MUGDAD: MULTILEVEL GRAPH DRAWING ALGORITHM IN A DISTRIBUTED ARCHITECTURE

Antoine Hinge, Gaëlle Richer and David Auber
LaBRI, Université de Bordeaux, France

ABSTRACT

This paper presents a multiparadigm force-directed graph drawing algorithm with horizontal scalability on distributed storage clusters. Adaptations of the classical force-directed scheme that function on a distributed environment are presented. This distributed force-directed scheme is associated with a distributed-compatible multilevel approach for a more efficient graph drawing algorithm. MuGDAD is compared in terms of layout quality and speed with other algorithms.

KEYWORDS

Computational geometry, Graph drawing, Distributed computing, Apache Spark, Big Data

1. INTRODUCTION

The Internet provides massive datasets. In the case of social networks or maps, these data are stored on clusters in a distributed manner. Visualization is necessary to help analyze this newly available large amount of data and graphs are the tool of choice to represent networks. Very large graph drawing is needed in community management, for example, to help build a community or fight against harassment. It can also be used in cybersecurity to secure a network. In Big Data visualization, data stored on clusters must be processed there to avoid data transfer from the cluster to a dedicated computer or cluster. Such a transfer is impossible because of the data volume. Thus distributed graph drawing algorithm are needed.

This is the basis of recent graph drawing papers like Arleo (2015) or Hinge (2015). Both papers present an adaptation of force-directed algorithms on a distributed cluster, using Giraph (Avery 2011) and Apache Spark (Zaharia 2010) respectively. These approaches have refined with a multilevel scheme by Arleo et al. (2016). By using Giraph, a multilevel scheme is computed and the force-directed algorithm developed in 2015 is used at each level. This algorithm demonstrate the limitations of a purely force-directed approach, as a multilevel approach performs better with a reduced computation time. Following this approach, MuGDAD propose a distributed multilevel algorithm using Spark and the centroid-directed graph drawing algorithm developed by Hinge and Auber.

Force-directed graph drawing algorithms are a class of graph drawing algorithms based on a physical analogy to describe the interactions between nodes. The idea behind the physical model is to find its energy minimum which gives the position of each node in the graph layout (Eades 1984, Fruchterman 1991, Quinn 1979). Two forces are usually used to model the physical system: attractive forces between connected nodes and repulsive forces between all nodes. Force-directed graph drawing algorithms give aesthetically pleasing layouts but at the cost of high computational complexity. In Fruchterman and Reingold, for graph G=(V,E), the all-pair repulsive forces has $O(|V|^2)$ complexity. Force-directed algorithms usually run O(|V|)

iterations to converge to an acceptable layout. Stress minimization algorithms (Kamada 1989) define an energy function to minimize and use general optimization methods to find the energy minimum, which corresponds to the final layout.

Due to complexity, force-directed graph drawing algorithms do not scale well when used with a global repulsive force. Quigley and Eades (2001) and Hu (2005) chose to limit the repulsive force computation complexity by using quadtrees. Neighbouring vertices in the layout are stored recursively in a tree by dividing the layout space into equal parts and assigning nodes to the corresponding branches. The tree is then used to compute subtree-node repulsive forces, reducing complexity from $O(|V|^2)$ to $O(|V| \log |V|)$, the time

needed to build the quadtree. Hachul and Jünger (2005) and Godiyal *et al.* (2009) limit the repulsive force computation complexity by using the Fast Multipole method. This method also divides the space into equal parts but the contribution of far-away nodes is computed with a multipole expansion. Using the potential field generated by nodes in their quadrants, an approximation of their effect can be computed for the other quadrants with an arbitrary precision. This also reduces complexity to O(|V| log |V|).

Many papers proposed techniques to reduce repulsive forces complexity but the O(|V|) iterations needed to converge still remain too high for large graphs. To reduce algorithmic complexity, force-directed graph drawing algorithms can be associated with a multilevel scheme (Gajer 2001, Hachul 2005, Hu 2005). This technique creates a vertex filtration, selecting decreasing subsets of nodes. From this filtration, increasingly simple graphs are built and laid out successively using the layout obtained at the previous level as initial positions. This process reduces iterations needed to converge: the initial layout is already close to the energy minimum. It can be associated with an intelligent node placement (Gajer 2001) to further improve performances, placing missing nodes at a position close to their final position in the layout. Using this method, FM^3 (Hachul 2005) only computes a fixed number of iterations at each level instead of the O(|V|) iterations usually needed. Thus the global complexity is O(|V| log |V| + |E|).

