An Experimental Evaluation of Giraph and GraphChi

Junnan Lu
University of Victoria, BC, Canada
byzantin@uvic.ca

Alex Thomo
University of Victoria, BC, Canada
thomo@uvic.ca

***Abstract*—We focus on the vertex-centric (VC) model introduced in Pregel, a Google system for distributed graph processing. In particular, we consider two popular implementations of the VC model: Apache Giraph and GraphChi. The first is a VC system for cluster computing, while the second is a VC system for a single PC. Apache Giraph became very popular after careful engineering by Facebook researchers in 2012 to scale the computation of PageRank to a trillion-edge graph of user interactions using 200 machines. On the other hand, GraphChi became popular, around the same time in 2012, as it made possible to perform intensive graph computations in a single PC, in just under 59 minutes, whereas the distributed systems were taking 400 minutes using a cluster of about 1,000 computers (as reported also by MIT Technology Review). Since then, new versions of Apache Giraph and GraphChi have been released, where new ideas and optimizations have been implemented. Therefore, it is time to validate again the claims made four years ago. In this work, we embark in this validation. We consider three cornerstone graph problems: computing PageRank, shortest-paths, and weakly-connected-components. Based on current experiments, we conclude that in the present, even for a moderate number of simple machines, Apache Giraph outperforms GraphChi for all the algorithms and datasets tested. This is in contrast to the claims of the GraphChi authors in 2012.**

I. INTRODUCTION

Graphs are very popular in analyzing data. Many interesting graph applications such as transportation routes, newspaper article similarity, disease outbreak and scientific work citations have been studied for decades. Graph algorithms that have been applied frequently include shortest paths computation, weakly-connected-components, connection-counting, and Pagerank. The main problem when applying graph analytics is the typically very big size of the available graphs. Taking the web graph, for example, the estimation from Google shows that the population of web pages is now exceeding 1 trillion. Social networks are other examples of very big graphs. In 2012, the number of users (as vertices) of Facebook exceeded over a billion with over 190 billion of friendship relationships, which can be viewed as links. Another example, LinkedIn, has more than 8 million users with more than 60 million of relationships.

Due to the irregular internal structure and the large scale of the data, processing such graphs is considered computationally hard in the traditional centralized way. Determining the best paradigm for computing systems in order to handle and process such data is a hot research area in the data analytics community.

Google's Pregel is a simple yet popular graph processing tool which is inspired by the BSP model [8]. Similar to BSP, Pregel consists of a sequence of iterations, called supersteps [4]. In each superstep in Pregel, a user-defined-function, *compute*, is invoked for each vertex of the graph in order to conduct computation in parallel. The user defined function, such as reading a message from the previous superstep or sending a message to be read in the next superstep, defines the logic and behavior of each single vertex in each of the iterations or supersteps. Such a model is also labeled as *vertex centric* because the computation task is running independently and locally.

Apache Giraph is an open source implementation of proprietary Pregel. All the essential functionality for processing a graph in parallel like in Pregel is implemented in Apache Giraph. Giraph became popular after careful engineering by Facebook researchers in 2012 to scale the computation of PageRank to a trillion-edge graph of user interactions using 200 machines [1].

Systems like Apache Giraph and Pregel require a distributed computing cluster to process large scale graph data quickly and effectively. Although distributed computing facilities such as cloud computing clusters are becoming more common and accessible, nevertheless, the question of how to process large scale graph data effectively without distributed commodity computing clusters is an interesting avenue for a data analyst who may need to analyze a large graph dataset but is unable to access a distributed computing cluster. GraphChi proposed by Kyrola and Guestrin [5] is a disk-based, vertex-centric system, which segments a large graph into different partitions. Then, a novel parallel sliding window algorithm is implemented to reduce random access to the data graph. Graphchi can process hundreds to thousands of graph vertices per second. GraphChi became popular, around the same time in 2012, as it made possible to perform intensive graph computations in a single PC in just under 59 minutes, whereas the distributed systems were taking 400 minutes using a cluster of about 1,000 computers (as reported also by MIT Technology Review). Since then, new versions of GraphChi and Apache Giraph have been released, where new ideas and optimizations have been implemented. Therefore, it is time to validate again the claims made four years ago. In this work, we embark in this validation.

