基于 BSP 图处理系统的高效并行 List Ranking 算法*

孟金涛[1+]，郭贵鑫[2+]，叶志强[2]，邱爽[2]，李胜康[2]，魏彦杰[1*]，王丙强[2*]，成杰峰[3]，冯圣中[1]

[1](中国科学院深圳先进技术研究院，高性能计算中心，深圳 518055)

[2](国家超级计算深圳中心，深圳 518055)

[3](华为诺亚方舟实验室，香港)

* 通讯作者：电话: +86-15814441566, E-mail: yj.wei@siat.ac.cn, wangbingqiang@gmail.com

An Ultra-fast Memory Efficient Parallel List Ranking Algorithm for BSP Based Graph Processing Systems

Meng Jintao[1+], Guo Guixin[2+], Ye Zhiqiang[2], Qiu Shuang[2], Li Shengkang[2], Wei Yanjie[1*], Wang Bingqiang[2*], Cheng Jiefeng[3], Feng Shengzhong[1]

[1](Shenzhen Institutes of Advanced Technology, The Chinese Academy of Sciences, Shenzhen 518055, China)

[2](National Supercomputing Center in Shenzhen, Shenzhen 518055, China)

[3](Huawei Noah's Ark Lab, Hong Kong, China)

* Corresponding author: Phn: +86-15814441566, E-mail: yj.wei@siat.ac.cn, wangbingqiang@gmail.com

Abstract: We proposed an ultra-fast memory efficient parallel list ranking algorithm, MEP-Ranking. MEP-Ranking has the lowest complexity on both computation and memory usage by updating traditional list ranking algorithm with two extra variables “lend” and “rend”. In addition, it adopts independent set to avoid the potential operation contention between neighbor nodes. In each communication round, every node in independent set bridges its left neighbor and right neighbor by adding edges with new distance, then all nodes in this independent set are excluded from previous linked lists. The probability of one node being selected into the independent set is about *1/3*. According to the stop criterion of selecting independent set, the number of communication round of MEP-Ranking is different. It is bounded by *log(p)* if the selection step stops when the number of nodes in the reminding lists is less than *(n/p)*, where n is the number of nodes in linked lists, and p is the number of processors, or *O(log w)* if all nodes in the remaining lists are end nodes, where w is the length of longest linked list. The complexity of computation and communication on both stop criteria is bounded by *O(n)*, specially the memory usage is lowest and bounded by *O(n)*. Experimental results confirm the above complexity analysis, and the MEP-Ranking implementation on GPS has achieved a speedup of 4X when the number of workers increases from 8 to 48.

Key words: list ranking; independent set; algorithm complexity; parallel algorithm

摘 要: 本文提出了一种高效的并行 list ranking 算法，MEP-Ranking. 该算法通过在已有的 list ranking 算法中加入两个额外的新的变量”lend”和”rend”用于记录独立集的选取切换的状态，从而在空间复杂度和时间复杂度上均达到该算法复杂度下限。此外该算法使用独立集来避免相邻节点潜在的读写冲突。在每一轮计算中，

Meng Jintao and **Guo Guixin** contributes equally in this work. **Meng Jintao** was born in 1982, He is an engineer in Shenzhen Institutes of Advanced Technology, the Chinese Academy of Sciences. His current research interests include high performance computing and graph algorithms. **Guo Guixin** is in BGI Shenzhen, His current research focuses in graph analysis, and bioinformatics.

独立集中的每个节点将会把左边的邻居和右边的邻居通过增加一条边来互相关联起来，同时边的长度为原来两条边的和。接着独立集中的每个节点都会被从原来的链表中暂时去掉。其中每个节点中被选中为中介节点的概率为 1/3. 根据 MEP-Ranking 的停止选举独立集的条件, MEP-Ranking 的计算轮数是不同的。如果停止条件是当每个链表中的节点数小于 *n/p* 时,这里是总的节点数,*p* 是总的核心数，那么计算轮数趋近于 *O(log(p))*. 如果停止条件是当每个链表中的节点都只剩下两端节点时，那么计算轮数趋近于 *O(log w)*，这里 *w* 是最长的一个链表的长度。最后在这两种情况下该算法的计算复杂度,通讯复杂度,空间复杂度都是线性复杂度 O(n). 实验结果验证了上述算法复杂度，同时在 GPS 上实现的 MEP-Ranking 在核心数从 8 扩展到 48 核心时可以加速 4 倍。

关键词: 链表排序; 独立集; 算法复杂度; 并行算法
中图法分类号: TP391 **文献标识码**: A

1 Introduction

List ranking is a very popular subroutine for obtaining numerous parallel tree and graph algorithms [1,2,3]. The list ranking algorithm can be generalized as to compute prefix or suffix sums [4, 5] for associative operators by replacing the addition operation for node distances with a respective associative operator.

