Latency Based Dynamic Grouping Aware Cloud Scheduling

Sheheryar Malik, Fabrice Huet, Denis Caromel

2012 26th International Conference on Advanced Information Networking and Applications Workshops

Latency based Dynamic Grouping aware Cloud Scheduling

Sheheryar Malik,
Research Team OASIS
INRIA - Sophia Antipolis
Sophia Antipolis, France
sheheryar.malik@inria.fr

Fabrice Huet
Research Team OASIS
INRIA - Sophia Antipolis
Sophia Antipolis, France
fabrice.huet@inria.fr

Denis Caromel
Research Team OASIS
INRIA - Sophia Antipolis
Sophia Antipolis, France
denis.caromel@inria.fr

***Abstract*—In a multi-cloud environment, distributed application may need to have minimal inter-node latency to improve their performance. In this paper, We propose a model for the grouping of nodes with respect to network latency. The application scheduling is done on the basis of network latency. This model is a part of our proposed Cloud Scheduler module, which helps the scheduler in scheduling decisions on the basis of different criteria. Network latency and resultant node grouping on the basis of this latency is one of those criteria. The main essence of the paper is that our proposed latency grouping algorithm not only have no additional network traffic for algorithm computation but also works well with incomplete latency information and performs intelligent grouping on the basis of latency. This paper addresses an important problem in cloud computing, which is locating communicating virtual machines for minimum latency between them and group them with respect to inter-node latency.**

***Keywords*- Latency grouping; Cloud scheduling; Group partitioning**

I. INTRODUCTION

In a multi-cloud environment, processing nodes (i.e. virtual machines) can be geographically co-located in the same cluster or far enough to be in different countries. Even they can belong to different cloud vendors. So there can be more inter-node latencies as compared to a typical cluster. If an application requires coordination among different cloud nodes, then the performance can be degraded due to the high inter-node latency. Thus there is a high demand to have the inter-node latency as minimal as possible. In order to have an optimal solution for this situation, we must have a group of nodes which have minimum inter-node latency and also have the minimum number of nodes required for the application.

Here we have come up with a problem statement. The problem statement has a number of requirements to address. First requirement is to group nodes on the basis of nodes connection value (i.e. latency) instead of number of connections to the nodes. The nodes in a group must have minimal inter-node latency. Second requirement is for the algorithm to work on incomplete latency information i.e. we do not have *NxN* communication instances (connections). And the third requirement is that the groups should be pre-computed, so that the user applications should not need to ask the algorithm to compute the groups every time it needs to run. Fourth requirement is to produce mutually exclusive groups, so a node in one group should not appear in other group. If the node is not-mutually exclusive in groups then a node which has already been assigned to an application can be a candidate to be assigned to another application because of appearing in another group.

As a solution to the above said problem, we propose an algorithm to group the nodes with minimum inter-node latency together. In this solution, the algorithm is quite capable to group the nodes with the minimum available latency information. Latency information is available only for those nodes, which have done some communication with other nodes. It does not rely on broadcasting to calculate the inter-node latency, as broadcasting consumes a lot of bandwidth and put a burden on network traffic. The proposed algorithm does not send special messages to other nodes to calculate latency. It uses the piggy back technique by determining the latency for the nodes which have done some communication. We aim to come up with a prudent solution, which heuristically performs the grouping decision on incomplete latency information. The algorithm produces multiple mutually exclusive groups with variable number of nodes. Thus a node can appear in only one group. It is ideally suitable when a process wants to execute its sub-tasks on multiple groups. Scheduler finds the group or groups with most suitable size for the demanding process. For a process which is not going to demand more resources during its course of execution, it should find a group closer to the minimum number of nodes required. For a process which can demand the scalable resources, scheduler chooses a group which can support execution in peak scalability demanded. The algorithm has different configurations, which differs according to the situation. The algorithm result is normally different for every configuration. The algorithms runs periodically and also when there is some change in cloud, like the process migration to other virtual machine, change of IP address etc. In this paper, we are focused on group discovery and creation. Here we are not addressing the issues related to the latency updates. As the latency update mechanism is not within the scope of our work, we use the already existing force based embedding techniques [12], [13], [14] for latency update.

