Research on Iterative Method in Solving Linear Equations on the Hadoop Platform

LIU Yi-di
City institute,Dalian university of technology, liuyidi80@yeah.net

***Abstract*—Solving linear equations is ubiquitous in many engineering problems, and iterative method is an efficient way to solve this question. In this paper, we propose a general iteration method for solving linear equations. Our general iteration method doesn't contain denominators in its iterative formula, and this relaxes the limits that traditional iteration methods require the coefficient a_{ii} to be non-zero. Moreover, as there is no division operation, this method is more efficient. We implement this method on the Hadoop platform, and compare it with the Jacobi iteration, the Guass-Seidel iteration and the SOR iteration. Experiments show that our proposed general iteration method is not only more efficient, but also has a good scalability.**

Keywords- Linear equations; Iterative method; Hadoop

I. INTRODUCTION

Linear equations play an important role in describing many significant engineering problems. Iterative method is a valid and rapid method for solving such linear equations. During those days before computer is invented, many artificial method are introduced to solve them. Those methods only solve these linear equations in theory. However, while facing practical engineering problems, these methods can't be used very well, and they cannot be implemented by computer programs. Iterative method is the best way to use computer to solve these problems. Besides linear equations, a lot of matrix computation can be solved by computer programs, such as solving Nonlinear Evolution Equations [1], Matrix Factorization [2], Inverse Eigenvalue Problem for Matrix [3], and game matrix modeling [4].

While iteratively solving the linear equations, traditional iterative methods, such as Jacobi iteration [5], Guass-Seidel iteration [6], and SOR iteration [7], all contain division operation in their iterative formulae. However, division in computer requires the coefficient a_{ii} to be non-zero, and consumes more computation resources than addition, subtraction and multiplication operations.

MapReduce [8] is a simple computation model for processing huge amounts of data in massively parallel fashion, using a large number of commodity machines. By automatically handling the lower level issues, such as job distribution, data storage and flow, and fault tolerance, it provides a simple computational abstraction. Many matrix problems, such as PageRank [9-12], HITS [13-15], and SALSA [16], can be efficiently solved by MapReduce-like systems. Among all the MapReduce-like systems, its open source implementation Hadoop is the most popular one, and it has been the product standard in many companies.

In this paper, we promote a general iterative method for solving linear equations. Our general iterative method does not contain division operation, and it can be easily implemented by the MapReduce programming model. We implement the general iterative method on the Hadoop platform [17]. Theoretical analysis and experiments show that our iterative method is not only more efficient than existing methods, but also has a good scalability.

A. Problem description

Given a coefficient matrix $A = (a_{ij})$, which contains n rows and n columns, a linear equations can be described as

$$Ax = b \tag{1}$$

where b is a column vector contained n items, and x is the solution vector also contained n items.

Herein and throughout the paper, we consume that the items of A, x, and b are all real numbers. For solving these linear equations, the problem is that given A and b, find a solution x, which make these equations equal from left to right.

In addition, formula (1) can also be described as formula (2) and (3).

$$(a_{i\cdot}, x) = b_i \qquad i=1, 2, \cdots, n \tag{2}$$

where $a_{i\cdot}$ is the ith row of A, b_i is the ith item of b, and $(\cdot, \cdot)$ is the inner product symbol.

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad i=1, 2, \cdots, n \tag{3}$$

where a_{ij} is the ith row and the jth column of A, x_j is the jth item of x, and b_i is the same with formula (2).

II. RELATED WORK

In this section, we review some iterative methods for solving linear equations.

A. Jacobi iteration

From formula (3), we can have that

$$x_i = (b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j) / a_{ii} \qquad i=1, 2, \cdots, n \tag{4}$$

where $a_{ii} \neq 0$ is the ith row of A, b_i is the ith item of b, and $(\cdot, \cdot)$ is the inner product symbol.

