A Parallel Computing Method for Community Structure Detection Based on BSP Model

Yi Sun[1, 2], Zhen Hua[1, 2] and Li-hui Zou[1, 2]
1 School of Computer and Communication Engineering,
University of Science and Technology Beijing, Beijing, China 100083
2 Beijing Key Laboratory of Knowledge Engineering for Materials Science, Beijing, China100083
Email: emailofsunyi@aliyun.com, pandiansinian@163.com, zoulihui@ustb.edu.cn

Abstract—**Since the conventional algorithm for community structure detection in a stand-alone environment cannot handle the giant network whose number of nodes is more than 10^5, and the widely used MapReduce method has a limitation on dealing with excessive I/O operations during the iterative process, an efficient parallel computing method based on BSP (Bulk Synchronous Parallel) model for detecting community structure is proposed in this paper. The Fast Newman method is improved into parallel calculations with multiple steps under the framework of BSP model. It is more efficient to discover community structures in the large scale network. In order to testify the performance of the proposed method, a hama platform was built up on the same cluster of the hadoop platform. And a dataset, at a scale of 10^6, was also simulated for the experiments. It is approved that the proposed method is not only able to solve the issue of memory overrun in the conventional calculation on a stand-alone computer, but also to improve the performance effectively comparing to the MapReduce model. The proposed method has high practical value in large scale networks.**

Index Terms—**complex networks, graph clustering, modularity, Fast-Newman algorithm, BSP model**

I. INTRODUCTION

With the development of the social network, the research on community structure detection has become a hot topic in the current data analysis. How to increase the data amount being processed without reducing the accuracy of the community structure detection has become a major research issue. Community structure detection derives from graph partitioning [1]. Graph is widespread in our daily life [2]. Many issues can be represented by the graphs in the real world, such as the biological networks, the social networks, etc. With the increase of the complexity of the modern system, the research on complex network [3-5] is playing more and more important roles in many fields of applications [6]. As people lucubrate the physical meaning and mathematical characteristics of complex networks, they found that many real networks have a common characteristic, being with community structures. Although there is no clear definition for the community, the networks have an obvious feature, i.e. a network is usually composed by several communities, and the nodes in the same community are gathered closely, whereas the nodes in different communities are connected sparsely [7-9]. It has great practical significance to study the community structures for many applications, such as in locating the user positions [10], analyzing and managing the social network [11], predicting the protein functions [12], identifying the master control genes, mining the Web communities and classifying the search engines, etc.

Currently, there are many ways to solve community structure detections, such as Kernighan-Lin algorithm based on the local search [13], Girvan-Newman algorithm based on the edge betweenness [7], Fast-Newman algorithm based on the modularity [13] and so on. They solved the problem of community structure detections in small-scale network efficiently. However, when they face the large-scale network, their processing capacities become inadequate in the stand-alone computing.

Since traditional methods for community structure detections require the entire network to be stored in memory, a stand-alone memory has been unable to store such a large volume of data if the data size reaches up to 10^5. Therefore, how to deal with large scale data becomes a key problem. As the data size increases, the processing time increases as well. At a certain point, the efficiency of community structure detections will hit a bottleneck in the stand-alone processing. Parallel and distributed technology [14, 15] is one optimum solution for the above problems. As well known, MapReduce [16] is the main parallel technique at present. Nevertheless, there are still some deficiencies in MapReduce model for detecting community structures, especially in iterative calculations. Its efficiency would easily be affected by the repeated I/O operations for storing the intermediate states during the iterative calculations. There are a lot of inevitable intermediate states during the whole process of community structure detections. With the increase of the data size, the number of the intermediate state is also going to be increased. Excessive intermediate states result in that much of the processing time is consumed on the I/O operations, which is obviously not good for the large scale data processing.

Manuscript received Sep 30, 2013; revised Oct 25, 2013, accepted Dec 13, 2013.
Corresponding author: Li-hui Zou (zoulihui@ustb.edu.cn).

doi:10.4304/jsw.9.7.1876-1885

In this paper, the Fast-Newman algorithm based on BSP (Bulk Synchronous Parallel) model is proposed to address the above problems. It can hash the data which the memory cannot afford to the nodes of different clusters in a data partitioning way, and exchange the information among nodes via a message passing mechanism to update the intermediate states and control the execution procedures orderly by the barrier synchronization. Compared with the stand-alone processing, the proposed method solves both the problems of memory overflows and parallel computations. Compared with the MapReduce method which stores the calculation states in the disk, it greatly reduces the I/O times by using the total RAM of the computer cluster as the shared memory to store computing states.

II. BSP Model Overview

The BSP (Bulk Synchronous Parallel) model, also called overall synchronous parallel computing model, is a kind of parallel computing and programming model proposed by Leslie Valiant, in the 1980s [17]. It aims to build a parallel computing architecture, without depending on a certain specified structure, whose parallel nodes can be extensible as needed. It brings a bridge between the software and the hardware in the parallel computer field [18]. The BSP model is not only a kind of parallel architecture, but also a parallel programming model that can accurately analyze and predict the execution performance of the parallel program [19].

A BSP computation proceeds in a series of global supersteps. Fig. 1 shows the structure of a superstep. Three components are involved in one superstep [17]:

(1) *Concurrent computation*: several kinds of computations occur on all the involved processors, each of which only uses and stores values in their local memory. Computations in this process are independent, while other processes, such as message passing, are asynchronous.

(2) *Global communication*: all the processors exchange data in this process. After the computing tasks, each processor will send their non-local results to the message queues of other processors in a simplex mode.

(3) *Barrier synchronization*: Since there is no time priority between the local concurrent computation and the global communication, the barrier synchronization is used to finish the superstep. When a process reaches the barrier synchronization point, it will not start the next computing process until all other processes have completed their communicating actions.

One superstep is an iteration of the above three processes. Firstly, local concurrent computations are conducted respectively. The consuming time of these processes is usually different among all the processors as shown in Fig. 1. It indicates that the tasks assigned to each processor or the scales of the tasks are different. Secondly, each processor sends their own non-local computing results to the corresponding processors through the network communication. And at last, the barrier synchronization will wait until all the messages are sent out. The longest calculation time is taken as the superstep time.

