Hadoop Recognition of Biomedical Named Entity Using Conditional Random Fields

Kenli Li, Wei Ai, Zhuo Tang, Fan Zhang, Lingang Jiang, Keqin Li, and Kai Hwang, *Fellow, IEEE*

Abstract—Processing large volumes of data has presented a challenging issue, particularly in data-redundant systems. As one of the most recognized models, the conditional random fields (CRF) model has been widely applied in biomedical named entity recognition (Bio-NER). Due to the internally sequential feature, performance improvement of the CRF model is nontrivial, which requires new parallelized solutions. By combining and parallelizing the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) and Viterbi algorithms, we propose a parallel CRF algorithm called MapReduce CRF (MRCRF) in this paper, which contains two parallel sub-algorithms to handle two time-consuming steps of the CRF model. The MapReduce L-BFGS (MRLB) algorithm leverages the MapReduce framework to enhance the capability of estimating parameters. Furthermore, the MapReduce Viterbi (MRVtb) algorithm infers the most likely state sequence by extending the Viterbi algorithm with another MapReduce job. Experimental results show that the MRCRF algorithm outperforms other competing methods by exhibiting significant performance improvement in terms of time efficiency as well as preserving a guaranteed level of correctness.

Index Terms—Biomedical named entity recognition, conditional random fields, MapReduce, parallel algorithm

1 Introduction

1.1 Motivation

With the rapid development of computational and biological technologies, biomedical literatures are expanding at an exponential rate. As one of the most concerned areas, papers on biomedicine have been published in a huge amount, reaching an average of 600,000 or more per year. Currently, the most authoritative biomedical literature database MEDLINE (Medical Literature Analysis and Retrieval System Online) in American National Library of Medical (NLM) has included the information of more than 7,000 kinds of important biomedical journals published in over 70 countries and regions since 1966, including more than 18 million articles [1]. The explosion of literatures in the biomedical domain promotes the application of text mining. Aiming to identify words or phrases referring to specific entities in biomedical literatures, biomedical named entity recognition (Bio-NER) is a critical step for the text mining. If biomedical named entities are not correctly and effectively identified, other tasks like relationship extraction, gene/protein normalization, and hypothesis generation cannot be performed effectively.

Current methods for Bio-NER fall into three general classes, i.e., dictionary-based methods [40], heuristic rule-based methods [24], and statistical machine learning methods [17]. Relying on dictionary-based methods could cause the low recall due to the continual appearance of new entities with the advancing biology research. Biological named entities do not follow any nomenclature, which makes rule-based methods hard to be perfect. Besides, rule-based systems require domain experts, and they are not portable to other NE types and domains. Machine learning methods are more robust and they can identify potential biomedical entities which are not previously included in standard dictionaries. More and more machine learning methods are introduced to solve the Bio-NER problem, such as Hidden Markov Model (HMM) [10], Support Vector Machine (SVM) [12], Maximum Entropy Markov Model (MEMM) [16], and Conditional Random Fields (CRF) [20], [33].

Conditional random fields, a type of conditional probability model, has been widely applied in biomedical named entity recognition [7], [28], [34]. The advantage of the CRF model is the ability to express long-distance-dependent and overlapping features. CRF has shown empirical success recently in Bio-NER, since it is free from the so-called label bias problem by using a global normalization. However, when facing large-scale data, the time efficiency of the CRF model with the traditional stand-alone processing algorithm is not satisfactory. For example, CRF takes approximately 45 hours (3.0 GHz CPU, 1.0 G memory, and 400 iterations) to train only 400 K training examples [30]. It is caused by the problem of CRF that the model parameter estimation cycle is long, because it needs to compute the global gradient for all features. The time complexity and space complexity of the whole algorithm show non-linear growth with the growth of the training data. To efficiently handle large-scale data, faster processing and optimization algorithms have become critical

- *K. Li, W. Ai, Z. Tang, and L. Jiang are with the College of Information Science and Engineering, Hunan University, Changsha, Hunan 410082, China. E-mail: lkl510@263.net, {aiwei, ztang}@hnu.edu.cn, jianglingang520@163.com.*
- *F. Zhang is with the Kavli Institute for Astrophysics and Space Research, Massachusetts Institute of Technology, Cambridge, MA 02139. E-mail: f zhang@mit.edu.*
- *K. Li is with the Department of Computer Science, State University of New York, New Paltz, New York 12561 and the College of Information Science and Engineering, Hunan University, Changsha, Hunan 410082, China. E-mail: lik@newpaltz.edu.*
- *K. Hwang is with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089. E-mail: kaihwang@usc.edu.*

Manuscript received 26 June 2014; revised 30 Oct. 2014; accepted 1 Nov. 2014. Date of publication 6 Nov. 2014; date of current version 7 Oct. 2015. Recommended for acceptance by Z. Du.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2368568

for biomedical big data. Hence, it is vital to develop new algorithms that are more suitable for parallel architectures.

The CRF model needs to consider three key steps, i.e., feature selection, parameter estimation, and model inference. The parameter estimation step is very time-consuming because of the large amount of calculations especially when the training data set is large, which becomes the most important reason that degrades the performance of the CRF model. An optimization algorithm called Limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) is a popular method that has been used to do parameter estimation of CRF [9], [23], [29]. However, since it is an iterative algorithm, achieving high parallelism is not easy and demands considerable research attention for developing new parallelized algorithms that will allow them to efficiently handle large-scale data. It is a challenging task to parallelize such a dependent iterative algorithm. The task of making iterations independent of each other and thus leveraging and boosting parallel architectures is nontrivial. In this paper, we solve such an inter-dependent problem with an efficient strategy.

Current methods of improving time efficiency of the CRF model focus on how to reduce the model parameter estimation time. However, the complexity of the model inference step increases quickly with the increase of constraint length of training data set as well. The model inference step can be performed using a modified Viterbi algorithm [5], [36], [41]. In this paper, we formulate the Viterbi algorithm within the MapReduce framework to parallelize the model inference step with a simple strategy.

1.2 Our Contributions

In this paper, we propose an improved parallel CRF algorithm by combining the parallel L-BFGS and Viterbi algorithms. The algorithm leverages the MapReduce framework to enhance the capability of estimating parameters. Furthermore, it infers the most likely state sequence by extending the Viterbi algorithm with another MapReduce job. We empirically show that, while maintaining a competitive accuracy on the test data, the algorithm achieves significant speedup compared to the baseline CRF algorithm implemented on a single machine. The proposed algorithms are designed to work in the Hadoop environment, where each mapper in the nodes only compute a subset of the data.

The major contributions of this paper are summarized as follows.

- We propose an efficient method called MapReduce CRF (MRCRF) to partition a large dataset across Hadoop nodes in order to keep the context of each word in each sentence of Bio-NER, balance the workload and minimize the need for replication. Compared to the CRF method, the proposed MRCRF method requires "partitioning" the data sets.
- We develop two efficient parallel algorithms, i.e., the MapReduce L-BFGS (MRLB) algorithm and the MapReduce Viterbi (MRVtb) algorithm to implement the parallel CRF for Bio-NER based on MapReduce. The algorithms have improved performance compared with an existing sequential algorithm.
- We conduct performance evaluation which can reveal the performance benefit of the MRCRF algorithm over the CRF counterpart. The performance is presented with reported speedup versus the sequential CRF under different data set sizes and varying Hadoop configurations.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 describes the CRF model and the related algorithms. Section 4 imports the algorithms into MapReduce and implements the parallelized CRF. Section 5 demonstrates the experimental results. Section 6 concludes this paper.

2 Related Work

There has been some prior works proposed in the literature for accelerating CRF. These methods essentially gain acceleration by omitting important information of labels and losing accuracy. Pal et al. proposed a Sparse Forward Backward (SFB) algorithm, in which marginal distribution is compressed by approximating the true marginal using Kullback-Leibler (KL) divergence [25]. Cohn proposed a Tied Potential (TP) algorithm which constrains the labeling considered in each feature function, such that the functions can detect only a relatively small set of labels [2]. Both of these techniques efficiently compute the marginal with significantly reduced runtime, resulting in faster training and decoding of CRF. Although these methods could reduce computational time significantly, they train CRF only on a small data set.

In order to handle large data, Jeong et al. proposed an efficient inference algorithm of CRF for large-scale natural language data which unified the SFB and TP approaches [11]. Lavergne et al. addressed the issue of training very large CRF, containing up to hundreds output labels and several billion features. Efficiency stems here from the sparsity induced by the use of penalty term [15]. However, none of these works described so far explore the idea of accelerating CRF in a parallel or distributed setting and thus their performance is limited by the resources of a single machine.