If we use iterative method to solve the linear equations, then we can use the kth solution to iteratively figure out the k+1th solution. The Jacobi iterative formula is as follows:

$$x_i^{k+1} = (b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(k)}) / a_{ii} \quad i=1, 2, \cdots, n \tag{5}$$

In the beginning, we initiate x with $x^{(0)}$, where $x_i^{(0)}$ can be any real number. In the k+1th iteration, we use $x^{(k)}$ to calculate the $x^{(k+1)}$. In theory, if we have $x^{(k)} = x^{(k+1)}$, then the iteration is terminated, and the solution for the linear equations is $x^{(k)}$. As a matter of fact, all items in x are real numbers, and if we want $x^{(k)} = x^{(k+1)}$, then the iteration will never be terminated. We introduce an m-dimensional constant column vector ε. In the iteration process, if $|x^{(k+1)}-x^{(k)}|< \varepsilon$, then the iteration will be terminated.

B. Guass-Seidel iteration

In order to accelerate the convergence of the Jacobi iteration, we use $x_1^{(k)}, x_2^{(k)}, \cdots, x_n^{(k)}$ to figure out $x_1^{(k)}$, use $x_1^{(k+1)}, x_2^{(k)}, \cdots, x_n^{(k)}$ to figure out $x_2^{(k+1)}$, ⋯, and use $x_1^{(k+1)}, x_2^{(k+1)}, \cdots, x_{(n-1)}^{(k+1)}$ to figure out $x_n^{(k)}$. Then we have the following Guass-Seidel iteration formula:

$$x_i^{k+1} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}}{a_{ii}} \quad i=1, 2, \cdots, n \qquad (6)$$

C. SOR iteration

The Guass-Seidel iteration formula is an improvement of the Jacobi iteration formula. However, $x_i^{(k+1)}$ is figured out by all items of $x^{(k)}$ except for $x_i^{(k)}$. That is $x_i^{(k+1)}$ is nothing about $x_i^{(k)}$. To address this problem, we can introduce a constant ω, and $x_i^{(k+1)}$ comes from $x_i^{(k)}$ by a factor of ω. From this point of view, formula (6) can also be changed into the following SOR iteration formula:

$$x_i^{k+1} = \frac{\omega\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)}{a_{ii}} + (1-\omega)x_i^k \quad i=1, 2, \cdots, n \qquad (7)$$

There is a common disadvantage in Jacobi iteration, Guass-Seidel iteration and SOR iteration. The coefficients a_{ii} are the denominators, and this requires that they can't be zero, and at the same time, the division operations in computer will take much more time.

D. Programming Model on MapReduce

In MapReduce programming model, computations are done in three phases, Map, Shuffle and Reduce. The Map phase reads a collection of values or key/value pairs from an input source, and by invoking a user defined Mapper function on each input element independently and in parallel, emits zero or more key/value pairs associated with that input element. The Shuffle phase groups together all the Mapper-emitted key/value pairs sharing the same key, and outputs each distinct group to the next phase. The Reduce phase invokes a user-defined Reducer function on each distinct group, independently and in parallel, and emits zero or more values to associate with the group's key. The emitted key/value pairs can then be written on the disk or be the input of a Map phase in a following iteration.

The basic MapReduce abstraction can sometimes be highly restrictive, as it requires any computation to be translated into the rigid framework of that contains only one Map and one Reduce operation. As the computations get logically more complicated, this becomes increasingly more challenging. Therefore, many models and system implementations have been proposed to extend the basic MapReduce framework [18-21]. These extensions provide higher level languages for the programmers (such as Pig-Latin or SCOPE), which include higher level primitives such as joins.

III. GENERAL ITERATION METHOD

In this paper, we propose a general iteration method for solving linear equations. Our method is not only simply implemented by computer programs, but also contains no denominators. The general iteration formula is as follows:

$$x^{k+1} = (A+E)x^k - b \qquad (8)$$

where E is the identity matrix of n rows and n columns, and e_{ij} equal to 1 if i equals to j, and 0 otherwise.

To illustrate formula (8) in detail, we can also rewrite this formula with all their items, and that is

$$x_i^{k+1} = \sum_{j=1}^{n} a_{ij}x_j^{(k)} + x_i^k - b_i \quad i=1, 2, \cdots, n \qquad (9)$$

A. Proof of convergence

In this subsection, we will give a detailed proof of the convergence of the general iteration formula.

