A parallel graph partitioning algorithm to speed up the large-scale distributed graph mining

ZengFeng Zeng  
School of Computer Science  
Beijing University of Posts and Telecommunications  
Beijing 100876, China  
zzfeng1987@gmail.com

Bin Wu  
School of Computer Science  
Beijing University of Posts and Telecommunications  
Beijing 100876, China  
wubin@bupt.edu.cn

Haoyu Wang  
School of Computer Science  
Beijing University of Posts and Telecommunications  
Beijing 100876, China  
muyiwhy0523@gmail.com

ABSTRACT
For the large-scale distributed graph mining, the graph is distributed over a cluster of nodes, thus performing computations on the distributed graph is expensive when large amount of data have to be moved between different computers. A good partitioning of distributed graph is needed to reduce the communication between computers and scale a system up. Existing graph partitioning algorithms incur high computation and communication cost when applied on large distributed graphs. A efficient and scalable partitioning algorithm is crucial for large-scale distributed graph mining.

In this paper, we propose a novel parallel multi-level stepwise partitioning algorithm. The algorithm first efficiently aggregates the large graph into a small weighted graph, and then makes a balance partitioning on the weighted graph based on a stepwise minimizing RatioCut Algorithm. The experimental results show that our algorithm generally outperforms the existing algorithms and has a high efficiency and scalability for large-scale graph partitioning. Using our partitioning method, we are able to greatly speed up PageRank computation.

Categories and Subject Descriptors
G.2.2 [Mathematics of Computing]: Discrete Mathematics—Graph Theory, Graph Algorithms; D.1.3 [Programming Techniques]: Concurrent Programming—Distributed programming

General Terms
Algorithms, Experimentation

Keywords
Graph Partitioning, distributed graph mining, RatioCut, Multi-level

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

1. INTRODUCTION

1.1 Large-scale graph mining
Graph datasets we face today are becoming much larger than before. The modern large search engines crawls more than one trillion links in the internet and the social networking web site contains more than 800 million active users [9]. Besides the large graph on Internet and social networks, the biological networks which represent protein interactions are of the same size [2]. Large graph processing has become more and more important in various research and we are rapidly moving to a world where abilities to analyzes very-large-scale dynamic graphs (billions of nodes, trillions of edges) are becoming critical.

The above graphs are far too large for a single computer to handle with. The common way to process the large graph is using a parallel computing systems to perform algorithms in which graph data is distributed across a cluster of machines. The various parallel computing models play an important role in handling these extremely large graphs. Some parallel graph processing systems are based on different computational model that have been proposed: Google's Pregel is based on the Bulk Synchronous Parallel model, Pegasus is based on Hadoop's MapReduce, CGMGRAPH /CGMLIB are based on MPI [15, 19, 6]. Unfortunately, none of these systems consider minimizing communication complexity by partitioning the graph data properly and saturating the network becomes a significant barrier to scaling the system up. Some complex graph algorithms, in which every vertex need access its neighbors frequently, such as triangle listing, clique percolation and Newman fastGN, could not be solved efficiently on these systems, because a graph is randomly distributed among the machines which will lead to a heavy traffic of the system’s communication when every vertex try to access its neighbors.

1.2 Graph partitioning
Good partition on large graph is critical for graph mining for the following reasons. First, graph partitioning is the key preprocessing step to divide-and-conquer algorithm-s, where it is often a good idea to break graph into roughly equal-sized dense subgraphs. The graph algorithm can respectively performs on these dense subgraphs and combines the intermediate results in the final phase. Second, a good partition that minimizes the number of cross partition edges can reduce the communications among the different machines at a large scale. As we know that inter-machine...
communication, even on the same local network, is much more expensive than inter-processor communication. A bad partition of graph will lead to the result that much more data to be moved among the machines when performing graph algorithms on these distributed graph data, which will largely increase the process time and cause network links saturated.

