UNIVERSITY OF AMSTERDAM

Virtual infrastructure partitioning and provisioning under nearly real-time constraints

Faculty of Physics, Mathematics and Informatics
MSc Security and Network Engineering
Research Project 1

Author: Andrey Afanasyev
Andrey.Afanasyev@os3.nl

Supervisors:
dr. Z. (Zhiming) Zhao, dr. ir. A. (Arie) Taal, Mr H. (Huan) Zhou MSc[1]
[1]{z.zhao, a.taal, h.zhou}@uva.nl

February 11, 2018

Abstract

Distributed or cloud application became a very common in today world. There are situations when virtual infrastructure used by application need to be partitioned in order to reduce costs and decrease a recovering time after failures. Further, partitions of the divided graph might be relocated across different data centers following a set of practical requirements and constraints.

A virtual infrastructure can be reduced tot a graph which need to be partitioned. A graph partition problem (GPP) is a well-known non-deterministic polynomial-time (NP-hard) complexity problem. There are many graph partition existing algorithms and some of them are implemented in software tools[1].

This research is focusing on investigating literature proposed constraints provided from developer, application and data center perspective in relation to virtual infrastructure. During research a partitioning METIS as a software tool was selected.A prototype based on Python wrapper of tool was invented in order to measure graph partitioning elapsed time of each implemented algorithm.

During experiment was used a 10, 100 and 1000 nodes graphs of 3 degree each with 15, 150 and 1500 edges retrospectively. A Fiduccia-Mattheyse algorithm with One-sided node-based refinement perform better then other algorithms for all types of defined graphs, but not significantly compare to two-sided node-based refinement. However, it was observed that this combination is more computational intensive compare to Greedy algorithm or even classical Fiduccia-Mattheyse.

Acknowledgements

I would like to thank my supervisors Dr. Zhiming Zhao and Dr. ir. Arie Taal for their support and guidance throughout this challenging research project. In addition, I would like to thank Mr H. (Huan) Zhou MSc for refinement of the research scope and aspects by providing research related information to get more focused results.

Contents

1 Introduction

In the epoch of ubiquitous use of cloud applications on client devices when information can be passed from different sources of data to cloud applications, developers of cloud applications may need an opposite activity. A situation application nodes connected in a virtual infrastructure should be separated between cloud data centers to provide: application services to the end users, decrease recovering time from failures and fulfill underlying constraints of cloud application, providers and other components. In order to optimize systems performances and costs, developers need to plan the partitioning virtual infrastructure infrastructure directly according their own requirements and understanding of their applications.

Cloud applications often consist of nodes which are connected in a virtual infrastructure located by one provider and one data center. According new trends and research direction developers are beneficial in spreading application infrastructure between different data centers and even providers[2]. Application's virtual infrastructure might be represented as a graph, where vertices are nodes and edges are link lines exchanging data.

Represented graph need to be partitioned according to the developer requirements. This leads to well known graph partitioning problem. Recent systematic reviews show that the graph partitioning problem (GPP) is in focus of many researchers[1]. Due to the high demand in science and IT industry this field of mathematics is in active development where different graph partitioning algorithms exist.

This research was conducted in the period from 8 January 2018 till 11 of February 2018. The goal of this paper is to categorize cloud application developer requirements related to virtual infrastructure proposed in literature. Select a software tool for partitioning a virtual infrastructure under near real-time constraints. Finally, provide an experiment by which set of graphs are partitioned in order to profile software tool implemented algorithms on the basis of the elapsed time efficiency.

Additionally, For this research was assumed that a virtual infrastructure is represented as a weighted undirected graphs. Measurements are done in terms of comparing algorithms elapsed time on defined sets of graphs. Only edge cutting algorithms are in scope of this research. No new or existing algorithms are created or coded during this stage.

Based on the aforementioned, the following main research question is defined:

How profiling of the graph partitioning algorithms might improve fractioning of a virtual infrastructure?