Given that CRF is weak in processing massive data, the idea of parallelization is introduced into the algorithms. Xuan-Hieu et al. proposed a high-performance training method of CRF on large-scale data by using massively parallel computers [38]. In [19], a novel distributed training method of CRF is proposed by utilizing the clusters built from commodity computers. The method employs Message Passing Interface (MPI) and improves the time performance on large datasets. Recently, in [21], an efficient parallel inference on structured data with CRF based on Graphics Processing Units (GPU) is introduced and it is testified that the approach is both practical and economical on very large data sets. These methods achieve significant reduction in computational time without losing accuracy. However, they are not suitable for a distributed cloud environment, where usually the communication cost is higher. In our approach, we overcome this limitation by a parallel implementation of CRF based on MapReduce which is suitable for huge data sets [32].

MapReduce is an excellent model for distributed computing on large data sets, which was introduced by Google in 2004. It is an abstraction that allows users to easily create parallel applications while hiding the details of data

distribution, load balancing, and fault tolerance. At present, it is popular in text mining of various applications, especially natural language processing (NLP) [8], [31], [37]. Laclavik et al. presented a pattern of annotation tool based on the MapReduce architecture to process large amount of text data [13]. Lin and Dyer discussed the processing method of data intensive text based on MapReduce, such as parallelization of EM algorithm and HMM model [18]. Palit and Reddy proposed two parallel boosting algorithms, i.e., ADABOOST.PL and LOGITBOOST.PL, scalable and parallel boosting with MapReduce [26].

3 Conditional Random Fields Model

In this section, we first describe conditional random fields. Then, we introduce the L-BFGS algorithm for CRF. Finally, we introduce the Viterbi algorithm for CRF.

3.1 Conditional Random Fields

Conditional Random Fields was first introduced by Lafferty et al. [14] as a sequence data labeling recognition model based on statistical approaches. CRF has shown empirical success recently in NER, since it is free from the so-called label bias problem by using a global normalization.

CRF uses an undirected graphical model to calculate the conditional probability $P(y|x)$. In an NER application, a given observed sequence x can be a set of words, and a state sequence y can be in a binary value set $\{C, O\}^{|x|}$, where $y_t = C$ indicates "word x_t is inside a name" and $y_t = O$ indicates the opposite.

The result of NER is a label sequence, so we often use a linear-chain CRF. Let y, x be random vectors. A linear-chain CRF defines the conditional probability of the state sequence y for a given input sequence x to be

$$P(y|x) = \frac{1}{Z(x)} \exp\left(\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right), \tag{1}$$

where $f_k(y_{t-1}, y_t, x_t)$ is the feature function (the number is K), λ_k is a learned weight of a feature , y_{t-1} and y_t respectively refer to the previous state and the current state, $Z(x)$ is the normalization factor over all state sequences as follows:

$$Z(x) = \sum_{y} \exp\left(\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right). \tag{2}$$

The key problem of linear-chain CRF is how to find the parameter vector $\vec{\lambda} = \{\lambda_1, \ldots, \lambda_k\}$. This is done during the training process. The training process is just for parameter estimation, and the most commonly used method is maximum likelihood.

Presume every $(x^i, y^i) \in D = \{(x^i, y^i)\}_{i=1}^{N}$ is independently and identically distributed, where each $x^i = \{x_1^i, x_2^i, \ldots, x_T^i\}$ is a sequence of inputs and each $y^i = \{x_1^i, y_2^i, \ldots, y_T^i\}$ is a sequence of corresponding predictions. Then the log-likelihood function of the training data D will be shown below

$$L\left(\vec{\lambda}\right) = \sum_{i=1}^{N} \log P\left(y^i|x^i\right). \tag{3}$$

From Eq. (1) and Eq. (3), the log-likelihood function should be as follows:

$$\begin{aligned} L(\vec{\lambda}) &= \sum_{i=1}^{N} \log\left[\frac{1}{Z(x^i)} \exp\left(\sum_{t=1}^{T}\sum_{i=1}^{K} \lambda_k f_k\left(y_t^i, y_{t-1}^i, x_t^i\right)\right)\right] \\ &= \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k f_k\left(y_t^i, y_{t-1}^i, x_t^i\right) - \sum_{i=1}^{N} \log Z(x^i). \end{aligned} \tag{4}$$

In order to avoid overfitting, a penalty term is involved, and Eq. (4) becomes

$$L(\vec{\lambda}) = L(\vec{\lambda}) - \sum_{k=1}^{K} \frac{\lambda_k^2}{2\sigma^2}. \tag{5}$$

From Eq. (4) and Eq. (5), the log-likelihood function should be

$$L(\vec{\lambda}) = \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k f_k\left(y_t^i, y_{t-1}^i, x_t^i\right) - \sum_{i=1}^{N} \log Z(x^i) - \sum_{k=1}^{K} \frac{\lambda_k^2}{2\sigma^2}. \tag{6}$$

To maximize Eq. (6), numerical optimization techniques, such as the improved iterative scaling (IIS) method [3] or the quasi-Newton method [4], can be applied to calculate the optimal value $\vec{\lambda} = \{\lambda_k\}$. As an equally good choice, we adopt the L-BFGS algorithm [22] to do that, because it is a quasi-Newton method with high efficiency compared to the IIS method.

After $\vec{\lambda} = \{\lambda_k\}$ is derived, Eq. (1) will then be used to do NER during the labeling process. The labeling process is to infer the most likely state sequence according to the known word sequence x, i.e., the maximum value of $P(y|x)$. The labeling process is called model inference. The linear-chain CRF model has efficient algorithms when using Eq. (1), such as the Viterbi algorithm [6], [35]. As one of the most commonly used methods, the Viterbi algorithm is adopted to do that, because it is an effective algorithm searching for the optimal path and considering the optimal of whole state sequence.

We adopt a two-phase approach based on CRF proposed by Yang and Zhou [39]. The approach explores novel feature sets for identifying the entities in text into five types. We divide the whole biomedical NER into two sub-tasks: term boundary detection and semantic labeling. In the first phase, the boundaries of the entities are detected, which is to label words with C and O. Here C indicates "word is a name entity" and O indicates the opposite. In the second phase, the entities detected in the first phase are labeled into five classes: protein, DNA, RNA, cell-line, and cell-type. We use the CRF method for both phases.

Section 1 of the supplemental file presents more detailed description of the above approach, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2014.2368568.

3.2 L-BFGS Algorithm for CRF

Limited-memory BFGS is an optimization algorithm in the family of quasi-Newton methods that approximates the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm using

a limited amount of computer memory. In the field of large-scale unconstrained optimization, one of the most versatile, effective, and widely used methods is the L-BFGS method.

In an NER application, training a CRF model needs to adjust the parameters $\vec{\lambda} = \{\lambda_1, \ldots, \lambda_k\}$, so that it is in conformity with the training data $D = \{(x^i, y^i)\}_{i=1}^{N}$. The essence of the L-BFGS algorithm is looking for a suitable $\vec{\lambda}$ for a given input sequence x, and making $P_{\vec{\lambda}}(y|x)$ the largest, i.e., maximizing Eq. (6). L-BFGS first sets parameters an initial value $\vec{\lambda_0}$. Then L-BFGS repeatedly improves the parameter estimates: $\vec{\lambda_1}, \vec{\lambda_2}, \ldots$. The whole process uses a loop with a given number L of iterations. From $\vec{\lambda_t}$ to $\vec{\lambda_{t+1}}$, L-BFGS finds the search direction $\vec{P_t}$, decides the step length a_t, and moves in this direction $\vec{P_t}$. The key of finding the search direction $\vec{P_t}$ is to calculate a gradient vector ∇L_t.

The pseudo code of the L-BFGS algorithm is described in Algorithm 1. The algorithm consists of three steps in each iteration and the process in each iteration is as follows.

Step 1. Calculate the gradient vector ∇L_t (line 4). The gradient vector of the model, i.e.,

$$\nabla L = \left(\frac{\alpha L(\vec{\lambda})}{\alpha \lambda_1}, \frac{\alpha L(\vec{\lambda})}{\alpha \lambda_2}, \ldots, \frac{\alpha L(\vec{\lambda})}{\alpha \lambda_k} \right),$$

can be obtained by calculating the partial derivative of each λ_k

$$\frac{\alpha L(\vec{\lambda})}{\alpha \lambda_k} = \sum_{i=1}^{N} \sum_{t=1}^{T} f_k\left(y_t^i, y_{t-1}^i, x_t^i\right) - \sum_{i=1}^{N} \sum_{y} P\left(y^{i'} | x^{i'}\right) f_k\left(y_t^{i'}, y_{t-1}^{i'}, x_t^i\right) - \frac{\lambda_k}{\sigma^2}. \quad (7)$$

Step 2. Calculate the search direction $\vec{P_t}$ (line 5).

