Parallel Triangle Counting over Large Graphs

Wenan Wang, Yu Gu, Zhigang Wang, and Ge Yu

Northeastern University, China
{wangwenan6,wangzhigang_mail}@yahoo.cn, {guyu,yuge}@ise.neu.edu.cn

Abstract. Counting the number of triangles in a graph is significant for complex network analysis. However, with the rapid growth of graph size, the classical centralized algorithms can not process triangle counting efficiently. Though some researches have proposed parallel triangle counting implementations on Hadoop, the performance enhancement remains a challenging task. To efficiently solve the parallel triangle counting problem, we put forward a hybrid parallel triangle counting algorithm with efficient pruning methods. In addition, we propose a parallel sample algorithm which can avoid repeated edge sampling and produce high-precision results. We implement our patterns based on bulk synchronous parallel framework. Compared with the Hadoop-based implementation, 2 to 13 times gains can be obtained in terms of executing time.

1 Introduction

The triangle counting over various graph data is a basic problem to support many important high-level applications, which has attracted more and more attention in both academical and industrial communities, such as [1, 2].

With the rapid growth of graph data, counting and listing triangles in such large graphs will cause serious performance concerns. Faced with such massive data, parallelization and sampling become two potential solutions. Some researchers attempt to extend and implement triangle counting algorithms based on Hadoop platform [3, 4]. However, some important issues such as communication optimization have not been sufficiently addressed. Besides, Hadoop may suffer performance problems when multiple-step map-reduce execution processes are needed. In addition, as a most prominent alternative, effective sampling can remarkably reduce the data volume and consequently improve the evaluation efficiency. While, how to gain high-precision results becomes quite challenging. *Doulion* is a typical representative which can guarantee the precision [4]. Unfortunately, the available sampling algorithms can not be easily executed in parallel due to the "repeated edge sampling" problem.

Our major contributions are twofold. First, we step forward to explore some essential optimization techniques to improve the efficiency of parallel triangle counting and listing in terms of local computation and across-node communication costs. The proposed optimization methods can be easily implemented utilizing more fundamental frameworks such as bulk synchronous parallel to avoid the limitation of Hadoop-like systems. Second, we attempt to crack the nut of injecting sampling techniques into our parallel framework while guaranteeing quite

W. Meng et al. (Eds.): DASFAA 2013, Part II, LNCS 7826, pp. 301–308, 2013.

high-precision analysis results, and hence further enhance the system capability in face of massive graph data. Specifically, (1) To tackle the problem of the overhead of communication and local computing, we propose a hybrid algorithm and a cut pruning technique. The hybrid algorithm combines the advantage of two available solutions namely *NodeIterator* and *EdgeIterator* [5]. And we propose the cut pruning to avoid repeated counting and reduce the message scale. (2) To solve the repeated edge sampling problem, we propose a partial-sampling method which can be embedded into our parallel framework.

The remaining sections are structured as follows. Section 2 reviews the related work. Section 3 proposes our optimization techniques and sampling algorithms. The experimental evaluation on various data sets is given in section 4 and we conclude in section 5.

2 Related Work

The centralized triangle counting and listing algorithms over graphs have been extensively studied. *NodeIterator* and *EdgeIterator* are two typical representatives [5]. *NodeIterator* is a vertex-centric algorithm which traverses every vertex and then checks the existence of an edge composed by any pair of the vertex's neighbors. While, *EdgeIterator* is an edge-centric algorithm, in which the source vertex and the destination vertex of every edge will be abstracted. Consequently, triangles can be found by searching common neighbors of these two vertices. In addition, some improved algorithms are proposed [6–8] which can gain better performance, but they are not suitable for parallel implementations as massive messages will be incurred. Some other works on graph data management can also indirectly offer the triangle counting function by issuing special queries. For example, R. Giugno et al. [9] propose a technique to count the three-node complete subgraph which composes a triangle actually. In [10], triangles can be counted as three-step-neighbors when the source vertex is assigned as the destination vertex.