The graph partition problem is NP-hard and has been researched for many years. A number of high-quality and computationally efficient algorithms have been proposed, even if these solutions are not necessarily optimal, such as Kernighan-Lin algorithm, spectral bisection method and multilevel graph partitioning [17, 4, 16]. However these methods do not scale to large-scale graph data. On one hand, because of the running time, on the other hand, because these algorithms require full information about the graph or large portions of the graph, which is impossible in distributed computing environment or will lead to large scale data to move among the machines. Recently, a streaming graph partitioning for large distributed graphs is proposed to read graph data serially from a disk onto a cluster based on simple heuristics [23]. Unfortunately, it is somewhat unrealistic for the distributed system where the graph data is loaded in a parallel way.

1.3 Overview of our approach

In this paper, we propose a parallel multi-level graph partitioning algorithm to make a k-way balance partitioning on large graph. The algorithm is divided into two phases: aggregate phase and partition phase. In aggregate phase, the algorithm uses a multi-level weighted label propagation to aggregate the large original graph into a small weighted graph. In partition phase, a k-way balance partitioning performs on the weighted graph based on a stepwise minimizing RatioCut method. In our algorithm, there is no need for adjacent list of vertexes to be exchanged which makes little data to be moved among different machines. Thus it can efficiently partition large-scale graphs on distributed system, where accessing vertex’s adjacent list located on different machines is expensive. The algorithm takes $O(|E|)$ time and well scales with the size of graph and partition number. In addition, in the traditional partitioning algorithms, the partition number $k$ must meet $k = 2^m$ as partitions into more than two clusters are usually attained by iterative bisectioning, while the partition number of our algorithm can be arbitrary.

1.4 Contributions

This paper makes the following contributions:

- The parallel multi-level weighted label propagation algorithm: the algorithm efficiently aggregates the large graph into a small weighted graph.

- The stepwise minimizing RatioCut algorithm: the algorithm minimizes the RatioCut step by step. In every step, a set of vertices are extracted by minimizing part of RatioCut and these vertices are removed from the graph. A k-way balance partitioning is obtained by this algorithm.

- Experimental study: We compare our partitioning algorithm to many other partitioning algorithms on various graph datasets. And the results show that our algorithm generally outperforms the others. Our algorithm also be evaluated on large-scale graph of different scale and the experiment shows the efficiency and scalability of the algorithm. We finally demonstrate the value of graph partitioning in graph mining by using our algorithm to partition graph for PageRank computation.

The rest of the paper is structured as follows. Section 2 reviews related works. Section 3 gives some notations and definitions used in this paper. In Section 4, we first give a detailed description of a parallel multi-level graph partitioning, and then present a parallel multi-level weighted label propagation algorithm and its implementation on MapReduce, finally, a stepwise minimizing RatioCut Algorithm is proposed to partition the weighted graph. Section 5 provides a detailed explanation of the evaluation of our algorithm compared with some existing algorithms and tests the efficiency improvement of the PageRank algorithm by only changing the data layout with our partitioning algorithm. And finally in Section 6, we draw a conclusions and discuss the future work.

2. RELATED WORK

Graph partitioning is an important issue in many research areas, such as parallel computing, circuit layout and the design of many serial algorithms. The graph partitioning problem is NP-hard. The Kernighan-Lin algorithm (Kernighan and Lin,1970) is one of the earliest heuristics methods which motivated by the problem of partitioning electronic circuit onto boards [17]. The Kernighan-Lin algorithm subsequently improved in terms of running time by Fiduccia and Mattheyses [10].

The spectral bisection method is another popular method, which is based on the spectrum of the graph’s Laplacian matrix [4]. A good approximation of the minimum cut size of partition can be attained by choosing the index vector $s$ parallel to the second lowest eigenvector.

Other popular methods for graph partitioning include level-structure partitioning, the geometric algorithm, and multi-level algorithms. A level-structure was provided in SPARSPAK [5]. The algorithm first finds a pseudo-peripheral vertex $v$ in the graph. A breath-first search from $v$ is used to partition the vertices into levels. The algorithm in SPARSPAK chooses the vertices in the median level as the vertex separator. The geometric partitioning algorithm is designed by Miller, Teng, Thurston, and Vavasis [20]. This algorithm partitions a graph by using a circle rather than a straight-line to cut the mesh. The basic structure of a multilevel partitioning algorithm is very simple. The graph $G = (V,E)$ is first coarsened down to a small number of vertices, a k-way partitioning of this much smaller graph is computed, and then this partitioning is projected back toward the original graph (finer graph) [16].