The rest of the paper is structured as follows. Section 2

978-0-7695-4652-0/12 $26.00 © 2012 IEEE
DOI 10.1109/WAINA.2012.91

gives the background and existing related work done in the area. Then in section 3, we present our proposed model. In Section 4, we describe the results of experimental evaluation of our proposed algorithm. We conclude in Section 5 and discuss the future research directions.

II. Related Work

Grouping of nodes with respect to some certain characteristics has been studied under various names. Most commonly it is referred as community detection. Community, group, or cluster detection have been done for various disciplines and fields, including internet, world wide web, biological network, citation network, social network, complex systems, graph theory etc. Different algorithms have been produced tailored to the needs of the discipline. Most of these algorithms are based on grouping of nodes on the basis of connection count for a particular node. However, in our problem scenario, we are grouping the nodes on the basis of connection value between nodes. The connection value is determined by the latency between the nodes. In general, a network community is a group of nodes with more interactions between its members than the other [5]. A node in a group or community normally has more interactions inside than outside the group. A community detection algorithm is considered better if its communities' external node connection to internal node connection ratio is lesser. This property is called conductance. Our algorithm produces groups with high conductance.

We have examined some of the existing algorithms for our problem statement. However, none of the existing algorithms fulfill all the requirements of it. As mentioned in our problem statement, every node connection has a value and the node proximity to its neighbor is determined by that value. In most of the existing community detection algorithms, the nodes are grouped with respect to the number of nodes connected to a particular node. Nodes which are far in terms of number of connection count (hop count) normally fall in different groups. Generally these algorithms produce non-mutually exclusive groups. Thus, none of these existing algorithms fulfill all the requirements of our problem statement. Here we list a few of the existing algorithms, which generally produce overlapping groups or communities. Our algorithm produces mutually exclusive groups according to our problem statement and performs well on incomplete latency information with a complexity of O(n^2).

A distributed community detection model has been introduced by Hui *et al.* [1]. In their work, they have three variations of their algorithm based on SIMPLE, *K*-CLIQUE and MODULARITY methods. The algorithm performs the community detection in a distributed fashion on mobile devices. It creates groups which are not mutually exclusive and there is no node regrouping. k-CLIQUE and MODULARITY have better performance than SIMPLE, but the SIMPLE is lesser complex i.e. O(n). Whereas, Modularity has a complexity of O(n^4) in worst case. k-CLIQUE has a complexity of O(n^2).

There also exist a few graph partitioning algorithms, which divides the graph in groups or communities. Local Spectral Partitioning algorithm [3] and Flow-based Metis+MQI [7] are widely used. Metis+MQI is better than Local Spectral in terms that its communities' internal connection is more stronger than external. So Metis+MQI has better conductance than Local Spectral. On the other hand, in worst case Metis+MQI can have communities with no inter-community connection. Whereas, in Local Spectral communities are always interconnected, which is a good property. In this situation, Local Spectral is better. Local Spectral is also more compact than Metis+MQI.

There are some hierarchical agglomerative clustering algorithms, which generate clusters or groups. Clauset has proposed an agglomerative algorithm [8] that greedily maximizes the modularity. The average complexity of the algorithm is O(n^2). It generates non-mutually exclusive groups. Zhao devises a mechanism for hierarchical agglomerative clustering with respect to ordering constraint [10]. The complexity of the algorithm is O(n^3). It generates overlapping clusters with nodes possibly common in different clusters. Garcia devised a general framework for agglomerative hierarchical clustering algorithms [9], in which he tested and compared different existing algorithms. Newman has proposed an algorithm [4], which is based on the edge removal mechanism. In the worst case, it has a complexity of O(n^3).

III. Proposed Model - Group Discovery Algorithm

The basis of our proposed solution is an algorithm that groups the nodes with respect to their inter-node latency. The cloud scheduler performs the scheduling decision on the basis of grouping information gained from the algorithm. The proposed algorithm targets specifically to the situation, when the latency information is incomplete and only available for the nodes, which have done some communication with other nodes. Thus it has a minimized effect on network traffic load. The produced groups are mutually exclusive, means that a node can only appear in one group.