Step 3. Select the step size a_t that meets the wolf condition and update the estimated parameters of the next $\vec{\lambda}_{t+1}$ according to the first and the second steps (lines 6-7).

Algorithm 1. L-BFGS Algorithm

Input: Training date set of N samples D.
Output: $\vec{\lambda}$.
Procedure:
1: $\vec{\lambda_0} \leftarrow (\frac{1}{n}, \ldots, \frac{1}{n})$
2: $t \leftarrow 0$
3: **repeat**
4: $\nabla L_t(\lambda_1, \lambda_2, \ldots, \lambda_k) \leftarrow (\frac{\alpha L(\lambda)}{\alpha(\lambda_1)}, \frac{\alpha L(\lambda)}{\alpha(\lambda_2)}, \ldots, \frac{\alpha L(\lambda)}{\alpha(\lambda_k)})$
5: $\vec{P_t} \leftarrow H_t \cdot \nabla L_t$
6: Select the step size a_t
7: $\vec{\lambda_{t+1}} \leftarrow \vec{\lambda_t} + a_t \cdot \vec{P_t}$
8: $t \leftarrow t + 1$
9: **until** convergence
10: **return** $\vec{\lambda}$

3.3 Viterbi Algorithm for CRF

The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states, called the Viterbi path that results in a sequence of observed events.

We use the Viterbi algorithm to label the entities in an NER application. $x^i = \{x_1^i, x_2^i, \ldots, x_T^i\}$ is a given sequence of input, and $\vec{\lambda} = \{\lambda_k\}$ is obtained by the L-BFGS algorithm. The Viterbi algorithm can find the optimal state sequence $y^i = \{x_1^i, y_2^i, \ldots, y_T^i\}$, i.e., the maximum value of $P(y^i, x^i|\vec{\lambda})$, in the m^T possible sate sequences. For m^T, m is the number of state in the CRF model and T is the length of the input sequence. In our two-phase approach, m is equal to 2 in the first phase, and the state can be in a value set $\{C, O\}$. And m is equal to 5 in the second phase, and the state can be in a value set {protein, DNA, RNA, cell-line, cell-type}.

The pseudo code for Viterbi is described in Algorithm 2. The algorithm consists of four steps and the algorithm proceeds as follows.

Step 1. Initialization (lines 2-4): calculate the probability (α_1) of all possible paths for the position $t = 1$ that has j as its state .

Step 2. Recursive Calculation (lines 5-8): calculate the maximum of the probability $\alpha_t(l)$ of all possible paths for the position $t = i$ that has j as its state, and use $\phi_t(l)$ to store the previous state of maximum probability path for the position $t = i$ that has l as its state.

Step 3. Termination Calculation (lines 9-10): calculate the maximum of the normalized probability of the position $t = T$ and return to the end of the optimal path (y_T^{i*}).

Step 4. Return Path (lines 11-13): according to the end of the optimal path (y_T^{i*}) reversely find the optimal path (y_t^{i*}), i.e., the most likely state sequence.

Algorithm 2. Viterbi Algorithm

Input: Training date set of N samples D, parameters vector $\vec{\lambda}$, model feature function $F(x, y)$.
Output: The state sequence $\{y_1^i, y_2^i, \ldots, y_T^i\}_{i=1}^{N}$.
Procedure:
1: **for** $i \leftarrow 1$ to N **do**
2: **for** $j \leftarrow 1$ to m **do**
3: $\alpha_1 \leftarrow \vec{\lambda} \cdot F_1(y_0^i = start, y_1^i = j, x^i)$
4: **end for**
5: **for** $t \leftarrow 1$ to T **do**
6: $\alpha_t(l) \leftarrow \max_{1 \le j \le m} \{\alpha_{t-1}(j) + \vec{\lambda} \cdot F_t(y_{t-1}^i = j, y_t^i = l, x^i)\}$, for $l = 1, 2, \ldots, m$
7: $\phi_t(l) \leftarrow \arg\max_{1 \le j \le m} \{\alpha_{t-1}(j) + \vec{\lambda} \cdot F_t(y_{t-1}^i = j, y_t^i = l, x^i)\}$, for $l = 1, 2, \ldots, m$
8: **end for**
9: $\max_y F(y^i, x^i)) \leftarrow \max_{1 \le j \le m} \alpha_t(j)$
10: $y_T^{i*} \leftarrow \max_{1 \le j \le m} \alpha_t(j)$
11: **for** $t \leftarrow T - 1$ to 1 **do**
12: $y_t^{i*} \leftarrow \phi_{t+1}(y_{t+1}^{i*})$
13: **end for**
14: **end for**
15: **return** $\{y_1^i, y_2^i, \ldots, y_T^i\}_{i=1}^{N}$

4 MapReduce Algorithm for CRF

In this section, we first implement the MapReduce L-BFGS algorithm for large data sets. Next, we decompose the basic Viterbi algorithm using MapReduce. Then, we present the MRCRF algorithm and use the flow of the MRCRF algorithm to illustrate the MapReduce scheme of CRF. Finally, we analyze the performance benefit of the MRCRF algorithm over its sequential counterpart.

4.1 Data Sets Partitioning

In MapReduce, the training data set is divided into many subsets, whose size depends on the number of map tasks that can be run in parallel. To ensure the context of each word in each sentence of Bio-NER, one sentence cannot be split into two map tasks. In addition, in order to achieve optimal resource utilization and minimize the need for replication, we will develop a load balancing technique to partition a large dataset. Let D_i represent the sentences number of the ith map task. Then, the D_i value must satisfy the following equation:

$$D_i = \begin{cases} \lceil N/M \rceil, & \text{if R} \neq 0, \text{i} = 1, \ldots, \text{R}, \\ \lfloor N/M \rfloor, & \text{if R} \neq 0, \text{i} = \text{R}+1, \ldots, \text{M}, \\ N/M, & \text{if R} = 0, \text{i} = 1, \ldots, \text{M}, \end{cases} \tag{8}$$

where M denotes the number of map tasks, and R is equivalent to $N \bmod M$.

This paper does not consider the length of the sentence. We can divide the training data into M random subsets with approximately equal size. If $N \bmod M = 0$, every map tasks has one input split with N/M sentences. If $N \bmod M \neq 0$, R map tasks have the input split with $\lceil N/M \rceil$ sentences and others have the input split with $\lfloor N/M \rfloor$ sentences.

4.2 MapReduce L-BFGS

The process of parameter estimation is also the process of training. For large-scale training data, the model will tremendously increase the training time consumption and take much time to study. Numerous experiments show that the first step of the L-BFGS algorithm which is described in Section 3.2 is the main part of the training process. About 90 percent of the total computation time of L-BFGS is used for the first step. If the first step be accelerated, we would cut down the whole training time sharply. Therefore, the main part of parallelization of the L-BFGS algorithm is parallelized objective function gradient calculation.

As in Eq. (7), we can extract the factor as follows:

$$\frac{\partial L(\vec{\lambda})}{\partial \lambda_k} = \sum_{i=1}^{N} \left(\sum_{t=1}^{T} f_k\big(y_t^i, y_{t-1}^i, x_t^i\big) - \sum_{y} p\big(y^{i'} | x^i\big) f_k\left(y_t^{i'}, y_{t-1}^{i'}, y_t^i\right) \right) - \frac{\lambda_k}{\sigma^2}, \tag{9}$$

where the first item is the expectations of characteristics f_k under the empirical distribution with a given vector x^i. The first item can be described as Eq. (10):

$$E_{(x^i,y^i)}(f_k) = \sum_{t=1}^{T} f_k\big(y_t^i, y_{t-1}^i, x_t^i\big). \tag{10}$$

The second item is the expectations of characteristics f_k under the model distribution with a given vector x^i. The second item can be described as Eq. (11):

$$\bar{E}_{(x^i,y^i)}(f_k) = \sum_{t=1}^{T} f_k\big(y_t^{i'}, y_{t-1}^{i'}, x_t^i\big). \tag{11}$$

So Eq. (11) can be written as

$$\frac{\partial L(\vec{\lambda})}{\partial \lambda_k} = \sum_{i=1}^{N} \Big(E_{(x^i,y^i)}(f_k) - \bar{E}_{x^i,y^i}(f_k) \Big) - \frac{\lambda_k}{\sigma^2}. \tag{12}$$

$\nabla L_{(x^i,y^i)}$ is the objective function of a set of (x^i, y^i) pairs:

$$\nabla L = \sum_{i=1}^{N} \nabla L_{(x^i,y^i)} - \frac{\lambda_k}{\sigma^2}. \tag{13}$$

From the above reasoning, the first step of Algorithm 1 can be revised as Algorithm 3.