From formula (8) we can see that $x_i^{(k+1)}$ is a linear function of $x_i^{(k)}$. if we replace A+E with B, then we have

$$x^{k+1} = Bx^k - b \qquad (10)$$

Theorem 1. Formula (10) converges if and only if the spectral radius of matrix B $\rho(B) < 1$.

Proof:

Suppose that x is a solution of formula (9), we can get

$$x = Bx - b \qquad (11)$$

If we subtract formula (10) with formula (11), then we have

$$x^{k+1} - x = B(x^k - x) \qquad (12)$$

With the recursive formula (12), we can get that $x_i^{(k+1)}$ is a linear function of $x_i^{(0)}$, and the formula is as follows:

$$x^{k+1} - x = B^k(x^0 - x) \qquad (13)$$

where B_k means that B times B in k times.

Taking the limit on both sides of formula (13), we have $\lim_{k\to\infty}(x^{k+1} - x) = \lim_{k\to\infty} B^k(x^0 - x)$, and we can further get that $\lim_{k\to\infty} x^{k+1} = x$ if and only if $\lim_{k\to\infty} B^k = 0$. In addition, we know that $\lim_{k\to\infty} x^{k+1} = \lim_{k\to\infty} x^k$, so we have

$$\lim_{k\to\infty} x^k = x \Leftrightarrow \lim_{k\to\infty} B^k = 0 \qquad (14)$$

Consume that λ is the eigenvalue of B, and ξ is its corresponding eigenvector, then $B^k\xi = \lambda^k\xi$. Hence, we have

$$\lim_{k\to\infty} x^k = x \Leftrightarrow \lim_{k\to\infty} B^k = 0 \Leftrightarrow |\lambda|<1 \Leftrightarrow \rho(B)<1 \tag{15}$$

B. Algorithm description on Hadoop

In this subsection, we will describe how we implement the general iterative method on the Hadoop platform. Our algorithm contains a main program, a Map function and a Reduce function, and the details are as follows.

Algorithm 1 GeneralIteration(A, b)

Input: a matrix A, a vector b, and a constant ε;
Output: a solution vector x for $Ax=b$;

```
1: let $x^T$={0, 0, ⋯, 0};
2: let $x'^T$={1, 1, ⋯, 1};
3: while | $x'^T$ - $x^T$ | > $\varepsilon$ do
4:   let $x^T = x'^T$;
5:   output = GeneralMapper($A$, $b$, $x^T$);
6:   $x'^T$ = GeneralReducer(output);
7: end while
8: return $x'^T$;
```

Algorithm 2 GeneralMapper (A, b, x^T)

Input: a matrix A, vectors b and x^T;
Output: key/value pair;

```
1: for all $b_i$ in $b$
2:     emit($i$, $b_i$);
3: end for
4: for all $a_{i\cdot}$ in $A$
5:     emit($i$, $a_{i\cdot}$);
6: end for
7: for $i$ in 1 to $n$
8:     emit($i$, $x^T$);
9: end for
```

Algorithm 3 GeneralReducer (key, $value$)

Input: output of GeneralMapper;
Output: a solution vector x for $Ax=b$;

```
1: for $i$ in 1 to $n$
2:     let $x_i' = \sum_{j=1}^{n} a_{ij}x_j + x_i - b_i$;
3: end for
4: return $x'^T$;
```

IV. Experiments

A. Experimental Setup

In this paper, we implement the general iteration method in Java on top of the Hadoop platform. Our experiments are executed on a cluster of 20 nodes, where each node is a commodity machine with a 2.16GHz Intel Core 2 Duo CPU and 1GB of RAM, running CentOS v6.0. In order to demonstrate the robustness of our algorithm and to show its performance, we present experiments with simulated datasets.

