Algorithms for Cost-efficient Network-aware Scheduling of Large-scale Graphs in Cloud Computing Environments

Safiollah Heidari and Rajkumar Buyya
Cloud Computing and Distributed Systems (CLOUDS) Laboratory
School of Computing and Information Systems
The University of Melbourne, Australia

Abstract

Large amount of data that is generated by Internet and enterprise applications is stored in the form of graphs. Graph processing systems are broadly used in enterprises to process such data. With the rapid growth in mobile and social applications and complicated connections of Internet websites, massive concurrent operations need to be handled. On the other hand, the intrinsic structure and the size of real-world graphs make distributed processing of graphs more challenging. Low balanced communication and computation, low preprocessing overhead, low memory footprint, and scalability should be offered by distributed graph analytics frameworks. To meet these requirements, we proposed two dynamic re-partitioning algorithms that consider network factors affecting public cloud environments to decrease the monetary cost of processing. A new classification of graph algorithms and processing is also introduced which will be used to choose the best strategy for processing at any operation. We plugged these algorithms to iGiraph and compared with systems such as Giraph and Surfer on Australian NECTAR Cloud. We observed up to 30% faster execution time, up to 50% network traffic decline and 3X-5X cost reduction is achieved by our algorithms.

Keywords: Graph processing; cloud computing; network-aware processing; cost saving

1. Introduction

Today many applications in domains such as the Internet, astronomy, social networks, information retrieval and particle physics are experiencing data flood and they have already reached peta-scale volume of data [1]. The growth in the volume of data needs large computing power to turn the original data into worthwhile insights. Nevertheless, massive amount of data is saved and modeled in the form of graphs. These graphs provide valuable sources of information for several applications. For instance, by studying social networks and the way that relationships are shaped between users, psychologists and sociologists can investigate their assumptions and hypothesis about people and communities. Analyzing web graphs can make search engines more accurate and effective [1]. By detecting social circles and their influential members in social networks, politicians can spread their thoughts in these communities [2]. Therefore, processing large-scale graphs and unveiling attributes of those graphs have become critical requirements.

Traditional approaches of processing Big Data such as MapReduce [3] are not suitable for graph processing because of the intrinsic behavior of graph algorithms. For example, MapReduce has a two-phase computation model – Map and Reduce, which is not exactly appropriate for the iterative nature of graph algorithms. It also does not retain the input graph and its states in main memory across these two phases and is very inefficient because of requiring repetitive disk I/O.
Many research works on large-scale graph processing frameworks concentrate on the platforms based on commodity clusters. However, not many studies have been done on cloud platforms, particularly public clouds [4]. Cloud computing is a model of computing that has modified hardware, software and datacenters implementation and design [5]. It has brought novel technologies and economical solutions such as elasticity and pay-as-you-go models by which service providers do not need to worry about previous obstacles of delivering services to their clients. Public cloud services are getting more popular than other cloud services such as private, hybrid and community clouds especially among small and medium size businesses. It is because they do not have sufficient funding to have their own private cloud or it is not efficient for their business models. So, public cloud is a true response to their needs. Another important feature of public clouds is the monetary modeling that different service providers offer to their customers. Amazon, for example, has three cost models: spot, on-demand and reserved provisioning models for providing resources. Using these commercial services, the client might select to pay more to get higher performance or better reliability. So, the challenge with using public clouds is making the right decision between utilizing the number of resources that the user needs and the amount of money he/she can pay for the service. In this research we only consider the reserved model.

Another less studied aspect of graph processing systems is the impact of the network environment on the performance of the whole system. Some systems such as Surfer [6] and Pregel.Net [7] are implemented to support graph processing on public clouds. Although, they consider some network features, none of these systems have explored the effects of provisioning and processing on monetary cost. For instance, Surfer proposes a graph partitioning approach based on the network bandwidth and claims that it could improve the performance. On the other hand, to the best of our knowledge, all existing graph processing frameworks –except iGiraph [4] - concentrate on decreasing the processing runtime, memory utilization and so on to degrade the cost of operation. They only take an unchanged pool of resources with known size into consideration. It means that all existing systems start and finish their computation with the same number of resources (machines). So, in many cases, idle machines have to wait for other busy machines to finish their jobs and all machines be released together which is a waste of money and time. These issues can be truly solved using the levels of scalability and elasticity that is provided by cloud computing.

In this paper, we propose a new network-aware graph processing approach considering both performance and monetary cost of the whole computation. We introduce a novel dynamic re-partitioning method that utilizes different factors including: a) the type of the graph application that is going to be used, b) some intrinsic features of natural graphs such as high-degree vertices, and c) the network features of the cloud environment that the system is running on.

