Network Traffic Measurement of Big Data Analytics Platforms

Mikael Mantis, Qizhen Zhang

Abstract

Graphs form the bases of many large scale data analytics in many real world applications such as Facebook and Google. In recent years several different scalable graph analytics systems to process such graphs have emerged which has thus sparked a new topic in literature. Studies have been done comparing several different systems performance in respect to different graph applications. We seek to improve upon the literature by observing the network traffic data utilized by Pregel+, a graph system that has been found to perform better than others. As a result of our observations we are able to further optimize the run time of several graph processing applications.

1. INTRODUCTION

Due to the significant scale of data, big data analytics consumes a large portion of workloads in today's data centers. Although a nature of such big data is the variety of data forms, some particular data forms have been highlighted and gained special attention because of their prevalence in the real world. The graph is one of them. According to [1], Facebook is running Giraph on hundreds of machines for their big graph analytics (on trillion scale graphs). There exist some works on optimizing the performance of graph analytics systems from different layers. For example, [2] proposes specialized file system using SSDs for big graph processing. Beside graph systems, general purpose big data analytics systems, such as Spark, also have been widely used in both industry and academia. It has significance to understand the network traffic in the data center of those big data systems. Findings related to network bottlenecks and traffic predictability may be critical to the design of such systems.

In this project, we seek to understand the network workload patterns of today's big data analytics by conducting a measurement study inside a data center. Further, based on the traffic study, we'd like to study how can we improve the performance of such systems from the network layer. For example, if we can change the routing strategies or network topology to gain a better network infrastructure for such systems. Particularly, we want to answer these questions: 1) are there particular network patterns (or predictability) for each big data workload (dataset, application, platform) in terms of intra-rack and inter-rack (optional) traffic? 2) (optional) what's the difference between intra-rack traffic and inter-rack traffic? 3) if such patterns exist, do they vary with datasets, applications or platforms, or are there general patterns for common datasets, applications or platforms? 4) what can we do to optimize the performance of such big data analytics systems in network layer? To answer these problems, we will set up a measurement environment in some data center first. We then select different datasets, applications, and big data analytics systems (particularly, we pay attention to graph systems). We will run those workloads in the prepared environment. At last, we will collect and analyze the results, and look for the answers to the questions that we are interested.

To our best knowledge, network traffic measurement for big data analytics systems has not been fully studied, especially for graph analytics systems. [3] conducted a measurement that only limited to Spark, while we will look at other analytics systems that are specifically designed for graphs such as the vertex-centric graph system, Pregel. [4] sought to characterize the workloads of graph processing systems, however, they did not consider the network traffic as we do. They focused on the workloads of CPU and memory usage. Also, their results are very elementary. [5] studied the impact of a fast network on the performance of graph analytics. However, they only compared the performance difference between slow and fast networks, and they did not attempt to understand the network traffic of graph analytics workloads as we do.

2. BACKGROUND

Graph Analytics Systems are computational frameworks designed to process large amounts of data in the form of graphs. They receive graphs in various forms as input and outputs various information depending on the application (e.g. PageRank, Single Source Short-

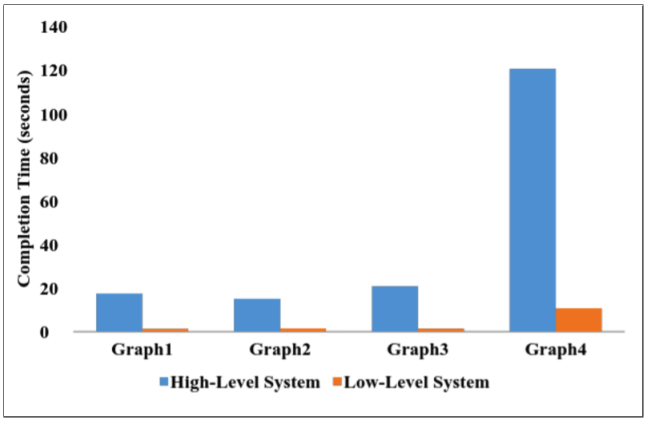

Figure 1: Comparison of SSSP performance using a High-Level System (SQL) and a Low-Level System (Pregel) on ascending real-world graph sizes