The algorithm discovers the neighbors within a specified latency and creates the groups. The algorithm has three different phases. The most important and core phase is configuration phase. This algorithm runs for every node and decides for node grouping by finding latency to its neighbor. This latency is compared to a threshold and group threshold value which acts as a mean in deciding for node grouping. The algorithm is iterated for a number of times equals to the number of nodes. Before going further into the algorithm description, we introduce the concepts that we are going to use in the algorithm.

Visit node is a node, which is selected during each iteration to check its latency with it's neighbors and possible grouping of neighbor. *Neighbor node* is a node, which has done some communication with a particular visit node. *Threshold* is the maximum latency allowed between two communicating nodes, candidate to be grouped together. Threshold value depends on the threshold policy. *Group Threshold* is the maximum latency allowed for grouping. It is the distance latency from the visit node to the root node of its neighbor. Group threshold

policy also depends upon the group threshold policy. Latency between visit node and neighbor node is compared with threshold and group threshold.

A. Initialization Phase

It is a pre-processing phase of the algorithm. In this phase, the initialization of environment variables is performed, which helps the algorithm to execute. The grouping decision in the next phase is done on the basis of these initial pre-configurations. The pre-configurations are calculated from different policies.

Threshold and Group Threshold Value is one of the most important thing to decide. Threshold and group threshold really play the most significant role in the output of the algorithm.

Threshold can be decided;

- by the scheduler administrator on the basis of his past experiences. In this case, administrator must have knowledge about the network infrastructure.
- on the basis of statistical information of inter-node latency. It can be set either on the basis of median or mean latency. It is more preferred to decide on the basis of mean latency and standard deviation. In normal case, the threshold will be equal to the mean or median latency of all the nodes. Group threshold can be either a combination of mean and standard deviation or median plus some fraction of median.

In the first case, the administrator needs to be very experienced in dealing with this type of situation and can do it with highly favorable result. In the second case (i.e. statistical based), mean latency can be affected by the outliers. Median seems to work better than mean latency as it is not affected by outliers. The next difficult decision is to choose a value for Group Threshold. In normal case, we select it as twice as the threshold.

B. Configuration Phase

This is the core phase of the group discovery algorithm. In this phase, we apply the main part of the algorithm. It groups the nodes with respect to their inter-node latency. The algorithm runs for each node and decides for its grouping by finding latency to its neighbor. The input to the algorithm is the latency and configuration information.

Algorithm

for each node in registry **do**
  select visitNode
  if numberOfNeighborOfVisit = 0 OR (previousOfVisit = null & visitNode $\neq$ grouped) **then**
    create newGroup
    add visit to newGroup
  end if
  for each neighbourNode of visitNode **do**
    input latency(visit, neighbor)
    if latency(visit, neighbor) $\leq$ threshold **then**
      if neighborNode $\neq$ grouped **then**
        add neighborNode into tempGroup
        call_proc: checkForGroupableToVisitRoot(visit, neighbor)
      else if neighborNode $=$ grouped **then**
        if neighborNode $\neq$ visited **then**
          add neighbor to tempGroup
        else if neighbor $=$ visited & latency(visit, neighbor) $<$ latency(neighbor, previousOfNeighbor) **then**
          call_proc: checkForGroupableToVisitRoot(visit, neighbor)
        end if
        add neighbor to tempGroup
      end if
    else if latency(visit, neighbor) $>$ threshold **then**
      if neighborNode $\neq$ grouped **then**
        create newGroup
        add neighborNode to newGroup
      end if
    end if
  end for
  check that the tempGroup is group-able to groupOfRoot or not
  if moreThanGroupThresholdCounter/thresholdCounter $\leq$ regroupingIndex **then**
    copy tempGroup to groupOfVisit
  else if thresholdCounter $>$ regroupingIndex **then**
    create newGroup
    if isVisitRemovable $=$ true **then**
      remove visit from existingGroup
      add visit to newGroup
    end if
    copy tempGroup to newGroup
  end if
  rank visitNode to 0
end for