Parallel force-directed algorithms have also been developed to harness the power of the Graphical Processing Unit (GPU), which gives access to many parallel cores. Force-directed algorithms (Auber 2007, Sharma 2011, Yunis 2012) and multilevel force-directed algorithms (Frishman 2007, Godiyal 2009) have been developed, transforming how force-directed algorithms are usually processed. With the access to parallel cores, the computation times can be improved up to a thousand times on the GPU compared to performances on the CPU only (for example see results of Auber 2007). GPU force-directed algorithms, with their fast running time, contribute to the scalability of graph drawing. More recently, Mi et al. (2016) proposed a multilevel algorithm on the GPU using a clustering approach to draw graphs.

MuGDAD is a new multilevel distributed algorithm with an emphasis on distributed compatibility. It has horizontal scalability for distributed architectures. While scalable, the number of iterations needed to converge with algorithms like Arleo *et al.* or Hinge and Auber was too high. Multilevel algorithms were developed to limit the number of iterations, using an optimized initial layout for each level. With MuGDAD, we present a way to compute and use a multilevel decomposition in a distributed environment. This scheme is also compatible with the approximate repulsive forces described in Hinge (2015). Results from both our distributed and non-distributed versions are presented.

2. MUGDAD: DISTRIBUTED MULTILEVEL DRAWING

$G_0 = (V_0,E_0) \leftarrow$ Initial graph; $i \leftarrow 0$;	**Initialization**	
While($\|V_i\| > 3$)	**Filtration phase**	
$V_{i+1} \leftarrow$ Filtration(V_i)	*Maximal independent set*	*2.1.1*
$i \leftarrow i+1$		
Create graph $G_i = (V_i,E_i)$	*Edge & node collapse*	*2.1.2*
EndWhile		
For(j **from** i **to** 0)	**Layout phase**	
Compute layout of graph G_j (with few iterations)	*Centroid graph drawing*	*2.2*
Place nodes in G_{j-1} according to the layout of G_j	*Layout propagation*	*2.1.3*
EndFor		

Algorithm 1. General structure of the MuGDAD algorithm

Multilevel force-directed algorithms use recursive simplifications of the input graph and a force-directed approach to draw each graph. This is done in two steps: a filtration step to recursively simplify the input graph and a layout step where graphs are drawn iteratively with a force-directed algorithm. Starting with the smallest graph, layouts are computed until convergence. The final layout is used as the initial layout for the next step, adding missing nodes. Thus, initial layouts are close to the energy minimum and only a few iterations of the force-directed algorithm are needed to converge. This is summarized in Algorithm 1. In this section, the distributed multilevel filtration of MuGDAD is presented. Then, the force-directed model is described.

2.1 Distributed Multilevel Decomposition

In MuGDAD, the multilevel decomposition is done by computing a Maximal Independent Set (MIS). The neighbourhood of nodes selected this way is clustered and positions of nodes are propagated using an intelligent placement after the graph drawing algorithm has been applied. Thus, the decomposition is similar to FM^3 (Hachul 2005) but clusters are only formed at distance one around sun nodes.

2.1.1 Distributed Maximal Independent Set

In FM^3 (Hachul 2005), computing the multilevel decomposition is done using a sequential algorithm returning a MIS with a distance of at least three between sun nodes. Nodes are then clustered into solar systems with no overlap. Parallel algorithms of MIS filtration exist and can be directly transposed in a distributed paradigm.

Luby (1985) describes several parallel algorithms to find a MIS. Instead of choosing one vertex at a time from the set of remaining vertices, it uses a global ranking scheme (a permutation of the nodes) and select locally minimal vertices in regards to this ranking. This process is repeated until each node is either selected or is a neighbour to a selected node. Of the several variants presented in the paper (Luby 1985), MuGDAD implements the version with the lowest expected complexity of O(log |V|). This version generates more random numbers on average which is not a problem in our situation.

