Memory-Optimized Distributed Graph Processing through Novel Compression Techniques

Katia Papakonstantinopoulou

Joint work with Panagiotis Liakos and Alex Delis

University of Athens

Athens Colloquium in Algorithms and Complexity
UoA, August 26th, 2016

Motivation

graph data whose size continuously grows

$\Downarrow$

distributed graph processing systems (e.g., Pregel & Apache Giraph)

scale of real-world graphs hardens graph processing
even in distributed environments

$\Downarrow$

we need efficient distributed representations of such graphs

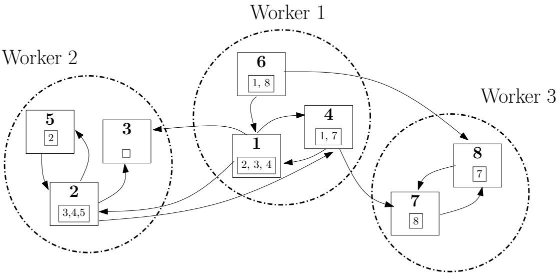
$\Downarrow$

We address this problem by exploiting empirically-observed properties demonstrated by behavior graphs

Distributed graph-processing approaches and systems

Most of these approaches adopt the *"think like a vertex"* programming paradigm introduced with `Pregel` [5] ($\Rightarrow$ intuitive parallelizable algorithms).

A graph partitioned on a vertex basis in a distributed environment:



- The proposed frameworks [1, 6, 4, 7] fail to handle the huge scale of real-world graphs, as a result of ineffective memory usage [2].
- The partitioning hardens the task of compression.

Indicative memory optimization approaches

The distributed graph-processing systems face significant memory-related issues. Some memory optimization approaches are:

`Apache Giraph` [1] (graph processing system that follows `Pregel`) with contributions by *Facebook*

- focused entirely on a more careful implementation for the representation of the out-edges of a vertex, without exploiting the redundancy in real-world graphs

`Gbase`[3] (a number of alternative compression techniques to reduce storage and hence network traffic and query execution time)

- does not follow the vertex-centric model
- requires decompression

Contribution

We follow the `Pregel` paradigm and partition the graph vertices among the nodes of a distributed computing environment.

In this context, we present a number of novel techniques that:

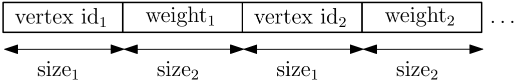
1. offer space efficient-representations of the out-edges of vertices,
2. allow fast mining (in-situ) of the graph elements without the need of decompression,
3. enable the execution of graph algorithms in memory-constrained settings, and
4. ease the task of memory management, thus allowing faster execution.

Our work lies in the intersection of distributed graph processing systems and compressed graph representations.

Distributed vs non-distributed settings

- In non-distributed settings, we can exploit the fact that vertices tend to exhibit similarities (copy property).
- In order to achieve memory optimization, we need representations that allow mining of the graph's elements without decompression.

`Giraph`'s adjacency-list representations: *ByteArrayEdges*

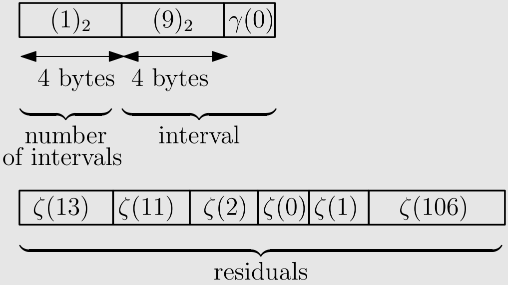


- The bytes required per out-neighbor are determined by the data type used for its id and weight; for integer numbers 4+4=8 bytes are required.

Representations based on consecutive out-edges

Consider the following sequence of neighbors to be represented:
$(2, 9, 10, 11, 12, 14, 17, 18, 20, 127)$.