We consider three cornerstone graph problems that are often used as subroutines for several graph analytics tasks: computing (1) PageRank, (2) shortest-paths, and (3) weakly-connected-components. Based on current experiments, we conclude that in the present, even for a moderate number of simple machines, Apache Giraph outperforms GraphChi for all the algorithms and datasets tested. This is in contrast to the claims of the GraphChi authors in 2012.

II. Algorithms

We evaluated four algorithms in our comparisons: PageRank, Single-Source Shortest Paths (SSSP), and Weakly Connected Components (WCC).

A. PageRank

PageRank [11] is a popular random walk algorithm that produces a ranking of the vertices in a network. The main goal of PageRank is to assign a numeric value to each of the vertices by exploring the structure of the graph. In the context of vertices being web pages, a page receives more importance, if it is linked from other pages of high importance. PageRank uses a damping factor, a probability value, to determine the likelihood that a user will jump to a random page by clicking an outgoing link from the current page when the user is browsing. In this evaluation, the damping factor is set to 0.85.

At superstep 0, each vertex is assigned a value of 1.0 as the vertex value. Then the vertex sends to each neighbor its "vote", which is 1/outdegree. At each subsequent superstep, each vertex will sum all the values it receives from its neighbors and store the summation in x. The vertex value will be updated by combining the in-edge summation value x with the dumping factor as $0.15 + 0.85 * x$. The vertex value is again broadcast to the neighbors. We perform 30 such iterations of PageRank computations. The implementation for PageRank is slightly different for Giraph and GraphChi. In the former we receive and send messages, in the latter we read from in-edges and write to out-edges. The pseudocode for Giraph is given in Alg. 1, whereas the pseudocode for GraphChi is omitted as being quite similar (this is not always so; see the next algorithm).

Algorithm 1 PageRank: Compute function for Giraph

```
1:  function COMPUTE(Vertex $v$, List $messages$)
2:      if $superstep = 0$ then
3:          $v.value \leftarrow 1$
4:      else
5:          $x \leftarrow 0$
6:          for all $m$ in $messages$ do
7:              $x \leftarrow x + m.value$
8:      $v.value \leftarrow 0.15 + 0.85 * x$
9:      $y \leftarrow x / v.numOutEdges$
10:     for all $outEdge$ in $v.outEdgeList$ do
11:         $sendMessage(outEdge.targetVertexId, y)$
```

B. SSSP

SSSP here stands for single-source shortest path. SSSP is a traversal algorithm which finds the shortest paths between a source vertex and all the other reachable vertices. The vertex-centric SSSP is a parallel variant of the Bellman-Ford algorithm. At the first superstep, the distance of the source vertex is set to 0 and the values of all other vertices are set to plus infinity. Also, the source vertex s is the only active vertex in the first superstep. At each point in time, a vertex v has as value its tentative distance d_v from the source. Vertex v sends $d_v + c(v, w)$ to each neighbor w, where $c(v, w)$ is the cost of the (v, w) edge. Upon receiving messages from its neighbors, vertex v updates its value to be the minimum of the received values. The number of supersteps executed by SSSP is limited by the graph's longest shortest path.

Whereas Giraph works using messages as described above, GraphChi works by reading values from in-edges and writing values to out-edges. Here we need to be more careful than in the case of PageRank because now the edges have already values, which are their costs. Therefore, we need to create special tuple objects to store on the edges (see Alg. 3).

On the other hand, the pseudocode for Giraph is given in Alg. 2. Observe that in Giraph we have the opportunity to use a combiner of messages. If there are many messages going to a vertex, it is clear that not all of them will cause the value of the vertex to change. In fact only the message with the smallest value (tentative distance) will be useful. Selecting the message with the smallest value and removing the other messages is the job of the combiner in this case. It aggregates messages sent to a vertex to only one message, the one with the minimum value.

The pseudocode for GraphChi is given in Alg. 3. Here we do not have the luxury of a combiner. The computation finishes when the value of variable $numVerticesUpdated$ becomes 0.

