http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

Information Technology Journal 13 (2): 269-277, 2014
ISSN 1812-5638 / DOI: 10.3923/itj.2014.269.277
© 2014 Asian Network for Scientific Information

Algorithm Acceleration of Compressed Sensing with Cloud

Zhang Yong-Ping, Zhang Gong-Xuan and Zhu Zhao-Meng
School of Computer Science and Engineering,
Nanjing University of Science and Technology, Nanjing, China

Abstract: Compressed Sensing (CS) is a new sampling approach. It can accomplish simultaneously sampling and compressing. But the signal is reconstructed by solving a numerical optimization algorithm with high computational complexity in the theory of CS. The longer time of reconstruction is disadvantageous to the applications of CS. To overcome this problem, a complete scheme named Cloud_CS is introduced in this study. The scheme can speed up CS algorithms written in Python. Cloud_CS is built on OpenStack cloud computing platform. It achieve algorithms acceleration by serializing function, rewriting the function pmap and defining two marks which are named #remote and #parallelize. The experiments show that the reconstructed effects of signals is substantially equivalent to running locally and the running time can be reduced significantly. Then, Cloud_CS scheme is correctness and effectiveness.

Key words: Compressed sensing, cloud, algorithm acceleration, code migration, parallelization algorithm

INTRODUCTION

In recent years, a new approach of signal acquisition named Compressed Sensing (CS) is attracting more and more attention in signal processing, information science, electrical engineering, statistics and computer science after the theory was put forward in 2006 (Eldar and Kutyniok, 2012). It can represent a sparse or compressible signal using a very small number of measurements in some bases and has the ability to reconstruct the original signal with high accuracy by solving a numerical optimization algorithm (Boufounos *et al.*, 2007) where the traditional methods, such as Nyquist rate sampling theorem, have failed. In other word, compressed sensing can directly acquire the compressed form by throw away the redundant information when the signal is sampled. Thus, the amount of collected data is relatively small and the full-length signal can be reconstructed from the small sampling. This characteristic presents a good prospect of applicability in the field of information acquisition.

But compressed sensing is an algorithm with high computational complexity. For example, the calculation scale of reconstruction is equivalent to solving a linear programming problem with 8192×262144 when the length of signal is 8192 in BP algorithm (that is one of the compressed sensing algorithms (Chen *et al.*, 2001). So, the acceleration of CS algorithm is essential. Cloud computing technology (Armbrust *et al.*, 2010) can provide speedup results in some case because it can utilize the existing computing resources in the networks and avoid one-off investment. Saving investment makes it very attractive, especially for private physical networks.

An acceleration scheme of CS algorithm based on cloud technology, named Cloud_CS, is described in this article. Cloud_CS can migrate automatically the complex reconstructed operations to the cloud environment, so it can get more computing resources provided by cloud to speed up the algorithm.

COMPRESSED SENSING

Compressed sensing (Donoho, 2006), also known as compressive sampling (Candes, 2006) or compressive sensing (Baramuk, 2007) is a new sampling approach to simultaneous sensing and compression. It can potentially reduce the sampling and computation costs for acquisition of signals which have a sparse or compressible representation in some basis. In this section, a brief description of compressive sensing theory will be presented (Tsaig and Donoho, 2006; Candes and Wakin, 2008).

In general, the compressible original signal X, not necessarily sparse, has the sparse representation Θ based on a transformation basis $\Psi_{N\times N}$:

Corresponding Author: Zhang Yong-Ping, School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China

$$X = \Psi\Theta \tag{1}$$

where, conversion coefficient Θ is sparse that the vast majority entries in matrix Θ are zero (Elad, 2010). Then, Θ can be compressed by using measurement matrix $\Psi_{M\times N}$ (M<<N). The compressed data Y is defined in Eq. 2:

$$Y = \Phi\Theta \tag{2}$$

where, the specially designed Φ must be independent of the transformation basis Ψ. The compression ratio here is N:M. Generally, the compression radio can become bigger if coeffecient Θ is sparser.

In order to reconstruct original signal X from the compressed data Y, according to the compressed sensing theory, the following 0-norm optimization problem need to be solved:

$$\min \|\Theta\|_0 \text{ s.t. } Y = \Phi\Theta \tag{3}$$

where, $\|\Theta\|_0$ is the number of nonzero entries in Θ.