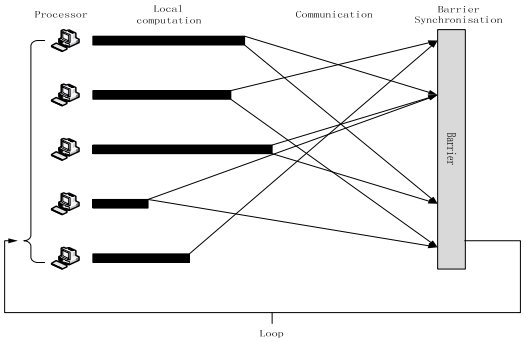

Figure 1. The structure of a superstep

III. Problem Descripions And Modeling

In this section, some important concepts are introduced and defined by mathematical descriptions, including complex network, community structure, node degree, connection of community and modularity, to formulate the problem of community structure detections according to a hierarchy clustering method. And the model of the problem is established combining with the characteristics of the BSP model.

A. Complex Networks and Community Structure

Complex network is an abstract description of a complex system. The complex system can be researched as a complex network if regarding its constituent unit as a node and abstracting the relationship between the units as an edge connection.

According to the definition of the complex network, we define a community structure network as a triple, $G=(V, E, C)$. Let $V=\{v_1, v_2, ..., v_N\}$ denote the set of nodes in a complex network G, in which N is the total number of vertices in the network. Let $E=\{(i_1, j_2), (i_2, j_2), \ldots, (i_M, j_M)\}$ be the set of connections between the nodes in the complex network G , in which M is the total number of edges in the network. If $(i, j) = (j, i)$, the network is an undirected graph, otherwise it is a directed graph. And let $C=\{c_1, c_2, \ldots, c_K\}$ as the set of communities in the complex network G, in which K is the total number of the communities.

The community structure C is defined as a quadruple, $C=(S, V, E_{in}, E_{out})$, where S represents the community label of the community, V represents the set of nodes in the community C, E_{in} represents the set of edges within the community C, and E_{out} represents the set of connections among this community and other communities.

An example of community structure which has been well divided in the network is shown in Fig. 2, in which the nodes in the same community are gathered closely, whereas the nodes in different communities are found in a lower density of edges.

The types of community can be classified by the tightness of community. If A is a community, it can be expressed by a quad, $C(A)=(A, V(A), E_{in}(A), E_{out}(A))$. Then two definitions are given as follows.

Definition 1 (Strong Community Structure): If $C(A)$ is a strong community structure, it must satisfy (1):

$$\begin{aligned} &h_i^{in}(A) > h_i^{out}(A), \\ &\forall i \in V(A), h_i^{in}(A) \in E_{in}(A), h_i^{out}(A) \in E_{out}(A) \end{aligned} \tag{1}$$

Definition 2 (Weak Community Structure): If $C(A)$ is a weak community structure, it must satisfy (2):

$$\begin{aligned} &\sum_{i \in A} h_i^{in}(A) > \sum_{i \in A} h_i^{out}(A) \\ &\forall i \in V(A), h_i^{in}(A) \in E_{in}(A), h_i^{out}(A) \in E_{out}(A) \end{aligned} \tag{2}$$

Since the constrain of the strong community structure is too strict, most of complex networks are analyzed by the weak community structure.

B. Degree and Community Connectivity

As a basic parameter of the network topology structure, degree is an index for measuring the importance of a node and it can reflect the ability of the node to establish a direct connectivity with its surrounding nodes. Community connectivity is described by degree. It can be divided into internal and external connectivity. Internal connectivity represents the degree formed by the connections among the internal nodes, and external connectivity represents the degree formed by the connections of the edge nodes between the community and other community.

Let E_{AA} denote the internal connectivity of the community A, and set N is the number of nodes in the community A. If there is an edge between the node v_i and node v_j, then it has:

$$d_{ij} = \begin{cases} 1 & i, j \in A \\ 0 & i, j \in A \end{cases} \tag{3}$$

$$E_{AA} = \sum_{i=1, j=1}^{N} d_{ij} \tag{4}$$

Let F_{AB} express the external connectivity of the community A related to the community B, and set M is the number of nodes in community A and N is the number of nodes in community B, then it has:

$$F_{AB} = \sum_{i \in M, j \in N} d_{ij} \tag{5}$$

C. Modularity

Modularity, an index invented by Girvan and Newman [20], is used to quantitatively evaluate the quality of network community partition. It is initially defined to be that Q = (number of edges within groups) − (expected number within groups). The modularity is measured relative to a null model which is defined by the probability of an edge between the nodes v_i and v_j. The modularity function Q can be defined as follows:

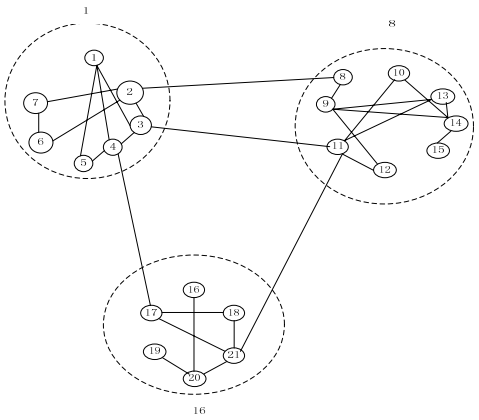

Figure 2. Diagram of community structure in networks

$$Q = \frac{1}{2M} \sum_{i,j} \left[\left(b_{ij} - k_i k_j / 2M \right) \delta(\sigma_i, \sigma_j) \right] \tag{6}$$

where b_{ij} represents there is an edge between the nodes v_i and v_j, δ is a membership function, if the vertices v_i and v_j belong to the same community, i.e. $\sigma_i=\sigma_j$, then $\delta(\sigma_i, \sigma_j)=1$, otherwise, $\delta(\sigma_i, \sigma_j)=0$, and $M = 0.5\sum_{i,j} b_{ij}$ describes the total nodes in the network. The probability of the edge between v_i and v_j is $\frac{k_i k_j}{2M}$ in a random network, in which k_i is the degree of v_i.