List ranking also plays a central role in graph concatenation in genome assembly problem [6, 7]. Each linked list in De Bruijn graph constructed from the sequencing reads needs to be concatenated into one edge. Most De Bruijn graphs constructed for large genomes are enormous, for example, the sequencing data from human can generate twenty billon nodes. These de bruijn graphs are generally distributed in a cluster or cloud. On these parallel systems with distributed memory, ranking the lists brings locality to this problem, and greatly minimizes the complexity of communication on post-processing task.

Improvement of list ranking algorithms generally follows the evolution of computational models, which has a roadmap from Von Neumann's RAM model, PRAM to BSP [8]. The basic approach for list ranking in RAM is "pointer jumping" [4, 9], and this is the simplest solution which has a computation workload of *O(nlogn)*. Several PRAM list ranking algorithms have been proposed [11-14], and according to the contention resolving strategy they can be divided into deterministic algorithms and randomized algorithms. For deterministic algorithms, pioneer works are "k-ruling set" technology[10,11]. With *n/logn* processors, Anderson and Miller' work has a complexity of *O(log(n))* for each processor, and the total computation workload can be reduced to *O(n)*[11]. For randomized algorithms, the "independent set" technology is used to improve the list ranking algorithm [12]. During each round a subset of nodes is selected as the independent set, and excluded after bridging their neighbors. With this strategy the size of nodes in the remaining lists can be reduced by a constant factor in every round. Reid-Miller's randomized algorithm, which uses the "sparse-ruling-set" algorithm [1, 13] can achieve a complexity of $O(n/p+log^2n)$ for each processor, and a total workload of $O(n+plog^2n)$.

As the *h*-relation at each superstep in BSP [8, 14] is time consuming, minimizing the number of communication round is the main concern in design algorithms on BSP model. Profound contributions on this issue in list ranking includes [4, 9, 15-16]. In 1997, Frank Dehne proposed a randomized parallel list ranking algorithm using "k-ruling set" method [15], the communication round of his work is limited by *log(p)*, and its computation complexity is *O(n)*. Two years later Sibeyn developed a new algorithm using sparse ruling set approach [1, 16], which requires only $6+2d\lceil \log longn \rceil$ communication round and has a total communication size of $6+\frac{3\ln d+6\ln p}{d+1}$,

here *d* is the number of recursion steps. This algorithm can handle the length of list up to *200* million in practice. In 2002, Isabelle Guerin Lassous presented a portable list ranking solution using independent set methods [4], which has limited communication round by *O(logp)* and the complexity of computation and communication by *O(n)*.

With the widely usage of cloud [17, 18] on storing and processing large volume of data, Pregel [19] as an example provides a portable framework for programming the graph algorithm on the cloud. As a vertex centric approach, Pregel and its implementations [20-24] are flexible enough for a broad set of algorithms, and these implementations automatically inherit Pregel's advantages on efficiency, scalability and fault-tolerance. However some graph algorithms need to modify two adjacent vertexes at the same time, this is difficult to implement on Pregel. To design a contention-avoiding mechanism for Pregel is one of the key issues [11, 25]. List ranking, is a representative example of this type of algorithms, and in this paper we aim at addressing this challenge.

In this paper, we propose a practical non-recursive parallel list ranking algorithm, MEP-Ranking. MEP-Ranking can avoid the potential operation contention between neighbor nodes by selecting a subset of nodes in each communication round in which no two nodes are neighbors. This subset is called an independent set. In each communication round, every node in independent set has to bridge its left neighbor and right neighbor by adding edges with new distance, then all nodes in the independent set is excluded from previous linked lists. In each communication round, a proportion of nodes in the linked lists are selected into independent set and then removed, and the proportion in this paper is proved to be about 1/3. According to the stop criterion of selecting the independent set, the number of communication round of MEP-Ranking is different. It is limited by *log(p)* if the number of nodes in the reminding lists is less than (*n/p*), or *O(log w)* if all nodes in the remaining lists are end nodes, where *w* is the length of longest list. The complexity of computation and communication on both stop criterion are bounded by *O(n)*.

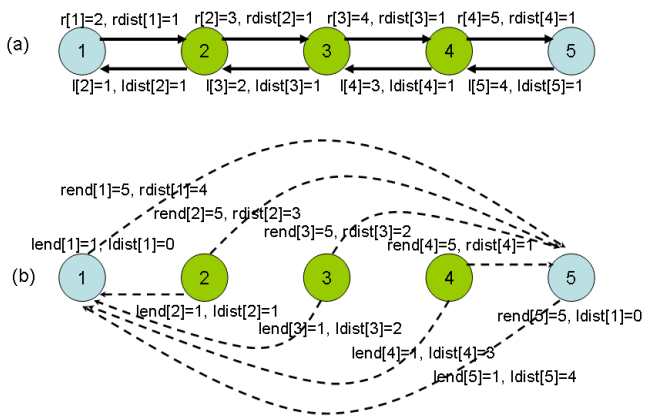

Figure 1. A linked list with five nodes are given in figure 1 (a), three internal nodes are colored green and two end nodes are colored blue. The initial value *(l[v], r[v], ldist[v], rdist[v])* is given for each node *v*. The final rank value *(lend[v], rend[v], ldist[v], rdist[v])* is given in figure 1 (b).