Equation 3 is a nonlinear optimization problem and can not be solved because it is a NP-hard (Baraniuk, 2007) problem. However, Chen *et al*. (2001) have proposed that a 1-norm optimization problem shown at Eq. 4 can substitute for the equation 3 when the $\Phi\Psi^T$ obey the Restricted Isometry Property (RIP) (Foucart and Lai, 2009) theorem:

$$\min \|\Theta\|_1 \text{ s.t. } Y = \Phi\Theta \tag{4}$$

Equation 4 is a convex optimization problem and it can be easily solved. The necessary and sufficient condition of RIP is that the measurement matrix Φ must be independent of the transformation basis Ψ.

The computing model of compressed sensing theory is shown in Fig. 1.

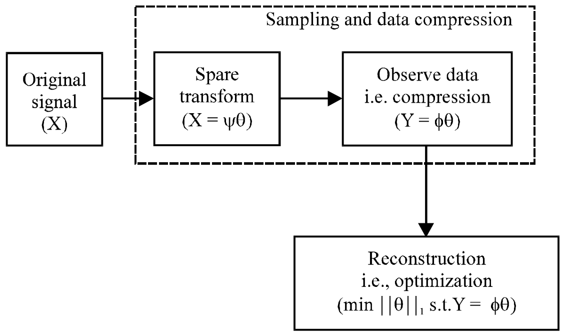
Sampling and data compression
Original signal (X)
Spare transform (X = ψθ)
Observe data i.e. compression (Y = ϕθ)
Reconstruction i.e., optimization (min ||θ||₁ s.t.Y = ϕθ)

Fig. 1: Theoretical framework of compressed sensing

DESIGN OF ACCELERATION SCHEME

Design: Usually, the algorithm acceleration is running programs by using more or better computing resources, such as more CPUs, more memories and faster hardware. The cloud can be convenient to provide a large number of computing resources for the user according to demand by networks. OpenStack (2012) is one of the most popular frameworks for running applications on cloud because it is simple to implement and massively scalable. OpenStack is an Infrastructure as a Service (IaaS; Bhardwaj *et al*., 2010) cloud computing project. It consists mainly of two modules: A cloud computing fabric controller-Nova and a massively scalable redundant storage system-Swift.

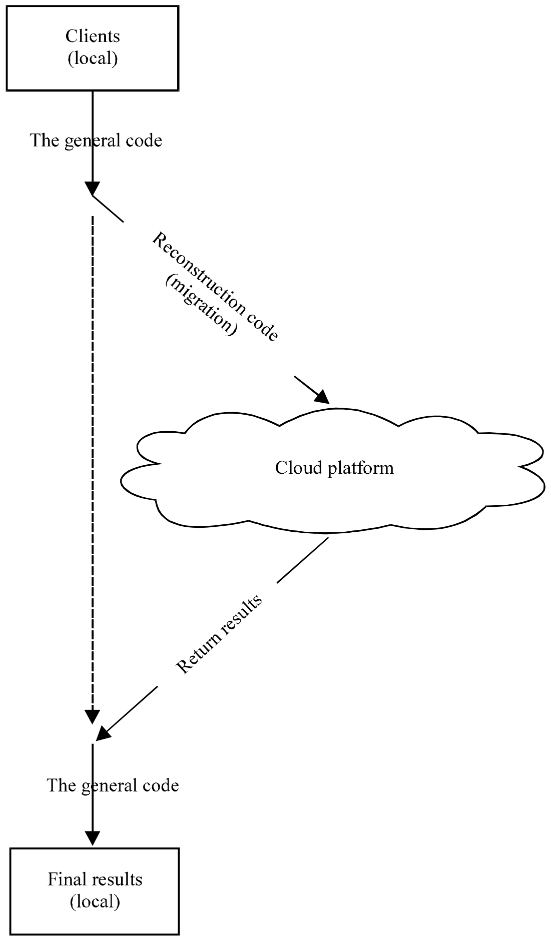

Fig. 2: Theoretical execution process of acceleration scheme

OpenStack can create and offer cloud computing services running on standard hardware and it can meet the needs of public and private clouds regardless of its sizes. The scheme of Cloud_CS is designed based on OpenStack. The theoretical execution process of Cloud_CS can be shown in Fig. 2.

Cloud_CS can use the computing resources of cloud platform to speed up CS algorithm written in Python (Cai *et al*., 2005) and provide on-demand services for users. In general, the collected data from a signal are acquired, transmitted and stored in compressed form in accordance with this scheme. The complex reconstruction operation will be migrated automatically to the cloud environment when users need to reconstruct the original signal. And then, cloud allocates some computing resources based on the needs of the users to speed up the reconstruction algorithm and eventually returns the calculated results.