Algorithm 3. Compute Gradient Algorithm

Input: Training date set of N samples D.
Output: ∇L_t.
Procedure:
1: $\nabla L_t \leftarrow 0$
2: **for** each $(x^i, y^i) \in D$ **do**
3: **for** $t \leftarrow 1$ to T **do**
4: $F_{(x_t,y_t)}(f_k) \leftarrow E_{(x_t,y_t)}(f_k) - \bar{E}_{x_t,y_t}(f_k)$
5: $\nabla L_{(x_t,y_t)}(f_k) \leftarrow F_{(x_t,y_t)}(f_1), F_{(x_t,y_t)}(f_2), \ldots, F_{(x_t,y_t)}(f_k)$
6: **end for**
7: $\nabla L_t \leftarrow \nabla L_t + L_{(x^i,y^i)}$
8: **end for**
9: $\nabla L_t \leftarrow \nabla L_t - \frac{\lambda}{\sigma^2}$
10: **return** ∇L_t

Algorithm 4. MR Compute Gradient Algorithm

Input: Training date sets of M workers $(D_{n^1}^1, D_{n^2}^2, \ldots, D_{n^M}^M)$.
Output: ∇L_t.
Driver:
1: Execute Mapper1
2: Execute Reducer1
3: Computepenalty(sum, ∇L_t)
Mapper1:
Method: Map(key, value)
//key: the name of subset $D_{n^m}^m$
//value: the value of $\vec{\lambda}_t$
4: $s \leftarrow 0$
5: **for** each $(x^i, y^i) \in T_{n^m}^m$ **do**
6: **for** $t \leftarrow 1$ to T **do**
7: $F_{(x_t,y_t)}(f_k) \leftarrow E_{(x_t,y_t)}(f_k) - \bar{E}_{x_t,y_t}(f_k)$
8: $\nabla L_{(x_t,y_t)}(f_k) \leftarrow F_{(x_t,y_t)}(f_1), F_{(x_t,y_t)}(f_2), \ldots, F_{(x_t,y_t)}(f_k)$
9: **end for**
10: $S[T, i] \leftarrow \bigtriangledown L_{(x_i,y_i)}(\lambda_1, \lambda_2, \ldots \lambda_k)$
11: **end for**
12: **EMIT**$(1, S)$
Reducer1:
Method: Reduce(key, value)
//key: the key emitted from Mapper1
//value: the value emitted from Mapper1
13: $sum \leftarrow 0$
14: **for** each val in value **do**
15: $sum \leftarrow sum + val$
16: **end for**
17: **EMIT**(1, sum)
Method: Computepenalty(sum, ∇L_t)
18: $\nabla L_t \leftarrow sum$
19: $\nabla L_t \leftarrow \nabla L_t - \frac{\lambda}{\sigma^2}$
20: **EMIT**(∇L_t)

In Algorithm 3, we first compute the gradient vector for each (x^i, y^i) (lines 2-6), and then add up these gradient

vector so that we can get the gradient vector in the whole training data (line 7). In addition, the ordered pairs (x^i, y^i) are mutually independent and identical, which means that we can use the MapReduce framework to get $\frac{\partial L(\vec{\lambda})}{\partial \lambda_k}$ rapidly. Formulation of the first step of the L-BFGS algorithm in one iteration within the MapReduce framework is shown in Algorithm 4.

In each iteration of Algorithm 4, we first specify different mappers to count the gradient vector for one input split. The Map(key, value) method emits $(1, S)$ for each gradient vector that occurs in the training sample. Then, the reducer iterates through all the values with the same key, parses the value, and adds up each value with the same key. Next, the Reduce (key, value) method emits the sum, and the sum corresponds to the gradient vector of the whole training data. Finally, the output key-value pair is used for computing penalty function entry, and the Computepenalty( ) method uses the immediate results to compute the final gradient vector.

4.3 MapReduce Viterbi

Algorithm 5 shows the formulation of the Viterbi algorithm within the MapReduce framework. The algorithm partitions the entire training data set into M smaller subsets, whose sizes follow the load balancing technique described in Section 4.1, and allocates each partitioned subset to a single map task. Each map function optimizes a partition in parallel. In the case of the Viterbi algorithm, the output of each map function is a partial state sequence for the local partition. Hence, we do not need a combined output, and we can save the reduce stage. The output of map which is no longer the intermediate result will be directly output and becomes the final result.

Algorithm 5. MRVtb Algorithm

Input: Training date sets of M workers $(D^1_{n^1}, D^2_{n^2}, \ldots, D^M_{n^M})$.
Output: the state sequence $\{y^i_1, y^i_2, \ldots, y^i_T\}_{i=1}^{n^m}$.
Driver:
1: Execute Mapper2
Mapper2:
Method: Map(key, value)
//key: the name of subset $D^m_{n^m}$
//value: the value of $\vec{\lambda}_t$
2: $s \leftarrow 0$
3: **for** i$\leftarrow$ 1 to n^m **do**
4: $\alpha_1 \leftarrow \vec{\lambda} \cdot F_1(y^i_0 = start, y^i_1 = j, x)$, for $j = 1, 2, \ldots, m$
5: **for** $t \leftarrow$1 to T **do**
6: $\alpha_t(l) \leftarrow \max_{1 \le j \le m} \{\alpha_{t-1}(j) + \vec{\lambda} \cdot F_t(y^i_{t-1} = j, y^i_t = l, x^i)\}$, for $j = 1, 2, \ldots, m$
7: $\phi_t(l) \leftarrow \arg\max_{1 \le j \le m} \{\alpha_{t-1}(j) + \vec{\lambda} \cdot F_t(y^i_{t-1} = j, y^i_t = l, x^i)\}$, for $j = 1, 2, \ldots, m$
8: **end for**
9: $\max_y F(y^i, x^i)) \leftarrow \max_{1 \le j \le m} \alpha_t(j)$
10: $y^{i*}_T \leftarrow \max_{1 \le j \le m} \alpha_t(j)$
11: **for** $t \leftarrow T - 1$ to 1 **do**
12: $y^{i*}_t \leftarrow \phi_{t+1}(y^{i*}_{t+1})$
13: **end for**
14: $S[T, i] \leftarrow y^{i*}$
15: **end for**
16: **EMIT**(key, S)

4.4 MapReduce Scheme of CRF

Our MapReduce CRF algorithm consists of a sequence of MapReduce rounds. Each MapReduce round consists of a map phase and a reduce phase in the MRLB algorithm, and each MapReduce round only consists of a map phase in the MRVtb algorithm. The MapReduce algorithm for CRF is shown in Algorithm 6.

Algorithm 6. MRCRF Algorithm

Input: Initial parameter $\vec{\lambda}_0$, training date sets of M workers $(T^1_{n^1}, T^2_{n^2}, \ldots, T^M_{n^M})$.
Output: $\vec{\lambda}$.
Driver:
1: **repeat**
2: Execute Mapper1
3: Execute Reducer1
4: Userprogram(sum, $\vec{\lambda}_{t+1}$)
5: $t \leftarrow t + 1$
6: **until** convergence
7: Execute Mapper2
Method: Userprogram(sum, $\vec{\lambda}_{t+1}$)
8: Computepenalty(sum, ∇L_t)
9: NextEstimate($\nabla L_t, \vec{\lambda}_{t+1}$)
10: **return** $\vec{\lambda}_{t+1}$
Method: NextEstimate(∇L_t, $\vec{\lambda}_{t+1}$)
11: $\vec{P}_t \leftarrow H_t \cdot \bigtriangledown L_t$
12: Select the step size a_t
13: $\vec{\lambda}_{t+1} \leftarrow \vec{\lambda}_t + a_t \cdot \vec{P}_t$
14: **return** $\vec{\lambda}_{t+1}$

In algorithm MRCRF, we first use the MRLB algorithm to estimate the parameter. We then find the state sequence in the MRVtb algorithm based on the parameter obtained from the MRLB algorithm. In the MRLB algorithm, the calculations of gradient vector are done by M mappers and one reducer present in the compute cluster. At the end of each MRLB algorithm iteration, the newly computed parameters need to be updated in the Hadoop distributed file system (HDFS), so that every mapper gets these values before the beginning of the next iteration. Thus, along with the calculation of gradient vector there is one more step to update parameters by using the Userprogram() method. In the MRVtb algorithm, only our defined map function is applied to find the most likely state sequence. Fig. 1 shows the flow of our MapReduce algorithm for CRF.