If Q is much closer to 1, then the partition of the community structure is much better.

D. Mathematical Model based on Modularity

According to the above descriptions of complex network, community structure, degree, community connectivity and modularity, assume that a network, $G=(V, E, C)$, has been partitioned into k communities. It has $e_{AA}=E_{AA}/2M$ expressing the fraction of all edges that link internal nodes of community A in the network, $f_{AB}=F_{AB}/2M$ representing the fraction of all edges that link the nodes in community A to the nodes in community B, and $a_A= \sum_{B \in C} f_{AB}$ denoting the fraction of edges that connect to the nodes in community A. According to the definition of the modularity function, the optimum community structure can be constructed at the $Q_{\max}$ as follows [21]:

$$Q_{\max} = \sum_{i=1}^{N} (e_{AA} - a_A^2) \tag{8}$$

Due to the rule that the larger value of Q, the better the community structures are, the principle of modularized increment is used to merge communities. The increment of Q is calculated as follows:

$$\Delta Q = f_{AB} + f_{BA} - 2a_A a_B \tag{9}$$

The communities can be merged iteratively in pairs in the way of increasing Q the most or decreasing Q the least until the calculation covering all the nodes goes into the convergence [22].

E. Community Structure in BSP Model

In the BSP model, the nodes in the network, as separate entities, are hashed into the whole cluster. Based on the description of community structure in the complex network, a triple C_S=(v_S, S, (C_{S_i} , C_{S_j})) is defined as the data structure of the community structure which can represent the community as well and a quad v_i=(i, S, N_{in}, N_{out}) is defined as the data structure of the nodes in BSP model, in which i is the ID of the node, S is the community label of the node, N_{in} is the set of the internal nodes of the community connecting to the node, N_{out} is the set of adjacent nodes connecting to the external communities of the community S. Several related definitions, the leader of community (LC), the parent community, the sub-merger and the main-merger, are given as follows for community merge in the BSP model.

Definition 3: v_S is the LC of the community S whose label is the same as that of the community. The LC is responsible for managing the unification of the internal nodes of the community C_S and combining with other communities.

Definition 4: the parent community C_S is composed of two sub-communities.

Definition 5: C_{S_j} is the main-merger if its community label does not change during the process of merging. The ID of the LC of the parent community C_s is the ID of the LC of the main-merger, and the community label of the parent community is that of the main-merger.

Definition 6: C_{S_i} is the sub-merger, referring to the community whose label is changed. Its LC needs to update the community labels of all the internal nodes in this sub-merger.

In the following section, a parallel computing algorithm will be built on the basis of the above model.

IV. Paralleled Fast-Newman Algorithm Oriented to BSP Model

A. Algorithm Description

The main idea of the algorithm is as follows. Firstly, the program hashes the nodes to the cluster as independent communities to manage them separately. And then the LC, representing its community, merges with other communities by computing their respective modularized increment in parallel during the process of merging. Each community filters out the potential merged object whose local modularized increment is the maximum. A predefined master node then collects all the potential merged objects and calculates the target that has maximal modularized increment globally. At last, based on the community label of this target, the LC data of the sub-merger is merged into the LC data of the main-merger to complete the community merge for once. In the above step of merging, the LC of the sub-merger needs to update the community labels of its internal nodes. The leader broadcasts the main-merger community label within its community, including all internal nodes and the nodes of sub-communities. Every sub-community updates in the same way until all nodes in the community have been updated with the new community label. Since updating the nodes and merging the nodes are exclusive with each other, they are executed in parallel. Recursively execute above steps until the modularized increment turns into negative, which implies the community partition is achieved at the maximum Q. The algorithm stops until all the nodes have been updated.

B. Community Merge Strategy

BSP model utilizes the leader merge strategy. The iterative merging process is shown in Fig. 3. Every node in the initial state in the network is an independent community. They are the leaders of their communities. Each LC passes the necessary information to its adjacent communities. The information is used to calculate the modularized increment between each adjacent pairs. Then each LC finds out the potential merged object, (C_{S_i} , C_{S_j} , ΔQ_{ij}), corresponding to the object with local maximum modularized increment, and sends it to the predefined master node. And the predefined master node compares all the modularized increments and finds out the pair of communities, (C_{S_i} , C_{S_j}), with global maximum modularized increment. At last, the two communities are merged into a new community C_S. The new LC, v_S , is equal to the LC of the main-merger, v_{S_j} , i.e. the new community and the main-merger have the same community label. During the process of merging, the new community sends its community label to the LCs of the sub-mergers for updating all nodes of the sub-communities. In the meantime, the LC of the new community continues to merge with other communities. The step of merging and the step of updating are executed in parallel.

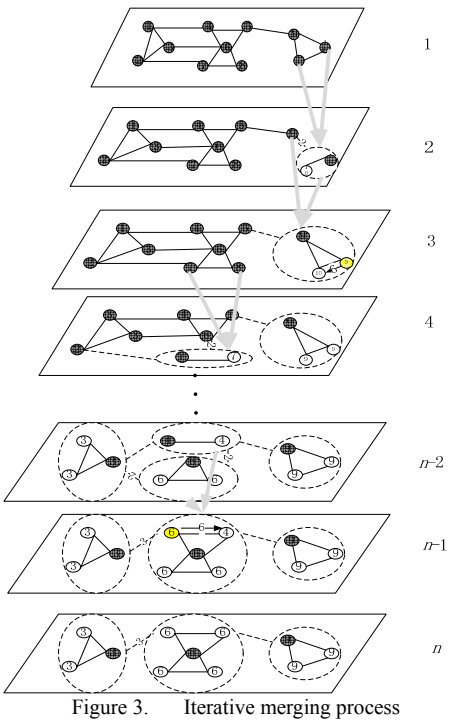

Figure 3. Iterative merging process

C. Multi-step Parallel Execution

The parallel computing is divided into two levels in BSP model. One is the parallel execution within the same superstep, e.g. multiple nodes in the same superstep compute the local modularized increments in parallel. The other one is the parallel execution of multiple steps, e.g. the step of merging and the step of updating are executed in parallel. Since community structure detection cannot be completed only via a single type of operation, the whole process is divided into the following steps:

Step 1: Calculate the modularized increment, select the merged objects, and merge the community.