The design scheme of Cloud_CS focuses on algorithm migration and parallelization.

Algorithm migration: How to migrate the code to cloud? It is based on functions in Cloud_CS scheme. In the scheme, a mark #remote is defined. If the mark #remote appears in the front of a function, the function will be migrated to cloud when it is called. For example, the presentation:

```
#remote
def reCloud (···):
```

indicates that the function reCloud will be migrated to cloud platform.

Generally, there are two ways to achieve the function migration. (1) To package the entire application and send it to cloud: This method is simple to implement, but it will send more redundant information and increase the time of data transmission. (2) To serialize function objects and variables, then package the function and send it to cloud: The serialization process involves serializing function, operating environment and results. This method can transmit the minimum amount of data, but it is complex and need to study the language specific features.

The second method is chosen in the design of Cloud_CS. The python objects are serialized by rewriting two functions. The functions are dumps function to serialize and loads function to de-serialize. They are based on the built-in modules of marshal (http://docs.python.org/2/library/marshal.html) and pickle (http://docs.python.org/lib/module-pickle.html). At the same time, another module named "Husky" is defined in the scheme. It can serialize more python objects; include functions, classes and running environment. The procedure of function migration can be shown in Fig. 3.

Parallelization loops: Multiple the same operations can be simultaneously dealt with if the loop can be processed in parallel. The parallel processing (Almasi and Gottlieb, 1989) of loops can accelerate the execution speed of the algorithm, but not all loops can be processed via parallel processing. In order to indicate parallelizable loops, another mark #parallelize is defined in the scheme. The codes will be processed in parallel if they are marked by #parallelize. For example, the presentation:

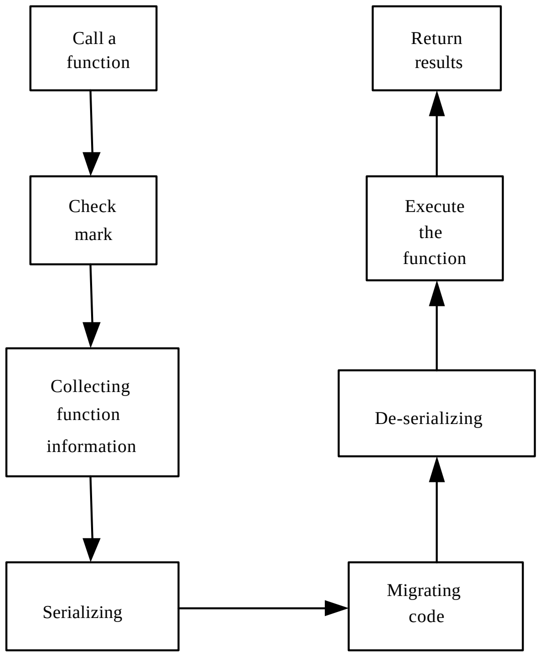

Fig. 3: Running procedure of a function with the mark #remote when it is called. The procedure includes function call, collecting information, serializing and de-serializing, running, return results. The related informations of function collected include variables, input and output, environmental parameters, the results, etc

```
#parallelize
for i in ···:
```

indicates the for statement will be processed in parallel. The loops are mainly for statement, while statement, list comprehensions, set comprehensions and dict comprehensions in Python. Because the CS algorithm mainly uses for statements, look at the parallelization of for statement, for example.

In Python, the list comprehensions can provide a concise way to create lists. Some for statements also can be transformed into list comprehensions. For example, there is a loop of for:

```
[(a, b) for a in [2, 3, 4] for b in [1, 3, 4]
   if a! = b]
```

The result is:

[(2, 1), (2, 3), (2, 4), (3, 1), (3, 4), (4, 1), (4, 3)]

So, the loop of for is equivalent to the list comprehension:

```
combs = []
for a in [2, 3, 4]:
    for b in [1, 3, 4]:
        if a ! = b:
            combs.append((a, b))
combs
```

Then, the for statement can be transformed into a list which can easily achieve the parallelization.

In addition, the "map", a built-in function in Python, is used to create recursive procedure. The function is not suitable for parallelization. Its migration and parallelization is achieved by rewriting it to pmap function. A new class that inherits the list comprehensions is defined in pmap function. It can not wait for all the results to continue to perform the following iteration by using the new class in the iteration operations and then parallel processing can be achieved.