We summarize the implementation of the MRCRF algorithm as follows.

The first phase (parameter estimation using the MRLB algorithm) - The MRLB algorithm is an iterative algorithm. Each iteration has the following steps.

1) Divide the training set into M subsets of fixed size.
2) Allocate each partition to a single map task.
3) Calculate the gradient vectors using the first map function, where the output of the first map function is a partial weight of gradient vectors.
4) Sum up the partial weight of gradient vectors using the first reduce function to produce the global weight of gradient vectors.

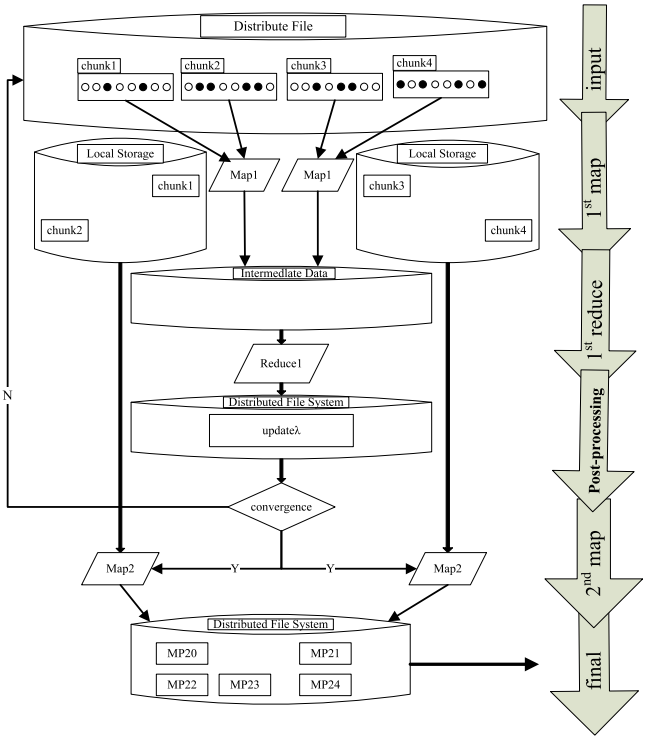


Fig. 1. Flow of the MapReduce CRF algorithm.

5) Output the value of the first reduce function to the HDFS, which is used to update the next estimated parameters in post-processing.

The second phase (model inference using the MRVtb algorithm) - The MRVtb algorithm uses the parameters generated by the MRLB algorithm to calculate the most likely state sequence of the training data set.

1) Divide the training set into M subsets of fixed size.
2) Allocate each partition to a single map task.
3) Calculate the observation of the subset using the second map function.
4) Output the values of the second map function to the HDFS, which become the final results.

4.5 Analysis of Execution Time

In this section, we analyze the performance benefit of the parallel MRCRF algorithm over the sequential CRF counterpart. In particular, we compute the speedup φ of the MRCRF algorithm compared to the CRF

$$\varphi = \frac{T_{CRF}}{T_{MRCRF}}, \tag{14}$$

where T_{CRF} and T_{MRCRF} are the computation times of the CRF and MRCRF algorithms respectively. For clarity, the correlated variables are listed in Table 1.

4.5.1 Computation Time of CRF

The total time of CRF consists of two parts, i.e., the L-BFGS and the Viterbi. The computational time of L-BFGS depends on the calculation of the gradient vector in line 4 of Algorithm 1. Assume that the time of calculating the gradient vector for N input records is $t_g N$ and the rest of the operations can be performed in time t_u. The overall cost of the L-BFGS algorithm with L iterations is $T_{\mathrm{lb}} = (t_g N + t_u)L$. In the Viterbi algorithm, assume that the computational time taken by N input records is $T_{vb} = t_v N$. Hence, the total time of CRF is

TABLE 1
Notations and Definitions

Notation	Definition
T_{CRF}	The computation time of CRF
T_{MRCRF}	The computation time of MapReduce CRF
T_{lb}	The computation time of L-BFGS
T_{vb}	The computation time of Viterbi
T_{mrlb}	The computation time of MapReduce L-BFGS
T_{mrvb}	The computation time of MapReduce Viterbi
T_{cc}	The communication cost of MapReduce CRF
L	The number of iterations of L-BFGS

$$T_{CRF} = T_{lb} + T_{vb} = (t_g N + t_u)L + t_v N. \tag{15}$$

4.5.2 Computational Time of MRCRF

As shown in Fig. 1, the computation time of MRCRF consists of four major parts, i.e., the first map phase, the first reduce phase, post-processing, and the second map phase. Unlike the sequential L-BFGS and Viterbi in the CRF algorithm, the time to transmit each key-value over the network (i.e., the communication cost) is considered in MRCRF.

The first map phase - In a distributed setting, where M workers participate in parallel and the data are distributed evenly among the workers, according to the implementation of MRLB, the computational time of the first map phase is $t_g \lfloor N/M \rfloor$.

The first reduce phase - Each mapper can emit one intermediate key. All values associated with the same intermediate key will generate one reduce task. Hence, the computational time of the first reduce phase can be defined as t_{rd}.

Post-processing - At the end of each iteration, the User-program( ) method is used to update the recently calculated values. As described in Section 4.5.1, the computational time of the rest of the operations is t_u.

Hence, the computational time taken by one iteration of the MRLB algorithm is

$$T_{mrlb} = t_g \lfloor N/M \rfloor + t_{rd} + t_u. \tag{16}$$

The second map phase - For MRVtb, the computational time of the second map phase is similar to that of the first map phase, which is $t_v \lfloor N/M \rfloor$.

Communication cost - For the communication cost analysis, let the cost of communication from the map function to the reduce function, from the reduce function to the user program, and from the user program to the map function be f, g, and h, respectively. Then, the communication cost of MRCRF will take $T_{cc} = L(f + g) + h$ time, where L is the number of iterations.

From the above analysis, the total time taken by MRCRF is as follows:

$$\begin{aligned} T_{MRCRF} &= T_{mrlb} + T_{mrvb} + T_{cc} \\ &= (t_g\lfloor N/M \rfloor + t_{rd} + t_u)L + t_v\lfloor N/M \rfloor \\ &\quad + (L(f+g)+h). \end{aligned} \tag{17}$$

4.5.3 Speedup

Based on Eqs. (15) and (17), we compute the speedup φ of the MRCRF algorithm compared to the sequential CRF as

$$\begin{aligned} \varphi &= \frac{T_{CRF}}{T_{MRCRF}} \\ &= \frac{T_{lb} + T_{vb}}{T_{mrlb} + T_{mrvb} + T_{cc}} \\ &= \frac{(t_g N + t_u)L + t_v N}{(t_g\lfloor N/M \rfloor + t_{rd} + t_u)L + t_v\lfloor N/M \rfloor + (L(f+g)+h)}. \end{aligned} \tag{18}$$

From the above equation, where we let M be a constant, we can expect the following conclusions.

1) As N increases, the numerator T_{CRF} increases at a higher rate compared to the denominator T_{MRCRF}. This indicates that the MRCRF algorithm is more effective for larger training data sets.
2) For very small values of N, the denominator in Eq. (18) will be greater than the numerator. In this case, MRCRF will be slower than sequential CRF ($\varphi < 1$), and at some point T_{CRF} exceeds T_{MRCRF}, making $\varphi > 1$.
3) For sufficiently large values of N, the following equations hold:

   $$t_g\lfloor N/M \rfloor \gg L(f+g) + t_{rd}, t_v\lfloor N/M \rfloor \gg h.$$

   We can approximate φ as follows:

   $$\begin{aligned} \varphi &= \frac{T_{CRF}}{T_{MRCRF}} \\ &= \frac{(t_g N + t_u)L + t_v N}{(t_g\lfloor N/M \rfloor + t_u)L + t_v\lfloor N/M \rfloor}. \end{aligned} \tag{19}$$

   In this equation, we observe that as N increases, the speedup is becoming greater and converges to a certain value. In this situation, the speedup is close to the linear growth.
4) Let N be a constant, the speedup value actually increases as we increase the value of M from Eq. (18). This demonstrates that the map number and performance of the cluster do affect the efficiency of the MRCRF algorithm.

5 Experiments

In this section, we first describe the experimental setup. We then test the recognition result correctness, and show the results of evaluating the effectiveness of the proposed MRCRF algorithm on different data set sizes and various Hadoop configurations.

