Received: 15 December 2017 | Revised: 4 May 2018 | Accepted: 17 June 2018

DOI: 10.1002/spe.2623

RESEARCH ARTICLE

WILEY

Cost-efficient and network-aware dynamic repartitioning-based algorithms for scheduling large-scale graphs in cloud computing environments

Safiollah Heidari | Rajkumar Buyya

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

Correspondence
Safiollah Heidari, Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia.
Email: sheidari@student.unimelb.edu.au

Funding information
ARC Discovery Project, Grant/Award Number: DP160102414

Summary

Large amount of data that is generated by Internet and enterprize applications are stored in the form of graphs. Graph processing systems are broadly used in enterprizes 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. Moreover, the effects of network factors such as bandwidth and traffic as well as monetary cost of processing such large-scale graphs and the mutual impact of these elements have been less studied. To address these issues, we proposed two dynamic repartitioning 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 our extended graph processing system (iGiraph) and compared them with those supported in other graph processing systems such as Giraph and Surfer on Australian National Cloud Infrastructure. We observed that up to 30% faster execution time, up to 50% network traffic decline, and more than 50% cost reduction are achieved by our algorithms compared to a framework such as the popular Giraph.

KEYWORDS

cloud computing, cost saving, graph processing, network-aware processing

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, ie, 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 the 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. Thus, 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, ie, 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. Therefore, 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). Therefore, 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. Even systems such as GraphP[8] and GraphR[9] that are reducing the number of messages passing between partitions and introducing new memory access techniques, respectively, do not discuss the monetary costs of the processing. These limitations can be overcome by dynamic management of resources in an elastic manner.

The aim of this paper is to develop scheduling algorithms that consider characteristics of application workloads and resources along with network factors to improve the performance and reduce the monetary cost of the whole computation. We propose a novel dynamic repartitioning method that utilizes different factors including (1) the type of the graph application that is going to be used, (2) some intrinsic features of natural graphs such as high-degree vertices, and (3) the network features of the cloud environment that the system is running on. Our algorithms were plugged in to our extended version of graph processing framework (iGiraph) and we compared them with those supported in other graph processing systems such as Giraph and Surfer on Australian National Cloud Infrastructure. We observed that up to 30% faster execution time, up to 50% network traffic decline, and more than 50% cost reduction is achieved by our algorithms in comparison with a framework such as popular Giraph.

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 repartitioning algorithm and a new computation-aware repartitioning 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, ie, iGiraph, which is a graph processing framework based on Giraph. A new classification of graph algorithms is explained in Section 4. Sections 5 and 6 introduce our new proposed bandwidth-and-traffic-aware and computation-aware dynamic repartitioning 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 works 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.[10] 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.[11] This method tries to solve the issue of the previous method. However, it still has problems with fault tolerance and scalability.[12] Another way of processing graphs is to adopt graphic processing units to accelerate different graph processing tasks. In the sampling method, the input graph will be divided into several subgraphs by the system and then the attribute of the main graph will be estimated based on the attributes of the smaller subgraphs. 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.[13] This can be specifically utilized for solving large graph problems. Pregel,[14] 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,[15] 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, which 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,[16] Apache Hama,[17] ExPregel,[18] GPS,[19] GraphLab,[20] 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, whereas 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, because 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. However, 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 the .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.

TABLE 1 Comparison of the most related work in the literature

System	Architecture	Partitioning Method	Traffic-aware	Bandwidth-aware	Computation-aware	Resource Scheduling
Pregel[14]	Distributed	Static	×	×	×	Static
Giraph[16]	Distributed	Static	×	×	×	Static
GPS[19]	Distributed	Dynamic	√	×	×	Static
GraphX[22]	Distributed	Static	×	×	×	Static
Surfer[6]	Distributed	Dynamic	×	√	×	Static
iGiraph[4]	Distributed	Dynamic	√	×	×	Dynamic
Our work - iGiraph-network-aware	Distributed	Dynamic	√	√	√	Dynamic

In another research,[21] 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% and 10%. GraphX[22] and Spark* were used in that experiment and some network factors such as the speed of the network was studied in different situations. However, McSherry[23] 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. Table 1 demonstrates the features of some of the most related works in the literature.