A practical Pregel framework, GPS, is selected to develop our MEP-Ranking. As a non-recursive algorithm, MEP-Ranking can be easily implemented on GPS. Experimental results show that our implementation has the following three properties:

1. If the stop criterion is that all nodes in the remaining lists are end nodes, the number of communication round of MEP-Ranking is linearly related to the length of longest list in the set of linked lists.

2. When the length of longest list is fixed, the number of communication round on processing the linked list is nearly the same. The total running time grows slowly with the increasing time usage of each communication round,

which is caused by the gradually increasing workload in each superstep.

3. Given a set of linked list with *50* million nodes, the time usage decreases when the number of workers increases from 8 to 48. A speedup of 4X is achieved by our implementation on GPS.

The rest of this paper is organized as follows: the list ranking problem is introduced in section 2, and then we present the algorithm of MEP-Ranking in section 3. The performance and scalability results of MEP-Ranking are given in Section 5. Section 6 concludes this paper.

2 List Ranking Problem

List and list ranking. List consists with nodes which are linked together, such that every node has one left neighbors and one right neighbor, except for the left end node and the right end node. **List ranking** determines the *rank* of every node, which is a node's distance to both its left and right end node of the list [1,2,3]. Our work follows the assumptions listed below:

- A set of linked lists L contains *n* listed nodes. In all these lists, the longest linked list is denoted as *W*, and its length is *w*.
- Each node *v* in *L* has a left neighbor *l[v]* and a right neighbor *r[v]*, except for the left end node and the right end node.
- *p* processors or workers in a cloud are used to solving the list ranking problem. Each processor or worker *i* holds a subset of list-nodes. For a given node *v*, it will be stored in processor *v%p*.

Following the above three assumptions, one example of list ranking problem and its solution is given in figure 1. The description of list ranking problem can be summarized as below:

Definition 1. For a set of linked lists *L*, list ranking includes two operations: (1) label each node *v* in *L* with its left and right end node (*lend[v]*, *rend[v]*), (2) compute the *rank* of each node, that is the distance to its left and right end node (*ldist[v]*, *rdist[v]*).

3 List Ranking Algorithm

In this section, we first describe MEP-Ranking，and then present the detail analysis on the communication round, and complexity of computation, communication, memory usage. Unlike most previously published recursive algorithms, MEP-Ranking is suitable to be implemented on Pregel, and this will be discussed in section 4.

3.1 Description of MEP-Ranking Algorithm

Independent set is first introduced by Jaja in 1992 [12], MEP-Ranking uses this technology to avoid the contention between neighbor nodes. Given a set of nodes in linked lists *L*, an independent set is a subset *I* of L such that no two items in *I* are neighbors in the linked lists. In fact such a set *I* only contains internal nodes, i.e. nodes that are not the terminal nodes of the sublists. These nodes in *I* are 'shortcut' in the algorithm: they can help exchange information between their left and right neighbors in order to connect the two neighbors directly. There are three advantages on using independent set in MEP-Ranking:

1. Independent set can be constructed in one communication round.
2. Independent set can help exchange data between its two neighbors, which is the key to shrink the length of the list.
3. In each communication round, independent set gives an order on shrinking the original list, which can avoid the contention of shrinking two neighbors at the same time.

```
input  : A set of linked lists L with n nodes, each node v has a distance value
         ldist[v] for its left neighbor l[v] and a distance value rdist[v] for its
         right neighbor r[v]. Each node v has ol[v] and or[v] to store its previous
         left and right neighbors.
output: For each node v, compute the distance ldist[v] between v and the left
         end node lend[v], the distance rdist[v] between v and the right end
         node rend[v].
1 while stop criterion is not satisfied do
2     I = IndependentSet(L);
      D = L - I;
3     for v ∈ I do
          Send (l[v], ldist[v]) to r[v];
          Send (r[v], rdist[v]) to l[v];
4     for v ∈ D and l[v] ∈ I do
          Let (nl, nldist) be the value received from l[v];
          Set ol[v] = l[v];
          Set l[v] = nl and ldist[v] += nldist;
          Let (nr, nrdist) be the value received from r[v];
          Set or[v] = r[v];
          Set r[v] = nr and rdist[v]+ = nrdist;
      L = D;
5 Send L to processor 0 and solve the problem sequentially;
6 while |L| < n do
7     for v ∈ L and ol[v] ≠ NULL do
          Send rank value (lend[v], rend[v], ldist[v], rdist[v]) to ol[v];
          Set ol[v] = NULL;
          Send rank value (lend[v], rend[v], ldist[v], rdist[v]) to or[v];
          Set or[v] = NULL;
8     for v ∉ L and l[v] ∈ L do
          if (lend, rend, ldist, rdist) is the value received from l[v] then
              Set lend[v] = lend and rend[v] = rend;
              Set rdist[v] = rdist − ldist[v], ldist[v] += ldist;
          if (lend, rend, ldist, rdist) is the value received from r[v] then
              Set lend[v] = lend and rend[v] = rend;
              Set ldist[v] = ldist − rdist[v] and rdist[v] += rdist;
          L = L + v;
```