5.1 Experimental Setup

We have searched for three groups of key words in MEDLINE by using GoPubMed. The first group is biological-process and disease, the second group is cellular-component and disease, and the third group is molecular-function and disease. 326937, 112998, and 157749 documents were downloaded respectively from these three fields. And among these documents, there were respectively 489, 170, and 237 documents selected randomly as the corpus. After eliminating the duplicate documents, there were still 595 ones related to gene function and disease. We named the corpus as GO-DO. The experiments are based on this corpus, and we adopt a two-phase approach based on CRF and our MRCRF. The unparallel CRF was carried out on a single machine.

There are two effective parallel implementations currently, i.e., the CRF based on Message Passing Interface and Graphics Processing Units. However, they are not suitable for large volumes of data in data-intensive applications. The strongest weakness of MPI is communication latency in a big data environment for data-intensive applications, because a large amount of data are exchanged between a large number of nodes, and network communications will spend long time, such that the MPI method shows low performance. Due to the capacity limits of global memory and the bottleneck of data transmission for data-intensive applications in a big date environment, the GPU method also shows low performance. Hence, we have the proposed algorithm compared with the sequential CRF algorithm, but not compared with other parallel implementations of the algorithm.

Hadoop, an implementation of MapReduce, has a master-slave file system HDFS, which is the underlying support for the MapReduce data processing function. With the HDFS, Hadoop can easily realize "computation to the data storage migration", thus greatly improve the computational efficiency of the system. MapReduce can deal with huge amount of data, especially for data-intensive applications.

Recognition of biomedical named entity using conditional random fields in this paper is a data-intensive application in the big data environment, so the Hadoop method is a suitable method.

Virtual machine instances are usually used in a public cloud to run Hadoop applications. The CPU instructions and memory space within a virtual machine need to be translated and mapped to its physical machine host. Therefore, this intermediate operation degrades the efficiency of running Hadoop jobs, and we choose to deploy them on physical machines directly. Meanwhile running Hadoop applications on a public cloud can be enabled by virtual machine templates and more execution nodes can be instantiated. Therefore, the scalability capacity will be much better, but this is not the focus of this paper. To analyze the speedup in a more efficient way, a local cluster with less interaction with the virtualization hypervisor, reveals the real performance of Hadoop jobs.

5.2 F-Score Check

First we test the recognition result correctness of MRCRF by comparing the results generated by MRCRF and CRF. The experiment results are measured with F-score valued below:

$$F_\beta = \frac{(1+\beta^2)(precision \times recall)}{\beta^2 \times precision + recall}, \tag{20}$$

TABLE 2
Comparison of CRF and MRCRF

Methods	First phase (Precision, Recall, F-score)	Second phase (Precision, Recall, F-score)
CRF	(75.22, 78.32, 76.74)	(70.79, 76.01, 73.31)
MRCRF	(75.20, 78.36, 76.75)	(70.80, 76.07, 73.34)

where *precision* is the percentage of the correct annotations, *recall* is the percentage of the total NES that are successfully annotated, F stands for the synthetic performance of a system, and the value of β is taken as 1. The value of β denotes the relative weight between *precision* and *recall*. The value of β is less than 1 if the *recall* is more important and vice versa. We consider *precision* and *recall* to be equally important in our experiments and set the value to 1.

Two experiments are carried out using CRF and MRCRF with 1 MB training samples. The experiment of CRF was carried out on a single machine. In the experiment of MRCRF, Hadoop 1.2 is deployed in a cluster with five nodes, and the size of its data block is limited to 64 MB. The software and hardware configurations in the Hadoop cluster are presented in Section 2 of the supplemental file, available in the online supplemental material.

We make comparisons for the results generated by MRCRF and CRF in each phase. Table 2 shows the results.

We conclude that the F-score of both methods are approximately the same. This happens due to the inherent similarity of the two implementations and thus we conclude that MRCRF is a correct implementation of CRF.

5.3 Varying Data Set Size

5.3.1 Results on Runtime

Our experiments were carried out on a single machine and a Hadoop cluster with 15 nodes using different scale of training samples, respectively. For clearly observing the results, we divide our experimental results into several charts to show different measures. In Fig. 2, with two charts, the CRF and MRCRF methods are compared in terms of the run time in the two phases.

We vary the data set size, using 0.5, 1, 5, 10, and 20 MB training samples for CRF and MRCRF respectively. Fig. 2a illustrates the two algorithms in a small instance. It is obvious that increasing the size of the training samples leads to increased run time, and the extent of growth is more and more big. However, there is no much difference between the run times of the CRF and the MRCRF.

Fig. 2b illustrates the two algorithms by varying the data set size, using 1, 2, 3, and 4 GB training samples. The time of the sequential method in the two phases keeps growing, and the extent of growth is more and more significant. While using the parallel method, the time has modest growth especially for large-scale samples. Therefore, under large training samples, there is significant difference between the run times of the CRF method and the MRCRF method. This is because the parallel method reduces time cost significantly.

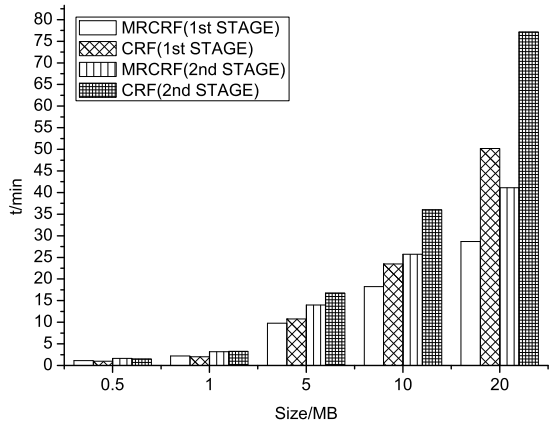
From the two groups of experimental results, we know that the run times of both the sequential CRF and MRCRF algorithms increase linearly with increasing sizes of data sets but at different rates.

5.3.2 Results on Speedup

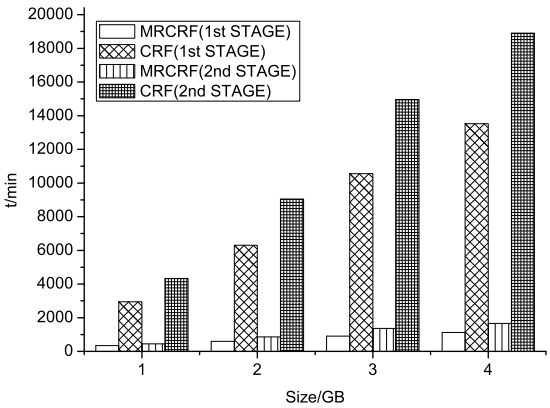
In order to better demonstrate the performance benefit of the MRCRF algorithm over the CRF counterpart, we calculate the parallel speedup and make comparison between them. Fig. 3 shows the speedup on different data sets.

In Fig. 3a, the speedup in case of small data sets is typically smaller. For small data sets, we can observe the minimum size of input data sets, which ensures the speedup to be greater than 1. This is primarily due to the fact that the communication cost is significantly larger compared to the computation cost, resulting in diminishing effect on the speedup. We observe that running Hadoop starts giving performance improvement for data sizes around 5 MB, where the cross-over point means that the sequential runtime exceeds the runtime on Hadoop, i.e., the speedup will be greater than 1.

From Fig. 3b, it can be seen that when facing a large amount of data, our algorithm can bring larger speedup. This is primarily due to the fact that the computation cost is

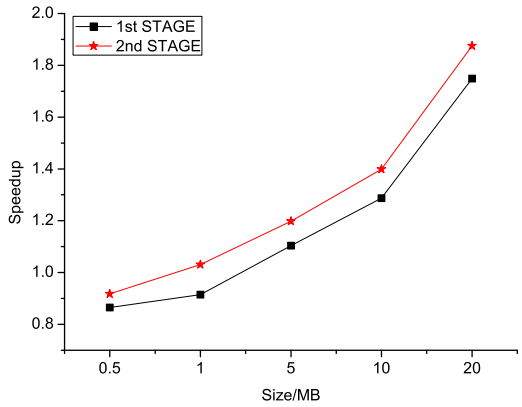


(a) Run time under small training samples

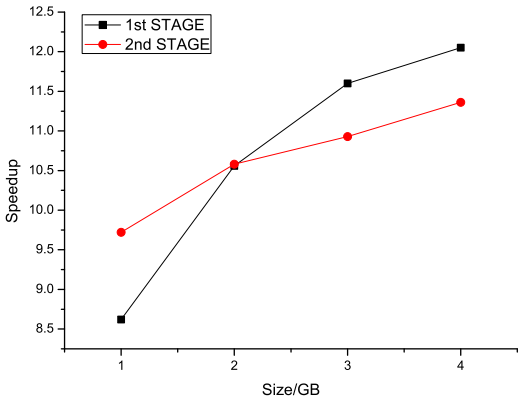


(b) Run time under large training samples

Fig. 2. Varying the data set size.