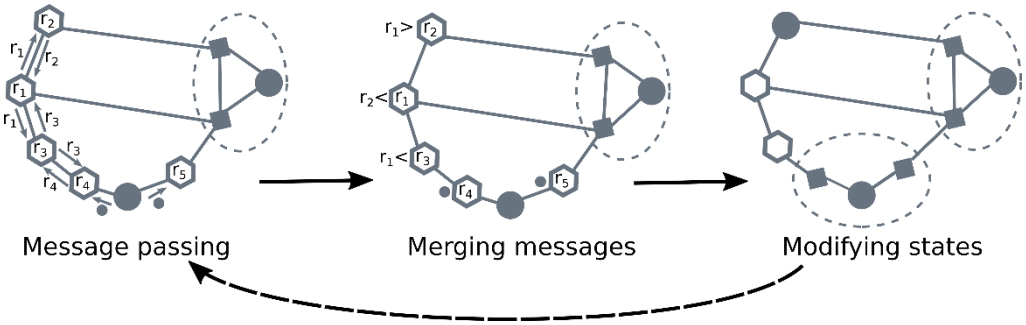

Figure 1. Pregel implementation of MIS and clustering algorithm. (*Left*) Messages are exchanged between unselected nodes. (*Right*) Messages received are merged. (*Bottom*) Nodes states are modified according to messages received

In MuGDAD, a Bulk Synchronous Parallel (BSP) model is used. This model defines supersteps during which tasks are executed for each node in the graph in parallel. Tasks include sending messages that will be read at the next superstep to another node, reading messages received from the previous superstep, as well as changing current node state.

To create clusters similar to the solar systems of FM^3, nodes track their state (unassigned, selected or clustered with a selected node) and the id of their cluster's sun node. In the message passing step, messages are exchanged between unselected nodes, containing the rank in the permutation of their neighbours. Selected nodes also send messages to their unselected neighbours. In the merging state, two cases are considered. Either the node has a selected neighbour and only this message is kept or it does not and the minimal rank of the unselected neighbours is computed. When modifying the node states, three cases can be considered. If a message has been received from a selected node and the node becomes clustered with a selected node. If not, if the reduced message rank is greater than its own rank it becomes selected. In the last case (message rank is lesser), the node does not change state. Then, new ranks are drawn and a new superstep begins.

2.1.2 Edge and Node Collapse Step

Clusters found in the previous step are collapsed into meta-nodes. New meta-edges are created between adjacent clusters. Weights of the meta-edges are computed. MuGDAD uses these weights in the layout phase as optimal edge length even though they do not reflect accurately the graph distance between nodes. These weights are also used to place yet unplaced nodes in the propagation phase, as described in next section.

In FM^3 (Hachul 2005), the weight of meta-edges is computed in two steps. First, the sum of weights of the shortest paths connecting solar systems is computed. Then, results are aggregated using a mean function. This proved to be inefficient in MuGDAD: the edge weight growth between levels is multiplied at each level

by the mean graph distance between neighbour solar systems, due to the first step of the aggregation. In MuGDAD, this exponential growth is problematic because more levels are computed when computing a layout. Indeed, the MIS is done at distance two in MuGDAD instead of three in FM^3 which results in more levels being computed.

To avoid this exponential growth, the maximal edge weight of edges in the shortest path is taken. For the contribution of the other edges, a random value is drawn following a uniform law between one and the current level number. This heuristic ensures both a continuity of edge weight between levels (strictly increasing) and avoids the effect of an exponential growth.

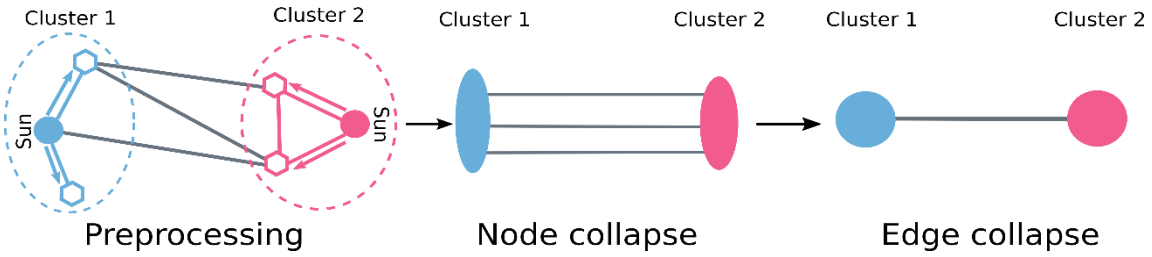

Figure 2. Edge collapse for two MuGDAD clusters. (*Left*) Pregel iteration: transmitting edge weights to set up map operation. (*Middle*) Map operation: creating meta-edges between clusters with total weight computation. (*Right*) Reduce operation: computing the mean weight of meta-edges between clusters

For our distributed implementation, we proceeded in three steps, see Figure 2. Let $G_i = (V_i,E_i)$ be the graph generated by our filtration at level i.