Step 2: Update communities.

Step 3: Judge the termination.

The parallel execution process of the three steps is shown in Fig. 4. The overlaps in time axis indicates the parallel executions of multiple steps happen. In Fig. 4, Step 1 is the merging process. Each merging process consists of five supersteps. Step 2 is the updating process and Step 3 is the terminating process. The five supersteps of Step 1 execute orderly. Step 2 is triggered by Step 1. The execution times of Step 2 are determined by the number of the sub-communities of the sub-merger. Step2 can be almost interspersed throughout the whole process. It updates the community labels of the nodes within the sub-merger through the iterative message passing among the sub-communities. Step 2 can be executed in parallel not only with Step 1, but also with the previous uncompleted updating process of Step 2, i.e., as shown in Fig. 4, the mth updating executes in parallel with the (n–1)th updating.

Step 3 is caused by two types of conditions. The first is triggered by the nth merging. This terminal condition, i.e. the global modularized increment becomes negative, is found in the superstep 3 of the nth merging, which shows that the community structure of the network has been divided into the optimal state. Thus, the superstep 4 and 5 become the virtual supersteps of Step 1. The virtual supersteps no longer perform the merging process, but terminate the calculations of all nodes. The predefined master node will inform these two merged communities to terminate their calculations. These two communities will terminate their calculation status and transmit the terminating messages to their adjacent communities and sub-communities to terminate their calculation status as well. The previous uncompleted updating process of Step 2 can be executed in parallel with Step 3. For Step 2 and Step 3, they are both executed step by step. And the execution of Step 3 on each node is after the execution of the final execution of Step 2. Therefore, such parallel execution can still ensure that all nodes in the network have no more operations when they terminate their calculations. As shown in Fig. 4, the (n–1)th updating and the nth terminating execute in parallel.

The second terminal condition for Step 3 is triggered when a new community has no adjacent communities. The new community becomes an independent community. In Fig. 4, after the (m-1)th merging, the new community becomes an independent community, and it will never merge with others. Spread this message to its nodes and sub-communities to terminate their calculations. At this time, Step 3 can be then executed in parallel both with Step 1 and Step 2.

D. Algorithm Details

Some notations used in the proposed algorithm are illustrated first.

Constant parameters are listed in Table I.

Input: $R= \{(v_i, v_j)|i, j \in G(V)\}$.

Output :$\{(C_S, S, V)|S \in G(V), V \neq 0)\}$.

The variable declarations are given as follows:

Send message format: *Send* (t, m), where *Send* is the function for sending messages, t expresses the goal of the message and m represents the content of the message.

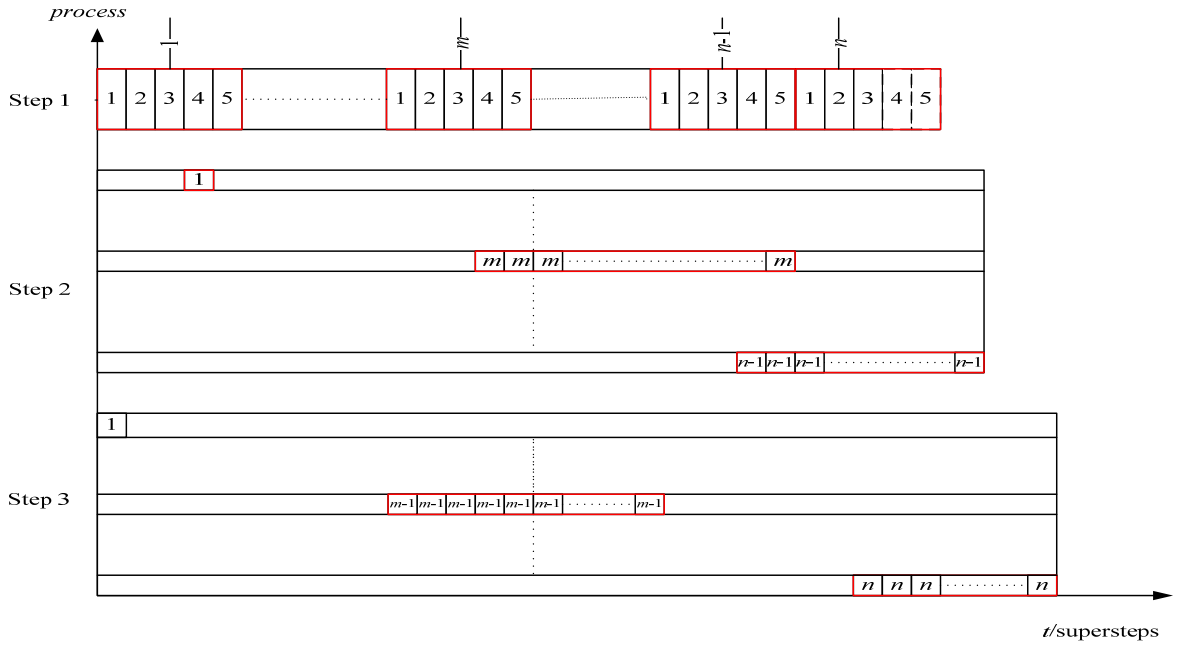

Figure 4. Parallel execution process

TABLE I
CONSTANT PARAMETERS

Parameter name	Symbol
Number of the cluster nodes	N
Number of edges	M
Synchronous node	I

Receive message format: *m*.getFirst(), representing extracting the first parameter from the message *m*. The method of extracting other parameters is similar to the above method.