Most of the design can be supported by famous scientific packages in Python, such as scipy (Jones *et al*., 2001), numpy (Developers, 2010) and matplotlib (Hunter, 2007).

ALGORITHM AND ITS MODIFICATION

Signal reconstruction is achieved by solving numerical optimization problems in compressed sensing. It has different reconstruction algorithms depending on different optimization methods. Currently, the popular reconstruction algorithms of compressed sensing can be grouped into three broad categories: Convex relaxation method, greedy algorithm and combination algorithm. Basis Pursuit (BP) algorithm is a representative algorithm of convex relaxation method. Greedy algorithm mainly includes Matching Pursuit (MP; Neff and Zakhor, 1997) algorithm and its improved algorithm.

OMP algorithm: As one of the greedy algorithms, OMP algorithm (Tropp and Gilbert, 2007) is a sparse approximation algorithm. The major advantages of this algorithm are easy to implement and its speed. Its computation complexity is lower than BP algorithm and its iteration times are less than MP algorithm. The experimental verification of this study is to use OMP algorithm.

The main idea of the OMP algorithm is to choose the column of measurement matrix Ψ, at each iteration, which is related to the greatest extent with the present redundant vector by solving the following optimization problem:

$$\lambda_t = \max_{i=1,2,3,\ldots,M} |\langle r_{t-1}, \varphi_i \rangle| \tag{5}$$

where, λ_t is the index of column at t-th iteration, r_{t-1} is the residual and φ_i is the i-th column of measurement matrix Φ; then a new estimated signal can be obtained by solving the least-square method:

$$x_t = \min_x \|\Phi_t x - y\|_2 \tag{6}$$

where, x_t is the solutions of t-th K-sparse approximation; followed by subtracting the related part from measurement vector and iterate on the residual.

The pseudo code of OMP algorithm can be shown in the following algorithm description (Tropp and Gilbert, 2007) and the flow chart is shown in Fig. 4.

Input:

- Measurement matrix Φ
- Sampling vector y
- Sparse degree K

Output: The K-sparse approximation solution x' of original signal x.

Restructuring procedure:

- Initialization: $r_0 = y$, $\Lambda_0 = \varnothing$, $t = 1$
- To obtain λ_t by solving Eq. 5
- Update $\Lambda_t = \Lambda_{t-1} \cup \{\lambda_t\}$ and $\Phi_t = [\Phi_{t-1}, \Phi_{\varepsilon t}]$
- To obtain new approximation solution x_t by solving Eq. 6
- Update $r_t = y - |\mu_t x_t$, $t = t+1$
- **IF** t<K **THEN** return 2)

Algorithm migration: Using OMP algorithm, the original signal can be reconstructed or solved an approximation solution. In order to migrate the algorithm to cloud platform, the mark of #remote must be added in front of the function. That is:

```
#remote
def OMP(y, Φ, k):
    function-body
```

Here, OMP(y, Φ, k) is the name of OMP algorithm. It has been previously defined. With the above definition, the OMP function can be automatically serialized and sent to the cloud environment in each calls.

Parallelization: The OMP algorithm can reconstruct the signal from a vector $X_{N\times1}$. It need to repeatedly perform OMP algorithm to reconstruct a signal when the signal X is N*L. This can be a substantial increase in execution time of reconstruction algorithm. The design scheme is mainly devoted to this case. This scheme can run multiple OMP algorithms at the same time and send them to