Our work makes the following contributions:

- A new classification of graph applications and processing is introduced in this paper which affects the policy that will be chosen to process the input graph. We have studied the impacts of combinations of different situations from this classification together on processing large-scale graphs on public clouds for the first time and reduced the monetary costs in each situation.

- A novel mapping strategy is designed to facilitate assigning partitions to the workers based on different features that each partition and worker has.

- A new bandwidth-and-traffic-aware dynamic re-partitioning algorithm and a new computation
aware re-partitioning algorithm have been proposed in this paper. These algorithms remarkably reduce the monetary cost of processing - which is a vital factor in the procedures of selecting services for any customer on a public cloud.

The rest of the paper is organized as follows: Section 2 explains the related work. Section 3 presents our background work - iGiraph which is a graph processing framework based on Giraph. A new classification of graph algorithms is explained in Section 4. Section 5 and 6 introduce our new proposed bandwidth- and traffic-aware and computation-aware dynamic re-partitioning algorithms of large-scale graphs respectively, with their implementation on iGiraph. We explain the architecture and details of our system (iGiraph-network-aware) in Section 7 followed by a discussion on the evaluation of our work in Section 8. Finally, Section 9 concludes the paper and proposes future works.

2. Related Work

To overcome the issues on traditional processing approaches, considerable endeavors are made to process large graphs. Some proposed systems try to process the entire graph on a single server whereas the main problem of this method is scalability [8]. However, the utmost size of graph to be processed is restricted by the single host’s memory in which the input graph has to be fully loaded. In addition, this method cannot use the strength of other hosts in terms of distribution and parallelization, to reduce the processing time. Another method is to utilize libraries that allow graph algorithms to be executed in parallel in the shared memory approach [9]. This method, tries to solve the issue of the previous method. However, it still has problems with fault-tolerance and scalability [10]. Another way of processing graphs is to adopt graphic processing units (GPU) to accelerate different graph processing tasks. In sampling method, the input graph will be divided into several sub-graphs by the system and then the attribute of the main graph will be estimated based on the attributes of the smaller sub-graphs. The major issue in this method is that there is a big distinction between the actual and estimated solutions.

Unlike the aforementioned methods, a distributed method utilizes a commodity of servers as a generic solution to performance, scalability, and availability issues [11]. This can be specifically utilized for solving large graph problems. Pregel [12], which was proposed by Google in 2010, is a computational model dedicated for processing large-scale graphs. The main inspiration for Pregel is the Bulk Synchronous Parallel (BSP) model [13] which streamlines the implementation of distributed graph algorithms. A program in Pregel contains sequences of iterations called superstep. Within a superstep, a user-defined function called Compute() is invoked by Pregel for each vertex that specifies the conduct of the node in the superstep. The Compute() reads messages that have been sent to the related node during the prior iteration, applies some processing and dispatches messages to other nodes, that will be collected at the next superstep. This function can also change the states of vertices and their outgoing edges. Pregel uses supersteps to accomplish fault tolerance and high scalability in a cluster of machines. Nevertheless, this might be an impasse for performance when the amount of communications grows in a graph with vertices in millions-scale. Many distributed graph processing frameworks have been introduced after Pregel. Systems such as Giraph [14], Apache Hama [15], ExPregel [16], GPS [17], GraphLab [18] and iGiraph [4] which have been developed based on Pregel are called Pregel-like systems. There are also other frameworks that are not developed based on Pregel.

Pregel-like frameworks are developed based on a distributed architecture in which one machine will act as the master while other machines will be workers (slaves) and do the computation. In this approach, the
input graph is split into partitions and partitions are assigned to workers by the master to be processed. Therefore, partitioning a graph is a critical job and since it has a direct influence on the performance of the system, various methods have been proposed for achieving better outcomes. A vast majority of graph processing systems propose some determined improvements on high performance computing clusters with fast interconnects. However, their behavior on cloud computing that provides virtualized commodity hardware and is available to a broader crowd of users is less investigated [4].

Despite introducing various partitioning methods by different frameworks, the impact of network factors on the system’s performance and the way that they can be used to optimize or improve the processing is not sufficiently studied. Surfer [6], is the closest framework to our proposed system. But according to the earlier discussion, it has many shortcomings and does not cover many aspects of network bandwidth; particularly its mapping strategy is not quite efficient. Another system that considers network traffic is Pregel.Net. Pregel.Net [7] is implemented based on Pregel but over .Net framework. It has used Microsoft Azure to analyze the impact of BSP graph processing model on public clouds. However, it does not investigate if its changes will affect the monetary cost of the operation.