Algorithm 1. MEP-Ranking algorithm

The major steps of MEP-Ranking is described in Algorithm 1. Explanations of this algorithm are given below:

Line 1: Each round of the while loop corresponds to one communication round. MEPrank has two stop criterions:

1. The number of nodes in the remaining lists is less than n/p ,

2. All nodes in the remaining lists are end nodes.

Two stop criteria have direct effects on the number of communication round. Users can select one stop criterion according to their requirements.

Line 2: The independent set subroutine is used to select a subset *I* from the nodes in linked list *L*. The remaining nodes in *L* are denoted as *D*. Here the probability that one node is selected into the independent set is denoted as ε,

$$\varepsilon = \frac{|I|}{|L|}, \qquad (1)$$

where $|I|$ and $|L|$ is the number of nodes in the independent set *I* and the set of linked lists *L*. This independent set subroutine and the value of proportion ε are to be discussed in the next subsection.

Line 3: A node *v* in the independent set I exchanges information by sending messages *(r[v]; rdist[v])* and *(l[v]; ldist[v])* to its left neighbor *l[v]* and right neighbor *r[v]*, respectively.

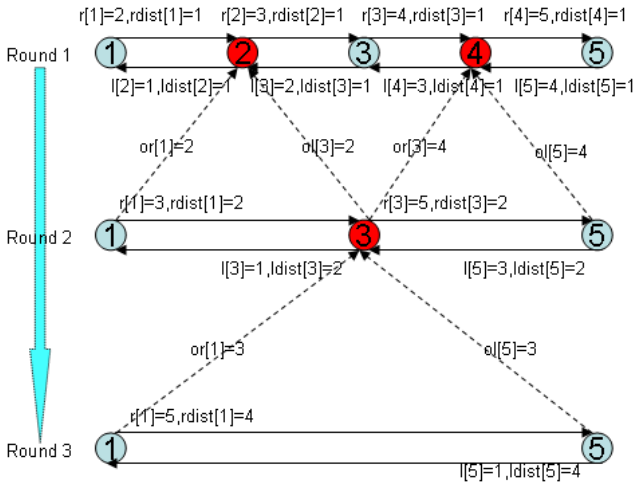

Figure 2. An example for the **top-down** part of algorithm 1. In communication round 1, node 2 and node 3 are selected into the independent set, here we color them red. After receiving messages from round 1, node 1 and node 3, node 3 and node 5 are connected; at the same time node 2 and node 4 are excluded from the original list. In order to restore the original list *ol[v]* and *or[v]* are used to store each node's previous value of *l[v]* and *r[v]*. The same situation happens in round 3, and node 3 are selected and excluded from the list. Finally after round 3, only two end nodes are left, and the top-down part stops.

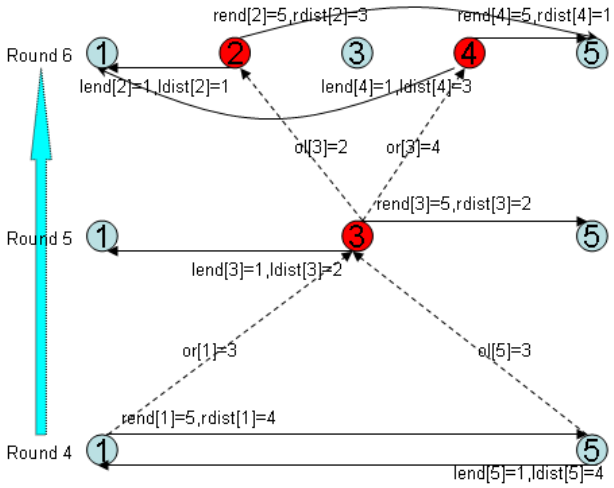

Figure 3. An example illustrates the **bottom-up** part of algorithm 1. In communication round 4, node 1 and node 5 send messages of their rank value (*lend, rend, ldist, rdist*) (1, 5, 0,4) and (1, 5, 4, 0) to node 3. Then in round 5, after received these messages, node 3 can compute out its rank value (1, 5, 2, 2) using its previous value of *ldist[3]* and *rdist[3]*,. Then node 3 sends messages of its rank value to node 2 and node 4. Finally node 2 and node 4 will calculate their rank value with the message received from node 3.

Line 4: For a node v in independent set *I*, its two neighbors *l[v]* and *r[v]* will keep their previous neighbor v in *or[l[v]]* and *ol[r[v]]* respectively. These two values will be used to restore the original linked list in later steps. Then *l[v]* and *r[v]* will be updated to be interconnected accordingly.