Algorithm 2 SSSP: Compute function for Giraph

```
1:  function COMPUTE(Vertex $v$, List $messages$)
2:      if $superstep = 0$ then
3:          if $v = source$ then
4:              $v.value \leftarrow 0$
5:          else
6:              $v.value \leftarrow \infty$
7:      $minDist \leftarrow v.value$
8:      for all $m$ in $messages$ do
9:          $dist \leftarrow m.value$
10:         $minDist \leftarrow \min\{minDist, dist\}$
11:     if $minDist < v.value$ then
12:         $v.value \leftarrow minDist$
13:         for $outEdge$ in $v.outEdgeList$ do
14:             $y \leftarrow minDist + outEdge.value$
15:             $sendMessage(outEdge.targetVertexId, y)$
16:     $v.voteToHalt()$
17: function COMBINER(List $messages$)
18:     return $\min(messages)$
```

C. WCC

The weakly connected components (WCC) problem is to assign to each vertex v the id of the weakly connected component that v belongs to. A component (subgraph) C is weakly connected if for every pair u, v of vertices, there exists a semi-path (ignoring edge directions) from u to v in C. The idea for this algorithm is simple. Each vertex has its own id and propagates this id to its neighbors. A vertex v, upon receiving the ids from its neighbors, checks to see whether there is some id which is smaller than its own. If so, v changes its id to this smaller id. At the end of the algorithm, which happens when there are no more id updates, each vertex will have an id that is the id of WCC the vertex belong to. At the first superstep, all vertices are active which is different from SSSP. At each of the supersteps, a vertex can vote to halt if there is nothing left to

Algorithm 3 SSSP: Update function for GraphChi

```
function UPDATE(Vertex $v$)
    if $iteration = 0$ then
        if $v = source$ then
            $v.value \leftarrow 0$
        else
            $v.value \leftarrow \infty$
    $minDist \leftarrow v.value$
    for all $inEdge$ in $v.inEdgeList$ do
        $edgeVal \leftarrow inEdge.pair.value$
        $minDist \leftarrow \min\{minDist, edgeVal\}$
    if $minDist < v.value$ then
        $v.value \leftarrow minDist$
        $numVerticesUpdated$++
        for $outEdge$ in $v.outEdgeList$ do
            $edgeVal \leftarrow outEdge.pair.value$
            $dist \leftarrow minDist + outEdge.value$
            $outEdge.setVal(new\ Pair(edgeVal, dist))$
```

update. The pseudocode for Giraph is given in Alg. 4, whereas the pseudocode for GraphChi is omitted as being similar.

Algorithm 4 WCC: Compute function for Giraph

```
function COMPUTE(Vertex $v$, List $messages$)
    if $superstep = 0$ then
        $v.value \leftarrow source.getId()$
    $changed \leftarrow false$
    for all $m$ in $messages$ do
        if $m.value < v.value$ then
            $v.value \leftarrow m.value$
            $changed \leftarrow true$
    if $changed = true$ then
        for $outEdge$ in $v.outEdgeList$ do
            $sendMessage(outEdge.targetVertexId, v.value)$
    $v.voteToHalt()$
```

III. Experiments

The Giraph experimentation is conducted on 20, 25, 30, 25, and 40 machines. All machines are T2.micro Amazon Elastic Computing instances which are located in us-west-2a. A single EC2 T2.micro instance is used for running GraphChi program. Each T2.micro instance has one 2.5 GHz, Intel Xeon Family CPU, 1 GB of RAM Memory. All instances run Amazon Linux AMI 2015.09.1 (HVM) Operating System. The T2.micro machines, used in this experimentation, are in the AWS Free Tier. All machines are EBS backed by default. No additional EBS attachment is added for the simplicity of management. There is one worker per machine. The Giraph, Hadoop and Java SDK and runtime environments were installed in both the master and workers.

The datasets we used are

- Soc-LiveJournal ($|V| = 4,847,571, E = 68,993,773$)
- Soc-Pokec ($|V| = 1,632,803, E = 30,622,564$)
- Cit-Patents ($|V| = 3,774,768, E = 16,518,948$).

We obtained them from http://snap.stanford.edu.

The running times of each algorithm for different machine numbers (giraph instances, GIs) are given in Figure 1. In the first, second, and third rows of results, we show the running times of PageRank, SSSP, and WCC for the Soc-LiveJournal, Soc-Pokec, and Cit-Patents datasets, respectively. All the running times are given in seconds.