With the rapid growth of graph data, some researchers are devoted to implementing classical centralized algorithms on parallel frameworks. S. Suri et al. [3] propose a parallel solution, *NodeIterator++*, by improving *NodeIterator*, and implement it on Hadoop. Although *NodeIterator++* counts the same triangle for several times repeatedly, the final result can be guaranteed to be correct due to designing different weights for edges.

Sampling techniques are regarded as feasible solutions on large data sets. Typically, C. E. Tsourakakis et al. [4] propose *Doulion* algorithm by using random sampling to process each edge, and *NodeIterator* to count triangles. Also, Rasmus Pagh et al. [11] introduce a new randomized algorithm for counting triangles in graphs. In the algorithm, one edge of a triangle is always sampled, if the other two have been sampled. However, these sampling algorithms can not be correctly executed in parallel because of the repeated edge sampling problem.

3 Optimization Policies and the Sampling Algorithm

3.1 SEN-Iterator

The definitions of symbols throughout the paper are given in Table 1.

Table 1. Symbols and Definitions

Sym	Definition	Sym	Definition
G	undirected graph(no self-edges)	V	vertex set of G
E	edge set of G	D_v	the degree of vertex v
$D(v)$	neighbor set of vertex v	$P(i)$	vertex set in $Node$ i
$Node$ i	a physical machine named i	N	the calculated number of triangles
M	the exact number of triangles		

Assume a *triangle* $\langle u, v, w\rangle$ exists and $u \in P(i)$, $v \in P(j)$, $w \in P(k)$, then triangles can be divided into three types: 1. Local-triangles, $i = j = k$. 2. Two-one-triangles, $i = j \neq k$ or $i = k \neq j$ or $k = j \neq i$. 3. Dis-triangles, $i \neq j \neq k$. Combining the partial-sampling algorithm (see section 3.2), *EdgeIterator* and *NodeIterator* [5], we propose a SEN-Iterator algorithm which has three phases. First, we generate a sampled graph $G^{'}$ by sampling edges which meet our policy with successful probability p (see section 3.2). Then local-triangles and two-one triangles in $G^{'}$ are counted by utilizing *EdgeIterator*, and we handle messages by using the concept of *NodeIterator* and cut pruning (see section 3.3). The third phase is to count the dis-triangles.

3.2 Partial-Sampling Algorithm

Assume an edge $\langle u, v\rangle \in E$, $u \in P(i)$, $v \in P(j)$, $i \neq j$. For existing sampling algorithms, in parallel environments, $\langle u, v\rangle$ will be processed on both *Node* i and *Node* j, which will be sampled twice. Therefore, for the parallel sampling process, *How to avoid sampling an edge repeatedly* is a critical problem. We propose a partial-sampling algorithm to overcome this issue. In the partial-sampling algorithm, the cross-*Node* edges are sampled with the successful probability 1. While, the edges in the same *Node* are sampled with the successful probability p.

Theorem 1. *The expected number of triangles in the sampled graph G' is equal to the actual number of triangles in G i.e. $E(N) = M$.*

Proof. For $G^{'}$, we assume that N_1 is the number of local-triangles, N_2 is the exact number of two-one-triangles and N_3 is the exact number of dis-triangles. For G, let M_1 denote the existing local triangles, M_2 be the number of two-one-triangles and M_3 be the number of dis-triangles. For each existing triangle with a specified ID i in G, ε_i is defined as a flag. Therefore, $\varepsilon_i = 0$ if triangle i does

not exist in G', otherwise, $\varepsilon_i = 1$. According to the partial-sampling algorithm, the expected values of N_1, N_2, N_3 are:

$$E(N_1) = E\sum_{i=1}^{M_1}(\frac{1}{p^3} \times \varepsilon_i) = \frac{1}{p^3} \times \sum_{i=1}^{M_1} p^3 = M_1 \tag{1}$$

$$E(N_2) = E\sum_{i=1}^{M_2}(\frac{1}{p} \times \varepsilon_i) = \frac{1}{p} \times \sum_{i=1}^{M_2} p = M_2 \tag{2}$$

$$E(N_3) = E\sum_{i=1}^{M_3}\varepsilon_i = \sum_{i=1}^{M_3} E(1) = M_3 \tag{3}$$

By above formulas, we can conclude that:

$$E(N) = E(N_1) + E(N_2) + E(N_3) = M_1 + M_2 + M_3 = M \tag{4}$$

Furthermore, we analyze the variance of N in Theorem 2.