Line 5: If the stop criterion in Line 1 has been satisfied, all the remaining nodes will be sent to one processor

and ranked sequentially using the pointer jumping algorithm introduced in [1,2].

Line 6: Each round of the while loop corresponds one communication round.

Line 7: Every nodes *v* in current set *L* will send its left end node *lend[v]* and the corresponding distance *ldist[v]*, and right end node *rend[v]* and the corresponding distance *rdist[v]* to *ol[v]* and *or[v]*, respectively.

Line 8: After receiving the rank value *(lend; rend; ldist; rdist)* from its neighbors, each node *v* calculates its rank value *(lend[v], rend[v], ldist[v], rdist[v])*.

Note that each while loop in line 0 and line 6 invokes one communication round. In algorithm 1, we can divided the pseudo-code into two part, the first part from line 0 to line 5 is the **top-down** part, which continuously shrinks the set of nodes using independent set, and the second part from line 6 to line 10 is the **bottom-up** part, which reversely restores the linked list and computes the rank value for each internal node. An example the **top-down** and **bottom-up** part of algorithm 1 with the second stop criterion are illustrated in figure 2 and figure 3.

The number of communication round of MEP-Ranking with these two stop criterions is given by lemma 1 and lemma 3 respectively.

Lemma 1. Given a set of linked lists with *n* nodes, in each communication round one node has a probability of ε being selected into the independent set and excluded from the original linked lists, and this process stops when the number of nodes in current linked lists is less then *n/p*. The total number of communication round is $O(\log_{1/1-\varepsilon} p)$.

Proof: we can denote the number of communication round as *t*, then the number of remaining nodes after *t* communication round is $n(1-\varepsilon)^t = n/p$, and we can get,

$$t = \log_{1/1-\varepsilon} p \tag{2}$$

Lemma 2. Given linked list with *n* nodes, in each communication round one node has a probability of ε being selected into the independent set and excluded from the original linked list, this process stops when all nodes in the remaining list are end nodes. The total number of communication round is $O(\log_{1/1-\varepsilon} n)$.

Proof: The number of communication round is written as *t*, then the number of remaining nodes after *t* communication round is $n(1-\varepsilon)^t = 2$, and we can get,

$$t = \log_{1/1-\varepsilon} n - \log_{1/1-\varepsilon} 2 \approx \log_{1/1-\varepsilon} n \tag{3}$$

Lemma 3. Given a set of linked lists *L* with *n* nodes, the set of lists is written as $\{l_1, l_1, \ldots, l_k\}$, and the length of these lists is $\{w_1, w_1, \ldots, w_k\}$, here the length of the longest link list is *w*. If the stop criterion on selecting the independent set is that all nodes in the remaining list are end nodes. The total number of communication round is $O(\log_{1/1-\varepsilon} w)$.

Proof: According to Lemma 2, the number of communication round *t* should be bounded by:

$$t = \max_{i=1}^{k} \left\{ \log_{1/1-\varepsilon} w_i \right\} = \log_{1/1-\varepsilon} w \tag{4}$$

According to Lemma 3, the number of communication round has no direct relation with the total number of nodes in linked lists, it increases linearly with the logarithm of the length of longest linked list.

In algorithm 1, each while loop in line 0 and line 6 will invoke one communication round. In the **top-down** part, each node in the independent set communicates (sending packets) only a constant number of times (at most two times) within every communication round; in the **bottom-up** part, each node in previous independent set communicates (receiving packets) at most two times. As each node can be selected into independent set only once, then the communication complexity of MEP-Ranking is *O(n)*. Only two extra arrays *ol* and *or* are introduced into algorithm 1, then the memory usage is bounded by *O(n)*.

3.2 Description of MEP-Ranking Algorithm

The independent set subroutine is the key part of MEP-Ranking algorithm. In order for MEP-Ranking algorithm to have the computational complexity within O(*n*), the independent set subroutine has to keep its computational complexity bounded by O(*n*), The independent set subroutine is presented in algorithm 2, and we will prove that algorithm 2 meets the above requirements.

input : A set of linked lists L with n nodes, and an integer number K.
output: The independent set I, which is a subset of nodes of L.
for each node $v \in L$ **do**
  Generate a random vote key $A[v]$ for node v in the interval $[1, K]$;
  Send $A[v]$ to its left neighbor $l[v]$ and right neighbor $r[v]$;
for each node $v \in L$ **do**
  if $A[v] \geq A[l[v]]$ and $A[v] > A[r[v]]$ **then**
    node v is selected into I;
return I;

Algorithm 2. Independent set algorithm.