In the preprocessing step, the edge weight computation is set up. Information required to compute the edge weight is transmitted to planet nodes of clusters by sun nodes. Messages are sent along edges connecting sun and planet nodes of the clusters and contain the edge weight. These data are necessary to compute the weight of paths connecting clusters in the next step. This step is done using the Pregel functions implemented in GraphX. Up to $|E_i|$ messages are transmitted at level i of the filtration and that equality holds when there are no edges between clusters of the filtration.

The second step of this process is a map operation. This operation outputs meta-edges with endpoints set to the sun nodes of their respective cluster. Meta-edges are created by mapping edges connecting different clusters. There is up to $0.5\ |E_i|$ edges between clusters at level i and that equality holds when edges between clusters only connect planet and sun nodes in different clusters and no edges connect planet nodes inside the clusters. This can be proved by noticing that meta-edges connecting planet nodes represent three edges and that meta-edges connecting sun and planet nodes only represent two. This map operation is divided over several mappers, each processing part of the edges. Thus, total complexity of this step is divided by the number of mappers.

The last step is a reduce operation. Meta-edges duplicated between clusters are reduced into one with a mean function. The reduce operation is applied on the $O(|E_i|)$ meta-edges obtained at level i in the previous map operations. The mean total weight is computed in a distributed manner using the mean design pattern, as seen in Miner and Shook (2012) for example. This step is also distributed over several reducers.

2.1.3 Propagating the Layout

Once the layout of G_i is computed, positions of its nodes can be propagated to G_{i-1}. First, the position of sun nodes is propagated. Then, nodes not positioned (i.e. nodes in $V_{i-1} \backslash V_i$) are placed at the barycenter of placed nodes with desired edge length used as weight. If unplaced nodes are on several of these paths, their final position is at the barycenter of all computed positions. This process is explained with more details in FM^3 (Hachul 2005).

Nodes unplaced after this step are placed in a circle around their respective sun nodes. Indeed, these leftover nodes are not in the paths between solar systems and are not sun nodes.

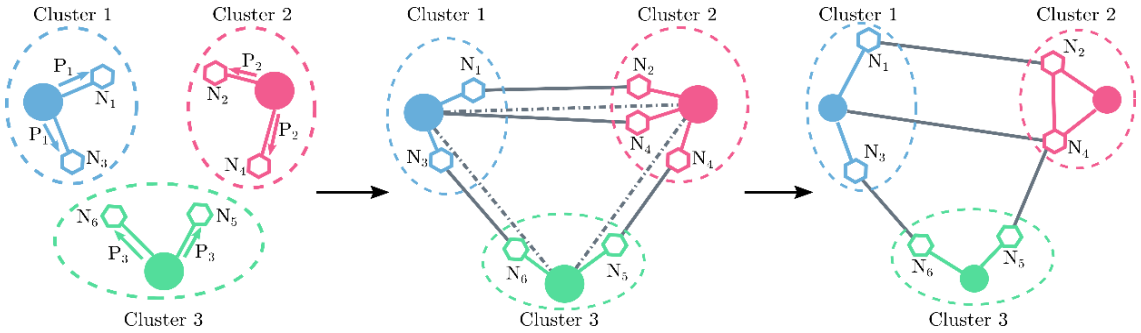

Figure 3. Propagating the layout for three MuGDAD clusters. *(Left)* Pregel iteration: transmitting sun node positions to set up map operation. *(Middle)* Map operation: positioning nodes on the segment connecting clusters. *(Right)* Reduce operation: computing the mean position of nodes

In distributed, propagating positions of sun nodes from graph G_i to graph G_{i-1} is done using a distributed join operation to match node positions between levels. Then, planet nodes are placed in three steps, see Figure 3. The three steps (preprocessing, map and reduce step) necessary to compute the initial placement of planet nodes are very similar to the ones necessary to compute meta-edges and their weight in the edge collapse step described in section 2.1.2. In the preprocessing step, messages are sent along edges connecting sun and planet nodes of the clusters to set up node positioning. They contain the edge weight connecting the two nodes as well as the position of the sun node. The map operation outputs nodes' positions for planet nodes. Nodes placed this way are created by mapping edges connecting two different clusters. Nodes' positions are computed using sun nodes' positions and the weight of the edges. With the reduce step, duplicated positions for planet nodes are reduced into one with a mean function, using a mean design pattern, see Miner and Shook (2012).