B. Dataset generation

In order to validate the efficiency of our algorithm, we do some simulated experiments on some artificial datasets. The idea of generation of datasets is as follows:

(1) for matrix A, we generate a $n \times n$ matrix, where each item is a random number between 0 and 1;

(2) for vector b, we generate a n-dimensional column vector, where each item is also a random number between 0 and 1.

C. Experimental results

In our experiments, we use JI to indicate the Jacobi iteration, GSI to indicate the Guass-Seidel iteration, SORI to indicate the SOR iteration, and our general iteration is abbreviated as GI. We compare our proposed GI with JI, GSI and SORI. Figures 1, 2 and 3 illustrate the execution time of n=100, n=500, and n=1000 respectively. From these figures we can see that our general iteration method is more efficient than the Jacobi iteration, the Guass-Seidel iteration and the SOR iteration.

Moreover, in order to test the scalability of our method, we change the size of problem, and observe if the execution time grows much more quickly than the problem size. If this happens, our method will be limited to small size of problems, and other our method can be applied to larger size of problems. Figure 4 gives the curve of execution time. From that figure we can see that with our method the execution time grows nearly linearly with respect to the size of problems, and thus we can conclude that our general iteration method is scalable to larger size of problems.

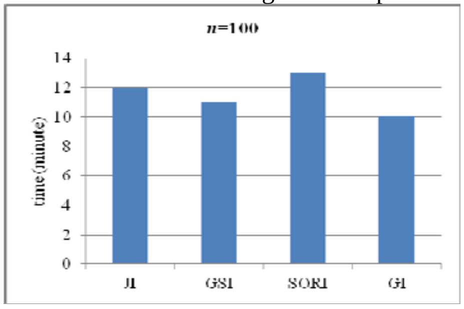

Figure 1. Execution time n=100

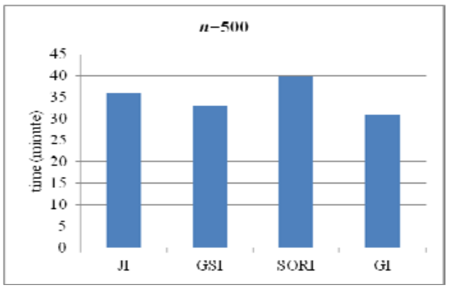

Figure 2. Execution time n=500

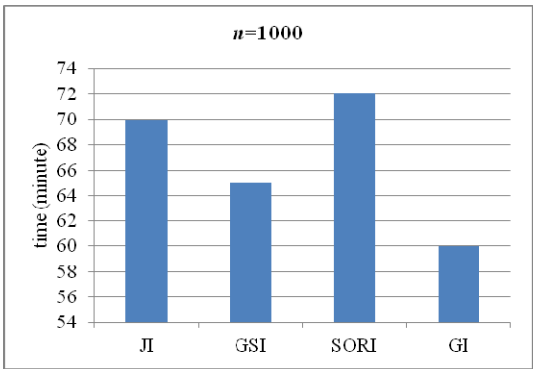

Figure 3. Execution time n=1000

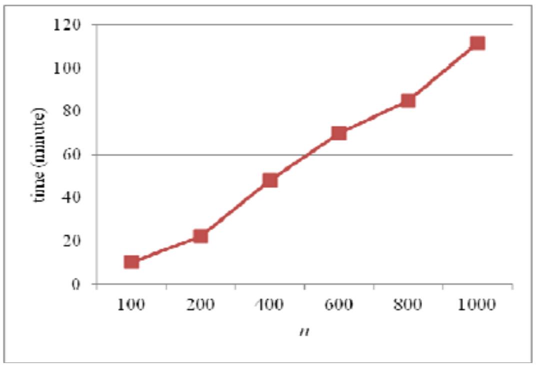

Figure 4. Scalability of GI

V. Conclusion

In this paper, we propose a general iteration method for solving linear equations. Our general iteration method doesn't contain denominators in its iterative formula, and this relaxes the limits that traditional iteration methods require the coefficient a_{ii} to be non-zero. Moreover, as there is no division operation, this method is more efficient. We implement this method on the Hadoop platform, and compare it with the Jacobi iteration, the Guass-Seidel iteration and the SOR iteration. Experiments show that our proposed general iteration method is not only more efficient, but also can be used in larger size of linear equations.