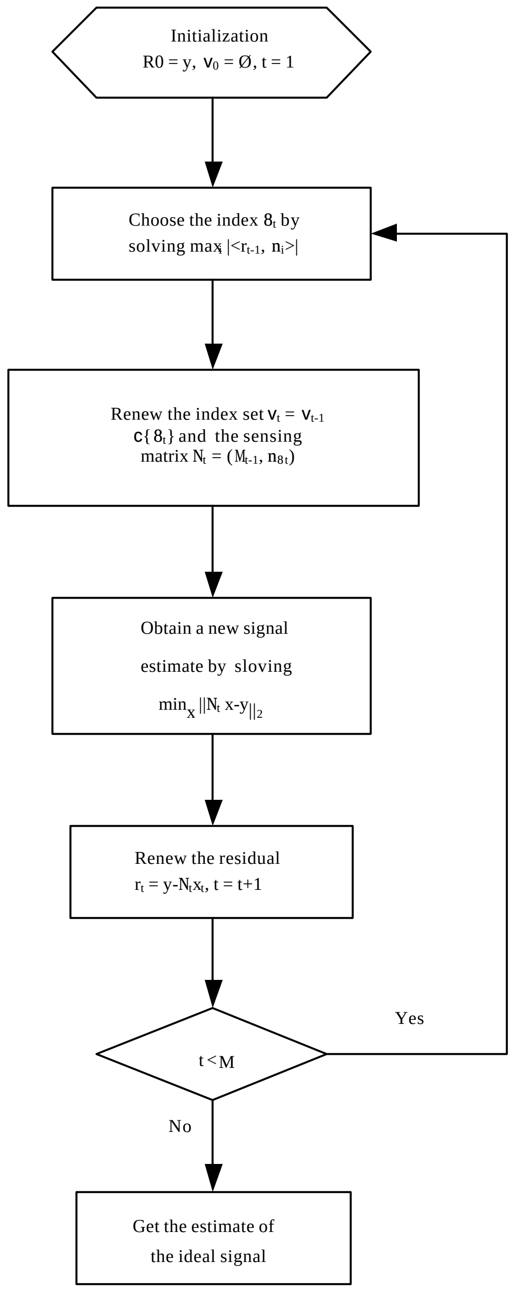
Initialization
R0 = y, V₀ = Ø, t = 1
Choose the index 8ₜ by solving maxᵢ |<rₜ₋₁, nᵢ>|
Renew the index set Vₜ = Vₜ₋₁ c{8ₜ} and the sensing matrix Nₜ = (Mₜ₋₁, n8ₜ)
Obtain a new signal estimate by sloving minₓ ||Nₜ x-y||₂
Renew the residual rₜ = y-Nₜxₜ, t = t+1
t<M
Yes
No
Get the estimate of the ideal signal

Fig. 4: Flow chart of OMP algorithm. It can approach the original signal by gradually iterative approximation

different computing resources to perform simultaneously, thereby improving execution speed of the reconstruction algorithm.

For the signal $X_{N\times1}$, the reconstructed time is smaller too and it does not require using any accelerated program when N is smaller. The signal can be divided into multiple small pieces with the same length when N is larger and then the signal $X_{N\times1}$ may be transformed into the other signal $X'_{N1\times N2}$, here the N1 is the length of a small piece and the N is equal to N1×N2 or the N+s is equal to N1×N2 (s<N1). So, the scheme can also be used to speed up the reconstruction algorithm.

For high-dimensional signals, they can also be converted to the forms of signal X_{N*L}. For example, consider the signal $X_{N\times S\times T}$. It can be transformed into the signal $X'_{N\times(S\times T)}$, i.e. $X'_{N\times L}$, here L is equal to S×T. The signals $X_{N\times S\times T\times U}$, $X_{N\times S\times T\times U\times V}$ and other high-dimensional signals can be taken the same treatment. Then, the scheme can also be used.

The parallelization of loops also requires the mark #parallelize. For example:

```
#paralleize
for i in xrange(N):
    X[:, i] = OMP(Y[:, i].reshape((-1, 1)), Φ, k)
```

Like the above for statement, the loops will be automatically executed in parallel if #parallelize is marked before the loops statement.

EXPERIMENTS AND ANALYSIS

Experimental environment: In the experiments of this study, the local computing platform for running test programs is a personal computer with 2.4 GHz Intel(R) Core(TM) 2 Duo dual-core processor (E4600), 2 GB memory, 64 bits, 10/100 Mbps wired Ethernet and running windows 7 SP1.

An OpenStack IaaS platform has been built on a cluster of IBM BladeCenter rack servers. For each server, there is a 2 GHz Intel Xeon quad-core processor, 24 GB memory, 64 bits, 1000 Mbps wired Ethernet and runs Ubuntu server edition. A various numbers of tiny VM instances (1VCPU/0GB Disk/512MB Ram) will be launched for these experiments in the cloud computing environment. Per computing resource is defined as a tiny VM instance in these experiments.

Next, this scheme will be verified via well-designed three groups of experiments. These three groups of experiments are used to verify the correctness of the algorithm, accelerating effects with different numbers of computing resources and accelerating effects for the different sizes of signals. For each group of experiments, the reconstruction algorithm should be repeatedly executed and then calculate its average running time.