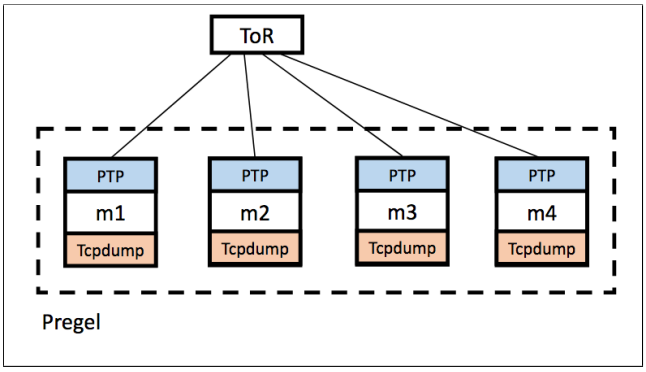

Figure 2: Network Configuration

est Path (SSSP), and Connected Components). The variety of Graph Analytics Systems out there can be categorized as either High-Level or Low-Level systems. High-Level meaning they are based on high-level declarative languages such as SQL or Datalog that offer an abstraction for implementing algorithms that allow for algorithms such as SSSP to be implemented in 3 lines of code. Low-Level meaning they are based on lower-level object oriented languages such as C++ and Java. Low-Level Systems also tend to be either vertex-centric, edge-centric, or subgraph-centric with vertex-centric being the most common. However, the negative aspect of low-level systems is that they require 100+ LOC for the same applications that can be expressed in much fewer lines for High-level systems. After measuring the performance of the SSSP algorithm on both a high and low-level graph processing system it is apparent that the performance gains from using a low-level system outweigh the increase in complexity. Thus we focus on vertex-centric systems.

In 2010, Pregel, the first scalable general-purpose framework for processing large-scale graphs was proposed by Google [8] One of Pregel's and all Pregel-like system's key features is its vertex-centric API. This allows for programmers to design their applications from the "perspective of a vertex". So rather than having to implement graph algorithms from a holistic view developers can implement such algorithms by specifying what each vertex must do. This allows for vertices to modify their own state, modify the state of their outgoing edges, send and receive messages, and even mutate graph topologies. Another key aspect is that these systems are iterative and they follow the Bulk Synchronous Parallel (BSP) model in the form of Supersteps. Supersteps are iterations that are separated by global synchronization points until the algorithm terminates and finishes with a single output. Upon each Superstep, vertices compute in parallel with each executing the same user-defined functions that form the application's logic. One last important aspect of these graph Systems is message passing. Because algorithm termination is determined by every vertex coming to a halt messages must be sent by each vertex and passed between each superstep. Pregel uses a pure message passing model that allows algorithms to avoid having to pass the entire state of the graph between stages as well as allows for latency to be amortized by asynchronously delivering messages in batches (supersteps). In section 6 we discuss other studies on such Graph Analytics Systems however an important finding from these studies is that Pregel+, an open source version of Pregel, outperforms other similar systems.

3. ENVIRONMENT

Before conducting the experiment a measurement environment had to be constructed. A cluster of 4 servers, each with 32 cores and 128 GB of memory running Ubuntu server 16.06 LTS was used. The graph system used for all measurements was Pregel+ implemented on top of MPI (3.2) and HDFS (Hadoop 2.6.5).

4. MEASUREMENT & ANALYSIS

In order to properly measure and keep measurements in sync amongst the network traffic data within the cluster we used PTP (ptpd2 2.3.1) for clock synchronization. The actual network profiling was done using Tcpdump (tcpdump 4.9.0), libcap (1.7.4), and OpenSSL (1.0.2). Figure 2 depicts the full network configuration with Pregel, ptp, and tcpdump.

4.1 Method

Our experiment involved running the SSSP, Connected Components, and PageRank applications on Pregel+ using the Orkut and UK-Web graph datasets. Orkut serves as an example of a real-world social network graph with approx. 3 million vertices and 200 million edges

Application
Single Source Shortest Paths
Connected Components
PageRank

Dataset	|V|	|E|	Type
Orkut	3,072,441	234,370,166	social network
UK-Web	105,896,555	3,738,733,648	web graph

Figure 3: Applications & Datasets

while UK-Web serves as an example of a real-world web graph with 100 million vertices and 3 billion edges. For each experiment we measured the total amount of data being transferred over the network, the length of each superstep interval in the application, and for each interval the amount of time each application spent in the computation stage and communication stage.

