A Comparison of Graph Processing Systems

Simon König
(3344789)
st156571@stud.uni-stuttgart.de

Leon Matzner
(3315161)
@stud.uni-stuttgart.de

Felix Rollbühler
(3310069)
@stud.uni-stuttgart.de

Jakob Schmid
()
@stud.uni-stuttgart.de

Abstract—In this paper we will analyze and compare the graph frameworks Galois, Ligra, Polymer, Gemini and Giraph in their parformance. All the frameworks will be tested in shared memory and Galois, Gemini and Giraph are tested on a distributed cluster as well. Furthermore we will give some insight on the complexity of writing custom applications based on these frameworks.

Index Terms—graphs, distributed computing, Galois, Ligra, Polymer, Giraph, Gluon, Gemini

I. INTRODUCTION

This paper makes the following contributions:

- Comparison of some of the most widely used graph based calculation frameworks
-
-
-

II. OVERVIEW OF THE FRAMWORKS

A. Galois and Gluon

Galois[1] is a general purpose library designed for parallel programming. The Galois system supports fine grain tasks, allows for autonomous, speculative execution of these tasks and grants control over the task scheduling policies to the application. It also simplifies the implementation of parallel applications by providing an implicitly parallel unordered-set iterators.

For graph analytics purposes a topology aware work stealing scheduler, a priority scheduler and a library of scalable data structures have been implemented. Galois includes applications for many graph analytics problems, among these are single-source shortest-paths (sssp), breath-first-search (bfs) and pagerank. For most of these applications Galois offers several different algorithms to perform these analytics problems and many setting options like the amount of threads used or policies for splitting the graph. All of these applications can be executed in shared memory systems and, due to the Gluon integration, with a few modifications in a distributed environment.

Gluon[2] is a framework written by the Galois team as a middleware to write graph analysis applications for distributed systems. It reduces the communication overhead needed in distributed environments by exploiting structural and temporal invariants.

The code of Gluon is embedded in Galois. It is possible to integrate Gluon in other frameworks too, which the Galois team showed in their paper[2].

B. Gemini

C. Giraph

Apache Giraph[7] is an iterative graph processing framework, built on top of Apache Hadoop. Because of this, expanding shared memory processing to a multi-node cluster is almost seamless.

Giraph is inspired by the Bulk Synchronous Parallel model of distributed computation introduced by Leslie Valiant[**?**]. Meaning that it is based on computation units that communicate using messages and are synchonized with barriers.

The input to a Giraph computation is always a directed graph. Not only the edges but also the vertices have a value attached to them. The graph topology is thus not only defined by the vertices and edges but also their initial values. Furthermore, one can mutate the graph by adding or removing vertices and edges during computation.

Computation is vertex oriented and iterative. For each iteration step called superstep, the Compute method implementing the algorithm is invoked on each active vertex, with every vertex being active in the beginning. This method receives messages sent in the previous superstep as well as its vertex value and the values of outgoing edges. With this data the values are modified and messages to other vertices are sent. Communication between vertices is only performed via messages, so a vertex has no access to values of other vertices or edges other than its own outgoing ones.

Supersteps are synchronized with barriers, meaning that all messages only get delivered in the following superstep and computation for the next superstep can only begin after every vertex has finished computing the current superstep. Edge and vertex values are retained across supersteps.

Any vertex can stop computing (i.e. setting its state to inactive) at any time but incoming messages will make the vertex active again. To end computation, a vote-to-halt method is applied. Each vertex outputs some local information (e.g. the final vertex value) as result.

Giraph is next to Galois probably the most actively developed framework in our comparison.

D. Ligra

E. Polymer

Polymer is very similar to Ligra, in fact Polymer inherits the programming interfaces `EdgeMap` and `VertexMap` from Ligra as its main interface.

Polymer is a vertex-centric framework, that tries to circumvent some of the random memory access drawbacks of such a design. It treats a NUMA machine as a distributed cluster and splits work and graph data accordingly between the nodes.[3, p.187] Application-defined data is not distributed. Other runtime state data is allocated in a distributed way but only accessed through a global lookup table.[3, p.184]

III. AND

A. An overview of some graph formats

A rather big portion of our time was invested in figuring out which graph framework requires which graph formats. We thus decided to give an overview over all the formats we encountered, with explanation on how they represent the graph.