Procedure definition: checkForGroupableToVisitRoot(visit, neighbor)

if sumOfLatency(neighborOfVisit, rootOfVisit) $>$ groupThreshold **then**
  increase moreThanGroupThresholdCounter
end if
if any latency(visit, neighbor) $<$ latency(visit, previousOfVisit) **then**
  remove isVisitRemovable = true
  add visit to tempGroup
end if

Algorithm Description:

This algorithm iterates for the number of nodes in the system. In each iteration, it selects a visit node (i.e. node with highest node rank among all the non visited nodes). Then it checks the latency between the visit node and all of its neighbor nodes. If the latency between the visit node and

neighbor node is less than or equals to the threshold, then it checks that whether the neighbor is already grouped or not. If the neighbor node is not grouped before, then it adds the neighbor node to a temporary group. After this, it checks that the sum of the latency from the neighbor of visit node to the root of visit node is less than the group threshold or not. If it is not, then it raises the flag that the neighbor cannot be grouped to the root of the visit node. Then it compares the latency of the neighbor node and the visit node with the latency of visit node and it's previous node. If any one of the neighbor nodes has lesser latency to the visit node than the latency from visit node to the previous node, then it removes visit node from the old group and adds it to the temporary group. In the other case, if the neighbor node is already grouped, then it further checks that whether it is also visited or not. If it is not visited, then it groups the node with the temporary group. But if it is already visited, then it checks, whether the latency of neighbor node and visit node is less than latency of neighbor node and it's previous node or not. If so, then it checks if the sum of latencies from the neighbor of visit node to the root of visit node is not less than the group threshold. If so, it raises the flag that neighbor node cannot be grouped with the root of visit node. It also checks that if any one of the neighbor node has lesser latency to the visit node than latency from visit node to the previous node then it removes the visit node from the old group and add to the temporary group.

On the other hand, if the latency between visit node and neighbor node is greater than the threshold, then it checks whether the neighbor node is already grouped or not. If it is already grouped, then it does not do anything. But if the neighbor node is not grouped before, then it creates a new group and adds the neighbor node to the new group. Then it checks that the temporary group is group-able to the root or not. It is checked by comparing the threshold-counter with the regrouping index (normally half of the number of neighbors within threshold). If threshold-Counter results in a number more than or equals to the regroupingIndex then it copies all the elements of the temporary group to the group of visit node. But if threshold-counter is less than regroupingIndex then it creates a new group. Then it checks whether the visit node can be removed from old group or not (if the latency between any neighbor node and visit node is less than latency between visit node and previous of visit node then the visit node is removable from old group). If so, then it removes visit node from old group and add it in the new group. Then it copies all the nodes from the temporary group to the new group. Then it ranks the visit node as 0 to indicate that it has been visited. It repeats all the above steps starting from finding of visit node till here.

C. Reconfiguration Phase

In this phase the algorithm creates new groups and adds the existing leaders of the old groups to each one of the new groups. Now each node checks its distance from the node leader. This phase of the algorithm tries to group the nodes which belong to a single node group. These nodes are candidate nodes to be grouped with some other nodes. In this phase, every candidate node probes for latency to the leaders of all the groups. It tries to find a leader node with minimum latency among all and within threshold. If it finds that the minimum latency is less than the threshold then it groups the candidate node with the group of leader node having minimum inter-node latency. If it fails to find a leader node with latency less than threshold then it does not group it with any one and appears as an independent group with one node.

IV. Experiments and Results

We have implemented our algorithm and experimented it with different configurations in different environments on clusters, grids and cloud. We have The results obtained from these experiments reflect the algorithm efficiency with different configurations in different conditions.