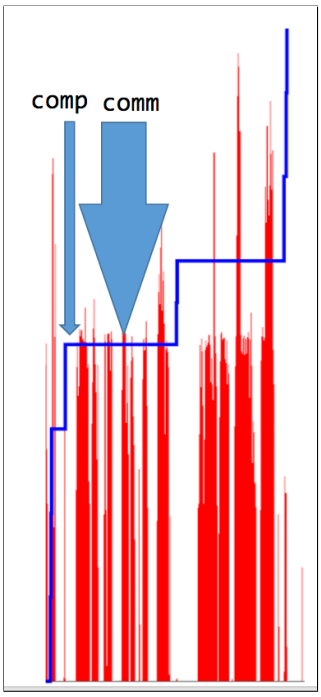

Figure 4: SSSP-Orkut

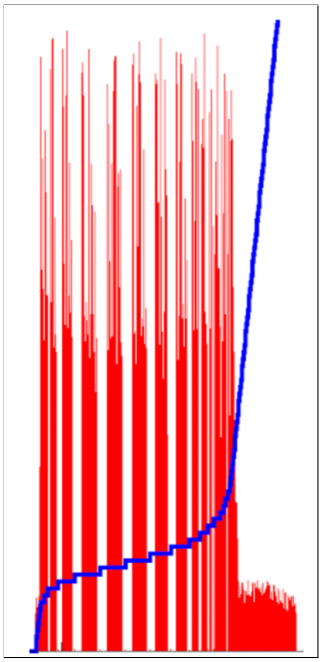

Figure 5: SSSP-UK-Web

4.2 Results

In this section we shall explain the results of our experiments shown in figures 4-9. For all charts the blue lines represent the length (in time) of each superstep interval for the applications and the red bars correspond with the amount of data in bytes transferred throughout the network over time. From these charts we have deduced that there are two stages in each interval. The first is the computation stage where each machine performs its vertex operations specified by the Pregel+ application. This is represented by the gap between each block of traffic data. The second is the communication stage where each machine in the cluster passes messages

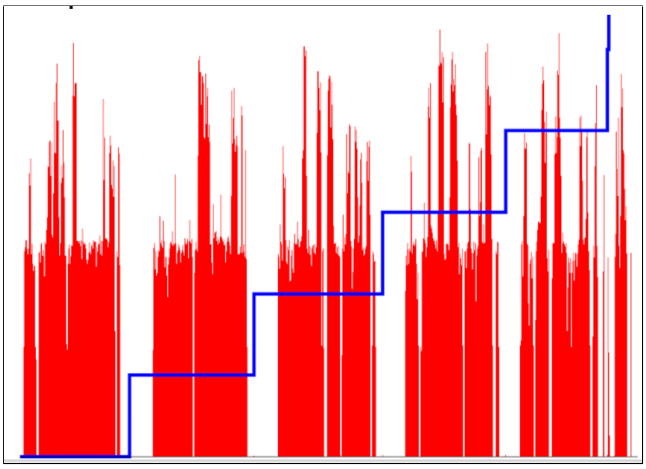

Figure 6: Connected Components-Orkut

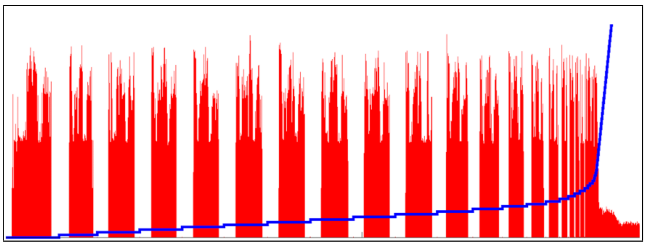

Figure 7: Connected Components-UK-Web

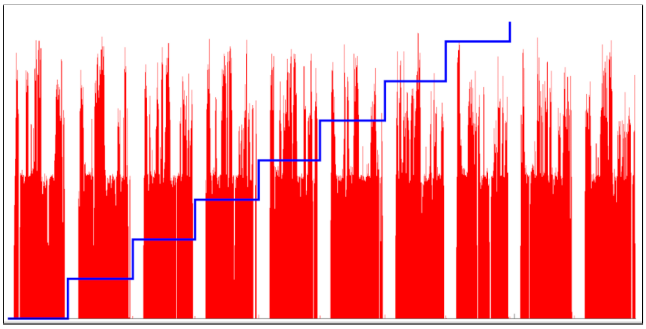

Figure 8: PageRank-Orkut

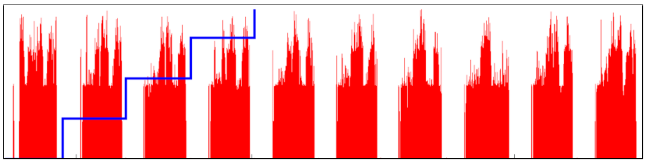

Figure 9: PageRank-UK-Web

to each other between the supersteps. Here it is inferred that the amount of traffic data corresponds to the number of vertices processed in the interval.