For nodes still unplaced, a single Pregel iteration is applied to send unplaced nodes the position of their respective sun node. Then, an angle is drawn at random for each of these nodes and they are placed at a constant radius around their sun node using this angle.

2.2 Distributed Force-Directed Graph Drawing

The initial layout is obtained by combining the layout of previous levels and the intelligent placement. Then, at each level of our graph filtration, a force-directed algorithm is applied to optimize the layout. In Fruchterman and Reingold (1991), two kind of forces are applied iteratively on each node to compute their displacement: attractive forces and repulsive forces.

Attractive forces, between adjacent nodes, can be computed directly in a distributed paradigm using edges of the graph, as shown in Hinge and Auber (2015) and Arleo *et al.* (2015). Regarding repulsive forces, Arleo *et al.* (2015) chose to limit their computation to nodes at graph distance k or less. This approach stems from the observation that in the final layout, nodes close to each other in the graph should also be close in the drawing. Hinge and Auber strike a compromise between locally optimal and globally optimal graph layout using centroids. Using node positions in the layout, clusters are formed and used to compute approximate repulsive forces. Far away nodes contribute as a cluster of nodes instead of contributing individually, generating $O(|V|)$ repulsive forces, instead of $O(|V|^2)$ for the all-pair repulsive forces computations.

For MuGDAD, we chose to implement centroid graph drawing. Our choice is motivated by the fact that this approach is compatible with the multilevel scheme, as explained in this section.

2.2.1 Centroid Graph Drawing and Multilevel Scheme

In Hinge and Auber, node clusters are computed using a process similar to k-means. K clusters are represented by centroids, whose positions are at the weighted position of all nodes in the cluster. Clusters are assigned every node closest to their centroid than any other centroid. Once nodes are assigned to a given cluster, the position of centroids are updated. Centroids are then used to compute approximate repulsive forces and the position of nodes is updated. This process is repeated each time the repulsive forces are computed. The advantage of using centroid graph drawing with a multilevel scheme is twofold.

First, centroid graph drawing is more efficient when the clusters are well defined. In their paper, Hinge and Auber start from a random layout to converge to the final layout. Clusters in the first steps of the force-directed method are not well defined which means that approximate repulsive forces are less accurate. In MuGDAD, the multilevel approach is combined with an intelligent placement and layouts at every level are already close to the final layout which improves the accuracy of centroid-repulsive forces greatly.

Secondly, centroids can be kept between levels to optimize repulsive forces. The clustering process of centroid graph drawing, which is similar to k-means, converges to a better solution with a good set of initial centroids. Algorithms like scalable k-means++ (Bahmani 2012) have shown that with a proper initialization the k-means algorithm converges to a better solution. Keeping centroids between levels ensures a good initialization to the k-means problem. Thus, resulting centroid-repulsive forces are more accurate.

2.2.2 Multi-Paradigm Graph Drawing

Computation in a distributed environment gives worst performances than a sequential algorithm when data is too small: the communication and synchronization overhead is way costlier than the gain obtained from the computation distribution and parallelism. Using a multilevel algorithm, the massive graph drawing problem is simplified until it can be processed on a single machine. At this point, computation is switched to this machine using the sequential algorithm.

MuGDAD is a good candidate as it functions similarly in a distributed and in a sequential environment. Using centroid graph drawing as our force-directed scheme in the two paradigms allows us to keep centroids between levels: when the computation is done in the sequential approach, the layout and clusters are transmitted to the distributed cluster that resumes computation where the sequential paradigm left it.

2.2.3 Adaptations to Centroid Graph Drawing

Minor changes are done to centroid graph drawing (Hinge 2015) to reflect edge weights. Edge weights represent theoretic distances between nodes in a multilevel scheme. Attractive forces are proportional to $d_{att} - d_0$ with d_{att} the distance between nodes and d_0 a nominal distance parameter set to the edge weight. Repulsive forces are multiplied by the average edge length in the level. This way, repulsive forces have as much impact as attractive forces.