- *Which user and application constrains might be applied during profiling of the graph partitioning algorithms?*
- *What key algorithms are implemented for evaluation?*
- *Which graph partitioning algorithms are most desirable in the specific scenario's?*

The paper structured is as follows. The Section 2, reviews a the state of the art of the collected developers virtual infrastructure partitioning requirements and selecting a right software tool. Next, the section 3 contains a discussion about proposed Virtual Infrastructure Graph Partitioner script. Section 4 describes the experimental setup and gathered the results. Section 5 discuss the results of the experiment and answers research questions. Finally, section 6 concludes the paper and light ups the way for future research.

2 State of the art

In this section, firstly the most important developer and graph partitioning software tool requirements gathered from literature are specified. Further, open source partitioning tools proposed in literature are compared. Finally, a partitioning tool with set of implemented algorithms will be selected to use in experiment setup.

2.1 Constrains landscape

Out of the defined research questions, trends and research directions in cloud proposed in literature [2][3] and mentioned in Section 1 a several functional requirements might be enumerated. Those requirements are spread across three parties: Developer of an application which has interest using a saving costs virtual infrastructure, a developed application which heavy rely on virtual infrastructure, and data center which provides a facilities to deploy a virtual infrastructure.

Developer:

- Costs. To reduce virtual infrastructure costs user might need to segment his infrastructure in multiple parts across different locations. It might be based on price of the exchanged data between nodes.
- Availability. In order to provide sufficient level of services to the end user infrastructure need to perform according developer defined requirements.
- Failures mitigation. To mitigate application unavailability and related services a fast reliable recovery from failures is required.
- Geo-location. To spread virtual infrastructure between different geographical locations in order to provide sufficient service to the end user or recovery from failures.

Application:

- Availability. In order to provide services to the end user an application need to have sufficient virtual infrastructure resources in response to elasticity requirements under specific sitation.
- Resiliency. Being able to adapt and provide services under stress or faults in order to avoid failures a virtual infrastructure as whole or partly need sufficiently fast recovers form failures.

Data center:

- Availability. A physical infrastructure should operate sufficient in order to support application required virtual infrastructure.
- Maintainability. Data centers need to have a time space for improving or supporting their physical infrastructure. This may lead to unexpected failures.

Therefore, finding a balance between application availability, mitigation of the virtual infrastructure components failures and geographical application spreading might save developers exchanging data costs and improve resiliency. To select the graph partitioning tool which might used to partition virtual infrastructure represented as a graph to seek the balance a following requirements need to be to fulfill:

- Open-source. Community need to have possibility to contribute and to be easy involved to development of the tool in order to purpose new features and fix bugs to keep application relevant according new trends in virtual infrastructure.
- Maintainability. The virtual infrastructure evolves rapidly a new features are added regularly in that sense tool must have possibility to be easy maintained.
- Usability. In order to reduce complexity tool needs to have a proper, understandable programming interface to interact with virtual infrastructure represented as a graph. Interacting via Python - a cross-platform general purpose programming language was chosen as a criteria.
- Cross-platform. Due to fact that virtual infrastructure may consist of multiple platforms it is urgent that tool may run on multiple platforms.

2.2 Tool selection

Over past two decades a multiple graph partitioning tools were developed as open and closed source. The paper of Buluç et al [1] summarize and discuss old and new graph partitioning software tools.

Table 1: Comparison of proposed graph partitioning tools and defined requirements

Tool	License	Latest stable release	Programming language	Cross-Platform	Python wrapper
Chaco [4]	GPL	v2.0, 1998	ANSI C	No (Unix-like)	No
KaHIP[5]	GPLv2	v2.00, 2017	C++	No (Unix-like)	Yes
KaHyPar-CA [6]	GPLv3	2017	C++	Yes(Unix-like, Windows)	No
METIS [7]	Apache 2.0	v5.1.0, 2013	ANSI C	Yes(Unix-like, Windows)	Yes (multiple)
Mondriaan [8]	LGPL	v4.2, 2017	C	No (Unix-like)	No
PaToH [9]	BSD	v3.2, 2011	C	No (Unix-like)	No
Zoltan [10]	LGPL	v3.8, 2016	C	No (Unix-like)	No

Table 1 compares the most important requirements defined in the section 2.1 and the tools proposed and described in literature in the last section. Among all other tools the METIS version 5.1.0 released in 2013 fulfill previously mentioned requirements. Additionally, tool is available as a package in Debian Stretch repository[11]. This make possible to automate deployment and maintainability of the tool. There is no windows package available for download, but according a BUILD-Windows.txt file presented in archive a source code can be compiled once using a Visual Studio C++ compiler.