Figure 4 depicts the traffic data and interval lengths of the SSSP algorithm on the Orkut social network graph. The traffic for this case starts off with short intervals followed by a series of longer intervals and fin-

ishes with short intervals again. This pattern is attributed to the design of the SSSP application. This is because SSSP starts from a single vertex and expands the set of vertices being processed via breadth-first search. Thus as time advances the application will be processing the most amount of vertices in the middle of its run time since that is when the diameter of the graph is reached. Figure 5 further exemplifies this traffic pattern for SSSP on the larger dataset, UK-Web. Because the data set is significantly larger we see a much smoother curve but it still exhibits the same pattern that SSSP shows for Orkut.

Figures 6 and 7 display the traffic patterns for the Connected Component application on Orkut and UK-Web. The traffic pattern for both charts is very similar in that the length of each interval slowly decreases until the algorithm has terminated. This again due to the pattern of the Hash-Min algorithm used where each vertex is given a numerical label which gets hashed and propagated in each iteration so that each vertex will adopt the minimum label found its neighborhood. Thus upon each iteration the set of vertices being processed decreases a bit and by the end of the algorithm all of the nodes with the same label are in the same connected component. For further details on the Hash-Min connected component algorithm please see [9].

Similarly, figures 8 and 9 exhibit the same patterns as well for PageRank on the different graph sizes. The PageRank traffic patterns show a consistent interval and communication lengths and this is because each iteration of PageRank processes the entire graph at once and terminates after a designated amount of time.

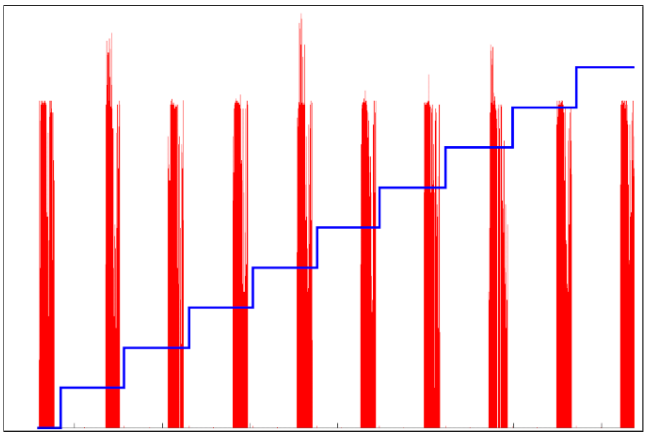

Figure 10: PageRank on UK-Web between m4 and m2

4.3 Machine to Machine Traffic

Upon observing the overall network traffic for the cluster with each experiment we decided to observe the individual machine to machine traffic per application. When looking at the machine to machine level traffic it can be seen that the computation stage takes up the most time. But this can be attributed to the fact that the Pregel+ is iterative and waits for all the computation from Machine 1 to its neighbors to complete, and then from Machine 2 to its neighbors, and so on. This pattern is known as sequential shuffling.

4.4 Findings

Several findings and opportunities for improvement were found after all of these experiments. The most obvious finding is that traffic shape is mainly determined by the application in use. However, further analysis shows that many of the data packets within the communication stage of each interval are duplicates. Thus causing communication to be a bottleneck for performance. Another observation is that network bandwidth is not fully utilized which is caused by the sequential shuffling described in 4.3. An attempt to remove the communication bottleneck yielded significant performance improvements. By removing duplicate packets between machines the run time of the applications was reduced greatly.