In another research [19], authors have shown that the network does not have a significant impact on the processing and the highest impact that any optimization solution can bring to graph processing system’s performance would be something between 2%-10%. GraphX [20] and Spark1 were used in that experiment and some network factors such as the speed of the network was studied in different situations. However, McSherry [21] showed that this assumption is completely wrong and many other factors have been missed from the study. He showed that using a dataflow framework can achieve much better results to 2X-3X compared to GraphX. This study and ours in this paper imply that there are still many features that can be taken into consideration and be mixed with novel solutions to leverage the impact of network to reach better performance.

3. Background – iGiraph

iGiraph [4] is a Pregel-like graph processing system that has been developed based on Giraph. Giraph itself is an open-source version of Pregel and is broadly used by big companies such as Facebook [22] for processing their large graph data. iGiraph has a distributed architecture which is implemented on top of Hadoop [23] and utilizes its distributed file system (HDFS) for data input/output. It employs a vertex-centric programing model similar to Giraph in which every vertex of the graph is identified by an exclusive allocated ID. It can also carry on more information such as a set of edges with an edge value for each edge, a vertex value and a set of messages sent to it.

iGiraph uses BSP computation model. In this model, each vertex might have either active or inactive state. When the processing starts, all the vertices of the graph are in their active mode and as the process continues, they might change their state to inactive. In each superstep during the computation using BSP model, each vertex that is involved in the processing, 1) obtains its neighbors’ new values from previous superstep, 2) updates its own value using the received values, 3) sends its updated value to its neighbors that will be accessible to them at the beginning of the next superstep and modifies its mode to inactive. A global synchronization barrier determines the end of each superstep. At any time, a vertex can be deactivated by calling VoteToHalt() function, if it does not receive any messages during an iteration. If a

1 http://spark.apache.org/
deactivated vertex receives a message from any of its neighbors, it will be activated again. The computation will be completed when there is not any active vertex left.

There is a difference between iGiraph and other existing graph processing frameworks including Giraph. In addition to proposing approaches to execute the processing faster and enhance the performance of the system, iGiraph also proposes solutions for the less studied part of such systems on cloud environments which is monetary costs of resource utilization. Nonetheless, cost is a pivotal element for every business that aims to utilize public cloud infrastructure. As cloud providers are using pay-as-you-go models for their services, considering the elements that influence the cost of the services is very important for clients to select the right services. The whole idea in iGiraph is that while other systems are using the same amount of resources during the processing period, it is also possible to reduce the number of resources that are idle or are not necessary to be kept for future computations (as illustrated in Figure 1).

The method that is used by iGiraph is a dynamic re-partitioning method which will be applied to the computation at every superstep based on the type of the graph algorithm that is used. It classifies graph algorithms into two categories: 1) convergent algorithms such as shortest path or connected components algorithms that will converge as the computation progresses, 2) non-convergent algorithms such as PageRank. When the system is processing the data using a convergent algorithm, the vertices that are processed and have changed their state to inactive, will be removed from the memory at the end of each iteration. Hence the graph is getting smaller during the processing. This means that the remaining graph may be fitted in less number of machines and we can terminate the machines that are not needed anymore. In non-convergent algorithms vertices are always in their active mode, so we cannot terminate machines. Instead, using the concept of high-degree vertices helps the computation to be executed faster and with less communication cost.

In a natural graph, a high-degree vertex is a vertex that has bigger degree (more links) than other vertices in the graph. For instance, in a social network, each member of the network is represented by a vertex and the relationship between two members is represented by an edge in the graph. In this graph, a celebrity or a president of a country is a high-degree vertex because they are usually followed by many other vertices (members) on social networks. When high-degree vertices are located at the border of a graph partition –
it means they have adjacent vertices on other partitions – this causes a very high communication cost by sending and receiving messages to/from its neighbors. This is due to the number of adjacent vertices that it has that is more than many other vertices in the graph. iGiraph utilizes high-degree vertices concept in both vertex and partition levels. It brings these types of vertices closer to their neighbors so that it reduces the communication cost by decreasing the number of messages that need to be passed through the network. In other words, it reduces the cost by reducing the network traffic.

In this paper, we extend iGiraph to support more network factors for its dynamic re-partitioning approach by providing a novel priority mapping solution to customize each machine for each partition. According to this solution, we provide a ranking method for this mapping. To distinguish between basic iGiraph and our proposed network-aware system in this paper, we refer to the new system as “iGiraph-network-aware” for the rest of the paper.

4. Processing Environment Categorization and Graph Applications

iGiraph utilizes a one dimension categorization of graph applications which divides them into convergent and non-convergent. In this work, we extend this categorization into two-dimensions by adding an extra layer. Figure 3 shows the new categorization for all sorts of processing where any kind of application can be either computationally-intensive, communicationally-intensive or a combination of them.