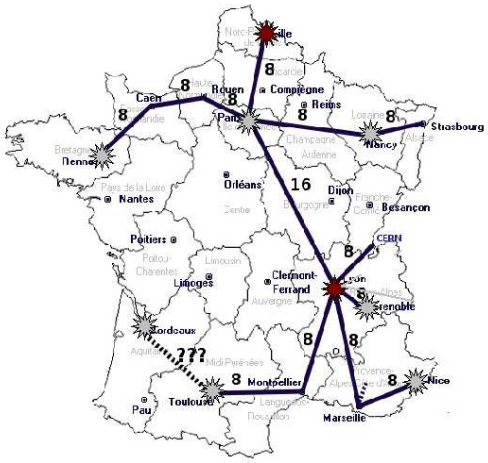

Figure 1. Sites of Grid5000 across France [11]

In this paper, due to the space limitation, we have included only the experiments performed on Grid5000 [11]. There are multiple sites in Grid5000, which are geographically located far from each other at variable distances as shown in Figure-1. In this type of network, the algorithm works better for the configuration value set by administrator, whereas configurations based on mean and median normally does not behave well. All of the following experiments were performed with spiral ranking policy.

We have selected multiple sites for our experiments. Here, we list a few of the experiments. All the time values shown in the experiments are in micro seconds. The experiments from 1 to 3 were conducted on 6 sites of Grid5000 with 8 nodes on each site. The node distribution was like this.
Node 00-07: Grenoble
Node 08-15: Lyon
Node 16-23: Bordeaux
Node 24-31: Toulouse
Node 32-39: Sophia
Node 40-47: Nancy

Experiment # 1: In the first experiment, we have set the threshold value at 1 millisecond (1000 microseconds). The group threshold value is set as its twice, i.e. 2 milliseconds. In this experiment, the nodes have communicated randomly with others. Some nodes have communicated more and some have less. The maximum number of nodes that can be contacted by a particular node is fifteen. During the experiment, we have 830 inter-node communication instances. It has a median latency value of 11112 microseconds. The mean latency is 10978 microsecond with a standard deviation of 1608 microsecond.

TABLE I
GROUPS AFTER CONFIGURATION PHASE

Group #	Leader	Nodes Number
0	6	0, 2, 4, 6, 1, 3, 7, 5
1	9	14, 9, 11, 12, 15
2	8	8, 13, 10
3	19	21, 18, 19, 22, 23, 16, 17
4	28	29, 24, 25, 26, 27, 31, 30, 28
5	37	36, 33, 35, 37, 38, 32, 39, 34
6	43	43, 40, 46, 47, 41, 45, 44
7	42	42
8	20	20

The result of configuration phase is shown in Table I. Here we can see that the configuration phase of the algorithm has generated 9 groups. Most of the groups are complete. Nodes in group 1 and group 2 belongs to the same site (i.e. Lyon), but they are not grouped together. There can be many reasons for this type of situation, either the nodes in both groups does not have communicated at all or either some of them have latency higher than threshold. In case of this experiment, none of the nodes in group 1 has done communication with the nodes in group 2. And none of the nodes in group 2 has done communication with the nodes in group 1. So completely lacking in information between these two resulted in the formation of two distinct groups from the same site. In case of node 42 in group 7, it has not done any communication with any node in the system. So it cannot be grouped with any other in this phase. Node 20 has done some communication with 4 other nodes, but the latency between them was more than the threshold. So it also cannot be grouped with any other node. In this experiment, we can see that despite having very less latency information, the algorithm has performed well and has come up with good groups formation.

TABLE II
GROUPS AFTER RECONFIGURATION PHASE

Group #	Leader	Nodes Number
0	6	0, 2, 4, 6, 1, 3, 7, 5
1	9	14, 9, 11, 12, 15
2	8	8, 13, 10
3	19	21, 18, 19, 22, 23, 16, 17, 20
4	28	29, 24, 25, 26, 27, 31, 30, 28
5	37	36, 33, 35, 37, 38, 32, 39, 34
6	43	43, 40, 46, 47, 41, 45, 44, 42

Table II shows the result of the reconfiguration phase. In this phase, both the single node groups (i.e. 7, 8) are processed. Node 42 and node 20 both find a leader node within the latency and they are grouped with them. Node 20 is grouped in group 3 and node 42 is grouped with group 6.