5. RELATED WORK

Similar work on analyzing big data analytics systems have been done. For example, [6] experimentally compared Giraph, GPS, Mizan, and GraphLab using several different metrics and algorithms. The first three are Pregel-based BSP (Bulk Synchronous Parallel) systems whereas GraphLab is based off of the open sourced PowerGraph and follows the GAS (Gather, Apply, Scatter) model for computation. The algorithms Han et al. tested included PageRank, SSSP, WCC, and DMST. They measured each graph system by its computation run time, setup time, memory usage, and network I/O. Overall, they found that Giraph's computation time's are longer than GPS's and GraphLab's for PageRank but has a much shorter setup time making Giraph's total time comparable to the other systems. Giraph ends up being faster than GPS for PageRank, SSSP, and WCC and is slower than GraphLab for PageRank. In terms of network usage, Giraph had the lowest for SSSP and WCC while GraphLab had the lowest for PageRank. For DMST, Giraph ran considerably slower than GPS when using byte arrays however when using hashmaps it has much faster computation and setup times. The Mizan system performed poorly across all experiments which led to the conclusion that features such as Mizan's dynamic migration should only be developed after built-in system optimizations have been made. The authors also compared their experiences with the systems in terms of development support and usability. They found Giraph to be the easiest system to understand and GraphLab to be a close second since they are well documented and have active developer

communities. As opposed to GPS or Mizan for they are not as well-documented. In terms of usability, GPS has a similar API to Giraph making it easier to port code despite the poor documentation. Criticisms of Mizan were due to it being an experimental system and still having bugs.

Lu et al. [7] also conducted a similar study to Han et al. [6] but differed in the number of graph computation systems they studied as well as the algorithms they used. The systems they compared were Pregel, Giraph, GPS, Pregel+, GraphLab, and GraphChi. They ran experiments on the following algorithms: PageRank, Diameter Estimation, SSSP, HashMin, Shiloach-Vishkin, Graph Coloring, and Bipartite Maximal Matching. They evaluated the performance of systems w.r.t algorithmic categories, graph characteristics, and performance of a distributed system as opposed to a single machine. Performance was measured by the total run time of each experiment. They found that GPS and Pregel+ have the best performance for always-active algorithms such as PageRank and Diameter Estimation while GraphLab outperformed Giraph and Giraph would run out of memory for diameter estimation. For graph traversal algorithms such as HashMin and SSSP Pregel+ outperformed all of the other systems, followed by GPS with Giraph and GraphLab at the end. Pregel+ also outperformed all other systems for multi-phase algorithms and graph mutation algorithms. The authors observed that Giraph and GPS are implemented in Java while Pregel+ and GraphLab are implemented in C++ which could be a reason for the performance gap. It was noted that the Java-based systems ran out of memory for very large graphs w.r.t Diameter Estimation. Pregel+ also demonstrated the best results on graphs of various natures such as skewed degree graphs, large diameter, small diameter, and high average degree graphs. GraphChi was compared separately to Pregel+ because it is single-machine based and similar to the other results, Pregel+ outperformed GraphChi. The authors also looked at the effects of algorithmic optimizations and dynamic repartitioning of the graphs which ultimately do not offer significant performance gains.

6. CONCLUSION

This paper presents a study of the network traffic patterns of Pregel+ running three different graph applications, Connected Components, Single Source Shortest Path, and PageRank. The applications were run on two real-world large scale graph data sets, Orkut, a social network that was owned and operated by Google from 2004 to 2014 and UK-Web, a web graph formed by a crawl of the .uk domain taken at the end of 2014. Overall we made three significant observations. 1) The shape of network traffic is highly dependent on the application in use. 2) Because many packets are duplicated in the communication stage of the applications' intervals, the communication stage served as a bottleneck for performance. 3) Sequential shuffling of data between intervals indicated that network bandwidth is not fully utilized. By removing duplicate packets we were able to improve upon observation 2) which demonstrated improvements in performance.

7. REFERENCES

[1] Avery Ching, Sergey Edunov et al. One Trillion Edges: Graph Processing at Facebook-Scale. VLDB'15.
[2] Da Zheng, Disa Mhembere et al. FlashGraph: Processing Billion-Node Graphs on an Array of Commodity SSDs. FAST'15.
[3] NSDI'15
[4] Scott Beamer, David Patterson et al. Graph Processing Workload Characterization. http://gap.cs.berkeley.edu/chara
[5] Frank McSherry and Malte Schwarzkopf. The impact of fast networks on graph analytics. http://www.frankmcsherry.org/pagerank/distributed/performan
[6] Minyang Han, Khuzaima Daudjee, et al. An Experimental Comparison of Pregel-like Graph Processing Systems. VLDB'14
[7] Yi Lu, James Cheng, et al. Large-Scale Distributed Graph Computing Systems: An Experimental Evaluation. VLDB'15
[8] Grzegorz Malewicz, Matthew H. Austern, et al. Pregel: A System for Large-Scale Graph Processing
[9] Vibhor Rastogi, et al. Finding Connected Components in Map-Reduce in Logarithmic Rounds