What we observe is that Giraph outperforms GraphChi significantly for a moderate number of machines (even 20 free-tier T2.micro machines). This is in contrast to what was reported in 2012 in [6] where the situation was reversed. Then, not even 1000 machines were able to beat GraphChi running on a single machine.

On the other hand, Giraph does not always scale linearly. For PageRank on Soc-LiveJournal, we achieve impressive scalability when the number of machines is increased. The runtime using 20 machines is 971 seconds, whereas the runtime using 39 machines (almost double) is 287. This translates to a 3.38 speed-up. On the other hand, for PageRank on Soc-Pokec and Cit-Patents, the speedup is not that impressive. As we go from using 20 machines to using 39 machines, we see speedups by a factor of 1.39 and 1.44, respectively. This is good but not ideal.

Doing a similar analysis for SSSP, we see speedups of 1.67, 1.15, and 1.29 on Soc-LiveJournal, Soc-Pokec, and Cit-Patents, respectively, when we go from 20 machines to 39. Similarly, for WCC, we see speedups of 1.21, 1.19, and 1.18 on Soc-LiveJournal, Soc-Pokec, and Cit-Patents, respectively. We can explain the lesser degree of speedup for SSSP and WCC compared to PageRank by the fact that they are less computationally intensive than PageRank which needs a considerable number of iteration to converge. Therefore, the start up time takes a more considerable portion of the overall time for SSSP and WCC.

IV. Related Works

The Pregel distributed graph processing framework was introduced by Malewicz et. al. in [8]. Apache Giraph (http://giraph.apache.org) is an open source implementation of Pregel based on Hadoop. An excellent reference on Giraph is the recent book by Martella, Shaposhnik, and Logothetis [9].

GraphChi was created by Aapo et. al. [6]. Its excellent speed compared to distributed vertex-centric systems at the time (2012) was commented with awe at MIT Technology Review [12].

Around the same time, a group of Facebook researchers introduced several optimizations to Giraph [1]. These and other optimizations to Giraph are described in a recent paper by Ching et. al. in [2]. As such, Giraph has become a much faster system that can easily outperform GraphChi even with few machines (as we showed with our experiments).

Thorough analysis of distributed vertex-centric systems have been presented by Han et. al. [5] and Lu et. al. in [7]. A recent survey of vertex-centric frameworks is by McCune et. al. [10].

As future work, we would like to analyze further more specialized shortest path approaches which are selective in