A parallel graph partition algorithm was proposed in JOSTLE. However, to do it in parallel JOSTLE must first distribute a graph sensibly amongst the processors and to distribute the graph sensibly it must first find a reasonable partition. Then The JOSTLE optimizes the initial crude distribution of graph. Another parallel multilevel graph partitioning is implemented on shared-memory multicore architectures which is not suitable for the distributed computing environment for the frequent information exchange among vertices [24].
Some methods are proposed to manage the partitions in large graphs. Yang etc. propose an environment (Sedge) [27] which can minimize inter-machine communication during graph query processing.

3. PRELIMINARIES

In this section, we briefly describe some notations and definitions used in this paper.

3.1 Graph notation

\( G = (V, E) \) is an graph where the \( V \) is a set of vertex and \( E \subseteq V \times V \) is a set of edges. The adjacent matrix of graph is a matrix \( W = (\omega_{ij}), i,j=1,2,\ldots \). \( \omega_{ij} > 0 \) indicates that two vertices \( v_i \) and \( v_j \) are connect by an edge and the weight is \( \omega_{ij} \). As \( G \) is unweighted, if the vertex \( v_i \) is adjacent to \( v_j \), then \( \omega_{ij} = 1 \), otherwise \( \omega_{ij} = 0 \). \( N(S) \) is the neighbors of vertex set \( S \) if \( N(S) = \omega \in V \setminus S : \exists v \in S s.t. (v, \omega) \in E \).

3.2 Graph partitioning

For any subset of vertices \( V_i \subseteq V \), its complement \( V \setminus V_i \) is denoted by \( \overline{V_i} \). For two sets which are not necessarily disjoint, we define

\[
W(A, B) = \sum_{i \in A, j \in B} \omega_{ij}
\]

The cut set induced by \( V_i \) is \( C_i = (u, v) \in V_i, \overline{V_i} \), so the edge cut of cut set \( C_i \) is \( W(V_i, \overline{V_i}) \). The subsets of \( P = V_1, V_2, \ldots, V_k \) are k-way partitioning of graph \( G \) iff (1) \( \cup_i V_i = V \) and (2) \( \forall i, j : i \neq j \rightarrow V_i \cap V_j = \emptyset \). The most directed way to construct a partition of graph is solving the mincut problem which chooses a partition \( P = V_1, V_2, \ldots, V_k \) that minimizes

\[
cut(V_1, V_2, \ldots, V_k) = \sum_{i=0}^{k} W(V_i, \overline{V_i})
\]

However the solution of mincut which simply separates one individual vertex from the rest of the graph in many cases does not lead to satisfactory partition. One of the most common objective functions to make a “balance partition” of graph is RatioCut [12]. The definition is:

\[
\text{RatioCut}(V_1, V_2, \ldots, V_k) = \frac{\sum_{i=1}^{k} W(V_i, \overline{V_i})}{|V_i|}
\]

The balancing conditions makes the previously mincut problem which is simple to solve become NP hard [7].

4. MULTI-LEVEL STEPWISE PARTITIONING ALGORITHM

4.1 Overview of the Multilevel Paradigm

Graphs that we meet in practice are not random. The distribution of edges is not only globally, but also locally inhomogeneous, with high concentrations of edges within special subgraph of vertices, and low concentrations between these subgraphs. This feature of graphs is called community structure or clustering [11]. A possible reason for this feature is that the vertices are geographically close in social networks, or related to topic/domain on the web. Our graph partition algorithm takes advantage of this locality to make a good partition. The main idea behind our graph partition algorithm is that a dense subgraph should not be divided among partitions. So the dense subgraph is treated as an indivisible atom, and the graph partition towards vertices are transformed into the graph partition towards the dense subgraph. The dense subgraph means that there must be more edges inside a subgraph than edges linking vertices of a subgraph with the rest of the graph. The above dense subgraph is known as community.