Additionally, to make life in the future a little bit easier, we wrote multiple tools to convert graphs acquired from Snap or Konect to the required formats. Additional information on this is available in the section Supplementary Data at the end.

1) AdjacencyList: The AdjacencyList and WeightedAdjacencyList formats[4] are used by Ligra and Polymer. They represent the directed edges of a graph as a number of offsets that point to a set of target nodes in the file. First the file contains the number of vertices n and edges m, followed by an offset for each vertex. This offset specifies at what point in the following list of numbers the information for a node begins. Lastly the file format contains a list of target nodes. The numbers are all separated by newlines.

$$
\begin{aligned}
&n \\
&m \\
&o_1 \\
&o_2 \\
&\vdots \\
&o_n \\
&t_1 \\
&t_2 \\
&\vdots \\
&t_m
\end{aligned}
$$

The offsets $o_i = k$ and $o_{i+1} = k + j$ mean that vertex i has j outgoing edges, these edges are

$$(i, t_k), (i, t_{k+1}), \ldots, (i, t_{k+j-1})$$

For the WeightedAdjacencyList format, the weights are just appended to the end of the file in the same order as the edges.

2) EdgeList: The EdgeList format is probably the easiest to understand and is one of the most commonly used in the online graph repositories. The directed eges $(s_1, t_1), (s_2, t_2), \ldots$ or $(s_1, t_1, w_1), (s_2, t_2, w_2), \ldots$ are represented in the following way.

$$
\begin{array}{ccc}
s_1 & t_1 & w_1 \\
s_2 & t_2 & w_m \\
& \vdots & \\
s_m & t_m & w_m
\end{array}
$$

The weights are optional, everything is ASCI encoded and the inline delimiter is a variable amount of any whitespace.

3) Binary EdgeList: The binary EdgeList format is used by Gemini. Finding information on this format required reverse engineering of the Gemini code.

We found that Gemini requires the following input format

$$s_1 t_1 w_1 s_2 t_2 w_2 \ldots$$

where s_i, t_i have `uint32` data type and the optional weights are `float32`. Gemini will derive the number of edges from the file size, so there is no file header or anything similar allowed.

4) Giraph's I/O formats: Giraph is capable of parsing many different input and output formats. All of those are explained in Giraph's JavaDoc. Both edge- and vertex-centric input formats are possible.

One can even define their own input graph representation or output format. For the purposes of this paper, we used the existing `JsonLongDoubleFloatDoubleVertexInputFormat`[**?**]. Our graph conversion tool outputs this format.

In this format, the vertex IDs are specified as `long` with `double` vertex values, `float` out-edge weights and the messages are of type `double` (all are Java data types). All this is in a JSON-like format, so each line in the graph file looks as follows

$$[s, v_s, [[t_1, w_{t_1}], [t_2, w_{t_2}]...]]$$

with s being a vertex ID, v_s the vertex value of vertex s. The values t_i are vertices for which an edge from s to t_i exists. The directed edge (s, t_i) has weight w_{t_i}.

There is surrounding pair of brackets and no commas separating the lines as it would be expected in a JSON format.

IV. TESTING METHODS

A. Hardware and Software

For testing all systems we used 5 machines with 96 CPU cores each (48 physical) and 256 GB of RAM. One of those machines was only used as part of the distributed cluster, since it only has 128 GB of RAM. All servers were running Ubuntu 19.x. Setup of each framework was performed according to our provided installation guides.

All benchmarks were initiated by our benchmark script that is available in our repository. Galois, Gemini and Giraph were benchmarked on both the 5-node cluster and a singe machine. Furthermore, – since Galois supports this parameter – we ran

TABLE I
SIZE COMPARISON OF THE USED GRAPHS

Graph	# Vertices (M)	# Edges (M)
flickr	0.1	2
orkut	3	117
wikipedia	12	378
twitter	52	1963
rMat27	63	2147
friendster	68	2586
rMat28	121	4294

TABLE II
NATIVELY SUPPORTED ALGORITHMS OF THE FRAMEWORKS

	Algorithm	Galois	Gemini	Giraph	Ligra	Polymer
PageRank	Push	✓			?	✓
	Pull	✓			?	
	Delta	?			✓	✓

multiple tests comparing Galois' performance with different core counts on a single machine.