According to algorithm 2, each node *v* in a given set *L* generates a random number *A[v]* in the interval *[1, K]*. For any node *v*, *v* can be included in the independent set *I*, if and only if its random number *A[v]* is larger than or equal to its left neighbor *l[v]*, *A[v]≥A[l[v]]*, and strictly larger than its right neighbor *r[v]*, *A[v]* > *A[r[v]]*. The following lemma is given to calculate the probability of one node being selected into independent set.

Lemma 4. Given three random number *x*, *y*, *z* in the interval *[1, K]*, the probability of $x \geq y$ and $x > z$ is:

$$\varepsilon_1 = \frac{1}{3}(1-\frac{1}{K^2}) \tag{5}$$

Proof. The observed probability of *x* larger or equal to *y* will be x/K and the probability of *x* larger than *z* is $x-1/K$. Finally the total probability over all values of x for $x \geq y$ and $x > z$ is:

$$\varepsilon_1 = \frac{1}{K}\sum_{i=1}^{K}\{\frac{i^2}{K^2}-\frac{i}{K^2}\} = \frac{1}{3}(1-\frac{1}{K^2}) \tag{6}$$

According to Lemma 4, the expected size of independent set *I* is:

$$E(|I|) = \frac{1}{3}(1-\frac{1}{K^2})|L| \tag{7}$$

Here |I| denotes the number of nodes in *I*, and |*L*| denotes the number of nodes in *L*.

When *K* is a large number, then the expected probability of one node *v* in *L being* selected into independent set *I* is:

$$\varepsilon_2 = \lim_{K\to\infty} \varepsilon_1 = \lim_{K\to\infty} \frac{1}{3}(1-\frac{1}{K^2}) = \frac{1}{3} \tag{8}$$

Then the expected size of independent set I is:

$$E(|I|) = \varepsilon_2 |L| = \frac{1}{3}|L|, \text{ when } K \to \infty. \tag{9}$$

From equations (8) and (9), the number of nodes in linked lists will shrink 1/3 at each communication round. It is clear that the computation complexity of algorithm 2 is *O(n)*, here *n* is the number of nodes in all input linked lists. The computation complexity of algorithm 1 on selecting independent set is:

$$C_1 = \sum_{i=1}^{t} n(1-\varepsilon_1)^{i-1} \le n\sum_{i=1}^{\infty}(1-\varepsilon_1)^{i-1} = \frac{n}{1-\varepsilon_1} \tag{10}$$

When $K \to \infty$, we have

$$C_2 = \frac{3}{2}n \tag{11}$$

From equations (10) and (11), the computation complexity of selecting independent set in algorithm 1 is O(*n*) regardless of the value of *K*. As two communications on sending and receiving message will recall one computation operation, the actual computation work in the other part of algorithms 1 is proportional to the number of communications. These computation work can also been bounded by O(*n*). Overall, the computation complexity of algorithm 1 is O(*n*).

In this section, we have presented a non-recursive algorithm on list ranking, MEP-Ranking. More importantly, we have shown MEP-Ranking has its computation, communication and memory usage complexity limited by O(*n*). Compared with previous works, this has the lowest complexity limit. The communication round of MEP-Ranking is *O(logp)* if MEP-Ranking adopts the first stop criterion, and *O(logw)* if MEP-Ranking adopts the second one.

4 Experiments and Performance Evaluation

MEP-Ranking has been implemented on GPS [24], a practical implementation of Pregel. In the experiment we use a high performance cluster with 6 servers; each server has 8 cores with 2.4GHz, 24 GB memory and 2.4T storage, and the operation system is CentOS 5.5. All these servers are interconnected with 1Gbit twisted-pair cables. The following parts of this section will evaluate and analyze the performance of MEP-Ranking algorithm on GPS.

4.1 Complexity evaluation

Firstly, we evaluate how the length of longest list affects the running time of MEP-Ranking. Seventeen datasets were generated with each having 10 linked lists. The maximum length of linked list in these datasets is from 1K to 8192K. Experimental results on the number of communication round and time usage are demonstrated in figure 4 and figure 5, respectively.

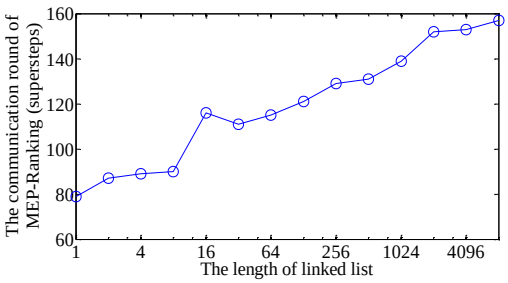

Figure 4. The number of communication round of MEP-Ranking implementation on processing linked list with its length from 1K to 8192K.

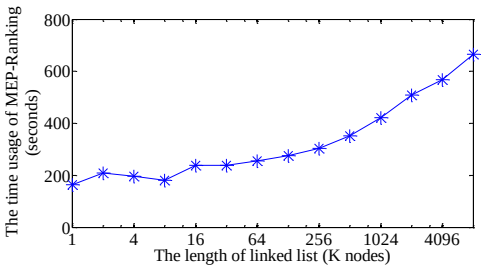

Figure 5. The time usage of MEP-Ranking implementation on processing linked list with its length varying from 10K to 100M.