(a) Speedup under small training samples



(b) Speedup under large training samples

Fig. 3. Speedup of MRCRF relative to CRF for varying data set size.

so dominant that the effect of communication cost on speedup is almost invisible. Moreover, we observe that as N increases, the speedup is becoming greater and close to the linear growth.

From these plots, we observe that the larger the data set is, the better the speedup will be for our proposed algorithm. These behaviors of Fig. 3 are consistent with our mathematical analysis in Section 4.5.3 and confirm that the input data size required can gain sufficient performance improvement.

5.4 Varying the Number of Hadoop Nodes

For the same training size (4 GB training samples), we vary the number of Hadoop nodes in the MRCRF algorithm. Fig. 4 shows the changes in the run time when using different numbers of Hadoop nodes. These experiments are run in the same experimental environment. Thus, the performance of the MapReduce job is affected by some external factors such as the network bandwidth.

In Fig. 4, we find that increasing the number of Hadoop nodes from 5 to 20 significantly reduces the running time.

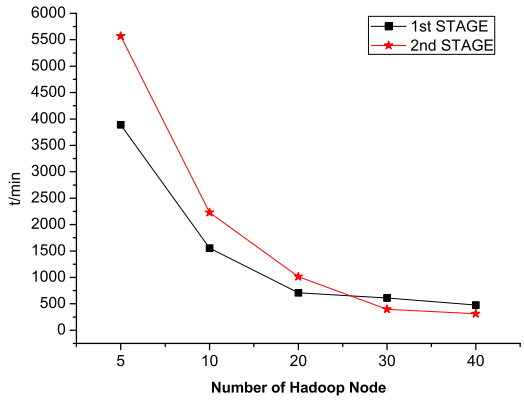


Fig. 4. Varying the Hadoop nodes with the same data set size.

This demonstrates that the number of Hadoop nodes does affect the time efficiency of MRCRF. However, when the number of Hadoop nodes is increased from 20 to 40, the total running time shows no obvious decrease. We can observe that for more than 20 nodes, compute nodes have little impact on computing performance. This is because the external factors and overheads of Hadoop are more obvious.

5.5 Varying Hadoop Parameters

The performance of Hadoop is related to its job configurations, e.g., the number of map tasks and the number of copy threads. Table 3 shows the different Hadoop parameters in our experiment.

We first run the MRCRF algorithm with 4 G training samples using 5, 10, 20, 30, and 40 map tasks with a different number of Hadoop nodes. We then use 5, 10, 15, 20, and 25 copy threads to run the MRCRF algorithm. The results are shown in Fig. 5.

It can be seen from Fig. 5a that by increasing the number of map tasks from 5 to 20, the run time decreases noticeably. However, when the number of map tasks is increased from 20 to 40, the total running time does not decrease so much. This happens because too many map tasks lead to oversplitting of the data, introducing too much overhead and internal communication delay.

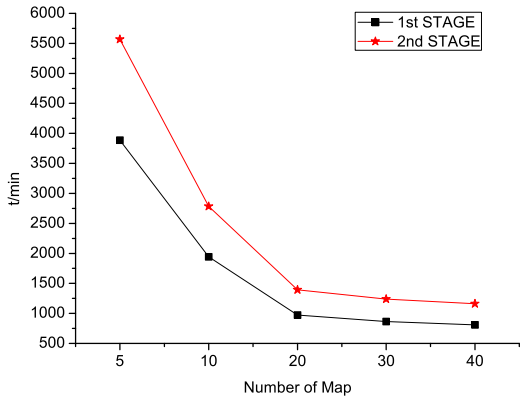
In Fig. 5b, by increasing the number of copy threads from 5 to 15, the run time decreases. When the number of copy threads reaches 15 and continues increasing, the run time will almost certainly be moderate. This is because increasing copy threads causes internal communication delay. So the right level of parallelism for copy threads is around 15 for the 5 nodes case.

6 Conclusions

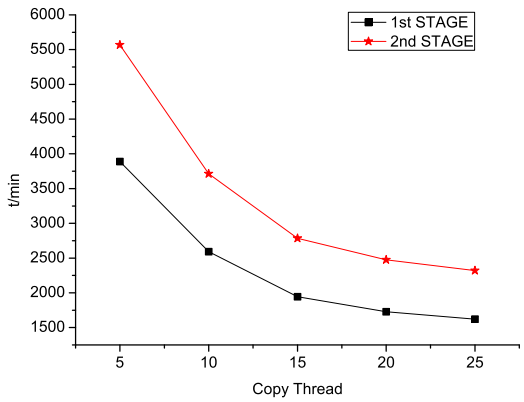
MapReduce is commonly used to distribute computation for vast amounts of data. In this paper, we apply the framework to a two-phase biomedical named entity recognition method using CRF. In this method, the L-BFGS algorithm is

TABLE 3
Different Hadoop Parameters

Configuration items	Configuration properties	Value1	Value2	Value3	Value4	Value5
Map Task	mapred.JobConf. setNumMapTasks (int n)	5	10	20	30	40
Copy Thread	mapred.reduce. parallel.copies	5	10	15	20	25



(a) Different map tasks



(b) Different copy threads

Fig. 5. Varying the Hadoop parameters with the same data set size.

used to learn the parameters while the Viterbi algorithm is used to model the inference procedure. Our work formulates both parameter estimation and model inference of the CRF model according to the MapReduce framework, and designs a parallel algorithm for these two steps. We present the details of our Hadoop implementation, report speedup versus the sequential CRF, vary different data set sizes and compare various Hadoop configurations for MRCRF. Experiments result show that the method can improve the data mining performance for biomedical literatures while guaranteeing the correctness of recognition result.

Spark is an open-source data analytics cluster computing framework, which provides primitives for in-memory cluster computing, making it particularly suited for machine learning algorithms. We intend to investigate parallel Spark implementation of conditional random fields for biomedical named entity recognition in the near future.

Acknowledgments

The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005), and the National Natural Science Foundation of China (Grant Nos. 61370095, 61472124).

References

[1] Wikipedia. MEDLINE[EB/OL] [Online]. Available: http://en.wikipedia.org/wiki/MEDLINE, 2014.

[2] T. Cohn, "Efficient inference in large conditional random fields," in *Proc. 17th Eur. Conf. Mach. Learn.*, 2006, pp. 606–613.

[3] S. Della Pietra, V. Della Pietra, and J. Lafferty, "Inducing features of random fields," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 4, pp. 380–393, 1997.

[4] J. E. Dennis, Jr. and J. J. Moré, "Quasi-newton methods, motivation and theory," *SIAM Rev.*, vol. 19, no. 1, pp. 46–89, Apr. 1977.

[5] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," in *Proc. 43rd Annu. Meeting Assoc. Comput. Linguistics*, 2005, pp. 363–370.

[6] G. D. Forney Jr., "The viterbi algorithm," *Proc. IEEE*, vol. 61, no. 3, pp. 268–278, Mar. 1973.

[7] C. M. Friedrich, T. Revillion, M. Hofmann, and J. Fluck, "Biomedical and chemical named entity recognition with conditional random fields: The advantage of dictionary features," in *Proc. 2nd Int. Symp. Semantic Mining Biomed.*, 2006, vol. 7, pp. 85–89.

[8] Y. Ganjisaffar, T. Debeauvais, S. Javanmardi, R. Caruana, and C. V. Lopes, "Distributed tuning of machine learning algorithms using mapreduce clusters," in *Proc. 3rd Workshop Large Scale Data Mining: Theory Appl.*, 2011, p. 2.

[9] A. Gunawardana, M. Mahajan, A. Acero, and J. C. Platt, "Hidden conditional random fields for phone classification," in *Proc. Int. Conf. Speech Commun. Technol.*, 2005, pp. 1117–1120.

[10] Z. GuoDong and S. Jian, "Exploring deep knowledge resources in biomedical name recognition," in *Proc. Int. Joint Workshop Nat. Lang. Process. Biomed. Appl.*, 2004, pp. 96–99.

[11] M. Jeong, C.-Y. Lin, and G. G. Lee, "Efficient inference of CRFs for large-scale natural language data," in *Proc. ACL-IJCNLP 2009 Conf. Short Papers*, 2009, pp. 281–284.