Message types:

(1) m_1: The content of m_1 is composed of the sub-merger label and the degree of the sub-merger. It is sent to the LC of other community for calculating the local modularity increment of the LC.

(2) m_2: The content of m_2 is composed of local sub-merger label, local main-merger label, local modularized increment and the degree of local sub-merger. It is sent to the predefined master node for finding out the global merged target.

(3) m_3: The content of m_3 is composed of global sub-merger label, global main-merger label, global modularized increment and the degree of global main-merger. It is sent to the LC of the main-merger and the LC of the sub-merger for merging into a new community.

(4) m_4: The content of m_4 is composed of the sub-merger label and the main-merger label. It is sent to the LCs of the adjacent communities of the sub-mergers for inform them to replace the sub-merger label with the new community label in the set of their adjacent communities.

(5) m_5: The content of m_5 is composed of the adjacent community label of the sub-merger. It is sent to the LC of the main-merger for expanding its set of their adjacent communities.

(6) m_6: The content of m_6 is composed of the main-merger label. It is sent to the internal nodes and sub-communities of the sub-merger for updating their community labels.

(7) m_7: The content of m_7 is composed of termination signal. It is triggered when the global modularized increment becomes negative. The message is then spread to the whole network for terminating the calculations.

(8) m_8: The content of m_7 is also composed of termination signal. It is triggered when the new community becomes an independent one. The message is sent to the internal nodes and sub-communities of the new community for terminating their calculations.

Since Step 2 and Step 3 execute in random, and they can both execute in parallel with Step 1, we add them into each superstep of Step 1. If the nodes need to execute these two steps, then they will stop executing Step 1. Therefore, these three steps can be executed in parallel in the cluster.

One merger is completed by five supersteps. The whole community division can be finished by repeating the supersteps below. The operations of update and termination can be executed throughout the five supersteps. If the nodes receive m_6, m_7 and m_8, they would perform update or termination operations. The nodes will handle them with these supersteps in parallel, in which *Handle*(*m*) is a function used to replace the processing procedure and its pseudo code is as follows.

```
Handle(m):
  if type(m)=m8
    then isStop←true
         for each t∈Nin
             Send(t, m8)
         End
         elseif type(m)=m6
                then Si←m.getFirst()
                     for each t∈Nin
                         Send (t, m6)
                     end
         elseif type(m)=m7
                then isStop←true
                     for each t∈(Nout∪Nin)
                         Send(t, m7)
                     end
  end if
```

The implement details of the paralleled Fast-Newman algorithm oriented to BSP model are as follows.

Initialization: *Initialize the nodes and averagely assign them onto the cluster.* Firstly, initialize the node v_i= (i, S_i, N_{in}, N_{out}, D_i, *isStop*). i is the ID of v_i. S_i is the community label of the node. S_i=i represents the node is the LC of the community. N_{in}=$\emptyset$ and N_{out}= $\{(i, j)|\ j\in G(V),\ i\neq j\}$. D_i and *isStop* are additional properties. D_i is the degree of the community, D_i = *CountDeg*(N_{out}) in which *CountDeg* is a function for counting the degree of the community. *isStop* is used to mark the calculating status of the node, if *isStop*=false, the node performs calculations, otherwise, it stops. Every node is its own LC in the beginning. Then, assign all the nodes to their corresponding cluster node by the calculation function *Hash* ($\{v_i|\ v_i\in G(V)\}$)%N which is a function for calculating the location of the node on the cluster.

Superstep 1: *Prepare for computing modularized increment.* Every LC will send the message, i.e. *Send*(t, m_1), $t\in N_{out}$, to its adjacent communities. The format of the message is m_1= (S_i, D_i). The content of the message is the label and the degree of the community. If there are no messages sent out, declare that the community is an independent one, stop its calculation and send the termination message to its internal nodes and sub-communities, i.e. *Send*(t, m_8), m_8=(*stop*), $t\in N_{in}$, to inform its internal nodes and sub-communities to stop their calculations. The pseudo code of this superstep is as follows.

```
Superstep 1:
for each m∈list(M)
     if type(m)= m6 || type(m)= m7 || type(m)= m8
        then Handle(m)
     end if
```

```
    if i= S_i
      if N_out=∅
        then for each t ∈N_out
               Send(t, m_1)
             end
      else
          for each t∈N_in
            Send(t, m_8)
          end
      end if
    end if
end
```

Superstep 2: *Compute local modularized increment*. The LC of the community S_j receives the list of m_1. Firstly, it extracts all the related S_i from every m_1 and counts the number of the connections, f_{ij}, between S_j and S_i in the node set N_{out} (f_{ij}=*Count* (S_i, N_{out}) where *Count* is a function for counting the number of S_i in N_{out}). Secondly, the LC of S_j calculates the modularized increment of the pair of communities, (S_i, S_j), i.e. $\Delta Q_{ij} = \frac{2f_{ij}}{2M} - \frac{(D_i * D_j)}{2M^2}$. Then, the LC of S_j compares all the ΔQ_{ij} and finds out the local pairs with the largest modularized increment, (S_i^{local}, S_j^{local}, $\Delta Q_{ij_{\max}}^{local}$, D_i^{local}), in which D_i^{local} equals to the D_i of S_i^{local}. At last, it sends the message to the predefined master node, i.e. *Send*(t, m_2), in which m_2=(S_i^{local}, S_j^{local}, $\Delta Q_{ij_{\max}}^{local}$, D_i^{local}) and t=I. The pseudo code of this superstep is as follows.

```
Superstep 2:
ΔQ^local_ij_max ←-Double.MAX_VALUE
for each m ∈list(M)
    if type(m)= m_6 || type(m)= m_7 || type(m)= m_8
      then Handle(m)
      else if type(m)= m_1
          f_ij=Count (S_i, N_out)
          ΔQ_ij = 2f_ij/2M - (D_i * D_j)/2M^2
            if ΔQ^local_ij_max <ΔQ_ij
              then ΔQ^local_ij_max ←ΔQ_ij
                   S_i^local ←m.getFirst()
                   D_i^local ←m.getSecond()
                   S_i^local ← S_j
                   Send(I, m_2)
              end if
      end if
end
```