In figure 4, the number of communication round increases linearly when the maximum length of linked list is growing exponentially. The same trend can also been seen in figure 5, where the time usage of MEP-Ranking also increases linearly. The experimental results confirm that the number of communication round and running time is linearly related to the logarithm of the length of longest linked list, which is consistent with our complexity analysis in section III.

Next, we have fixed the length of linked list to be 100K, and generated another series of datasets by increasing the number of linked lists from 10 to 100. Experimental results on the number of communication round, time usage and average of time usage at each communication round are illustrated in figure 6, 7, 8.

Table.1 The running time and communication round statistics on MEP-Ranking algorithm processing 500 linked lists with a length of 100k.

Number of processors	Number of supersteps	Total Time usage(seconds)	Time usage per superstep (seconds)
8	134	1688.7	12.6
16	147	989.1	6.73
24	152	714.2	4.7
32	138	570.4	4.13
40	131	468.9	3.58
48	129	426.7	3.31

According to figure 6, the number of communication round holds its value around 125 supersteps and varies between 120 and 140 supersteps. However the time usage of MEP-Ranking in figure 7 is growing steadily when the number of linked lists increases from 10 to 100. More detail can be found in figure 8, where the average time usage in one communication round sharing the same trend with the total time usage of MEP-Ranking. As the

communication and computation workload in one processor have a complexity of $O(n/p)$, here n is the number of nodes in the linked lists, the workload and time usage in one communication round will increase along with the expending problem size n.

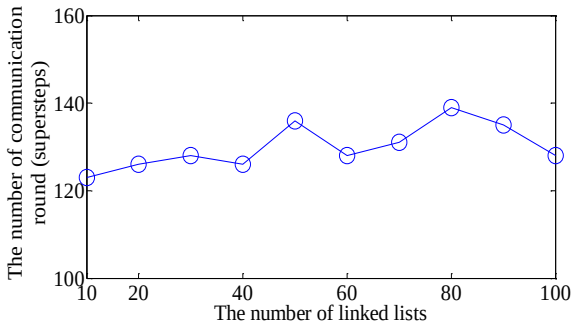

Figure 6. The number of communication rounds when the number of linked lists with fixed length increases from 10 to 100.

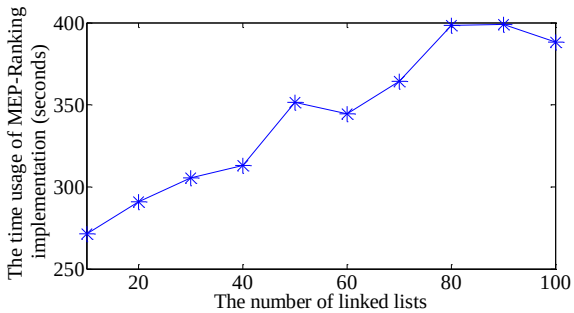

Figure 7. The time usage with GPS when the number of linked lists with fixed length increases from 5 to 100.

4.2 Scalability evaluation

In order to evaluate the scalability of MEP-Ranking, we have randomly created a dataset including *500* linked lists with a length of *100k* in average, and this dataset has 10^7 nodes in total. The simulation results are illustrated in Table.1.

In table 1, the number of communication round (supersteps) on processing this dataset is almost constant when the number of workers increases from 8 to 48. However the overall running time of MEP-Ranking is decreasing following a trend of decreasing time usage on each superstep, and more than 4 times speedup has been achieved when the number of workers scales from 8 to 48. Finally we conclude that, given a fixed dataset of linked list, the running time per superstep deceases when the number of jobs increases and the number of communication round remains constant.

5 Conclusion

List ranking is the kernel in genome assembly. However a parallel practical solution for enormous large volume of data is still a hard problem for many years. This situation blocks many genome assembly projects on the cloud, such as SOAP-Hecate and Contrail.

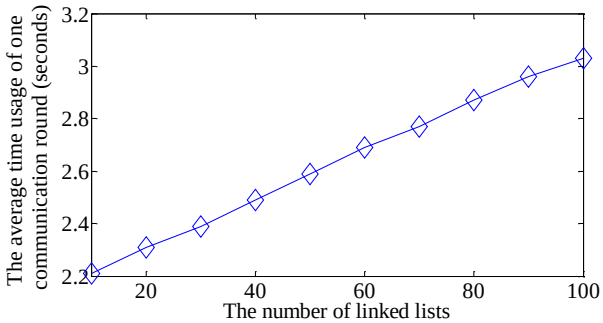

Figure 8. The average time usage in one communication rounds when the number of linked lists increases from 10 to 100.