The basic structure of our graph partitioning algorithm is similar to the multilevel k-way partitioning algorithm. Our graph partitioning algorithm is also a multilevel algorithm and uses the bottom-up method. The algorithm is divided into two phase: aggregate phase and partition phase. In the aggregate phase, we continue to aggregate the original graph to a smaller graph on a higher level. In each level, we use a label propagation algorithm to detect dense subgraphs and these dense subgraphs will be a super vertex of upper level. Then the label propagation algorithm continues on the upper level graph until the vertex size of upper level graph smaller than the threshold we set. In the partition phase, a balance graph partition performs on the top-level graph which vertex size is much smaller than the original graph. As the top-level graph is a vertex-weighted and edge-weight graph, where the edge-weight reflects the number of edges connecting two different super vertices and the vertex-weight reflects vertex number in the super vertex, the traditional partition algorithms such as Kernighan-Lin (KL) or spectral method are not suitable to this graph [17, 4]. We propose a novel stepwise partition algorithm using a greedy method to get a small RatioCut step by step. At every step, we extract a connected subgraph with vertex size approximates \( \frac{|V|}{k} \) which has minimal edge cuts with the remaining graph, and then remove the subgraph from the original graph. In this way, \( k \) subgraphs are iteratively extracted from the graph. A partition example on synthetic graph is shown in Figure 1.

4.2 Weighted label propagation

The main reason we use the label propagation algorithm to detect the dense subgraph is: the label do not need to access other vertex’s adjacent list, what it need is sending the label to its neighbor [21]. For this property, it is very suitable for parallel computing environment where access the vertex’s adjacent list is expensive. Besides, the time complexity of label propagation algorithm is linear. However the original label propagation algorithm that performs on unweighted graph can not adapt to the weighted graph on
upper level we face in the aggregate phase. In this paper, we propose a weighted label propagation algorithm to handle with the above weighted graph.

Our label propagation algorithm is based on the idea that which community the vertex \( v \) belongs to is determined by its close neighbors. We assume that each vertex in the graph chooses to join the community that most of its close neighbors belong to. Suppose that a vertex \( v \) has neighbors \( N(v) = \{u_1, u_2, ..., u_k\} \) and each neighbor owns a label \( L_u \) denoting which community they belong to. The neighbor who is more close to the vertex \( v \) has more influence on vertex \( v \). Hence we define the \( u \)'s affinity to \( v \)
\[
aff(v, u) = \frac{W(u, v)}{\sum_{i \in N(v)} W(u, i)}
\]
The degree of membership indicates how much the vertex belongs to a community \( C \) is defined as
\[
d(v, C) = \frac{\sum_{u \in N(v), L_u = C} aff(v, u)}{\sum_{u \in N(v)} aff(v, u)}
\]
The vertex \( v \) belongs to the community \( C \) which has the maximal degree \( d(v, C) \).

The algorithm is performed iteratively, where at iteration \( t \), each vertex updates its label by computing the degree \( d(v, C) \) based on labels of its neighbor vertices at iteration \( t-1 \). Hence, \( L_v(t) = f(L_{v_{t-1}}, ..., L_{v_{t-1}}) \), where the function returns the label with the maximal degree of membership. As the labels propagate, closely and densely connected subgraphs of vertices quickly reach a consensus on a unique label. At the end of the propagation process, vertices with the same label are aggregated as one community.

Ideally, the algorithm should be ended when all vertices' label unchanged after iterations. But in terms of ambiguous vertices with the same membership degree of several communities, their label may change after each iteration. So the algorithm should be ended, when most of nodes' labels are stable. Here, we will set the maximal number of iteration of the algorithm in the practice. The algorithm will be intermitted when iteration times equals the maximal number.

We can describe our weighted label propagation algorithm in the following steps.