Verification of correctness: This group of experiments will test the impact on reconstruction results when the

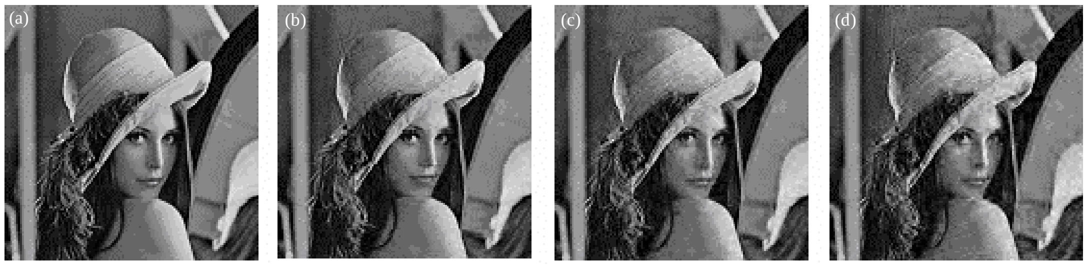

Fig. 5(a-d): For different compression ratio, the reconstruction effects of signal when the reconstructed algorithm run in cloud, (a) Origional image, (b) Reconstructed image (N:M = 2:1), (c) Reconstructed image (N:M = 3:1) and (d) Reconstructed image (N:M = 4:1)

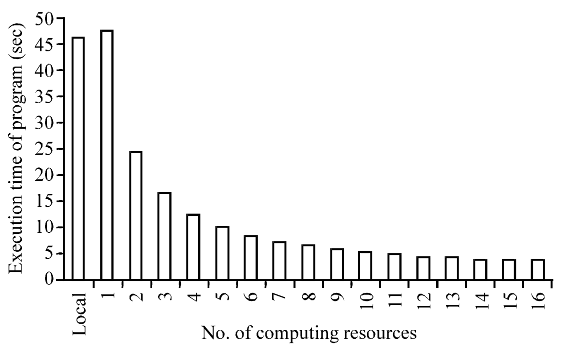

Fig. 6: Comparison of running time when the No. of computing resources requested are different. The x-axis shows the No. of computing resources. The running time is the average time after repeated running for the same signal

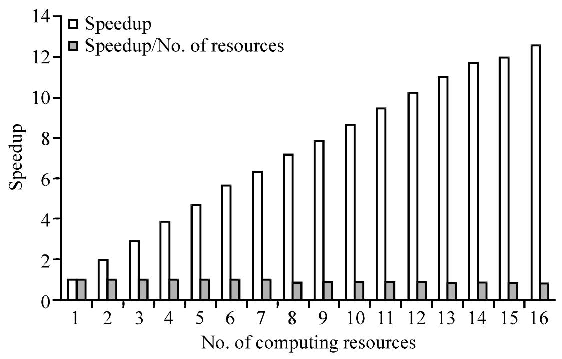

Fig. 7: Comparison of speedup when the No. of computing resources requested is different. The blue item shows the speedup size and the purple item shows contribution of per computing resource. The No. of computing resources change from 1 to 16

Table 1: PSNR comparison of running in local and cloud platform when the compression ratio is different. The first row shows the PSNR when algorithm is running locally. The second row shows the PSNR when algorithm is running in cloud

	Comparison of PSNR values		
Location of code to run	N:M = 2:1	N:M = 1:3	N:M =1:4
Local	30.7480	29.0053	26.4821
Cloud platform	30.9726	28.8986	26.3639

algorithm is migrated and parallelized. It will reconstruct a standard image named lena.bmp with 256*256 pixels on the local platform and in the cloud computing environment. These reconstruction effects are shown in Table 1 and Fig. 5.

Consider the comparison of PSNR (Power Signal-to-Noise Ratio) in Table 1. The PSNR of reconstructed signal is almost the same whether the codes are run in the local or sent to cloud, the nuances exist only because there is difference in the sparse transform and observation matrix. As Fig. 6 shows, the approximating value of original signal can be reconstructed and accepted.

Effects of different numbers of computing resources: This group of experiments will observe the effects of algorithm acceleration when the number of computing resources varies. Similarly, this group of experiments will reconstruct the standard image named lena.bmp with 256*256 pixels. It changes the number of computing resources from 1 to 16. For various numbers of computing resources, the average execution time and speedup are computed after the reconstruction algorithm is repeatedly executed. The results of this group of experiments can be shown in Fig. 6 and 7.