Superstep 3: *Compare global modularized increments and select global merged target.* The predefined master node I receives the list of m_2. Then it compares all the $\Delta Q_{ij_{\max}}^{local}$ in the list of m_2 and finds out the largest one as the global maximum increment ($\Delta Q_{ij_{\max}}^{global} = \Delta Q_{ij_{\max}}^{local}$). If $\Delta Q_{ij_{\max}}^{global}$ >0, the potential merged object of the m_2, corresponding to $\Delta Q_{ij_{\max}}^{global}$, constitutes the global merged object , m_3 = (S_i^{global}, S_j^{global}, $\Delta Q_{ij_{\max}}^{global}$, D_i^{global}), in which the S_i^{global} is the S_i^{local} of the m_2, the S_j^{global} is the S_j^{local} of the m_2, the D_i^{global} is the D_i^{local} of the m_2. Then *Send*(t, m_2), t= S_i^{global}, S_j^{global}. If $\Delta Q_{ij_{\max}}^{global}$ <0, it is the end of merging, and *Send*(t, m_7), m_7=(*stop*), t=i, j. The pseudo code of this superstep is as follows.

```
Superstep 3:
ΔQ^global_ij_max ←-Double.MAX_VALUE
for each m∈list(M)
    if type(m)= m_6 || type(m)= m_7 || type(m)= m_8
      then Handle(m)
      elseif type(m)= m_2
          if m.getThird()> ΔQ^global_ij_max
            then ΔQ^global_ij_max ← m.getThird()
                 S_i^global ← m.getFirst()
                 S_j^global ← m.getSecond()
                 D_i^global ← m.getFourth()
          end if
end for
if i=I
  if ΔQ^global_ij_max >0
      then Send( S_i^global , m_3)
           Send( S_j^global , m_3)
      else Send( S_i^global , m_7)
           Send( S_j^global , m_7)
  end if
end if
```

Superstep 4: *Prepare for merging*. For one situation, if the ID of the node is equal to the S_j^{global} in the m_3, then the node is the LC of the main-merger. The S_i^{global} of the m_3 is moved into its N_{in} by the LC. Then D_j= ΔD +D_j, in which ΔD is the D_i^{global} of the m_3 and D_j is the degree of the new community. At last, the LC of the main-merger removes the S_j^{global} of the m_3 from N_{out}.

For another situation, if the ID of the node is equal to the S_i^{global} of the m_3, then the node is the LC of the sub-merger. Firstly, the LC of the sub-merger updates its community label S_i to be the S_j^{global} of the m_3. Secondly, *Send*(t, m_4), t∈N_{out}, m_4=(S_i^{global}, S_j^{global}) for informing its adjacency LCs to replace S_i^{global} to S_j^{global} in their N_{out}. Then, *Send*(t, m_5), $t = S_j^{global}$, m_5=(*out*), *out*∈ $\left\{E \mid N \in N_{out} \text{ and } E \neq S_j^{global}\right\}$ for merging the community data. At last, *Send*(t, m_6), t∈E_{in}, m_6=(S_j^{global}) for updating the internal nodes and sub-communities of the sub-merger. If the LC of the main-merger and the LC of the

sub-merger receive m_7, they will stop calculations and *Send*(t, m_7), m_7=(*stop*), $t \in (N_{out} \cup N_{in})$ to inform others.

The pseudo code of this superstep is as follows.

```
Superstep 4:
for each m∈list(M)
    if type(m)= m6 || type(m)= m7 || type(m)= m8
      then Handle(m)
      else
        if i=m.getSecond()
         then Dj← m.getFourth()+Dj
              Nin.add(m.getFirst())
              Nout.remove(m.getFirst())
         elseif i= m.getFirst()
              then Si← m.getSecond()
                   for each out∈{E|N∈Nout and
                                 E≠ Sjglobal }
                       Send( Sjglobal , m5)
                   end
                   for each t∈ Nout
                       Send(t, m4)
                   end
                   for each t∈Nin
                       Send(t, m6)
                   end
        end if
    end if
end
```

Superstep 5: *Merge the communities*. If the LC of the main-merger receives the list of m_5, it will add the variable *out* of the m_5 to N_{out}. If the other LCs receive the list of m_4, they will replace S_i^{global} with S_j^{global} in their N_{out}. If the nodes receive the massage m_6, they are the LCs belonging to the sub-communities of the sub-merger. They will update their community labels. If they has sub-communities, then *Send*(t, m_6), $t \in N_{in}$, m_6=(j). If the nodes receive the massage m_7, they will continue to *Send*(t, m_7), m_7=(*stop*), $t \in (N_{out} \cup N_{in})$. The pseudo code of this superstep is as follows.

```
Superstep 5:
for each m∈list(M)
    if type(m)= m6 || type(m)= m7 || type(m)= m8
      then Handle(m)
        elseif type(m)= m4
              then replace(m.getFirst(),
                           m.getSecond(), Nout)
        elseif type(m)= m5
             then Nout.add(m.getFirst())
    end if
end
```

The above supersteps consist of the whole process of paralleled Fast-Newman oriented to on the BSP model.

V. Experiment and Analysis

The experimental software platform is HAMA - 0.6.1. The computer cluster used to test and compare the parallel programs (BSP-based Fast-Newman and MapReduce-based Fast-Newman) is composed of 5 PCs. The CPU is Core i5, the capacity of RAM is 4G. Classic stand-alone-based Fast-Newman was tested by a PC whose CPU is also Core i5 but the capacity of RAM is 8G.

In order to test the efficiency of the program, we counted the running time of the three algorithms at different orders of magnitude. The results are shown in Table II. Since the MapReduce program runs slowly, the estimated time is used, i.e. the total time ≈ (time of a merging) * (the number of merging).