1. Initial the label of vertices in graph. For a given vertex \( v \), \( L_v(0) = v \).
2. Set \( t = 1 \);
3. For each vertex \( v \) computes its affinity to its adjacent vertices \( aff(v, u) \), create a label with weight equals \( \frac{aff(v, u)}{|v|} \), where the divisor \(|v|\) is used to avoid excessively rapid growth of large-size vertex in upper level, and then send the label to the adjacent vertex.
4. For each vertex \( v \), \( L_v(t) = f(L_{v_{t-1}}, ..., L_{v_{t-1}}) \), where the function \( f \) returns the label with the maximal degree of membership.
5. For most of vertex, the label is unchanged, then stop the algorithm. Else, set \( t = t+1 \) and go to (3).

### 4.3 Implementation on MapReduce

MapReduce is a parallel computing framework which is simple and easy to use [8]. A application, which is based on the Map-Reduce framework, can run on large-scale commercial clusters, process data in a parallel and fault-tolerant way. MapReduce framework relies on the operation of \( <\text{key}, \text{value}> \) pair, both the input and output is a \( <\text{key}, \text{value}> \) pair. Users specify a map function that processes a \( <\text{key}, \text{value}> \) pair to generate a set of intermediate \( <\text{key}, \text{value}> \) pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The overall execution can be simply described as the following streaming:

Input | Mappers | Sort | Reducers | Output

The weighted label propagation algorithm is very easy to be implemented in a MapReduce way. In the mapper, the input key is the id of a vertex and the value is the vertex object. For each vertex, the map computes the affinity to its neighbors and sends the weighted label to its neighbors. The output key of the mapper is the neighbor's id and the value is the label and affinity. In the reducer, the vertex collects the labels that it receives, then computes the membership degree of different labels and updates its label with the label that has the maximal membership degree. The reducer will output the vertex's id as key and vertex object with the new label as value. After the label propagation, the vertices labeled with the same label will be grouped together as super vertex of upper level and the edge weight of these super vertices are computed for the further aggregation.

### 4.4 Stepwise minimizing RatioCut Algorithm

The top-level graph \( G_m(V_m, E_m) \) produced by multi-level weighted label propagation algorithm is a vertex-weighted and edge-weighted graph, where the edge-weight reflects the number of edges connecting two different super vertices and the vertex-weight reflects vertex number of the super vertex. It is very important to note that the partitioning algorithm must be able to handle the edges and vertex weights. In this section, a partitioning algorithm based on stepwise minimizing RatioCut is proposed to compute a k-way partitioning of graph \( G_m \) such that each partition contains roughly \( \frac{V}{k} \) vertices of the original graph. Instead of minimizing RatioCut \( (V_1, ..., V_k) \), which is a part of \( \text{RatioCut}(V_1, ..., V_k) \), as our first partition, and then remove the vertex set \( V_i \) from \( G_m \) and try to find the next partition in the same way. The partial-RatioCut of vertex set \( V \) is defined as
\[
\text{PRC}(V_i) = \frac{W(V_i, V)}{|V_i|}
\]
It is obvious that the problem of minimizing PRC\( (V) \) is also NP-hard. Because the problem is a 2-way graph partition problem. And we approximate the PRC\( (V_i) \) using a simple greedy algorithm. At each iteration, we greedily select the vertex that minimizing PRC of the currently constructed vertex set. Before the algorithm is proposed formally, we first give a proposition and proof it.

**Proposition 1.** There exist weighted graph \( G_m \) for which the PRC\( (V_i) \) is non-monotonic as the vertex size \( |V_i| \) grows.

**Proof.** Consider a current vertex set \( V_i \). Assume that the next vertex selected for inclusion in the set is \( v \). Then,
the new set is $V_i \cup \{v\}$. By the partial-RatioCut definition, the $PRC(V_i \cup \{v\})$ is \( \frac{W(V_i \cup \{v\}, \{v\})}{|V_i| + |v|} \) (note that the vertex of upper level is vertex-weighted super vertex). The $PRC(V_i \cup \{v\})$ will decrease when:

\[
\frac{W(V_i \cup \{v\}, \{v\})}{|V_i| + |v|} < \frac{W(V_i, \{v\})}{|V_i|}
\]