VI. References

[1] C. Qiu, M. Diao and Shubo Yue, A New Algebra Method and Its Applications for Nonlinear Evolution Equations, IJACT: International Journal of Advancements in Computing Technology, Vol.4, No.2, pp.41-49, 2012.

[2] T. Li, A New Algorithm for Triangular Factorization of the Inversion of Toeplitz Type Matrix, JDCTA: International Journal of Digital Content Technology and its Applications, Vol.6, No.23, pp.698-706, 2012.

[3] L. Feng, S. Yan, Y. He, Y. Yang and Ping Li, Inverse Eigenvalue Problem for Jacobi Matrix, JDCTA: International Journal of Digital Content Technology and its Applications, Vol.6, No.16, pp.395-402, 2012.

[4] H. Bing, C. Heshan and Li Tianzeng, The Mathematical Modeling for a Game of "Bears-change color", JDCTA: International Journal of Digital Content Technology and its Applications, Vol.6, No.12, pp.141-148, 2012.

[5] S. Sun, S. Wang, W. Shen, W. Xu and Y. Zheng, A Study of the Memory Wall within the Jacobi Iteration Method, pp.964-969, 2012.

[6] H. Yan and Q. Zheng, A LOOP DISTRIBUTED GUASS-SEIDEL ITERATIVE ALGORITHM FOR SOLVING LINEAR EQUATIONS, Jisuanji Yingyong yu Ruanjian, Vol.28, No.7, pp.262-263, 2011.

[7] W. Zheng and Z. Zhao, Analysis of Block-SOR Iteration for the three dimensional Laplacian, Anziam Journal, Vol.50, 2009.

[8] J. Dean and S. Ghemawat, Mapreduce: simplified data processing on large clusters, in OSDI, pp.1-10, 2004.

[9] L. Page, S. Brin, R. Motwani and T. Winograd, The PageRank citation ranking: bringing order to the web, 1999.

[10] T.H. Haveliwala, Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search, Knowledge and Data Engineering, IEEE Transactions on, Vol.15, No.4, pp.784-796, 2003.

[11] M. Richardson and P. Domingos, The intelligent surfer: Probabilistic combination of link and content information in pagerank, MIT Press, Cambridge, MA, 2002.

[12] A.N. Langville and C.D. Meyer, Deeper inside pagerank, Internet Mathematics, Vol.1, No.3, pp.335-380, 2004.

[13] J.M. Kleinberg, Authoritative sources in a hyperlinked environment, Journal of the ACM (JACM), Vol.46, No.5, pp.604-632, 1999.

[14] H. Deng, M.R. Lyu and I. King, A generalized Co-HITS algorithm and its application to bipartite graphs, pp.239-248, 2009.

[15] L. Li, Y. Shang and W. Zhang, Improvement of HITS-based algorithms on web documents, Proceedings of the 11th international conference on World Wide Web, pp.527-535, 2002.

[16] R. Lempel and S. Moran, The stochastic approach for link-structure analysis (SALSA) and the TKC effect, Computer Networks, Vol.33, No.1, pp.387-401, 2000.

[17] Hadoop,http://hadoop.apache.org/common/docs/r0.16.4/hdfsdesign.html

[18] Pig, http://hadoop.apache.org/pig.

[19] R. Chaiken, B. Jenkins, P.A. Larson, B. Ramsey and D. Shakib, et al., SCOPE: easy and efficient parallel processing of massive data sets, Proceedings of the VLDB Endowment, Vol.1, No.2, pp.1265-1276, 2008.

[20] C. Olston, B. Reed, U. Srivastava, R. Kumar and A. Tomkins, Pig latin: a not-so-foreign language for data processing, SIGMOD, pp.1099-1110, 2008.

[21] H. Yang, A. Dasdan, R. Hsiao and D. Parker, Map-reduce-merge: simplified relational data processing on large clusters, SIGMOD, pp.1029-1040, 2007.