METIS supports multilevel recursive-bisection, multilevel k-way partitioning schemes. Might be used for both ***edge cutting*** and node clustering. METIS uses a classical Fiduccia-Mattheyses(fm) algorithm with optional One-sided node-based(sep1sided), which is default and Two-sided node-based (sep2sided) refinement components. Those components developed by a team behind METIS. Additionally, a Greedy algorithm can be used too.

However, there are three different Python wrappers which might be used by prototype or in real life environment.

Table 2: Comparison of METIS's python wrappers

Wrapper	License	Python version	Latest stable release	Cross-Platform	Functionality
NetworkX-METIS[12]	Apache 2.0	2.7 and 3,2	1.0, 2015	Yes	Limited
METIS for Python[13]	MIT	2 and 3	0.2a4, 2018	Yes	Full
PyMetis[14]	MIT	2 and 3	2016.2, 2016	Yes	Basic

Table 2 illustrates the differences between available wrappers. Despite fact that METIS for Python is still in development it provides a best functionality among all available wrappers. Further, in the prototyping and experimenting stage a METIS for Python will be used.

3 Virtual Infrastructure Graph Partitioner

In this section the architecture of the prototype of the Virtual Infrastructure Partitioner script will be described. The relation between developer and partitioning script and how prototype is build.

3.1 Overview

Figure 1 illustrates the process of generating graph or getting it from a file, getting developer requirements file which are all parts of Input component. Graph partitioning and visualizing are parts of Processing component. Visualizing is use only for testing purpose and might not needed in production. Processing component produce a measurements file, visualizing of original and partitioned graphs. Measurements file stores elapsed time of each algorithm used during experiment. It is useful for further analyzing. Analyzing component process process measurements and delivers clear representative plots which are useful for this research.