Table 2: Comparison of speedup effects when the sizes of signal are different. The first column is the size of reconstruction signal. For every signals, the No. of computing resources used to test are 1, 8, 16

Image size	No. of computing resources	Average execution time (sec)	Speedup	Speedup/No. of resource
64*64	1	0.9262	1.00	1.00
	8	0.2710	3.42	0.43
	16	0.1911	4.85	0.30
128*128	1	5.9534	1.00	1.00
	8	1.1390	5.23	0.65
	16	0.7895	7.54	0.47
256*256	1	47.5001	1.00	1.00
	8	6.6415	7.14	0.89
	16	3.7026	12.57	0.79
512*512	1	455.1829	1.00	1.00
	8	60.5982	7.51	0.94
	16	31.3774	14.51	0.91
1024*1024	1	6056.9360	1.00	1.00
	8	794.6400	7.62	0.95
	16	400.8561	15.11	0.94

Figure 6 shows the execution time of OMP algorithm in the scheme when the number of computing resources is different. The local means that the algorithm runs directly on a computing resource and without migrating. The number "1~16" is the numbers of required computing resource and the execution time decreases with increasing in computing resources, which shows that the speedup scheme is feasible and effective.

Figure 7 shows the speedup and contribution of per computing resource of the OMP algorithm in the scheme when the number of computing resources is different. It is obvious that the speedup is increasing with increasing in computing resources, which is consistent with the change of execution time in Fig. 6. However, the contribution of per computing resource to speedup contrary gradually reduced when computing resources is increasing. This is because the costs of system scheduling and synchronization between the various computing resources are also increasing with increasing in computing resources. Therefore the contribution of per computing resource is reduced with increasing in computing resources.

Effects of different sizes' signal: Finally, the effect of signal with different sizes will be tested when the number of computing resources is the same. In these experiments, execution time and speedup are present when the sizes of signal are 64*64, 128*128, 256*256, 512*512 and 1024*1024. Table 2 shows the results of the group of experiments when the numbers of computing resources are 1, 8 and 16.

As it is obvious in Table 2, the scheme can significantly reduce execution time of the reconstruction algorithm for all signals of different sizes and only the decreased proportion is different. It can be found that the speedup and the contribution of per computing resource are all increasing with increasing in sizes of signals when computing resources are the same by seeing the rightmost two columns in Table 2. This is mainly because the reconstructed time is relatively small when the sizes of signal are smaller; then there is relatively large proportion of the costs in code migration and program scheduling.

CONCLUSION AND FUTURE WORK

In this study, an algorithm acceleration framework based on the cloud computing platform is proposed and implemented. The framework can speed up the program written in Python (Not limited to CS algorithms, although it is developed for compressed sensing). The accelerated scheme can achieve algorithm acceleration by defining two marks #remote and #parallelize and rewriting the function pmap. The mark #remote can send automatically the function to cloud environment; the mark #parallelize can execute the loop in parallel and the function pmap can break the order of iteration. The experimental results prove the framework's correctness and effectiveness. For example, the execution time is decreased from 47 to 4 sec by using 16 computing resources in Fig. 6. The research of scheme is significant because it can solve the problem of high time complexity of traditional compressed sensing algorithms and broaden its applications.

Next work: The future researches will be focused on the following two aspects:

- Optimization of scheme and algorithm. In this study, the algorithm acceleration framework and test algorithm is achieved by our own. All codes are written in Python language and without any optimization. So the execution time is higher than Matlab program that can call the standard library. The future, these algorithms will be optimized in the experiment in order to improve the performance of the acceleration framework
- To study the applications of compressed sensing in Internet of Things. The Internet of Things (Ashton, 2009; ITU, 2005), probably be also known as wireless sensor networks by somebody, is one the important technologies which can change humanity's life. It has become a national strategic emerging industry (Zhu *et al*., 2009; EU, 2009; ITS

Headquarters, 2009; MIIT, 2011). There are lots of collected data in large-scale Internet of Things. For example, a logistics monitoring system supported by Internet of Things in a supermarket, if you need to track the location and status information of 10 million products, the amount of data generated will reach 10 GB in one day and 3.65 TB in one year

To overcome the problem, the application of compressed sensing is studied in Internet of Things because it can potentially reduce the amount of collected data. However, the high computational complexity of reconstruction algorithms will increase the costs of data processing. The Cloud_CS framework can have a good interaction with Internet of Things. The cloud computing platform can be deployed in Internet of Things because there are lots of computing resources and storage resources in Internet of Things, such as CPUs, memories, personal computers, servers, monitoring equipments and even GPGPU. Then, the acceleration framework can decrease the execution time of compressed sensing algorithm and Internet of Things can provide computing resources for the framework.

In addition, the OMP algorithm and some authentication algorithms are used in this experiment; some other reconstruction algorithm will also be explored.