- **Computationally-intensive processing**: This type of processing often has a large impact on CPU utilization because it spends more time on computing than communication and the memory side. Sometimes the graph processing application itself is computationally intensive and sometimes other applications keep the CPU busy in VMs and the graph application has to find a way to be processed faster. This situation happens mostly in a public cloud environment.

- **Communication-intensive processing**: This type of processing usually has a big impact on network and memory especially when an application needs to keep the intermediate states of a computation.
In this paper, we utilize two typical algorithms (convergent and non-convergent) to show the impacts of each algorithm on both types of processing. Here, we give a brief description of sample algorithms that we are going to use for our experiments.

1. **PageRank**: PageRank algorithm was proposed to weigh the importance of web pages and websites by calculating the number of links connected to them. The more hyperlinks the page gets from other websites, the more significant the page is. PageRank assesses every page individually and will not weigh the whole website as a unit. In this algorithm, the importance of a typical web page will not be affected by the PageRank of other pages because each page has its own exclusive approximated weight. According to the categorization we presented in this section, PageRank is a non-convergent algorithm due to generating a constant number of messages in each iteration during the processing.

2. **Single source shortest path**: The aim of solving the shortest path problem is to find a route between two nodes in a graph while the sum of the weights of its edges is minimized. Shortest path is a famous problem in graph theory and various approaches have been suggested to solve it. Single-source-shortest-path (SSSP) problem is a special case of the original shortest path problem. SSSP is about discovering the shortest route between a typical source vertex and all other nodes in the graph. Before SSSP starts, the values (distance) of all vertices are set to INF (∞) except the source vertex which is set to 0. Any possible route from the source vertex in the graph will be shorter than INF. During each superstep, vertices receive messages from their adjacent nodes, update their value using the minimum value received from their neighbors and send any recently found minimum value to all neighbors. In the initial iteration, only the adjacent vertices of the source node will be updated. In each superstep, the updated nodes will send their new values to their neighbors until the computation ends. The processing finishes when the status of all nodes in the graph is changed to inactive and no more updating happens. According to this definition, SSSP is categorized as a convergent algorithm.

5. **Bandwidth-and-Traffic-aware Dynamic Re-partitioning**

Assume that the average amount of network traffic sent along each cross-partition is \( N_{M}(P_i, P_j) \), the networks bandwidth between the machines stored \( P_i \) and \( P_j \) to be \( B_{i,j} \), and \( C(P_i, P_j) \) to be the number of cross-partition edges from partition \( P_i \) to \( P_j \). Because network bandwidth is a scarce resource in the cloud environment, it is considered as the major index for network performance. So, the approximate data transfer time (DTT) from \( P_i \) to \( P_j \) will be as follows:

\[
DTT(i,j) = \frac{C(P_i, P_j) \times N_M (P_i, P_j)}{B_{i,j}}
\]

This estimation is adequate for large-scale graph processing in both public and private cloud environments. Suppose we have stored \( P \) graph partitions on \( P \) disparate machines; the overall data transfer time (DTT\(_{\text{Total}}\)) in the network caused in all partition pairs is as follows:

\[
DTT_{\text{Total}} = \sum_{i=0}^{P-1} \sum_{j=0}^{P-1} DTT(i, j)
\]

Obviously if network bandwidth amongst different machine pairs is constant, the total network data transfer time will be minimized when the total number of cross-partition edges is minimized.
Nevertheless, the network bandwidth amongst different machine pairs can change remarkably in the cloud. Cloud providers have noticed such network bandwidth unevenness. The network bandwidth of every machine pair amongst 64 and 128 small Amazon EC2 instances is shown in Figure 4. On the other hand, research shows that in public cloud, the network bandwidth between two instances is provisionally steady. This allows us to perform our mapping calculation before each superstep.

Because of the network bandwidth unevenness, an important factor for an efficient graph processing is the mechanism of partitioning the graph and storing its partitions on the VMs. According to [15], because there might be a large number of partitions and workers for processing the graph, there is P! possible ways to store partitions on workers which is a huge solution space. Another issue is finding a solution by which both graph processing and graph partitioning algorithms can be aware of the bandwidth variability for networking efficiency.

![Network bandwidth unevenness in Amazon EC2 small instances with (a) 64 instances and (b) 128 instances](image)

Fig. 4. Network bandwidth unevenness in Amazon EC2 small instances with (a) 64 instances and (b) 128 instances [6]

To address these issues in a public cloud environment, a new dynamic re-partitioning method is proposed in this paper. The idea is to place the partitions with larger number of high-degree border vertices – which means they have larger number of cross-partition edges – on workers with higher network bandwidth. This is because those graph partitions need more network traffic. It also helps the partitions to be processed faster.