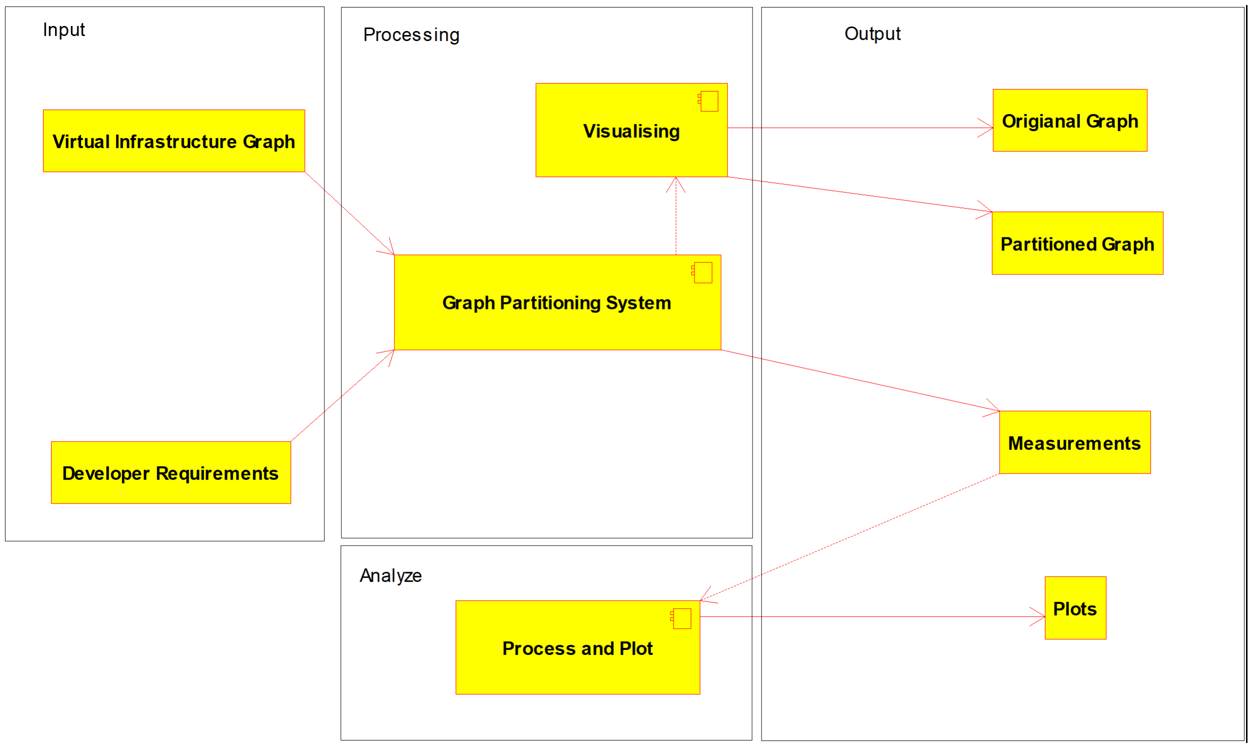


Figure 1: Architecture of the Virtual Infrastructure Graph Partitioner

3.2 Prototype

To demonstrate the benefit of the design a prototype with experimental purpose was build. The prototype is a script tool written in Python v3.5.2. Besides aforementioned METIS for Python module was used another modules:

- networkx: in order to convert or generate graph in to METIS readable format,
- os and csv: to manage measurements stored in a csv file,
- time: to measure elapsed time of graph partitioning per algorithm.
- matplotlib: to analyze and visualize measurements in form of plots,
- numpy: to simplify data ordering and some calculations.

Current prototype may read and write Gaphviz(*.dot) and GraphML(*.xml) files. This allows to use a real virtual infrastructure graphs in future. During visualization process a PNG (*png) files are generated. Results from the Analyze architectural component among with Output component were actively used during experiments discussed in following section. The developed code is publically available and can be accessed on Github[1].

4 Experiment

In order to illustrate benefit of chosen graph partitioning tool and related python wrapper for Virtual Infrastructure Graph Partitioner a several experiments were conducted. These experiments aim to answer following question:

Which graph partitioning algorithms are most desirable in the specific scenario's?

The goal of the first question is to measure if there is a difference between graphs partitioning algorithms in elapsed time. During experiments a certain assumptions were taken in mind.

- A graph partitioning elapsed time which takes less then 10 millisecond considered as a real-time partitioning.
- There are three different size (10, 100, 1000 nodes) regular graphs in degree of 3 were generated.Those graphs represent different scenario's.
- Partitioning done using a balanced mode with edge-cuts minimizing.

Measurements of CPU's and Memory are desirable, but doe to time limitations and required amount of work are out of the scope.

4.1 Environment

All experiments were conducted on the physical machine. Table 3 illustrates hardware components of used machine. An fully updated Ubuntu Server 16.04.3 operating system in default configuration was installed and used during experiments.

Table 3: The hardware components of a physical machine used during experiment

Item	Description
CPU	AMD A10-7870K Radeon R7, 12 Compute Cores 4C+8G
Memory	7106MiB System memory, DDR3, PC3-10700 (667 MHz)
Mainboard	ASRock FM2A88X-ITX+
Storage	Crucial_CT512M550SSD3, 476,9GiB
Network	1x 1Gb/s

Over the years a great performance improvements are done in Hypervisors software. Based on that knowledge an assumption was taken in account. Assumption consist of statement that a fluctuations between tests on virtual machine with the same configuration as a physical machine should be not significant and results should be similar to results gathered from a physical machine.