From Table II, it can be seen that the program running on stand-alone PC showed obvious superiority when the amount of the nodes was less than $1*10^4$. Its performance was much better than those of BSP and MapReduce. When the amount of the nodes was more than $1*10^4$, the stand-alone program appeared memory overflows. Both MapReduce and BSP based program can still operate. However, the running time of Fast-Newman based on MapReduce model had become intolerable.

From Table II, we can also obtain the increment ratio of the running time among different data scales. The increment ratio represents the incremental volume of the running time with the growth of the data scale. It is an objective index for evaluating the effectiveness of the parallelism. The smaller the increment ratio is, the better the effectiveness of the parallelism is. The experimental results are shown in Table III and Fig. 5. It can be seen that the increment ratios of the BSP and the MapReduce model were relatively small, which means the growth of the data size made a little influence on the effectiveness of the parallelism on the BSP and MapReduce models. However, the influence was deep on the stand-alone. Combined with the increment ratio of the running time on the data size in Fig. 5, it can be predicted that when the amount of the nodes exceeds 10^5, even increasing the RAM of the stand-alone model until it can run any size of a single serial program, the running time of BSP-based parallel algorithm will be also less than that of the stand-alone-based. Although both MapReduce and BSP based parallel programs had a similar increment ratio, the MapReduce-based parallel program was limited by the running time as shown in Table II. To sum up, compared with the FastNewman algorithm based on stand-alone and MapReduce models, the BSP-based has obvious advantages in processing large-scale data.

The influence of the number of cluster nodes on the

TABLE II
Running Time Table

Data size (nodes/edges)	Running time(s)		
	Stand-alone	MapReduce	BSP
10/14	0.016	1089.9	12.711
10^2/196	0.094	10899	44.339
10^3/1997	2.27	108990	400.7
10^4/19992	2620.64	989920	3333.012
10^5/199993	NULL	10804700	51282.05
10^6/1999995	NULL	202000000	1111111.1

TABLE III
THE INFLUENCE OF DATA SIZE

Data size (nodes)	Increment ratio		
	Stand-alone	MapReduce	BSP
10	1	1	1
10^2	5.875	10	3
10^3	24.149	10	9
10^4	1154.47	9.08	8.33
10^5	NULL	10.91	10.91
10^6	NULL	18.695	21.67

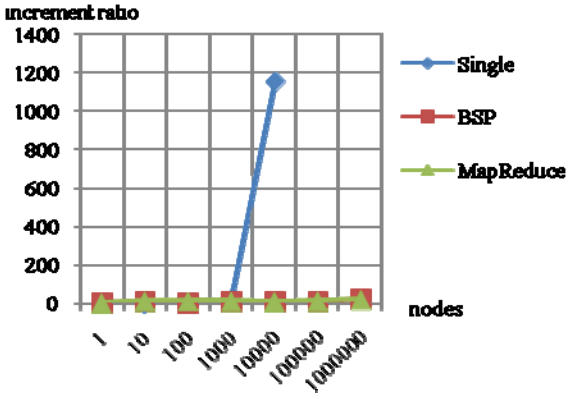

Figure 5. The influence of data size

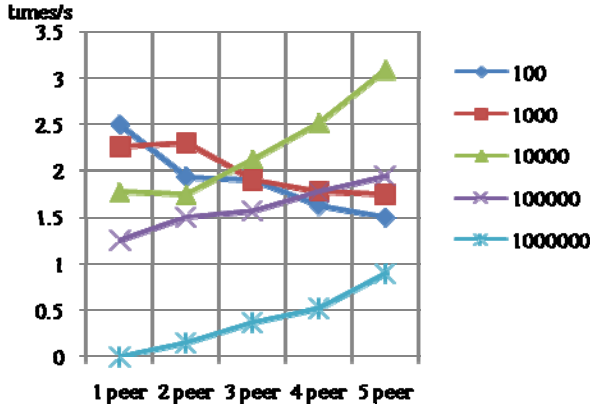

Figure 6. The influence of the number of cluster nodes on the BSP-based Fast-Newman algorithm

BSP-based Fast-Newman algorithm is also tested in our experiments. The purpose of the experiment is to evaluate t the degree of parallelism in processing large-scale data. Due to the message passing mechanism used on BSP model，increasing cluster nodes cannot ensure the high efficiency of parallelism. When the amount of the calculation and the communication achieves an equilibrium, the efficiency of parallelism becomes the highest. The experimental results are shown in Fig. 6. It can be seen that when the nodes of the network were less than 10^2, the merging times in per second reduced with the increase of the cluster nodes. The reason is that when the data size was small, the calculation could be completed instantly, but most of the processing time was spent on waiting for network communications. The more cluster nodes was, the longer the communication time was. Moreover, the time of task allocation in the whole running time was also quite considerable.

When the nodes of the network reached to 10^3, the merging times in per second increased at first, and then decreased, as shown in Fig. 6. It achieved the maximum point when the cluster nodes were 2, which can be deemed that it achieved the equilibrium between the calculations and the communications.

When the nodes of the network reached up to 10^4, the merging times in per second were proportional to the increase of cluster nodes. In this situation, the calculating time was more important. The time for calculation became dominant. The more the cluster nodes were, the higher the merging efficiency was.

When the nodes of the network reached up to 10^6, the merging times in per second were significantly less than those whose nodes were less than 10^6 since the increase of data size would easily result in increasing the calculation time. Nevertheless, the merging times in per second were still proportional to the increase of cluster nodes as shown in Fig. 6.

Through the above experiment, it showed that the paralleled Fast-Newman on BSP model could increase the degree of parallelism. It was helpful for processing large-scale data. Nevertheless, it was not the more cluster nodes, the better the degree of parallelism, just as the instance of 10^3. It can be inferred that the merging times in per second at different order of magnitude has the same trend, i.e. increased at first and then decreased. In order to achieve the best efficiency of the parallelism, the balance point between the data scale being processed and the number of the cluster nodes should be found in advance for constructing the optimum processing environment.