To achieve performance improvement, we implemented a mapping strategy (illustrated in Figure 5) in iGiraph. The processing starts with a random partitioning approach as we use this method for all our experiments to start with. This is because random partitioning is shown to have the worst performance among most of the existing well-managed partitioning approaches. So, we aim to improve this situation as the cheapest implementing strategy which is not good performance-wise. According to this strategy, the first iteration (superstep 0) starts with a random partitioning method, the processing of the iteration completes and the global synchronization barrier occurs. Before going to the next superstep, we use the information we collected from the first iteration to plan a new partitioning (re-partitioning) for the next iteration.

![Mapping strategy for 5 partitions and 5 workers](image)

Fig. 5. Mapping strategy for 5 partitions and 5 workers
After the completion of the first superstep, each partition is assigned a factor called Partition Priority (PP). The partition with PP=0 is the one that receives the larger number of messages over the network when compared to other partitions. In other words, this partition contains more high-degree border vertices than other partitions. It is also a candidate for being merged with other partitions or its vertices being migrated to other partitions. All other partitions also get their own PP which shows their importance based on the amount of network traffic they generate. On the other hand, each worker also will be assigned a factor called Worker Priority (WP). The worker with WP=0 is the one with the highest bandwidth among all workers (machines). All other workers also will be given their own WP based on their bandwidth rating in the network. In case in which two or more partitions have the same priority after calculation, one of them will get the higher PP randomly. The same logic also applies to workers. After assigning PPs and WPs to partitions and workers respectively, the partitions with specific PPs will be assigned to the workers with the same WPs. The calculations and assignments are done after each superstep \( i \) and before each superstep \( i+1 \).

**Algorithm 1: Bandwidth-and-traffic-aware dynamic re-partitioning**

1: Partition the graph randomly
2: Set \( PP=0 \) for each partition and \( WP=0 \) for each worker
3: For the rest of the computation do
4: Calculate \( PP \) for each partition based on the number of messages that each partition receives
5: Calculate \( WP \) for each worker using end-to-end mechanism
6: If global synchronization happened then
7: Merge the partitions or migrate vertices if needed
8: Set the priorities based on \( PP \) and \( WP \)
9: Map partitions(based on \( PP \)) and workers(based on \( WP \))
10: If VoteToHalt() then
11: Break

Another issue that should be considered is the time when the priority setting should be done. Due to the possibility of merging or removing the partitions after each superstep, the priority setting is done after these operations, immediately before the next iteration starts. Therefore, the partitions that have received migrated vertices will be given the highest priorities. This is because the reason for vertex migration is to bring high-degree vertices closer to their neighbors. If there is more than one partition receiving migrated vertices, the one that has got more migrated vertices will get the highest priority and so on. Also for the partitions that get merged, the priority of the final partition (combined partition) will be set as the priority of the partition with highest priority (its priority from the previous iteration). At the beginning of the processing (superstep 0), all partitions’ priorities will be set to 0 (highest priority).

According to our experiment results, using a mapping strategy that assigns partitions to workers based on the traffic in the network and the bandwidth capacity of workers, combined with iGiraph’s re-partitioning method (for both convergent and non-convergent types of algorithms) gives much better results compared to previous solutions. These results would be in regard to reducing the monetary cost of the processing by reducing the cost of resource utilization, reducing network traffic and accelerating the execution time of the whole process.

**6. Computation-aware Dynamic Re-partitioning**

Although many graph algorithms are communication intensive, computation unit can still affect the execution of applications. In a public cloud, each VM can host different applications at the same time.
Some applications might be computation-intensive and keep the CPU busy while other applications are not very CPU-dependent but still can be affected by the former. Computation-intensive algorithms or applications can delay the computation and execution time of others.

Various approaches can be applied to deal with such situations. For example, each job can have a different priority by which the host can schedule the computation time for that. There are many prioritization strategies such as first-in-first-out, first-in-last-out, assigning priority numbers to tasks, etc. Another approach for when there is no priority or preference for job execution can be using equal time-slots for computing jobs in an interwined way.

**Algorithm 2: Computation-aware dynamic re-partitioning**

1: Partition the graph randomly
2: \textbf{Set} $PP=0$ for each partition and $WP=0$ for each worker
3: \textbf{For} the rest of the computation \textbf{do}
4: \hspace{1em} Calculate $PP$ for each partition based on the number of messages that each partition receives
5: \hspace{1em} Calculate $WP$ for each worker using CPU utilization on each worker and CPU idle time
6: \hspace{1em} \textbf{If} global synchronization happened \textbf{then}
7: \hspace{2em} Merge the partitions or migrate vertices based on $machineType$ if needed
8: \hspace{1em} \textbf{Set} the priorities based on $PP$ and $WP$
9: \hspace{1em} Map partitions(based on $PP$) and workers(based on $WP$)
10: \hspace{1em} \textbf{If} VoteToHalt() \textbf{then}
11: \hspace{2em} \textbf{Break}

We propose a similar \textit{mapping strategy} as we discussed for traffic and bandwidth-aware re-partitioning, but we consider CPU utilization instead of bandwidth in the algorithm and re-partition the graph differently. We have implemented this strategy on iGiraph. As in last section, the computation starts with a random partitioning for superstep 0. At the end of superstep 0 when the global barrier happens –before superstep 1- we use the information we have got so far to initiate the re-partitioning.