4.2 Description

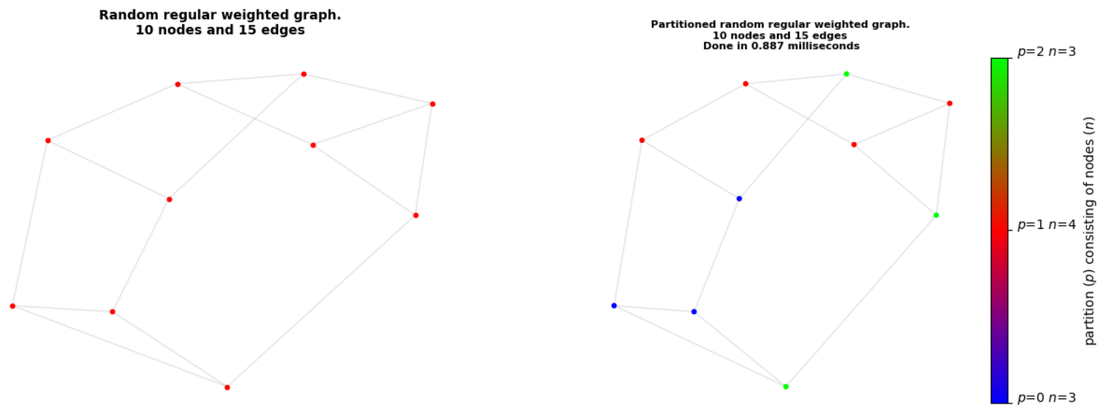
For experiment were chosen three random undirected graphs of different sizes. Table 4 represents characteristics of those graphs. Each of this graph represent a different situation.

[1]https://github.com/aafanasyev/RP1-VIGP

Table 4: Graphs characteristics used during experiments

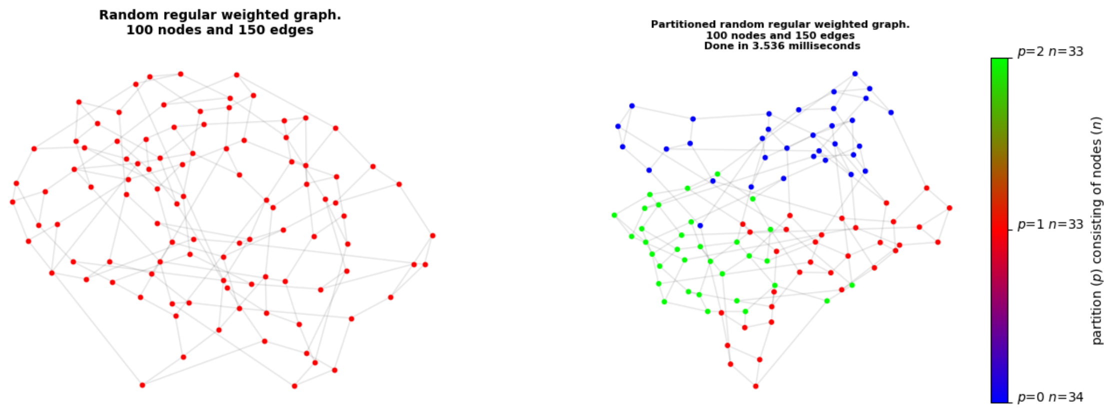
Id	Nodes	Edges	Maximum Degree
1	10	15	3
2	100	150	3
3	1000	1500	3

Each classical algorithm and its refinement component will partition each original unpartioned graph 1000 times in 3 partitions. Partitioning done using balanced mode. After each iteration a pause of 1 second taken in order to save results. In order to get statistical data an elapsed time per algorithm per each iteration were produced. This data was saved in $n.csv$ file. Where n is a quantity of nodes per graph.


Random regular weighted graph.
10 nodes and 15 edges
Partitioned random regular weighted graph.
10 nodes and 15 edges
Done in 0.887 milliseconds
p=2 n=3
p=1 n=4
p=0 n=3
partition (p) consisting of nodes (n)

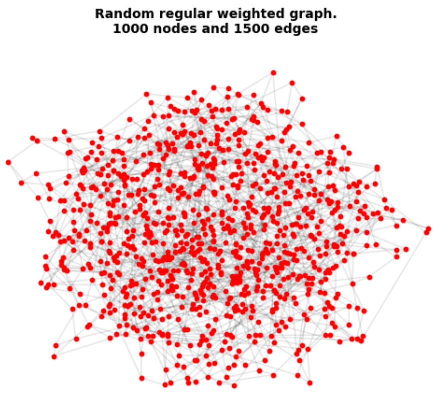
(a) Original (b) Partitioned

Figure 2: A 10 nodes graph.