BVedges

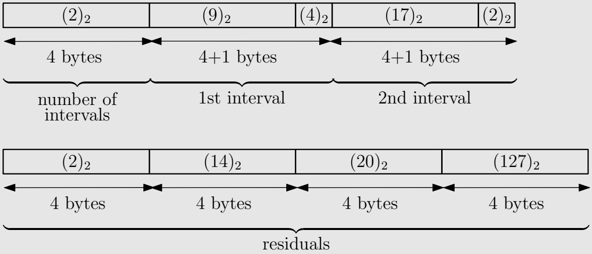


In the context of graph compression, Elias' γ coding is preferred for the representation of rather small values of x, whereas ζ coding is more proper for potentially large values.

Representations based on consecutive out-edges

Consider again the sequence of neighbors: $(2, 9, 10, 11, 12, 14, 17, 18, 20, 127)$.

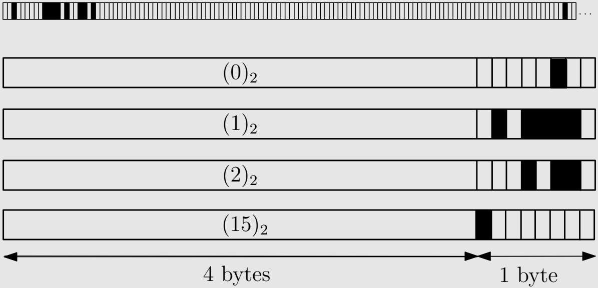
IntervalResidualEdges



Representation based on concentration of out-edges

Again using the sequence: $(2, 9, 10, 11, 12, 14, 17, 18, 20, 127)$.

IndexedBitArrayEdges



Experimental Evaluation

- How much more space-efficient is each of our three compressed out-edge representations compared to `ByteArrayEdges`?
- Are our techniques competitive speed-wise when memory is not a concern?
- How much more efficient are our compressed representations when the available memory is constrained?
- Can we execute algorithms for large graphs in settings where it was not possible before?

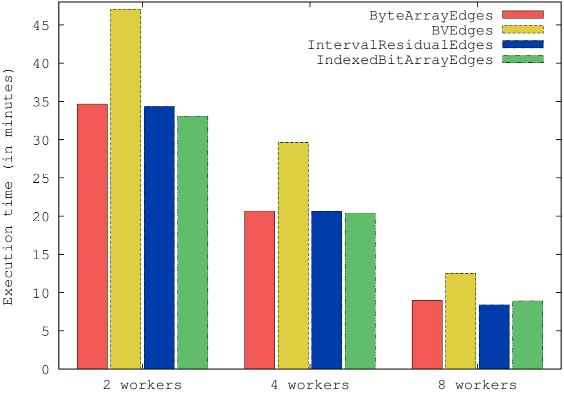
Space Efficiency Comparison

Memory requirements of `Giraph`'s `ByteArrayEdges` and our three representations for the small and large-scale graphs of our dataset:

graph	`ByteArray-Edges`	`BVEdges`	`IntervalResidualEdges`	`IndexedBitArrayEdges`
uk-2007-05@100000	22.61MB	6.41MB	7.92MB	8.91MB
uk-2007-05@1000000	279.16MB	67.36MB	82.7MB	97.79MB
indochina-2004	1,511.67MB	442.34MB	646.03MB	554.23MB
hollywood-2011	1,381.91MB	287.53MB	613.52MB	676.88MB
uk-2002	2,733.6MB	1,092.82MB	1,224.07MB	1,255.67MB
arabic-2005	4,820.09MB	1,428.97MB	1,674.75MB	1,849.83MB
uk-2005	7,401.88MB	2,383.54MB	2,728.74MB	2,928.81MB
sk-2005	14,829.64MB	4,889.85MB	5,657.79MB	6,354.17MB