At this stage, on one side based on the number of messages passed between workers through the network, we define Partition Priority ($PP$) again by which the partitions with high-degree vertices can be recognized. On the other side, a scalable monitoring tool called Ganglia\textsuperscript{2} is used to monitor the CPU utilization on each worker. Therefore, the information regarding the computational conditions of all machines will be written and saved on a separate file on the master machine. The information include the percentages of CPU idle times at the end of each superstep so that it can be possible to find which machines are still busy, or how busy they are, and which one is free and ready to use. The reason for choosing the CPU idle time to use in the algorithm instead of CPU working time is that the former is more reliable. There might be situations that a very small task can use most of computation resources for a short time and increase the utilization percentage remarkably but the reality is that the CPU will be idle the rest of the time. From this information, a map of available computation resources can be depicted which will be used for dynamic re-partitioning during the rest of computation. Figure 6 shows the computation map of a system with 15 workers where some random computation-intensive applications are running on some machines.

\footnote{\url{http://ganglia.sourceforge.net/}}
According to the aforementioned strategy, there will be four types of machines in the environment after the first superstep: 1) a machine with both a computation-intensive application and high-degree vertices of graph dataset on it, 2) a machine with computation-intensive application running on it but the graph partition that has been assigned to that does not have high-degree vertices, 3) a machine with a partition containing high-degree vertices but no computation-intensive application on it, and 4) a machine that has neither computation-intensive application running on it nor the partition that have been assigned to it has any high-degree vertices.

The idea is to move high-degree vertices with their neighbors to the machines that have higher CPU idle time. This is because more computation is needed to be done on these vertices in terms of the number of messages they receive. So the algorithm would be like this: partitions in machine type 1 need to be migrated to or merged with partitions on machines type 4 or 2 respectively. Partitions on machine type 2 can be merged with the one on type 3 and 4. Partitions on machine type 3 can be migrated to type 4 or be merged by partitions on machines type 2. Based on this algorithm, at the start of the processing, all workers have their types set as 0 which will change after the first superstep. Then, at the end of each superstep, this algorithm will re-partition the graph and assign the proper partitions to their best worker. iGiraph-network-aware also considers the available memory on the destination before moving the vertices.

As will be shown in next sections, our experiments prove that under the equal situation, the computation-aware re-partitioning on iGiraph (iGiraph-network-aware) significantly reduces the execution time of the entire processing compared to the original Giraph. It is also shown that this approach can reduce the monetary cost of the processing for both convergent and non-convergent types of applications.

7. System Design and Implementation

Figure 7 shows the design of our proposed software system and the components that we have added to iGiraph. The architecture and placement of different components of our system is shown in Figure 8.
7.1. Bandwidth Measurement
Bandwidth measurement component is implemented on all machines in the system to be able to calculate the bandwidth between workers by an end-to-end calculation mechanism which is used in [24].

7.2. Traffic Measurement
To calculate the network traffic between each pair of machines, the traffic measurement module is implemented and installed on all workers. It basically works based on the number of messages transferring between machines. Using this information, the system ranks the most congested paths and uses that for partitioning purposes.

7.3. CPU Measurement
As part of a network characteristic, CPU workload shows the amount of computations occurring on each machine and in the whole network. In a public cloud, there may be different jobs running on each machine at the same time and some of these jobs might be computation-intensive. By knowing how busy each worker in the network is, we can avoid overloading occupied workers by assigning more tasks to them. This module uses the information it gets from Ganglia- a tool by which we can measure many specifications of a network- to calculate the CPU idle times per worker.

7.4. Policy Selector
Policy selector is a component of iGiraph which we have expanded to cover our network-aware scheduling algorithms. Using this component, the user can choose between two policies that can be bandwidth-aware or computation-aware.
7.5. Network KPI Aggregator
The network KPI aggregator is implemented on the master to aggregate the information from all workers and pass them to the next component for partitioning decision making. Having this component as an independent module that gathers all information in one place helps to reduce the burden of workers and make the execution faster.