Because $W(V_i \cup \{v\}, \{v\}) = W(V_i, \{v\}) - k_{v, V_i} + d_v - k_{v, V_i}$ where $k_{v, V_i} = \sum_{v, j \in V_i} W(i, j)$ and the degree of vertex $v$ $d_v = \sum_{v, j \in |V_i|} W(i, j)$.

\[
W(V_i, \{v\}) - k_{v, V_i} + d_v - k_{v, V_i} < \frac{W(V_i, \{v\})(|V_i| + |v|)}{|V_i|}
\]

Conversely, The $PRC(V_i \cup \{v\})$ will increase when:

\[
d_v - 2k_{v, V_i} < \frac{W(V_i, \{v\})}{|V_i|}
\]

From the proof of proposition 1, we can know that in the order to minimize the $PRC(V_i)$, in every step the next vertex which minimize $\frac{d_v - 2k_{v, V_i}}{|V_i|}$ is selected to add to $V_i$, as shown in Algorithm 1. The parameter $\alpha$ in line 5 and 8 is used to avoid the case that an exceptional larger vertex add to the vertex set whose size approximate $\frac{|V|}{k}$ and will making the set size much larger than $\frac{|V|}{k}$.

In the following, we will present the $k$-way balance partitioning algorithm based on above stepwise minimizing $PRC$ algorithm. The algorithm runs with $k$ steps. In every step, each remaining vertex is selected as a start vertex and gets a vertex subset $S$ by stepwise minimizing $PRC$ algorithm. And get the best subset $S'$ with minimal $PRC$ from these subsets. Finally, remove the vertices of $S'$ from graph and start next step.

Algorithm 2 Stepwise partitioning Algorithm.

**Input:**
- Graph $G_m = (V_m, E_m)$; the partition number $k$.
- the vertex set List $setList$.

**Output:**
- the vertex set List $setList$
- $V_m = V_m \ \text{bestSet}$

1: $setList = \emptyset$;
2: $bestSet = \emptyset$;
3: for $i = 1; i < k; i + +$ do
4:   for $v \in V_m$ do
5:     $S = \text{minimize}PRC(G_m, \frac{|V|}{k}, v)$
6:     if $PRC(S) < PRC(bestSet)$ then
7:       $bestSet = S$
8:     end if
9:   end for
10: $setList = setList \cup bestSet$
11: $V_m = V_m \ \text{bestSet}$
12: end for

5. EXPERIMENTAL EVALUATION

In this section, we begin to evaluate the performance of our partitioning algorithm compared with some other existing partitioning algorithms on various graphs. And the scalability of our partitioning algorithm is also be evaluated on different scale large graph. Finally, we evaluate our partitioning algorithm in a real cluster application.

5.1 Hardware Description

The cluster environment used to test our algorithm composed of one master node and 32 computing nodes (Intel Xeon 3.20GHz x2, 2GB RAM, Linux RH4 OS) with 2TB total storage capacity, and deployed a Hadoop platform(a wildly accepted open source implementation of MapReduce). As a contrast, the running environment of stand-alone graph partitioning algorithm is: Intel Core2 Duo 2.66GHz processor, with 2GB memory, using WindowsXP operating system.

5.2 Dataset Description

We evaluated the performance of our multi-level stepwise graph partition algorithm(MSP) on different graphs obtained from various sources. There are a total of 21 different networks: 9 synthetic graph datasets and 12 real-world graph datasets. The synthetic graph datasets are produced by popular generative models, preferential attachment (BA) [3], Watts-Strogatz [26] and a power-law graph generator with clustering [14]. The synthetic datasets based on BA, WS, and PL were created with the NetworkX python package [22]. For graphs BA1, BA2, and BA3, which are generated by barabasi-albert model, we choose parameters $(n = 10000, m = 3)$, $(n = 100000, m = 5)$ and $(n = 500000, m = 5)$ respectively. And graphs PL1, PL2 and PL3, which are generated by powerlaw-cluster model with the same parameters as barabasi-albert model. For graphs WS1, WS2, and WS3, which are generated by Watts-Strogatz model, we
choose parameters \((n = 10000, m = 10, p = 0.1), (n = 100000, m = 10, p = 0.1)\) and \((n = 500000, m = 10, p = 0.1)\) respectively. The real-world graph datasets are mainly come from SNAP[18] and Graph Partitioning Archive [25]. The SNAP graphs used are: email-EuAll, email-Enron, amazon0312, amazon0302, web-NotreDame, web-Stanford, ca-CondMat, ca-help, LiveJournal and Twitter. The Partitioning Archive graphs used are: elt4, and cs4. The characteristics of these graphs are shown in Table 1.