In order to radically change this situation, we have proposed a MEP-Ranking algorithm for list ranking problem. MEP-Ranking has its communication, computation, and memory usage complexity bounded by O(*n*), and this reaches the lower bound among its previous solutions. According to the stop criterion of selecting independent set, the number of communication round of MEP-Ranking is different. It is limited by log(*p*) if the number of nodes in the reminding lists is less than (*n*/*p*), or O(log*w*) if all nodes in the remaining lists are end nodes, where p is the number of processors, n is number of nodes in linked lists, w is the length of longest list. As a non-recursive solution, MEP-Ranking can be directly implemented on Pregel system. In the experiment, we have developed MEP-Ranking algorithm on GPS, which is a practical implementation of Pregel on hadoop map-reduce platform, statistical results confirm the above complexity analysis, and the MEP-Ranking implementation on GPS has achieved a speedup of 4X when the number of workers increases from 8 to 48.

Acknowledgement This work is supported by National Science Foundation of China under grant No. 11204342, the Science Technology and Innovation Committee of Shenzhen Municipality under grant No. JCYJ20120615140912201. The authors also thanks for the computing resources provided by Dawning TC5000 supercomputing cluster, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences.

References:

[1] Reid-Miller M. List ranking and list scan on the Cray C-90. In Proc. 6th ACM Symp. on Parallel Algorithms and Architectures (SPAA' 94), 1994, pp. 104–113.

[2] Atallah M, Hambrusch S. Solving tree problems on a mesh-connected processor array, Information and Control, 1986, 69(1-3):168-187.

[3] Baase S. Introduction to parallel connectivity list rankingand euler tour techniques. Synthesis of Parallel Algorithms, Morgan Kaufmann Publisher, 1993.

[4] Lassous I, Gustedt J. Portable list ranking: an experimental study. 2002, 7:7-25

[5] Cole R, Vishkin U. Faster optimal parallel prefix sums and list ranking. Information and Computation, June, 1989, 81(3): 334–352.

[6] Jackson B, Schnable P, Aluru S. Parallel short sequence assembly of transcriptomes, BMC Bioinformatics, 2009, 10(Suppl 1):S14. doi:10.1186/1471-2105-10-S1-S14

[7] Jackson B, Regennitter M, Yang X, Schnable P, Aluru S. Parallel de novo assembly of large genomes from high-throughput short reads. In Proc. 24th International Symposium on Parallel & Distributed Processing (IPDPS'10), April, 2010, pp. 1-10.

[8] Valiant L. A bridging model for parallel computation. Communications of the ACM, Aug. 1990, 33(8):103-111.

[9] Sibeyn J, Guillaume F, Seidel T. Practical parallel list ranking. Journal of Parallel and distruted computing, 1999, 56(2):156-180.

[10] Cole R, Vishkin U. Deterministic coin tossing with applications to optimal parallel list ranking. Information and Control, 1986, 70(1): 32-53.

[11] Anderson R, Miller G. deterministic parallel list ranking, Algorithmica, 1991, 6: 859-868.

[12] Jaja J. an introduction to parallel algorithms, Addison-Wesley, MA, 1992.

[13] Reid-Miller M, Miller G, Modugno F. List-ranking and parallel tree contraction, Synthesis of parallel algorithms, 1993, pp.115-194.

[14] Skillicorn D, Hill J, McColl W. Questions and answers about BSP, Scientific Programming, 1997, 6(3): 249-274.

[15] Dehne F, Song S. Randomized parallel list ranking for distributed memory multiprocessors. International journal of parallel programming, 1997, 25(1): 1-16.

[16] Sibeyn J. Better trade-offs for parallel list ranking. In Proc. 9th ACM Symposium on Parallel Algorithms and Architectures (SPAA'97), 1997, pp. 221–230.

[17] Schatz M, Langmead B, Salzberg S, Cloud Computing and the DNA Data Race, Nature biotechnology. July, 2010, 28(7): 691-693.

[18] Wall D, Kudtarkar P, Fusaro V, Pivovarov R, Patil Prasad, Tonellato P. Cloud computing for comparative genomics. BMC Bioinformatics , 2010, 11:259, doi:10.1186/1471-2105-11-259

[19] Malewicz G, Austern M, Bik A, Dehnert J, Horn I, Leiser N, Czajkowski G. Pregel: A System for Large-Scale Graph Processing. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, 2010, pp. 135-146.

[20] Apache Incubator Giraph. http://incubator.apache.org/giraph

[21] GoldenOrb. http://www.raveldata.com/goldenorb

[22] The apache hama project. http://incubator.apache.org/hama

[23] Phoebus at github. http://github.com/xslogic/phoebus

[24] Salihoglu S, Widom J. GPS: A Graph Processing System. Technical Report, 2012, http://ilpubs.stanford.edu:8090/1039/7/full_paper.pdf

[25] Hong S, Salihoglu S, Widom J, Olukotun K. Compiling GreenMarl into GPS. Technical Report, November, 2012. http://ppl.stanford.edu/papers/tr_gm_gps.pdf