6.6. Re-partitioner
The re-partitioning component partitions the graph again based on the information that has been gathered from other parts of the system. Since the system utilizes a synchronous approach for execution, re-partitioning happens after each superstep and before the next superstep begins. We will show that using these components and the new re-partitioning strategy, the performance of the system will increase significantly compared to similar frameworks such as Giraph and Surfer.

8. Performance Evaluation

8.1. Experimental Setup
We use m1.medium NECTAR VM instances for all partition workers and the master role. NECTAR [25] is the Australian national cloud infrastructure facilities. Medium instances have 2-cores with 8GB RAM and 70GB disk including 10GB root disk and 60GB ephemeral disk. All the instances are in the same zone and use the same security policies. We also installed NECTAR Ubuntu 14.04 (Trusty) amd64 on each instance. We implement our algorithms for iGiraph [4] (our extended version of Giraph system) with its checkpointing characteristic turned off. To distinguish between the original iGiraph and the current work, we refer to the new system as “iGiraph-network-aware” for the rest of the paper. We also use Apache Hadoop version 0.20.203.0. All experiments run using 16 instances where one takes the master role and others are set up as workers.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon (TWEB)</td>
<td>403,394</td>
<td>3,387,388</td>
</tr>
<tr>
<td>YouTube Links</td>
<td>1,138,499</td>
<td>4,942,297</td>
</tr>
<tr>
<td>Pokec</td>
<td>1,632,803</td>
<td>30,622,564</td>
</tr>
</tbody>
</table>

We chose shortest path and PageRank for communication-bound convergent and non-convergent algorithms respectively. We also use three real datasets of different sizes: Amazon, YouTube and Pokec [26] and properties of these datasets are shown in Table 1.

8.2. Results
To evaluate the proposed algorithm we chose Giraph as a popular graph processing framework to compare the performance of our system with. We also implemented the bandwidth-aware graph processing method proposed by Surfer on Giraph to use it as another baseline. Although Giraph has been improved since Surfer was developed, the implemented algorithm still shows Surfer’s behavior on the network. In addition, the size of messages in all experiments is the same. Therefore, the communication cost is independent from message size and is calculated based on the number of messages that are transferred through the network.
The first group of experiments is carried out for communication-intensive scenarios. Most graph processing applications are classified in this category. As the results show, iGiraph-network-aware could achieve better performance compared to Giraph and Surfer on both convergent and non-convergent applications. Both Giraph and Surfer start computing with a constant number of machines and finish the computation with the same number of machines; no matter if the graph is shrinking or not during the execution. On the other hand for convergent algorithms, as the processing continues, the number of active vertices decreases. So, iGiraph-network-aware removes deactivated vertices from the memory which means the graph is shrinking during the processing. Our experiments (Figures 9-11) show that the number of messages in the network is reduced significantly by using dynamic bandwidth-and-traffic-aware re-partitioning and mapping approach on iGiraph-network-aware. This results in reducing the number of active workers during the processing. As a result, when the number of machines declines, the cost of
processing will also drop significantly. The results even show that the number of workers tends to be reduced faster compared to original iGiraph paper [4].

It is also shown that the new mechanism works well on non-convergent algorithms such as PageRank. According to Figures 14 and 15, not only the average number of messages in the network is reduced in iGiraph-network-aware compared to Giraph and Surfer, but also the processing has been completed faster using our bandwidth-and-traffic-aware dynamic repartitioning algorithm.

The second group of experiments is carried out for computation-intensive scenarios. It is shown that using computation-aware re-partitioning that considers CPU idle time on each worker for mapping, the system performs better compared to Giraph. For this experiment, we have created two 500×500 matrices with random integer numbers and multiply them to keep the CPU busy on a random number of machines. The results of multiplication will not be saved because we do not want to decrease the memory of workers during the experiment. The results of the experiments have only been compared to original Giraph under the same conditions. It means that, for example we have done the experiments on both iGiraph-network-aware with computation-aware dynamic re-partitioning algorithm and Giraph when matrices multiplication is running on six workers and the same workers every time. The results have not been compared with Surfer because it does not have such capability to process the graph using computation information on the network.
As shown in the Figure 18, again the number of machines has noticeably decreased in iGiraph-network-aware using computation-aware dynamic re-partitioning approach for a convergent algorithm such as shortest path algorithm. So, processing the graph on iGiraph-network-aware is much cheaper than doing so on Giraph. The same results have been obtained for non-convergent algorithm PageRank. It shows that our proposed mechanism has reduced the average number of messages in the network while completing the computation faster.

8.3. Complexity Analysis

We analyzed the time complexity of the two proposed algorithms (traffic-and-bandwidth-aware and computation-aware algorithms) which are very similar in terms of the structure. Both algorithms are dependent to the number of supersteps \((N)\) which \(N\) varies based on the application and the number of vertices in the graph. Also, prioritizing partitions \((P)\) and worker machines \((W)\) affect the algorithms as they need to be calculated in each iteration. Therefore, the complexity of these algorithms is \(O(N(P+W))\). Since both \(P\) and \(W\) are dependent to the number of machines \((m)\) (one partition per worker), the complexity also can be written as \(O(N(log m))\).