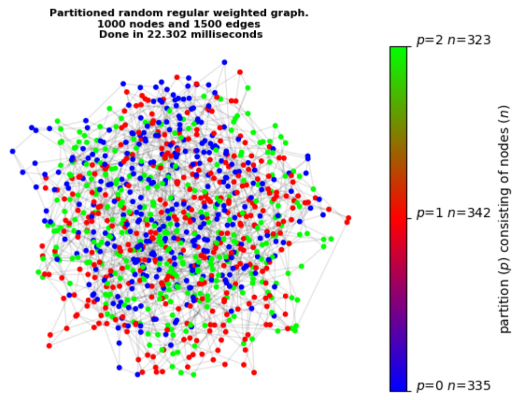

Random regular weighted graph.
100 nodes and 150 edges
Partitioned random regular weighted graph.
100 nodes and 150 edges
Done in 3.536 milliseconds
p=2 n=33
p=1 n=33
p=0 n=34
partition (p) consisting of nodes (n)

(a) Original (b) Partitioned

Figure 3: A 100 nodes graph.



(a) Original


Partitioned random regular weighted graph.
1000 nodes and 1500 edges
Done in 22.302 milliseconds
p=2 n=323
p=1 n=342
p=0 n=335
partition (p) consisting of nodes (n)

(b) Partitioned

Figure 4: A 1000 nodes graph.

Figures 4.2, 4.2 and 4.2 illustrate originally generated graphs and partitioned graphs conducted during one iteration of classical Fiduccia-Mattheyses algorithm(fm) without any sided refinement. The matplotlib Python module may represent original and partitioned graph visually different but both graphs stays isomorphic.

5 Discussion

In this section the results of the experiment are discussed.

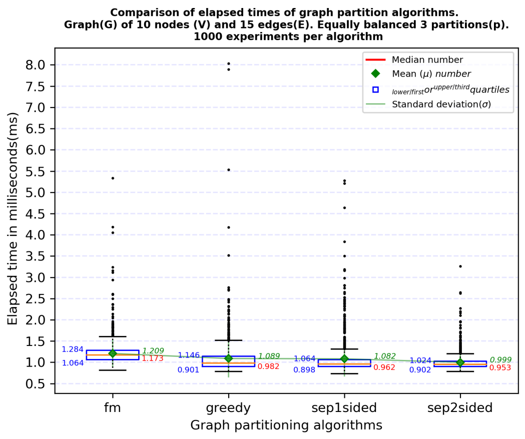
5.1 Results

Which user and application constrains might be applied during profiling of the graph partitioning algorithms?

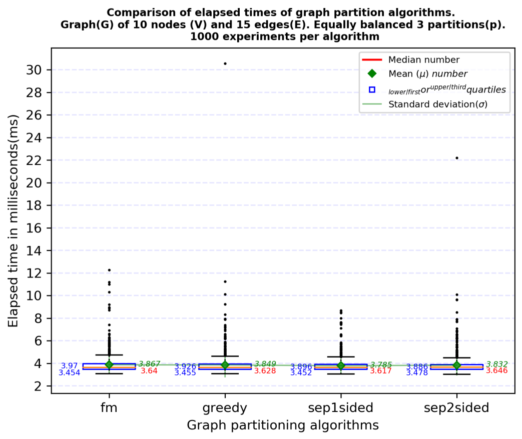
Keeping in mind failures mitigation to fulfill availability requirements and spreading application across different Geo-located data centers costs saving on data exchange might be one of the most important constraint for developer. From application point of view availability of resources to provide stable services is another important constraint. Therefore a virtual infrastructure graph partitioning tool may provide a balance between all these constraints.

What key algorithms are implemented for evaluation? After evaluating a tools proposed by literature a METIS graph partitioning software which may run cross-platform was selected. A best Python available wrapper for this software is a metis for python. METIS software has implemented a classical Fiduccia-Mattheyses(fm) algorithm with optional One-sided node-based(sep1sided) and Two-sided node-based (sep2sided) refinement components and Greedy algorithm.