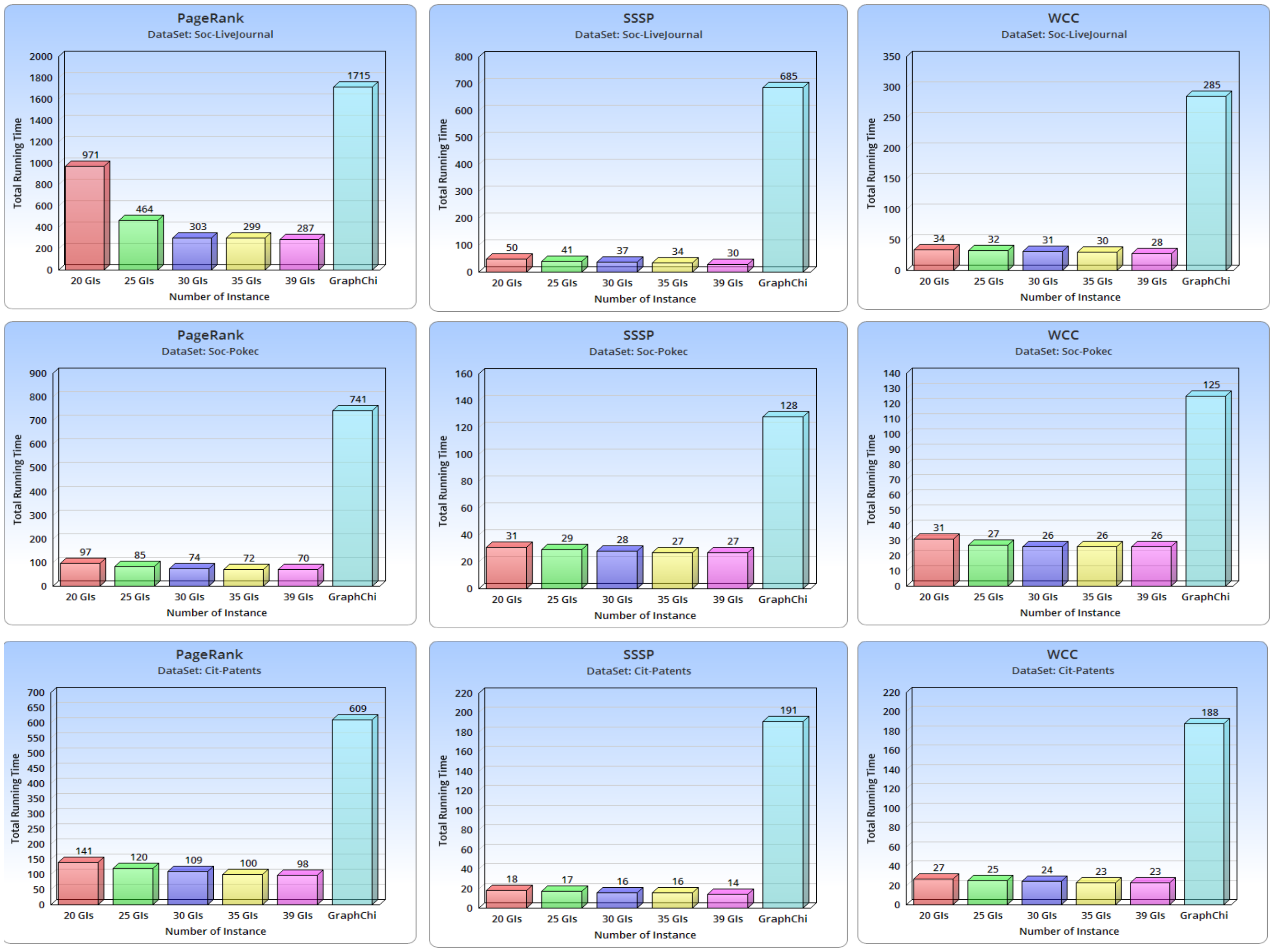

Fig. 1. Running times of Giraph and GraphChi (sec).

the edges they use. For example the type of edges to use can be specified by regular-path-queries (c.f. [3], [4]). It will be interesting to see how to devise vertex-centric algorithms for shortest paths guided by an automaton representing RPQs. Initial attempts are the algorithms presented in [13], [14] for a general distributed, message passing system.

References

[1] Scaling apache giraph to a trillion edges. http://bit.ly/1TomAkh. Accessed: 2016-05-22.

[2] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan. One trillion edges: graph processing at facebook-scale. *PVLDB*, 8(12), 2015.

[3] G. Grahne and A. Thomo. Algebraic rewritings for optimizing regular path queries. *Theoretical Computer Science*, 296(3), 2003.

[4] G. Grahne, A. Thomo, and W. Wadge. Preferentially annotated regular path queries. In *ICDT*. Springer, 2007.

[5] M. Han, K. Daudjee, K. Ammar, M. T. Özsu, X. Wang, and T. Jin. An experimental comparison of pregel-like graph processing systems. *Proceedings of the VLDB Endowment*, 7(12):1047–1058, 2014.

[6] A. Kyrola, G. Blelloch, and C. Guestrin. Graphchi: Large-scale graph computation on just a pc. In *USENIX OSDI*, pages 31–46, 2012.

[7] Y. Lu, J. Cheng, D. Yan, and H. Wu. Large-scale distributed graph computing systems: An experimental evaluation. *PVLDB*, 8(3):281–292, 2014.

[8] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD*. ACM, 2010.

[9] C. Martella and R. Shaposhnik. Practical graph analytics with apache giraph. 2015.

[10] R. R. McCune, T. Weninger, and G. Madey. Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing. *ACM Computing Surveys (CSUR)*, 48(2):25, 2015.

[11] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[12] J. Pavlus. Your laptop can now analyze big data. *MIT Technology Review*, July 17, 2014.

[13] M. Shoaran and A. Thomo. Fault-tolerant computation of distributed regular path queries. *Theoretical Computer Science*, 410(1):62–77, 2009.

[14] D. C. Stefanescu, A. Thomo, and L. Thomo. Distributed evaluation of generalized path queries. In *SAC*. ACM, 2005.