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[24] for processing their large graph data. iGiraph has a distributed architecture, which is implemented on top of Hadoop[25] and utilizes its distributed file system (HDFS) for data input/output. It employs a vertex-centric programming 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 the 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, ie, (1) obtains its neighbors' new values from previous superstep, (2) updates its own value using the received values, and (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 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 s 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 repartitioning 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, ie, (1) convergent algorithms such as the shortest path or connected components algorithms that will converge

*http://spark.apache.org/

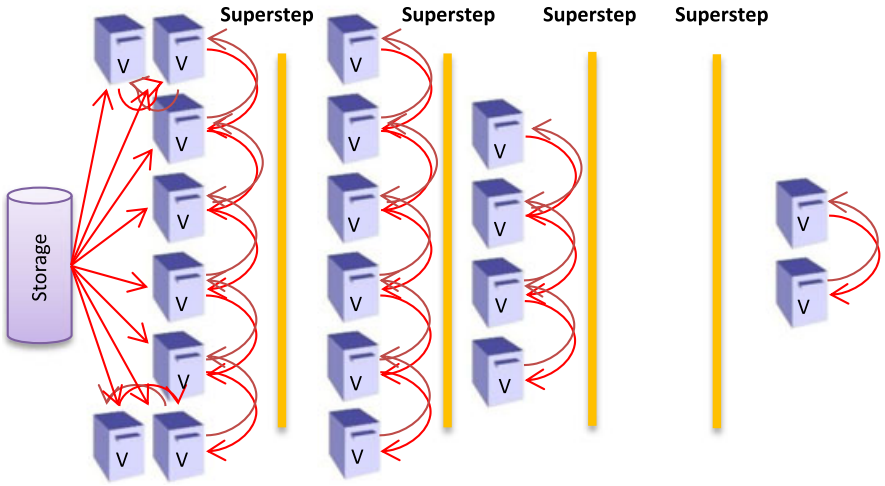

FIGURE 1 Steps in iGiraph system reduces resource requirements [Colour figure can be viewed at wileyonlinelibrary.com]

as the computation progresses and (2) nonconvergent 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 nonconvergent 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 the 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 (Figure 2). 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 repartitioning 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 the 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 this paper.

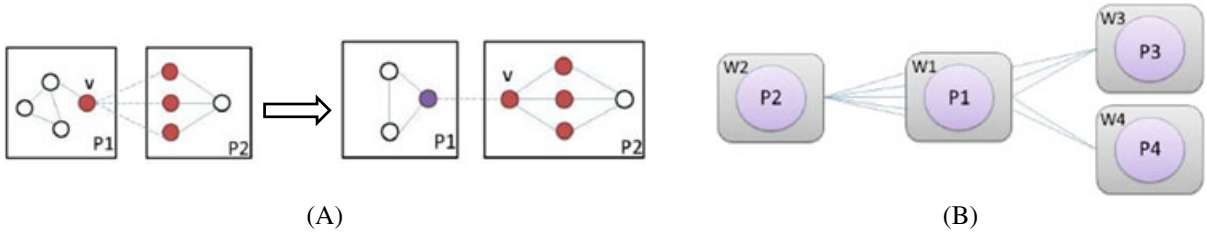

FIGURE 2 High-degree border vertices in both (A) vertex level and (B) partition level [Colour figure can be viewed at wileyonlinelibrary.com]

4 | PROCESSING ENVIRONMENT CATEGORIZATION AND GRAPH APPLICATIONS

Different works use different categorizations for graph applications. For example, GBASE[26] and TurboGraph[27] categorize the queries to *global queries* and *targeted queries*. Algorithms such as diameter estimation that need to traverse the entire graph are identified as global queries, whereas other algorithms such as single source shortest are put in the targeted queries category. A framework such as Giraph++[28] uses three categories including graph traversal, random walk, and graph aggregation for graph algorithms, whereas iGiraph[4] utilizes a one-dimension categorization for all graph applications, which divides them into convergent and nonconvergent.

In this work, we extend the iGiraph's 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 communicating and the memory side. Sometimes, the graph processing application itself is computationally intensive and sometimes other applications keep the CPU busy in virtual machines (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 nonconvergent) 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 nonconvergent 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, whereas the sum of the weights of its edges is minimized. The shortest path is a famous problem in the 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. The 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.