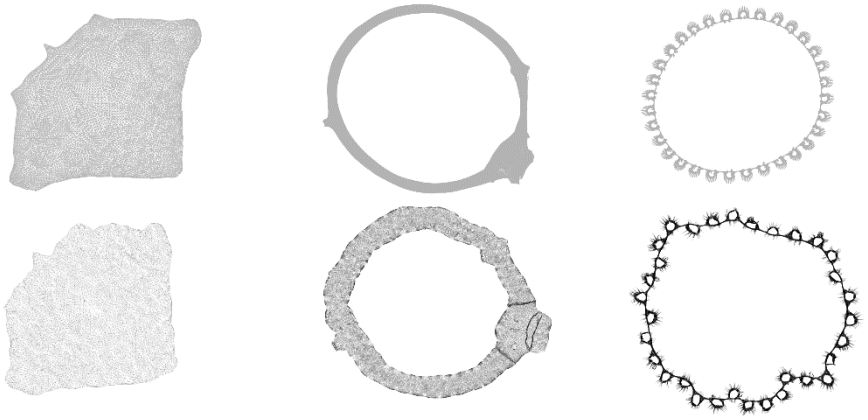

Figure 4. Layouts of crack (Left), fe_pwt (Middle) and finan512 (Right). (Top) Layouts obtained with FM3. (Bottom) Layouts obtained with MuGDAD

3. EXPERIMENTAL RESULTS

Table 1. Computation times (expressed in seconds)

Dataset	V	E	FM^3	MuGDAD	MuGDAD (Dist.)	MultiGila
crack	10,240	30,380	4.54	1.19	23.3	-
fe_pwt	36,463	144,794	15.69	2.77	54.2	-
finan512	74,752	261,120	36.97	12.93	133	-
Amazon0302	262,111	899,792	-	-	538	1577
ASIC_320	321,523	515,300	-	-	1124	1102
Com-DBLP	317,080	1,049,666	-	-	-	2366
Com-Amazon	334,863	925,872	-	-	4332	2242
Roadnet-PA	1,087,562	1,541,514	-	-	1129	2241

3.1 Implementation

MuGDAD has been implemented in two different versions: one developed to run sequentially in C++ and the other developed using Spark (Zaharia 2010). In Spark (Zaharia 2010), and more precisely in the graph library called GraphX (Xin 2013), an implementation of Pregel (Malewicz 2010) is available.

Furthermore, as described in section 2.2.2, the Spark implementation of MuGDAD relies on the C++ implementation to draw graphs too small to benefit from distribution. To do this, the Spark implementation relies on the Java Native Interface (JNI), allowing to run a native program through a Java interface.

3.2 Results

In Figure 4, layouts for FM^3 and MuGDAD are compared for three datasets (crack, fe_pwt and finan512). For crack and fe_pwt, MuGDAD layouts are comparable to the ones obtained with FM^3. Both the global structure and the lattice pattern are present. For finan512, the general structure is present in MuGDAD but the finer details of the structures are not clearly visible. More generally, the centroid repulsive force does not seem to be able to manage details at very fine scales.

Table 1 contains computation times for layouts shown in Figure 3. For the smaller graphs (upper part of Table 1), the CPU implementation of MuGDAD runs more quickly than FM^3, as can be expected since centroid graph drawing has a complexity of O(|V|). The distributed version of MuGDAD performs relatively quickly considering that this data is too small to benefit from data distribution. This is due to the JNI implementation that computes the layout on a single machine.

For the larger graphs (lower part of Table 1), our implementation performs better than MultiGila for a series of graphs and significantly worse for some others. In the case of Com-DBLP, MuGDAD was stopped after 12h of computation. MultiGila uses a distributed clustering similar to the one in FM^3, a Maximal Independent Set at distance three. As a result, in some cases, our multilevel implementation takes many more levels than the one implemented in MultiGila. This is reflected on the computation times.

Algorithm comparison was conducted with a Core i7-4710HQ for the CPU algorithms. The implementation used for FM^3 is the OGDF implementation. Times from the MultiGila algorithm are taken from Arleo (2016). They are obtained on a cluster comparable to the one used for our results. Our distributed infrastructure is composed of 16 computers (non virtualized) with 64GB of RAM, 2x6 hyperthreaded cores at 2.1GHz and 2 hard drives of 1 TB each. Computers are linked by a 1Gb/sec network infrastructure.