Experiment # 2: We have also tested the algorithm for the case when we do have complete latency information. i.e. *N x N-1* communication is done, where N is the number of nodes. In this case we set the value of threshold at 1 millisecond (1000 microsecond) and group threshold value as its twice. During this experiment, we have N x N-1 (i.e. 2256) inter-node communication instances. It has a median latency value of 11099 microseconds. The mean latency is 11124 microseconds with a standard deviation of 975 microseconds. The algorithm resulted in the 100% correct result after the configuration phase as shown in Table III. In the reconfiguration phase of algorithm, it has nothing to do except to simply remove the empty groups if any. The result of reconfiguration is the same as shown in Table III, as the configuration phase has already produced the correct results.

TABLE III
GROUPS AFTER CONFIGURATION PHASE & RECONFIGURATION PHASE IN BROADCAST COMMUNICATION

Group #	Leader	Nodes Number
0	7	0, 1, 2, 3, 4, 5, 6, 7
1	15	8, 9, 10, 11, 12, 13, 14, 15
2	23	16, 17, 18, 19, 20, 21, 22, 23
3	31	24, 25, 26, 27, 28, 29, 30, 31
4	39	32, 33, 34, 35, 36, 37, 38, 39
5	47	40, 41, 42, 43, 44, 45, 46, 47

Experiment # 3: In the next experiment we tested the broadcast communication scenario by setting the threshold value to statistical based instead of administrator heuristic. The groups that were formed are given below. The groups formed are identical for both mean and median based threshold. Table

TABLE IV
GROUPS AFTER CONFIGURATION PHASE IN BROADCAST COMMUNICATION USING STATISTICAL BASED THRESHOLD

Group #	Leader	Nodes Number
0	26	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 32, 33, 34, 35, 36, 37, 38, 39, 24, 30, 31, 40, 41, 42, 43, 44, 45, 46, 47, 25, 27, 28, 29, 26

IV shows that the algorithm has grouped all the nodes in one group. It is due to the fact that the statistical based threshold gives poor results in this type of situation. Because there is much more communication outside the clusters than the inside. Every node is communicating to 40 nodes out of its cluster which are geographically far away, whereas intra-cluster communication is done with only 7 nodes. Latency to the nodes of the other cluster is much higher, thus resulting in a higher value of mean and median threshold. This mechanism is well suited to the situation when there is more communication within a cluster. Therefore, we recommend using the administrator heuristics to determine the threshold value, except

in the case where automated threshold calculation has to be performed. Table IV is same for reconfiguration phase as there is no change in reconfiguration phase from the configuration phase. All the nodes were grouped in one single group and there was no group with a single node.

Experiment # 4: This experiment is conducted on 4 sites of Grid5000 with 10 nodes on 3 sites and 8 nodes on the last site. The node distribution was like this.
Node 00-09: Grenoble
Node 10-19: Lyon
Node 20-29: Bordeaux
Node 30-37: Toulouse

TABLE V
GROUPS AFTER CONFIGURATION PHASE USING STATISTICAL (MEAN) BASED THRESHOLD

Group #	Leader	Nodes Number
0	1	0, 7, 9, 14, 3, 8, 13, 19, 15, 17, 1, 2, 5, 12, 11, 18, 4, 10, 16
1	20	25, 20, 21, 24, 26, 29
2	28	23, 27, 22, 28
3	37	37, 34, 39
4	30	30
5	6	6
6	32	31, 33, 38, 35, 36, 32

In this experiment, we have set the threshold value equals to mean latency i.e. 7524 microseconds. The group threshold value is set as mean plus standard deviation. In this experiment the nodes has communicated randomly with others. Some nodes have communicated more and some have less. During the experiment, we have 324 inter-node communication instances. It has a median latency value of 8843 microsecond. The mean latency is 7524 microsecond with a standard deviation of 2574 microsecond.

TABLE VI
GROUPS AFTER RECONFIGURATION PHASE USING STATISTICAL (MEAN) BASED THRESHOLD

Group #	Leader	Nodes Number
0	1	0, 7, 9, 14, 3, 8, 13, 19, 15, 17, 1, 2, 5, 12, 11, 18, 4, 10, 16, 6
1	20	25, 20, 21, 24, 26, 29
2	28	23, 27, 22, 28
3	37	37, 34, 39
4	32	31, 33, 38, 35, 36, 32, 30

Initially, the configuration phase has created 7 groups, as shown in Table V. This is quite closer to the real. In this situation, two sites Grenoble and Lyon has lesser inter site latencies as compare to the others. So we can see that the nodes from both these sites are grouped together as the threshold value is quite higher than their average inter-site latency. Nodes 34, 37 and 39 on Toulouse site have done communication between them, and also with the nodes with which they have latency higher than threshold. Thus they are grouped together. After the reconfiguration phase we came up with 5 groups as shown in Table VI. In this phase, node 30 found the group 4 to be grouped with and node 6 found group 0 to be grouped with.