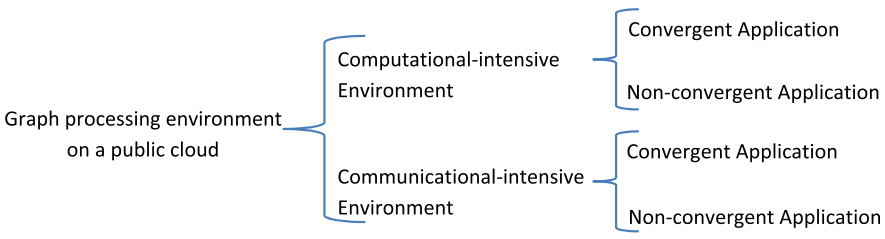

FIGURE 3 Graph applications and processing environment categorization [Colour figure can be viewed at wileyonlinelibrary.com]

The total cost of processing in a graph system is depending on two major factors (considering equal size for the messages in the network), ie, (1) the number of machines and (2) the time in which a specific type of machine is being used, as shown in the following:

$$\text{Cost}_{\text{Total}} = \sum_{j=0}^{m} \sum_{i=1}^{n} \left(C\left(VM_i\right) \times T\left(VM_i\right)\right). \tag{1}$$

In the aforementioned equation, C (VM_i) is the price of each machine and T (VM_i) is the time within the machine is used. To reduce the total cost of the operation, either less costly machines must be used or the total time that each machine is being used should be reduced. In order to achieve this in a graph processing system, partitioning plays an important role. For instance, there is no need to keep all the initial machines in the system for convergent algorithms if there is a way to repartition the graph and place the remaining of the graph on less number of machines and reschedule the resources. In this paper, we show that, to provide an effective dynamic repartitioning mechanism, considering factors such as traffic, bandwidth, and computation burden in the network can help to reduce the monetary cost and improve the performance.

5 | BANDWIDTH-AND-TRAFFIC-AWARE GRAPH SCHEDULING ALGORITHM WITH DYNAMIC REPARTITIONING

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. Therefore, the approximate data transfer time (DTT) from P_i to P_j will be as follows:

$$\text{DTT}\left(\text{i,j}\right) = \frac{C\left(P_i, P_j\right) \times N_M\left(P_i, P_j\right)}{B_{i,j}}. \tag{2}$$

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 DTT ($\text{DTT}_{\text{Total}}$) in the network caused in all partition pairs is as follows:

$$\text{DTT}_{\text{Total}} = \sum_{i=0}^{P-1} \sum_{j=0}^{P-1} \text{DTT}\left(i, j\right). \tag{3}$$

Obviously, if network bandwidth among different machine pairs is constant, the total network DTT will be minimized when the total number of cross-partition edges is minimized. Nevertheless, the network bandwidth among different machine pairs can change remarkably in the cloud. Cloud providers have noticed such network bandwidth unevenness. The network bandwidth of every machine pair among 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 the work of Valiant,[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.

To address these issues in a public cloud environment, a new dynamic repartitioning 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. Therefore, 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 (repartitioning) for the next iteration.

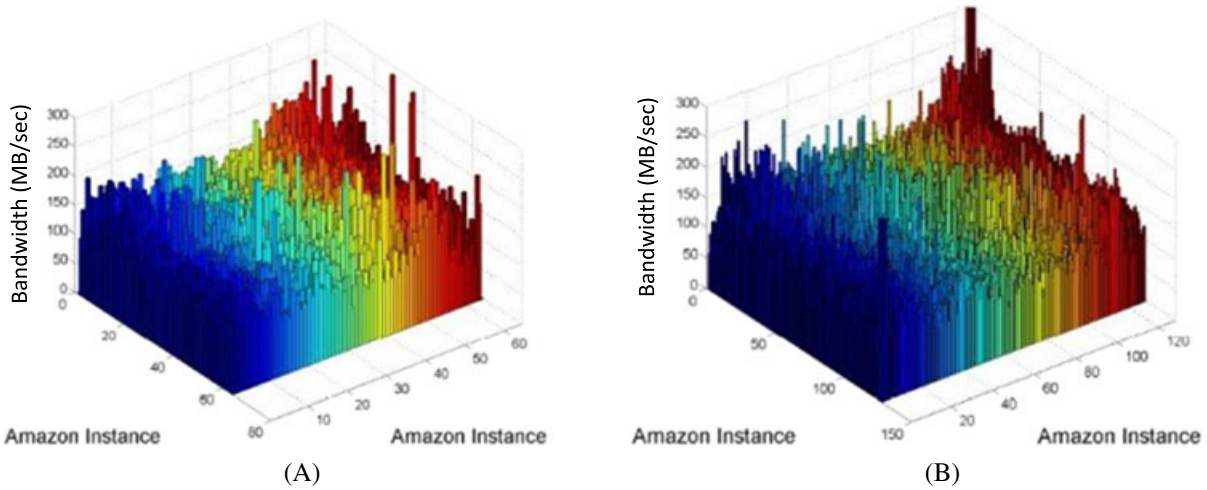

FIGURE 4 Network bandwidth unevenness in Amazon EC2 small instances with (A) 64 instances and (B) 128 instances[6] [Colour figure can be viewed at wileyonlinelibrary.com]