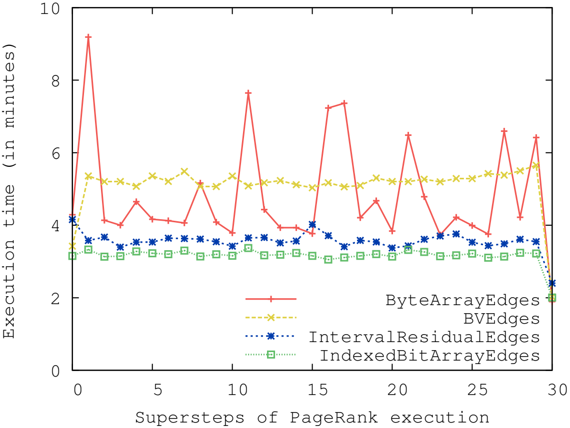
Execution Time Comparison (small-scale graphs)

Execution time of PageRank algorithm for graph *indochina-2004* using a setup of 2, 4, and 8 workers:



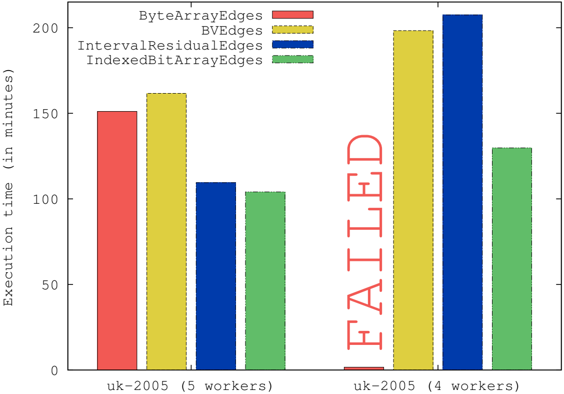
Execution Time Comparison (large-scale graphs)

Execution time for each *superstep* of PageRank algorithm for graph *uk-2005* using 5 workers:



- `ByteArrayEdges` performance fluctuates due to extensive garbage collection.

Execution time of PageRank algorithm for graph *uk-2005*:



- `IntervalResidualEdges` and `IndexedBitArrayEdges` outperform `ByteArrayEdges`.

Conclusion

Our experimental results indicate significant improvements on space-efficiency for all proposed techniques.

- We reduced memory requirements up to 5 times in comparison with currently applied techniques.

In settings where earlier approaches were able to execute graph algorithms, we achieve significant performance improvements.

- We reduced execution time up to 31% due to memory optimization.

These findings establish our structures as the preferable option for web graphs, or any other type of behavior graphs.

Future Directions

- Design representation methods that favor mutations of the graph.
- Examine the compression of edge weights.

References

[1] Apache Giraph.
`http://giraph.apache.org/`.

[2] Minyang Han, Khuzaima Daudjee, Khaled Ammar, M. Tamer Özsu, Xingfang Wang, and Tianqi Jin.
An Experimental Comparison of Pregel-like Graph Processing Systems.
Proc. of the VLDB Endowment, 7(12):1047–1058, 2014.

[3] U. Kang, Hanghang Tong, Jimeng Sun, Ching-Yung Lin, and Christos Faloutsos.
GBASE: an efficient analysis platform for large graphs.
VLDB J., 21(5):637–650, 2012.

[4] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein.
Distributed GraphLab: A Framework for Machine Learning in the Cloud.
Proc. of the VLDB Endowment, 5(8):716–727, 2012.

[5] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski.
Pregel: A System for Large-Scale Graph Processing.
In *ACM SIGMOD*, 2010.

[6] Semih Salihoglu and Jennifer Widom.
GPS: a graph processing system.
In *SSDBM*, 2013.

[7] Da Yan, James Cheng, Yi Lu, and Wilfred Ng.
Effective Techniques for Message Reduction and Load Balancing in Distributed Graph Computation.
In *WWW*, 2015.

Thank you for your attention!

for further details visit:
http://hive.di.uoa.gr/network-analysis/

or email me at: katia@di.uoa.gr