5.3 Experimental Results

5.3.1 Performance of partitioning algorithm

We compared our partition algorithm with two existing partitioning algorithms: multi-level graph partitioning algorithm and spectral method. The multi-level graph partitioning algorithm[MP] is currently considered to be a popular algorithm and is used extensively. The multi-level graph partitioning algorithm and the spectral partitioning algorithm are provided by Chaco [13]. Note that the bisections produced by spectral were further refined by using a KL refinement algorithm. Due to space constraints and serial execution of Chaco, the comparison between our partition algorithm and algorithms provided by Chaco are made on small graph datasets. However, the relative performance of our algorithm performed parallelly on a cluster of commodity machines remains the same for larger graph datasets. We ran each experiment on 4 partitions and fixed the imbalance such that no partition held more than 1% more vertices than its share by setting the parameter \(\alpha = 0.01\). The fraction of edges cut obtained by the different partition algorithm across the different datasets are shown in Figure 2.

From figure 2, we can see that all the partition algorithms obtain a larger fraction of edges cut on the graphs: BA1, BA2, BA3, PL1, PL2 and PL3 than other graphs. These graphs have a low average clustering coefficient and more

| Name          | \(|V|\) | \(|E|\)  | \(|C|\)  | Type         |
|---------------|-------|--------|--------|--------------|
| BA1           | 10,000| 29,992 | 0.0057 | Synth.       |
| BA2           | 100,000| 499,975| 0.0011 | Synth.       |
| BA3           | 500,000| 2,499,975| 0.0010 | Synth.       |
| WS1           | 10,000| 549,735| 0.5931 | Synth.       |
| WS2           | 100,000| 2,750,988| 0.5882 | Synth.       |
| WL1           | 10,000| 29,990 | 0.0605 | Synth.       |
| WL2           | 100,000| 499,965 | 0.0331 | Synth.       |
| WL3           | 500,000| 2,499,961| 0.0319 | Synth.       |
| email-Enron   | 265,009| 364,481| 0.3093 | Social       |
| amazon0312    | 36,692 | 183,831| 0.4970 | Social       |
| amazon0302    | 400,027| 2349,960| 0.4113 | Product      |
| web-NotreDame | 262,111| 899,792 | 0.4240 | Product      |
| web-Stanford  | 62,586 | 1,090,108| 0.4540 | Web          |
| ca-CondMat    | 281,903| 1,992,636| 0.6109 | Web          |
| ca-help       | 23,133 | 93,439 | 0.6334 | Citation     |
| elt4          | 9,875  | 25,973 | 0.4715 | Citation     |
| cs4           | 15,606 | 504,230| 0.4076 | FEM          |
| LiveJournal   | 14,010 | 14,010 | 0.4715 | FEM          |
| Twitter       | 4.6 * 10^6| 77.4 * 10^6| 0.330 | Social       |
|               | 41.7 * 10^6| 1.468 * 10^7| 0.1060| Social       |

Figure 2: The fraction of edges cut of the different partition algorithm.

5.3.2 Scalability of partitioning algorithm

The graph datasets used in above performance comparison are tiny when compared with some of the graphs used in practice. While the above results are promising, it is important to understand whether our partitioning algorithm well scales with the size of the graph. As synthetic graphs produced by same generative model with similar parameter settings have similar graph statistics, we used the synthetic datasets in experiment in order to control for the variance in different graphs. We will present only the results for the Watts-Strogatz graphs, but all other graphs have similar results. We created 10 Watts-Strogatz graphs with a scale of 1 to 10 million vertices. For the Watts-Strogatz model, the probability of rewiring each edge is 0.1 and each node is connected to 10 nearest neighbors in ring topology.