Theorem 2. *Let M be the exact number of triangles in graph G. The variance of N is:*

$$Var(N) = \frac{M \times (p^3 - p^6) + 2k \times (p^5 - p^6)}{p^6} + \Delta \times p \times (1 - p) \tag{5}$$

where, k is the number of triangles which share an edge with other triangles.

Proof. The deviation mainly comes from three parts: overall triangles' deviation, edge-shared triangles' deviation and local-triangles' deviation. In [4], the author gives the variance estimate of the overall triangles' deviation and edge-shared triangles' deviation as follows:

$$Var(N') = \frac{M \times (p^3 - p^6) + 2k \times (p^5 - p^6)}{p^6} \tag{6}$$

Here, we will derive the deviation of the third case. First, we assume that Δ is the number of two-one-triangles. With the partial-sampling algorithm, only one edge will be sampled. The variance of the third case is:

$$Var(N'') = \Delta \times p \times (1 - p) \tag{7}$$

Finally, we get the variance estimate:

$$Var(N) = \frac{M \times (p^3 - p^6) + 2k \times (p^5 - p^6)}{p^6} + \Delta \times p \times (1 - p) \tag{8}$$

Using Theorem 2, we can get the following theorem to evaluate the stability of the expected number of triangles.

Theorem 3.

$$Pr(|X - M| \leqslant \varepsilon) \geq 1 - \frac{M \times (p^3 - p^6) + 2k \times (p^5 - p^6)}{P^6 \times \varepsilon^2} - \frac{\Delta \times p \times (1-p)}{\varepsilon^2} \tag{9}$$

Proof. By using the Chebyshev's inequality, we have:

$$Pr(|X - M| \leqslant \varepsilon) \geq 1 - \frac{Var(X)}{\varepsilon^2} \tag{10}$$

and by substituting the Formula 8, we can analyze the bound.

This theorem gives an evaluation about the performance of our partial-sampling. The accuracy of the approximate value is affected by the number of triangles in the graph, the structure of the graph and the value of p. The larger the number of triangles in the graph, the more the probability to obtain a good approximate value is. Also, the fewer edge-shared triangles which exist in the graph, the better the approximate value is.

3.3 Cut Pruning for Messages

In parallel environments, messages are used to confirm the existence of dis-triangles. And in *EdgeIterator* or *NodeIterator* [5], the content of messages generated by vertex u is $D(u)$. If messages are generated on *Node* i and sent to the same *Node*, we define them as $self - messages$, while others are called $normal - messages$. For *EdgeIterator* or *NodeIterator* [5], to confirm the existence of a dis-triangle $\langle u, v, w \rangle$, where $u \in P(i)$, $v \in P(j)$, $w \in P(k)$ and $i < j < k$, there are six kinds of $normal - messages$: *Node* i to *Node* j, *Node* i to *Node* k, *Node* j to *Node* k, *Node* j to *Node* i, *Node* k to *Node* i and *Node* k to *Node* j. In fact, only one message is necessary to confirm the existence of dis-triangles. Therefore, we design an optimization policy to reduce the message scale in SEN-Iterator: First, Node i only generates messages whose destination Node ID is larger than i. Second, for a message sent to vertex u, its content only includes neighbors whose ID is larger than that of u. Then, only one message will be sent from *Node* i to *Node* j to confirm whether triangle $\langle u, v, w \rangle$ exists.