V. CONCLUSIONS

Our proposed algorithm provides a viable solution for grouping of nodes with respect to their inter-node latency. It specifically targets the situation when we do not have complete information about all the inter-node latencies. It works well on the latency information for the nodes who have communicated. It does not introduce any additional network traffic for the algorithm. It is very simple to understand and use. Algorithm complexity O(n^2) is quite low as compare to its counterparts. During the execution of the algorithm a node can be regrouped. Importantly it produces mutually exclusive groups according to the problem statement, whereas most of the existing grouping algorithms produce overlapping groups or communities. It can also produce overlapping groups by making small changes in the algorithm. This can be done by removing the regrouping steps in the configuration phase of the algorithm. There is no false positive, but there exist a few chances of false negative. In case of false negative, a node will fail to join a group with whom it should be grouped.

REFERENCES

[1] P. Hui, E. Yoneki, S. Chan, J. Crowcroft, "Distributed Community Detection in Delay Tolerant Networks", *Proceedings of the 2nd ACM/IEEE international workshop on Mobility in the Evolving Internet Architecture*, Kyoto, Japan, August 2007

[2] E. Yoneki, P. Hui, J. Crowcroft, "Visualizing Community Detection in Opportunistic Networks", *Proceedings of the second ACM workshop on Challenged networks*, Montreal, Canada, September 2007

[3] R. Andersen, F. Chung, K. Lang, "Local graph partitioning using PageRank vectors", *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, California, USA, October 2006

[4] M. E. J. Newman, "Detecting Community Structure in Networks", *Eur. Phys.*, J.B 38, pp. 321330, 2004

[5] J. Leskovec, K. J. Lang, M. W. Mahoney, "Empirical Comparison of Algorithms for Network Community Detection", *Proceedings of the 19th international conference on World wide web*, New York, USA, 2010

[6] C. Pang, F. Shao, R.Sun, S. Li, "Detecting Community Structure in Networks by Propagating Labels of Nodes", *Proceedings of the 2009 Proceedings of the 6th International Symposium on Neural Networks: Advances in Neural Networks* Wuhan, China, May 2009

[7] K. Lang, S. Rao, "A Flow-based Method for Improving the Expansion or Conductance of Graph Cuts", *Proceedings of the 10th International IPCO Conference on Integer Programming and Combinatorial Optimization* , New York, USA, June 2004

[8] A. Clauset, "Finding Local Community Structure in Networks", *Physical Review*, E 72, 026132, 2005

[9] R. J. Gil-Garcia, J. M. Badia-Contelles, A. Pons-Porrata, "A General Framework for Agglomerative Hierarchical Clustering Algorithms", *18th International Conference on Pattern Recognition*, Hong Kong, 2006

[10] H. Zhao, Z. Qi, "Hierarchical Agglomerative Clustering with Ordering Constraint", *Proceedings of 3rd International Conference on Knowledge Discovery and Data Mining*, Phuket, Thailand, June 2010,

[11] Grid 5000, https://www.grid5000.fr/

[12] F. Dabek, R. Cox, F. Kaashoek, R. Morris, "Vivaldi: A Decentralized Network Coordinate System", *Proceedings of the ACM 2004 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '04)*, Portland USA, 2004

[13] P. Sharma, Z. Xu, S. Banerjee, S. Lee, "Estimating Network Proximity and Latency", *ACM SIGCOMM Computer Communication Review*, Vol 36, No. 3, July 2006

[14] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, L. Zhang, "IDMaps: A Global Internet Host Distance Estimation Service", *IEEE/ACM Transactions on Networking*, Vol. 9, No. 5, October 2001