Which graph partitioning algorithms are most desirable in the specific scenario's?



(a) 10 nodes



(b) 100 nodes

Figure 5: An experiment results.

Figure 5.1 illustrate results of experiment. It is clear that a classical Fiduccia-Mattheyses(fm) algorithm with Two-sided node-based (sep2sided) refinement component developed by a researchers behind METIS performs better in small scale graph. However in high scale one-sided node-based performs a bit better.

5.2 Observations

Despite the results that differences in elapsed time between two-sided node-based refinement and one-side node-based refinement are significantly small it was observed without empirical data that CPU usage by two-sided node-based refinement is higher then by one-sided node-based refinement. However, one-sided node-based refinement uses more memory. The classical Fiduccia-Mattheyse and Greedy algorithm are perform less and use more CPU and RAM resources.

6 Conclusions and Future work

Selecting a correct partition algorithm might be profitable for developer to save costs on exchanged data and mitigate failures. Using a METIS software with implemented Fiduccia-Mattheyses with Two-sided node-based refinement requires less execution time to partition graphs with small amount of nodes and low maximum degree. However, it may require more CPU and RAM resources.

Relation of each algorithm to CPU and RAM resources might be a great improvement of this research. Detection of a virtual infrastructure topology as a graph type is not really researched yet an need some effort. Combination of nodes clustering and edges cutting is available as a different components in METIS but it is not clear how it can be combined together. Researching and implementing more algorithms might be profitable not only for METIS but for community in general. Additional, research on user and virtual infrastructure application constraints might improve usage of virtual infrastructure.

References

[1] Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. https://www.researchgate.net/publication/310515157_Recent_Advances_in_Graph_Partitioning, 11 2016. (Accessed on 28-Nov-2017).

[2] Blesson Varghese and Rajkumar Buyya. Next generation cloud computing: New trends and research directions. *CoRR*, abs/1707.07452, 2017.

[3] Junchao Wang, Huan Zhou, Yang Hu, Cees de Laat, and Zhiming Zhao. Deadline-aware coflow scheduling in a dag. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 341–346. IEEE, 2017.

[4] B Hendricksonand R Leland and B Hendrickson. The chaco user's guide: version 2.0. Technical report, Tech. Rep. SAND94-2692, Sandia National Labs, Albuquerque, NM, 1995.

[5] Peter Sanders and Christian Schulz. Think Locally, Act Globally: Highly Balanced Graph Partitioning. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *LNCS*, pages 164–175. Springer, 2013.

[6] Tobias Heuer and Sebastian Schlag. Improving coarsening schemes for hypergraph partitioning by exploiting community structure. In *16th International Symposium on Experimental Algorithms, (SEA 2017)*, pages 21:1–21:19, 2017.

[7] D. Lasalle and G. Karypis. Multi-threaded graph partitioning. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 225–236, May 2013.

[8] Daniël M Pelt and Rob H Bisseling. A medium-grain method for fast 2d bipartitioning of sparse matrices. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 529–539. IEEE, 2014.

[9] Kamer Kaya, Ümit V Çatalyürek, and Bora Uçar. Integrated data placement and task assignment for scientific workflows in clouds. 2011.

[10] K.D. Devine, E.G. Boman, L.A. Riesen, U.V. Catalyurek, and C. Chevalier. Getting started with zoltan: A short tutorial. In *Proc. of 2009 Dagstuhl Seminar on Combinatorial Scientific Computing*, 2009. Also available as Sandia National Labs Tech Report SAND2009-0578C.

[11] George Karypis. Package: metis (5.1.0.dfsg-5 and others). https://packages.debian.org/stretch/metis, March 2013. (Accessed on 21-Jan-2017).

[12] NetworkX Developers. Networkx-metis. http://networkx-metis.readthedocs.io/en/latest, August 2015. (Accessed on 22-Jan-2017).

[13] Ken Watford. Metis for python. http://metis.readthedocs.io/en/latest/, January 2018. (Accessed on 23-Jan-2017).

[14] Andreas Klöckner. Pymetis. https://mathema.tician.de/software/pymetis/, August 2016. (Accessed on 21-Jan-2017).