We analyze the effect of the policy. Assume $D_l(v, u) = \{w \mid w \in V, w \in D(v), w > u\}$ and the length of one message is measured by the number of neighbors included in its content. For $v, z \in V, D(v) = \{y_1, y_2, y_3 ... y_{k-1}, z, y_{k+1} ... y_n\}$, where $y_1 < y_2 < y_3 < ... y_{k-1} < z < y_{k+1} ... < y_n$, $v \in P(i)$, $z \in P(j)$, $i < j$, *Node* i will send a message to *Node* j according to our policy. Then we can compute the total length of messages sent by *Node* i as:

$$rLen(v) = (n-k+1)+(n-k)+(n-k-1)+...+1 = \frac{1}{2} \times (n-k+2) \times (n-k+1) \tag{11}$$

The length of messages based on *EdgeIterator* or *NodeIterator* is computed as:

$$Len(v) = n + n + n + ... + n = n^2 \tag{12}$$

Let $f(n,k) = Len(v) - rLen(v)$, we get:

$$f(n,k) = \frac{1}{2} \times n^2 + n \times k - \frac{1}{2} \times k^2 - \frac{3}{2} \times n + \frac{3}{2} \times k - 1 \tag{13}$$

Then we evaluate the derivative functions of $f(n,j)$:

$$\frac{\partial f}{\partial n} = n + k - \frac{3}{2} \tag{14}$$

$$\frac{\partial f}{\partial k} = n - k + \frac{3}{2} \tag{15}$$

By analyzing the Formula 14 and Formula 15, we can infer the following properties: (1) A larger n will enhance the effect of the policy, in other words, it will have better performance for dense-graphs. (2) A larger k will enhance the effect of the policy. It means each vertex has fewer neighbors. Considering the number of edges is fixed, we can infer that this policy is more suitable for the scenario where the number of every vertex's neighbors is nearly equivalent.

4 Experiment

We implement the SEN-Iterator algorithm on the bulk synchronous parallel model and compare the performance with NI-Hadoop. NI-Hadoop is implemented on Hadoop by using the similar idea proposed by [3]. All of the datasets we used are publicly available [12] and described in Table 2. Self-loops and the direction of edges are removed. Our cluster is composed of 21 nodes. Every node contains 2 hyperthreaded 2.00GHz CPUs, 8GB RAM and a Hitachi disk drive with 500GB capacity and 7,200 RPM. All nodes are connected by gigabit Ethernet to an Ethernet switch.

Table 2. Characteristics of data sets

Data Set	Vertices	Edges	Triangles	Data Set	Vertices	Edges	Triangles
Ast	18,772	396,160	1,351,441	Web	875,713	5,105,039	13,391,903
Soc	131,828	841,372	4,910,076	Am2	262,111	1,234,877	717,719
Hep	12,008	237,010	3,358,449	Am5	410,236	3,356,824	3,951,063

4.1 Performance Analysis and Scalability of SEN-Iterator

We evaluate the SEN-Iterator algorithm over a large amount of real graphs. As shown in Fig 1(a), the overall gain of SEN-Iterator is tremendous. Exemplified by Web, the speedup of SEN-Iterator compared to NI-Hadoop is a factor of up to 13. Fig 1(b) demonstrates the scalability of SEN-Iterator. For the data set Am2, when the number of Nodes increases from 12 to 20, the running time reduces from 15s to 10s.

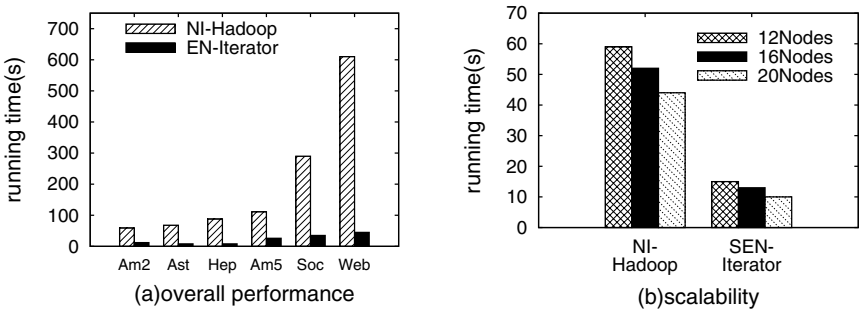

Fig. 1. Overall performance and scalability of SEN-Iterator

4.2 Analysis of Cut Pruning

The cut pruning policy improves the performance by reducing the number of messages. This suit of experiments is used to analyze the effect of the cut pruning policy by comparing SEN-Iterator with None-Iterator. The latter does not adopt the pruning policy. The message scale of None-Iterator is 2 times more than that of SEN-Iterator. For Web, SEN-Iterator only has 3845986 messages, while None-Iterator has 8644102 messages.