We will consider the following questions in the experiment: (1) whether our algorithm’s partition performance on large graph is stable with increasing graph size. (2) how the partitioning quality scales with the number of partitions. (3) how the partition on MapReduce framework scales with graph size and computing nodes.

Firstly, the partition performance of 10 Watts-Strogatz graphs obtained is presented on the Figure 3. The Figure shows that the fraction of edges cut well scale with the size of the graph.
The next question is how the partitioning quality scales with the number of partitions. We only present the partition result of one graph in Figure 4, the 5 million vertex Watts-Strogatz graph, but all graphs have similar characteristics. As the figure show, the fraction of edges cut must necessarily increase as we increase the number of partitions. Our partitioning algorithm performs better than other partitioning algorithms when the partition number is small and obtains a approximate performance when the partition number is large. To show how our partitioning algorithm scales with graph size in MapReduce, Figure 5 presents runtimes of our partitioning algorithm for above 10 Watts-Strogatz graphs on the MapReduce framework, using a fixed reducer number 32.

As an indication of how our algorithm scales with computing nodes, Figure 6 shows runtimes for a Watts-Strogatz graph with 5 million vertices when the number of reducer varies from 2 to 64.

5.3.3 Results on real-world large datasets

In this section, we show whether a better graph partitioning will improve the performance of graph mining in real computation systems. To evaluate our partitioning algorithm in a real cluster application, we employed an implementation of PageRank on our cluster.

We used two datasets previously mentioned in our experiment, one is LiveJournal with 4.6 million vertices and 77.4 million edges, and the other is the Twitter graph with 41.7 million vertices and 1.468 billion edges. This two datasets were partitioned into 100 pieces with imbalance of at most 2% by our partitioning algorithm and Hashing approach, a popular method currently used by many real systems [19]. In most cases, Hashing equivalents to a random partitioning. For LiveJournal, our partitioning algorithm reduced the number of edges cut to 14,403,161 edges compared with 70,224,763 for the Hashing partitioning. For twitter, our partitioning algorithm cut 0.273 billion edges, while the Hashing partitioning cut 1.263 billion. We ran 10 iterations of PageRank of above two datasets, and repeated this experiment 5 times. The average runtime of a iteration of PageRank on two datasets with two different partitioning approach is shown on the Table 2. The result shows that using our graph partitioning algorithm as a preprocessing step for the large-scale distributed graph mining can yield a large improvement in the running time.

As it is shown in the experiment, our algorithm has a better performance than some existing algorithms. This is because that we use a weighted label propagation algorithm and a novel method based on stepwise minimizing Ratio-Cut. This algorithm can also be refined from the following respects. First, the performance of the proposed algorithm does not show too much advantage over the baseline algorithm on some of the graphs. It may be the case that whatever you partition the graph, the ratio can not be reduced too much. So more graphs with different structures should be investigated. Second, as other greedy algorithms have limitation, this greedy minimization of the partial Ratio-Cut algorithm also may fall into the local optimal solutions. So we should analyze to what extent does this greedy strategy influence the whole process. Finally, more state-of-the-art
algorithms should be referred, such as the Multilevel Algorithms for Partitioning Power-Law Graphs[1].

6. CONCLUSIONS
In this paper, we proposed a multi-level stepwise partition algorithm that drastically reduces the number of edges cut in distributed graph data. The algorithm first uses a weighted label propagation algorithm to aggregate the large original graph to a small upper level graph iteratively. And then a k-way balance partition is obtained by a novel method based on stepwise minimizing RatioCut. We conduct various experiments to evaluate our graph partitioning algorithm on Hadoop’s MapReduce with different large datasets and make a performance comparison with the other existing partitioning algorithms. The results show our partitioning algorithm has a better performance and scalability and can largely improve the efficiency of graph mining on real distributed computing system. For the future work, we plan to further investigate the interplay between the partition performance and various graph structural properties and their effect on partition.

7. ACKNOWLEDGMENTS
The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of this paper. This work is supported by the National Natural Science Foundation of China (Grant No.61074128, 60905025, 90924029).

8. REFERENCES