The complete log files and extracted raw results will be available in our repository as well.

B. Measurements

For every framework, we measured the *execution time* as the time from start to finish of the console command.

For the *calculation time*, we tried to extract only the time the framework actually executed the algorithm. We came up with the following:

- For Galois, we resulted to extracting console log time stamps. Galois outputs `Reading graph complete..`. Calculation time is the time from this output to the end of execution.
- Polymer outputs the name of the algorithm followed by a measured time.
- Gemini outputs a line `exec_time=x`, which was used to measure the calculation time.
- Ligra outputs a measured time with `Running time : x`.
- Giraph has built in timers for the iterations (supersteps) which we used to extract the computation time.

Each test case consisting of graph, framework and algorithm was run 10 times, allowing us to smooth slight variations in the measured times.

C. Graphs

The graphs used in our testing can be seen in detail in Table I. We included a variaty of different graph sizes, from relatively small graphs like the flickr graph up to an rMat28 with 4.2 billion edges. All graphs except the rMat27 and rMat28 are exemplary real-world graphs and were retrieved from the graph database[1] associated with the Koblenz Network Collection (KONECT)[8]. Both the rMat27 and rMat28 were created with a modified version (we changed the output format to EdgeList) of a graph generator provided by Ligra.

D. Algorithms

We ran Breadth-first search (BFS), PageRank (PR) and Single-source shortest-path (SSSP) algorithms on each framework and graph. For frameworks that support multiple implementations (e.g. PR in push and pull modes), we included the alternatives in our testing.

[1]http://konect.uni-koblenz.de/

E. Comparison of setup

F. Implementing new application

V. RESULTS

A. Komplexität Aufsetzen

Systeme wie Galois/Gluon, bei denen man erstmal ewig suchen muss bis man überhaupt einen Guide findet sollten hier schlechter abschneiden. *vorausgesetzt, wir nehmen das hier überhaupt mit auf..*

B. Komplexität eigene Apps schreiben

Schwierig objektiv zu vergleichen.

C. Pure Performance-Ergebnisse

D. Ergebnisse Calc time

der offensichtliche, wichtige Vergleich

E. Ergebnisse exec time

hier erhoffe ich mir einen Vergleich der Ladezeiten und erwarte, dass Systeme wie Giraph, die erstmal auf irgendwas warten schlecht abschneiden. Aber vielleicht ist auch die setup time bei gleichen frameworks zwischen verteilt und shared memory ganz interessant zu vergleichen.

VI. DISCUSSION

VII. CONCLUSION

The conclusion goes here.

SUPPLEMENTARY DATA

We have written a number of conversion tools and installation guides to help users or developers with the use of the tested frameworks.

Our GitHub repository: http://www.github.com/serengti/Forschungsprojekt.

REFERENCES

[1] D. Nguyen, A. Lenharth, and K. Pingali, "A lightweight infrastructure for graph analytics," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 456–471. [Online]. Available: https://doi.org/10.1145/2517349.2522739

[2] R. Dathathri, G. Gill, L. Hoang, H.-V. Dang, A. Brooks, N. Dryden, M. Snir, and K. Pingali, "Gluon: A communication-optimizing substrate for distributed heterogeneous graph analytics," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 752–768. [Online]. Available: https://doi.org/10.1145/3192366.3192404

[3] K. Zhang, R. Chen, and H. Chen, "Numa-aware graph-structured analytics," in *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 183–193. [Online]. Available: https://doi.org/10.1145/2688500.2688507

[4] J. Shun, G. Blelloch, J. Fineman, P. Gibbons, A. Kyrola, K. Tangwonsan, and H. V. Simhadri. (2020, Jun.) Problem Based Benchmark Suite. graphIO.html. [Online]. Available: http://www.cs.cmu.edu/~pbbs/benchmarks/

[5] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," in *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 135–146. [Online]. Available: https://doi.org/10.1145/2442516.2442530

[6] X. Zhu, W. Chen, W. Zheng, and X. Ma, "Gemini: A computation-centric distributed graph processing system," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 301–316. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/zhu

[7] A. S. Foundation. (2020, Jun.) Apache Giraph. [Online]. Available: https://giraph.apache.org

[8] J. Kunegis, "Konect: the koblenz network collection," 05 2013, pp. 1343–1350.