4.3 Accuracy Analysis for Partial-Sampling Algorithm

We run SEN-Iterator by five different values of p which ranges from 0.01 to 0.2. The examination is evaluated on real graphs. We define $Accuarcy = \frac{N}{M}$, where N is the calculated value of triangles and M is the exact value. Fig 2 shows the experimental results. We notice that the accuracy is always greater than 99%, when $p = 0.1$ or 0.15.

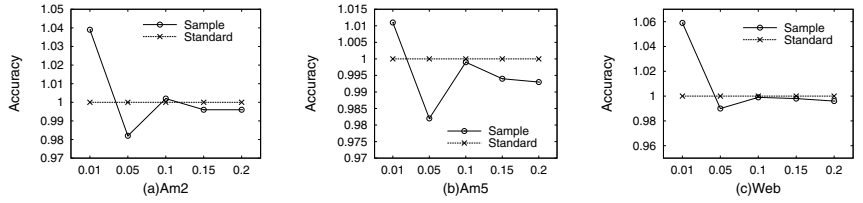

Fig. 2. The accuracy for different p in sampling

4.4 Performance Analysis of Partial-Sampling Algorithm

We analyze the gain of sampling by comparing SEN-Iterator and EN-Iterator without sampling. They are run on 12 Nodes with $p = 0.1$. We define the speedup as $Acc = \frac{R}{B}$, where R is the running time of EN-Iterator without cut pruning

and B represents the running time of SEN-Iterator without cut pruning. The Acc is more than 200% for all data sets. What's more, the Acc of Web is 297%, which is the largest.

5 Conclusions

In this paper, we propose a new solution to efficiently solve the parallel triangle counting problem. A cut pruning policy is designed to optimize the overhead of communication, and we propose a partial-sampling method to avoid the repeated sampling in parallel environments. It can be embedded into our framework and improve the performance.

Acknowledgments. This research is supported by the National Natural Science Foundation of China (61272179, 61003058) and the Fundamental Research Funds for the Central Universities (N110404006, N100704001).

References

1. McPherson, M., Smith-Lovin, L., Cook, J.M.: Birds of a feather: Homophily in social networks. Annual Review of Sociology 27, 415–444 (2001)
2. Eckmann, J.-P., Moses, E.: Curvature of co-links uncovers hidden thematic layers in the World Wide Web. Proc. of the National Academy of Science, 5825–5829 (2002)
3. Suri, S., Vassilvitskii, S.: Counting triangles and the curse of the last reducer. In: Proc. of WWW, pp. 607–614 (2011)
4. Tsourakakis, C.E., Kang, U., Miller, G.L., et al.: DOULION: counting triangles in massive graphs with a coin. In: Proc. of KDD, pp. 837–846 (2009)
5. Alon, N., Yuster, R., Zwick, U.: Finding and counting given length cycles. Algorithmica 17(3), 209–223 (1997)
6. Schank, T., Wagner, D.: Finding, Counting and Listing All Triangles in Large Graphs, an Experimental Study. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 606–609. Springer, Heidelberg (2005)
7. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. In: Proc. of STOC, pp. 20–29 (1996)
8. Tsourakakis, C.E.: Counting triangles in real-world networks using projections. Knowl. Inf. Syst. 26(3), 501–520 (2011)
9. Giugno, R., Shasha, D.: Graphgrep: A fast and universal method for querying graphs. In: Proc. of ICPR, pp. 112–115 (2002)
10. Kang, U., Tong, H., Sun, J., et al.: Gbase: a scalable and general graph management system. In: Proc. of KDD, pp. 1091–1099 (2011)
11. Pagh, R., Tsourakakis, C.E.: Colorful triangle counting and a mapreduce implementation. Inf. Process. Lett. 112(7), 277–281 (2012)
12. SNAP, `http://snap.stanford.edu/data/soc-LiveJournal1.html`