ACKNOWLEDGMENTS

We wish thank my classmate Rui Zhi and Professor Xiaohui Liu of Brunel University for their valuable suggestions. The study's work is supported by Jiangsu 973 project (No. BK2011022), Natural Science Foundation of China (No. 61170035), Nature Science Foundation of China (No. 61272420) and NUST project (No. 2011YBXM18).

REFERENCES

Almasi, G.S. and A. Gottlieb, 1989. Highly Parallel Computing. Benjamin-Cummings Publishers Co., Inc., USA.

Armbrust, M., A. Fox, R. Griffith, A.D. Joseph and R. Katz *et al.*, 2010. A view of cloud computing. Commun. ACM., 53: 50-58.

Ashton, K., 2009. That internet of things thing. RFID Journal, July, 22. http://www.rfidjournal.com/articles/view?4986.

Baraniuk, R.G., 2007. Compressive sensing. IEEE Signal Process. Magazine, 24: 118-121.

Bhardwaj, S., L. Jain and S. Jain, 2010. Cloud computing: A study of Infrastructure As A Service (IAAS). Int. J. Eng. Inform. Technol., 2: 60-63.

Boufounos, P., M.E. Duarte and R.G. Baraniuk, 2007. Sparse signal reconstruction from noisy compressive measurements using cross validation. Proceedings of the IEEE/SP 14th Workshop on Statistical Signal Processing, August 26-29, 2007, Madison, USA., pp: 299-303.

Cai, X., H.P. Langtangen and H. Moe, 2005. On the performance of the python programming language for serial and parallel scientific computations. Scient. Programming, 13: 31-56.

Candes, E.J. and M.B. Wakin, 2008. An introduction to compressive sampling. IEEE Signal Process. Magazine, 25: 21-30.

Candes, E.J., 2006. Compressive sampling. Proceedings of the International Congress of Mathetmaticians, Volume 3, August 22-30, 2006, Madrid, Spain, pp: 1433-1452.

Chen, S.S., D.L. Donoho and M.A. Saunders, 2001. Atomic decomposition by basis pursuit. Siam Rev., 43: 129-159.

Developers, N., 2010. Scientific computing tools for python-numpy. http://www.numpy.org/.

Donoho, D.L., 2006. Compressed sensing. IEEE Trans. Inform. Theory, 52: 1289-1306.

EU, 2009. Internet of Things-an action plan for Europe. Commission of the European Communities, COM(2009) 278 Final, Brussels, June 18, 2009.

Elad, M., 2010. Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing. Springer Publishers, Inc., UK.

Eldar, Y.C. and G. Kutyniok, 2012. Compressed Sensing: Theory and Applications. 1st Edn., Cambridge University Press, UK.

Foucart, S. and M. Lai, 2009. Sparsest solutions of underdetermined linear systems via lp-minimization for 0<P<1. 395-407.

Hunter, J.D., 2007. Matplotlib: A 2D graphics envir. Comput. Sci. Eng., 9: 90-95.

ITS Headquarters, 2009. i-Japan strategy 2015. Striving to Create a Citizen-Driven, Reassuring and Vibrant Digital Society, Towards Digital inclusion and innovation, July 6, 2009.

ITU, 2005. ITU internet reports 2005-the internet of things: Executive summary. ITU, Pages: 17.

Jones, E., T. Oliphant and P. Peterson, 2001. SciPy: Open source scientific tools for Python. http://www.citeulike.org/group/2018/article/2644428.

MIIT, 2011. The internet of things development planning in twelfth five-year. Ministry of Industry and Information Technology, China. http://www.gov.cn/zwgk/2012-02/14/content_20 65999.htm.

Neff, R. and A. Zakhor, 1997. Very low bit-rate video coding based on matching pursuits. IEEE Trans. Circuits Syst. Video Technol., 7: 158-171.

OpenStack, L., 2012. OpenStack install and deploy manual-Ubuntu. http://docs.openstack.org /folsom/openstack-compute/install/apt/content/.

Tropp, J.A. and A.C. Gilbert, 2007. Signal recovery from random measurements via orthogonal matching pursuit. IEEE Trans. Inform. Theory, 53: 4655-4666.

Tsaig, Y. and D.L. Donoho, 2006. Extensions of compressed sensing. Signal Process., 86: 549-571.

Zhu, J., X. Fang, Z. Guo, M.H. Niu, F. Cao, S. Yue and Q.Y. Liu, 2009. IBM cloud computing powering a smarter planet. Cloud Comput., 5931: 621-625.