[12] J.-D. Kim, T. Ohta, Y. Tsuruoka, Y. Tateisi, and N. Collier, "Introduction to the bio-entity recognition task at JNLPBA," in *Proc. Int. Joint Workshop Nat. Lang. Process. Biomed. Appl.*, 2004, pp. 70–75.

[13] M. Laclavík, M. Šeleng, and L. Hluchỳ, "Towards large scale semantic annotation built on mapreduce architecture," in *Proc. 8th Int. Conf. Comput. Sci.*, 2008, pp. 331–338.

[14] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proc. 18th Eur. Conf. Mach. Learn*, 2001, pp. 282–289.

[15] T. Lavergne, O. Cappé, and F. Yvon, "Practical very large scale CRFs," in *Proc. 48th Annu. Meeting Assoc. Comput. Linguistics*, 2010, pp. 504–513.

[16] C. Lee, W.-J. Hou, and H.-H. Chen, "Annotating multiple types of biomedical entities: A single word classification approach," in *Proc. Int. Joint Workshop Nat. Lang. Process. Biomed. Appl.*, 2004, pp. 80–83.

[17] L. Li, R. Zhou, and D. Huang, "Two-phase biomedical named entity recognition using CRFs," *Comput. Biol. Chem.*, vol. 33, no. 4, pp. 334–338, 2009.

[18] J. Lin and C. Dyer, "Data-intensive text processing with mapreduce," *Synthesis Lectures Human Lang. Technol.*, vol. 3, no. 1, pp. 1–177, 2010.

[19] X. Lin, L. Zhao, D. Yu, and X. Wu, "Distributed training for conditional random fields," in *Proc. Int. Conf. Nat. Lang. Process. Knowl. Eng.*, 2010, pp. 1–6.

[20] R. McDonald and F. Pereira, "Identifying gene and protein mentions in text using conditional random fields," *BMC Bioinformat.*, vol. 6, no. Suppl 1, p. S6, 2005.

[21] K. Morik and N. Piatkowski, "Parallel inference on structured data with CRFs on GPUs," in *Proc. Int. Workshop ECML PKDD Collective Learn. Inference Structured Data*, 2012.

[22] J. Nocedal, "Updating quasi-newton matrices with limited storage," *Math. Comput.*, vol. 35, no. 151, pp. 773–782, 1980.

[23] S. Nowozin, P. V. Gehler, and C. H. Lampert, "On parameter learning in CRF-based approaches to object class image segmentation," in *Proc. 11th Eur. Conf. Comput. Vis.*, 2010, pp. 98–111.

[24] F. Olsson, G. Eriksson, K. Franzén, L. Asker, and P. Lidén, "Notions of correctness when evaluating protein name taggers," in *Proc. 19th Int. Conf. Comput. Linguistics*, 2002, pp. 1–7.

[25] C. Pal, C. Sutton, and A. McCallum, "Sparse forward-backward using minimum divergence beams for fast training of conditional random fields," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2006, vol. 5, p. V.

[26] I. Palit and C. K. Reddy, "Scalable and parallel boosting with mapreduce," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 10, pp. 1904–1916, Oct. 2012.
[27] D. Pinto, A. McCallum, X. Wei, and W. B. Croft, "Table extraction using conditional random fields," in *Proc. 26th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2003, pp. 235–242.
[28] B. Settles, "Biomedical named entity recognition using conditional random fields and rich feature sets," in *Proc. Int. Joint Workshop Nat. Lang. Process. Biomed. Appl.*, 2004, pp. 104–107.
[29] F. Sha and F. Pereira, "Shallow parsing with conditional random fields," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics Human Lang. Technol.*, 2003, vol. 1, pp. 134–141.
[30] C. Sun, Y. Guan, X. Wang, and L. Lin, "Rich features based conditional random fields for biological named entities recognition," *Comput. Biol. Med.*, vol. 37, no. 9, pp. 1327–1333, 2007.
[31] H. Tamano, S. Nakadai, and T. Araki, "Optimizing multiple machine learning jobs on mapreduce," in *Proc. IEEE 3rd Int. Conf. Cloud Comput. Technol. Sci.*, 2011, pp. 59–66.
[32] Z. Tang, J. Zhou, K. Li, and R. Li, "A mapreduce task scheduling algorithm for deadline constraints," *Cluster Comput.*, vol. 16, no. 4, pp. 651–662, 2013.
[33] R. T. Tsai, C.-L. Sung, H.-J. Dai, H.-C. Hung, T.-Y. Sung, and W.-L. Hsu, "Nerbio: Using selected word conjunctions, term normalization, and global patterns to improve biomedical named entity recognition," *BMC Bioinformat.*, vol. 7, no. Suppl 5, p. S11, 2006.
[34] T.-h. Tsai, W.-C. Chou, S.-H. Wu, T.-Y. Sung, J. Hsiang, and W.-L. Hsu, "Integrating linguistic knowledge into a conditional random fieldframework to identify biomedical named entities," *Expert Syst. Appl.*, vol. 30, no. 1, pp. 117–128, 2006.
[35] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory*, vol. 13, no. 2, pp. 260–269, Apr. 1967.
[36] D. Z. Wang, E. Michelakis, M. J. Franklin, M. Garofalakis, and J. M. Hellerstein, "Probabilistic declarative information extraction," in *Proc. IEEE 26th Int. Conf. Data Eng.*, 2010, pp. 173–176.
[37] P. Wittek and S. Darányi, "Accelerating text mining workloads in a mapreduce-based distributed GPU environment," *J. Parallel Distrib. Comput.*, vol. 73, no. 2, pp. 198–206, 2013.
[38] P. Xuan-Hieu, L.-M. Nguyen, Y. Inoguchi, and S. Horiguchi, "High-performance training of conditional random fields for large-scale applications of labeling sequence data," *IEICE Trans. Inf. Syst.*, vol. 90, no. 1, pp. 13–21, 2007.
[39] L. Yang and Y. Zhou, "Exploring feature sets for two-phase biomedical named entity recognition using semi-CRFs," *Knowl. Inf. Syst.*, vol. 40, no. 2, pp. 439–453, 2014.
[40] Z. Yang, H. Lin, and Y. Li, "Exploiting the performance of dictionary-based bio-entity name recognition in biomedical literature," *Comput. Biol. Chem.*, vol. 32, no. 4, pp. 287–291, 2008.
[41] B. Zhu and M. Nakagawa, "A MRF model with parameter optimization by CRF for on-line recognition of handwritten japanese characters," in *Proc. SPIE-IS&T Electron. Imaging*, 2011, vol. 7874, p. 787407.

Kenli Li received the PhD degree in computer science from the Huazhong University of Science and Technology, China, in 2003. He is currently a professor of computer science and technology at Hunan University and the deputy director of National Supercomputing Center in Changsha. His major research contains parallel computing, grid and cloud computing, and DNA computer.

Wei Ai received the master's of engineering degree from the School of Information Science and Engineering, Central South University in 2008. She is currently working toward the PhD degree in Hunan University, China. Her current research focuses on big data, cloud computing, and parallel computing.

Zhuo Tang received the PhD degree in computer science from the Huazhong University of Science and Technology, China, in 2008. He is currently an assistant professor of computer science and technology at Hunan University. His research interests include security model, parallel algorithms, resource scheduling for distributed computing systems, and grid and cloud computing.

Fan Zhang received the PhD degree from the National CIMS Engineering Research Center, Tsinghua University, Beijing, China. He is currently a postdoctoral associate with the Kavli Institute for Astrophysics and Space Research at MIT. His research interests include simulation-based optimization approaches, cloud computing resource provisioning, characterizing big-data scientific applications, and novel programming models for cloud computing.

Lingang Jiang is working toward the master's degree at the College of Information Science and Engineering, Hunan University, China. His research interests include modeling and scheduling for distributed computing systems, and parallel algorithms.

Keqin Li is a SUNY distinguished professor of computer science and an Intellectual Ventures endowed visiting chair professor at Tsinghua University, China. His research interests are mainly in design and analysis of algorithms, parallel and distributed computing, and computer networking. He has more than 310 research publications. He is currently on the editorial boards of *IEEE Transactions on Computers* and *IEEE Transactions on Cloud Computing*.

Kai Hwang received the PhD degree from the University of California, Berkeley, in 1972. He is a professor of electrical engineering and computer science at the University of Southern California, Los Angeles. He specializes in computer systems, parallel processing, Internet security, and distributed computing. He is the founding editor-in-chief of the *Journal of Parallel and Distributed Computing*. He is a fellow of the IEEE and the IEEE Computer Society.