VI. CONCLUSION

The paralleled Fast-Newman algorithm based on BSP model is proposed in this paper. It takes full advantage of the characteristics of the parallelism and the distributed computing framework. It can solve the network scale problem which the stand-alone computation cannot handle. Under the same conditions, it is more efficient than the MapReduce-based Fast-Newman algorithm. The experiment proved that, when the data size was between 10^4 and 10^6, the proposed algorithm appeared a better efficiency without reducing the accuracy of community structure detection. It can be used in community structure detections in the order of magnitude at 10^6 nodes.

The time complexity of the improved Fast-Newman algorithm in the BSP model is $O(n^2)$. It reduces the running time of modularity increment calculations, but the merging times do not reduce. When the network scale enlarges to a certain degree, the merging times will be the

main factor that affects the efficiency of community structure detections. Therefore, the next step is to address the problem how to merge unconcerned multi-communities in parallel during one time calculation.

ACKNOWLEDGMENT

This work is supported by the National Key Technology R&D Program in 12th Five-year Plan of China (No. 2013BAI13B06) and China Postdoctoral Science Foundation funded project (No. 2013M540863).

REFERENCES

[1] G. Karypis, and V. Kumar, "A Coarse-Grained Parallel Formulation of Multilevel k-way Graph Partitioning Algorithm," SIAM: Parallel processing for scientific computing, 1997, pp. 1-12.
[2] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. W. Berry, "Challenges in parallel graph processing," *Parallele Processing Letter*, vol. 17, pp. 5-20. January 2007.
[3] B. Yang, D. Y. Liu, J. M. Liu, D. Jin, H. B. Ma, "Complex Network Clustering Algorithms," *Journal of Software*, vol. 20, no. 1, pp. 54-66, 2009.
[4] D. Chen, X. Xu, "A Novel Approach for Finding Clusters from Complex Networks," *Journal of Software*, vol. 6, no. 1, 2011.
[5] S. Z. Li, X. M. Wang, "Complex Networks Community Structure Division Algorithm Based on Multi-gene Families Encoding," *Journal of Computers*, vol. 8, no. 1, pp. 3021-3026, Dec 2013.
[6] M. Rosvall, "Information horizons in a complex world," Umea: Umea University, 2006, pp. 19-29.
[7] M. E. J. Newman, "Fast algorithm for detecting community structure in networks," *Physica Review E*, vol. 69. no. 6, 2004.
[8] S Fortunato. "Community detection in graph," *Physics Reports*, vol. 486, pp. 75-174, 2010.
[9] M. Girvan, and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, 2002.
[10] B. Krishnamurthy, and J. Wang, "On network aware clustering of web clients," *Proceedings of the Conference on Applications, Technologies, Architectures and Protocols for Computer Communication,* New York: ACM, 2000, pp. 97-110.
[11] P. K. Reddy, M. Kitsuregawa, and P. Sreekanth, *A graph based approach to extract a neighborhood customer community for collaborative Works-hop on Databases in Networked Information Systems*, London: Springer Berlin Heidelberg, 2002, pp. 188-200.
[12] Y. Dourisboure, F. Geraci, and M. Pellegrin, "Extraction and classification of dense communities in the web," *International Conference on the World Wide Web*, New York: ACM, 2007, pp. 461-470.
[13] M. E. J. Newman, "Detecting community structure in networks," *European Physical Journal*, vol. 38, no. 2, pp. 321-330, 2004.
[14] J. Zhang, G. Q. Wu, X. G. Hu, S. Y. Li, S. L. Hao, "A Parallel Clustering Algorithm with MPI – MKmeans," Journal of Computers, vol. 8, no. 1, pp. 10-17, Jan 2013.
[15] K. Chen, W. M. Zhen, "Cloud Computing:System Instances and Current Research," *Journey of Software*, vol. 20, no. 5, pp. 1337-1348, 2009.
[16] J. Dean, and S. Ghemawat, "MapReduce Simplified Data Processing on Large Clusters," *Symposium on Operating Systems Design and Implementation*, 2004, pp. 137-150.
[17] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "A System for Large-Scale Graph Processing," *ACM SIGMOD International Conference on Management of data* , 2010, pp. 135-146.
[18] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, Auguest 1990.
[19] J. J. Climent, C. Perea, L. Tonosa, and A. Zamora, "A BSP recursive divide and conquer algorithm to solve a tridiitional linear system," *Applied Mamematics and computation*, vol. 159 , pp. 459-484, 2004.
[20] M. E. J. Newman, and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, 2006.
[21] K. Kteinhaeuser, and N. V. Chawla, "Identifying and evaluating community structure in complex networks," *Pattern Recognition Letters*, 2010
[22] X. J. Li, P Zhang, Z. R. Di, and Y Fan, "Community structure of complex networks," *Complex systems and complexity science*, vol. 5, no 3, pp. 19-42, 2008.

Yi Sun, born in 1963, received the B. Eng. degree in computing application technology from Beijing Institute of Technology, P. R. China, in 1985.

Currently, she is an associated professor at University of Science & Technology Beijing. She has been involved in many important national projects, such as "973", "863", National Key Technology R&D Programs and National Natural Science Foundation of China. Her research interests include artificial intelligence, data mining and knowledge discovery, semantic technology and ontology-based knowledge base construction.

Zhen Hua, born in 1986. He is a M.S. candidate at University of Science & Technology Beijing.

He has been involved in National Key Technology R&D Programs 12th Five-year Plan of China. His research interests include data mining artificial intelligence, semantic technology and ontology-based knowledge base construction and social network analysis, etc.

Li-hui Zou, born in 1982, received her B. Eng. degree and M. Eng. Degree in electrical engineering and automation in 2005 and 2007, respectively, from Northeast Forestry University, P. R. China, and received her Ph.D. degree in control science and engineering from Beijing Institute of Technology in 2012.

She was selected for an international academic exchange with full scholarship and studied at Universidad Politécnica de Madrid (UPM) in 2009. Now she is doing her postdoctoral work in University of Science & Technology Beijing. Her research interest covers complex system analysis, artificial intelligence and machine vision.