On the other side, the complexity of partitioning algorithm for Surfer is \(O(m^2)+O[P+log_2 P (n+log P)]\) where \(P\) is the number of partitions and random partitioning is used instead of METIS. For Giraph the complexity is \(O(N(n))\) \((n=number\ of\ nodes)\). As can be seen, algorithms are dependent to the applications’ complexities as well. According to [27], for instance, the complexity of SSSP and CC algorithms are \(O(ne)\) and \(O((e+n)log n))\) respectively, where \(n\) is the number of nodes and \(e\) is the number of edges in the graph. In Surfer, the user should define the number of partitions for the processing hence the complexity of the algorithm is dependent to the number of partitions \((P)\).

8.4. Discussion

We compared our algorithm with Surfer’s algorithm [6], due to its relevance to our work. Both approaches use mapping strategy to map partitions and worker machines for computation. They both
consider bandwidth as an important factor that affects the performance of processing which shows the role of network to make the processing costly. Both methods try to reduce the number of cross-partition edges to reduce the number of messages transferred between machines so that they can decrease the communication cost.

Apart from the similarities, there are significant differences between Surfer and our work. First, Surfer partitions the graph before the processing starts and never repartitions data during computation. It creates the partition map at the beginning of the operation along with the workers map, but it only changes the workers map during the processing. The problem is that after each iteration, a new map is generated for workers and partitions have to be moved to a different worker every time. It is specifically very costly when all active and inactive vertices are meant to be transferred together. This is the reason that iGiraph-network-aware distinguishes between convergent and non-convergent algorithms and is using a re-partitioning algorithm to make a new partition map and workers map after each superstep. Second, the Surfer authors evaluate their approach using METIS and ParMETIS to initiate the partitioning the graph while iGiraph-network-aware uses a random approach. METIS and ParMETIS have been shown to give better partitioning results than random partitioning. So, we believe that this is the reason that Surfer’s approach does not work well by being initiated with random partitioning. However, initiating iGiraph-network-aware by either METIS or ParMETIS will still give better results compare to Surfer because of different strategies that they are using. Third, all experiments on Surfer have been done on random graph datasets which is generated by a graph generator and not real-world datasets. Therefore, the impact of high-degree vertices has not been investigated by Surfer, although it is an important feature of real-world graphs. Fourth, Surfer has not investigated monetary cost of the processing. This is the unique feature of iGiraph-network-aware as it reduces the number of using machines as the operation progresses whereas both Surfer and Giraph maintain the same higher number of machines during the entire operation.

Overall, there are many factors that need to be considered for scheduling resources in cloud environments [28]. However, factors such as monetary cost and networks aspects of clouds have not been investigated much in graph processing context. Our work is one of the first works that combines all those factors to not only improve the performance but also to minimize the cost of using public clouds.

9. Conclusions and Future Work

As the amount of data is growing every day, processing and analyzing them in a cost-efficient way is a challenge. Distributed graph processing frameworks have emerged in the past few years to facilitate the processing of large-scale graphs that are made and stored by applications such as social networks and mobile applications. On the other hand, cloud computing has brought new facilities to streamline large-scale computing and storage. It has brought different models of computing with new paradigms such as pay-as-you-go model, scalability and elasticity. In this paper, a new graph processing framework was proposed to analyze large-scale graph data. To achieve this, a new two-dimension classification of graph applications was used for the processing strategy. A novel dynamic re-partitioning was also introduced which considers network factors such as bandwidth and network traffic to process the graph by reducing network, communication and monetary costs. According to our experiments, this model could significantly outperform other frameworks such as famous Giraph.

As future work, we plan to consider other different scenarios such as processing graphs using different types of machine configurations (e.g. large, medium, small instances) that are available on cloud
platforms. We will investigate the impact of other partitioning methods such as METIS instead of a simple random partitioning on our framework and identify the factors having an effect on the quality of graph processing services.

Acknowledgements

This work is partially supported by ARC Future Fellowship and ARC Discovery Project grants. We thank NECTAR research cloud for providing the infrastructure for the experiments. We thank Satish Narayana Srirama, Adel N. Toosi and Maria Rodriguez for their contributions on improving this paper. We also thank anonymous reviewers for their comments on improving the paper.

References


[27] Khan, A., "Vertex-centric Graph Processing: Good, Bad, and the Ugly", in the Proceeding of 20th International Conference on Extending Database Technology (EDBT), Venice, Italy, 2017