color Image Restoration, IEEE transactions on image processing, vol. 17, NO. 1, january 2008

[3] Qiang Zhang, and Baoxin Li, Discriminative K-SVD for dictionary learning in face recognition, Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on

[4] David L. Donoho, Michael Elad, and Vladimir Temlyakov. Stable recovery of sparse overcomplete representations in the presence of noise. IEEE Transactions on Information Theory, 52(1):6–18, January 2006

[5] Gene H Golub and Charles F Van Loan Matrix computations, volume 3. JHU Press, 2012

[6] Lin Xiao and Stephen Boyd Fast linear iterations for distributed averaging. Systems Control Letters, 2004

[7] Joel Tropp, Anna C Gilbert, et al Signal recovery from random measurements via orthogonal matching pursuit. Information Theory, IEEE Transactionson, 53(12), 2007

[8] R. Tron and R. Vidal, "Distributed computer vision algorithms through distributed averaging," in Proc. IEEE CVPR, 2011, pp. 57–63

[9] S. P. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, “Mixing times for random walks on geometric random graphs.” in Proc. ALENEX/ANALCO. SIAM, 2005, pp. 240–249

[10] L. Xiao, S. Boyd, and S.-J. Kim Shorter version appeared in Proceedings of the 17th International Symposium on Mathematical Theory of Networks and Systems (MTNS), pages 2768-2776, July 2006, Kyoto

[11] H. Raja and W. U. Bajwa, “Cloud K-SVD: Computing data-adaptive representations in the cloud,” in Proc. 51st Allerton Conf., 2013

[12] H. Raja and W. U. Bajwa, “A convergence analysis of distributed dictionary learning based on the K-SVD algorithm,” in Proc. IEEE Intl. Symp. Information Theory (ISIT), 2015.

[13] R. Olfati-Saber, J. A. Fax, and R. M. Murray, “Consensus and cooperation in networked multi-agent systems,” Proc. IEEE, Jan. 2007