4. CONCLUSION

MuGDAD is a distributed graph drawing algorithm that combines centroid repulsive forces with a multilevel scheme to draw distributed graphs efficiently. The scalability is proved in a distributed environment. Experimental results show that layouts are obtained quickly compared with methods like FM^3 but fewer details can be seen, due to the use of a distributed-compatible approximate repulsive force.

ISBN: 9978-989-8533-66-1 © 2017

ACKNOWLEDGEMENT

This work has been carried out as part of “REQUEST” project supported by the French “Investissement d’Avenir” Program (Big Data - Cloud Computing topic - PIA O18062-645401).

REFERENCES

Arleo A. et al, 2015. A million edge drawing for a fistful of dollars. *In Graph Drawing and Network Visualization*, pp 44-51.

Arleo, A. et al., 2016. A Distributed Multilevel Force-Directed Algorithm. *International Symposium on Graph Drawing and Network Visualization*, pp 3-17.

Auber D. and Chriricota Y., 2007. Improved efficiency of spring embedders: Taking advantage of gpu programming. *In Visualization, Imaging, and Image Processing*, Vol. 2007, pp 169-175.

Avery C., 2011. Giraph: Large-scale graph processing infrastructure on hadoop. *Proceedings of the Hadoop Summit*.

Bahmani B. et al., 2012. Scalable k-means++. *Proceedings of the VLDB Endowment*, Vol. 5, No 7, pp 622-633.

Peter Eades, 1984. A heuristic for graph drawing. *Congressus numerantium*, Vol. 42, pp 146-160.

Frishman Y. and Tal A., 2007. Multi-level graph layout on the GPU. *Visualization and Computer Graphics, IEEE Transactions on*, Vol. 13, No.6, pp 1310-1319.

Fruchterman T. and Reingold M, 1991. Graph drawing by force-directed placement. *Softw., Pract. Exper.*, Vol. 21, No. 11, pp 1129-1164.

Gajer P. and Kobourov S., 2001. Grip: Graph drawing with intelligent placement. *In Graph Drawing*, pp 222-228. Springer.

Godiyal A. et al., 2009. Rapid multipole graph drawing on the GPU. *In Graph Drawing* , pp 90-101. Springer.

Hachul S. and Jünger M., 2005. Drawing large graphs with a potential-field-based multilevel algorithm. *In Graph Drawing*, pp 285-295. Springer.

Hinge A. and Auber D., 2015. Distributed graph layout with Spark. *In Information Visualisation (iV), 2015 19th International Conference on*, pages 271-276.

Hu Y., 2005. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, Vol. 10, No.1, pp 37-71.

Kamada T. and Kawai S., 1989. An algorithm for drawing general undirected graphs. *Information processing letters*, Vol. 31, No 1, pp 7-15.

Luby M., 1985. A simple parallel algorithm for the maximal independent set problem. *In Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pp 1-10. ACM.

Malewicz G. et al., 2010. Pregel: a system for large-scale graph processing. I*n Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pp 135-146. ACM.

Mi, P. et al., 2016. Interactive Graph Layout of a Million Nodes. *Informatics*, Vol. 3, No. 4, p 23.

Miner, D. and Shook, A, 2012. MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems. O'Reilly Media, Inc.

Quigley A. and Eades P, 2001. Fade: Graph drawing, clustering, and visual abstraction. *In Graph Drawing*, pp 197-210. Springer.

Quinn N. et al., 1979. A forced directed component placement procedure for printed circuit boards. *Circuits and Systems, IEEE Transactions on*, Vol. 26, No. 6, pp 377-388.

Sharma P. et al., 2011. Speeding up network layout and centrality measures for social computing goals. *In Social Computing, Behavioral-Cultural Modeling and Prediction*, pp 244-251. Springer.

Xin R. et al., 2013. Graphx: A resilient distributed graph system on spark. *In First International Workshop on Graph Data Management Experiences and Systems* , page 2. ACM.

Yunis E. et al., 2012. Scalable force directed graph layout algorithms using fast multipole methods. *In Parallel and Distributed Computing (ISPDC), 2012 11th International Symposium on*, pp 180-187. IEEE.

Zaharia M. et al., 2010. Spark: cluster computing with working sets. *In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, Vol 10, p 10.