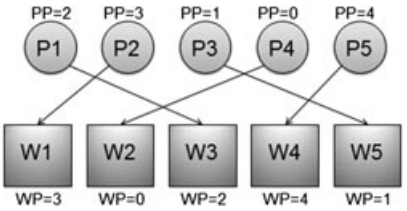

FIGURE 5 Mapping strategy for 5 partitions and 5 workers. Partitions with higher priorities are assigned to the machines with higher bandwidth

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 (Figure 5). 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. Furthermore, for the partitions that get merged, the priority of the final partition (combined partition) will be set as the priority of the partition with the 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).

In a nutshell, according to Algorithm 1, after each superstep, initial priorities for the partitions and workers will be calculated based on the measurement of various network factors (traffic and bandwidth here) that have been completed during the iteration. Then, if needed, partition merges and vertex migrations might happen based on the aforementioned mechanism. Eventually, final priorities will be set for partitions and workers and they will be mapped accordingly.

According to our experiment results (Section 8.2), 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 repartitioning method (for both convergent and nonconvergent 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 GRAPH SCHEDULING ALGORITHM WITH DYNAMIC REPARTITIONING

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, whereas 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, and assigning priority numbers to tasks. Another approach for when there is no priority or preference for job execution can be using equal time-slots for computing jobs in an intertwined way.

Algorithm 2 Computation-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 CPU utilization on each worker and CPU idle time
6:     If global synchronization happened then
7:         Merge the partitions or migrate vertices based on machineType 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
```

We propose a similar *mapping strategy* as we discussed for traffic and bandwidth-aware repartitioning, but we consider CPU utilization instead of bandwidth in the algorithm and repartition the graph differently. We have implemented this strategy on iGiraph. As in the 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 repartitioning.

At this stage, on one side, based on the number of messages that has been passed between workers through the network, we define PP again by which the partitions with high-degree vertices can be recognized. On the other side, a scalable

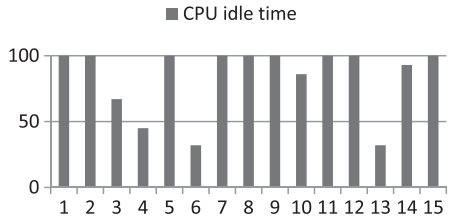

FIGURE 6 Percentage of CPU idle time in a system with 15 workers

monitoring tool called Ganglia[†] 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 supestep 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 repartitioning 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.

According to the aforementioned strategy, there will be four types of machines in the environment after the first superstep, ie, (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 have 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. Therefore, the algorithm would be like the following: partitions in machine type 1 need to be migrated to or merged with partitions on machines types 4 or 2, respectively. Partitions on machine type 2 can be merged with the one on types 3 and 4. Partitions on machine type 3 can be migrated to type 4 or be merged by partitions on machine type 2. Based on this algorithm, at the start of the processing, all workers have their type set as 0, which will change after the first superstep. Then, at the end of each superstep, this algorithm will repartition 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.

To summarize, similar to Section 5, after each superstep, initial priorities for the partitions and workers will be calculated based on the measurement of various network factors, which, in this case, are the traffic and the computation capacity of each machine. After that, some partitions might get merged and some vertices might get migrated using the dynamic repartitioning mechanism. Finally, priorities will be set for partitions and workers and they will be mapped accordingly (Algorithm 2).

As will be shown in Section 8.2, our experiments prove that, under the equal situation, the computation-aware repartitioning on iGiraph-network-aware significantly reduces the execution time of the entire processing compared to Giraph. It is also shown that this approach can reduce the monetary cost of the processing for both convergent and nonconvergent 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.

[†]http://ganglia.sourceforge.net/

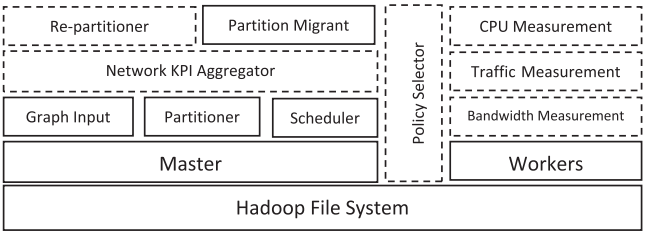

FIGURE 7 The components that we added to original iGiraph are shown in dotted rectangles

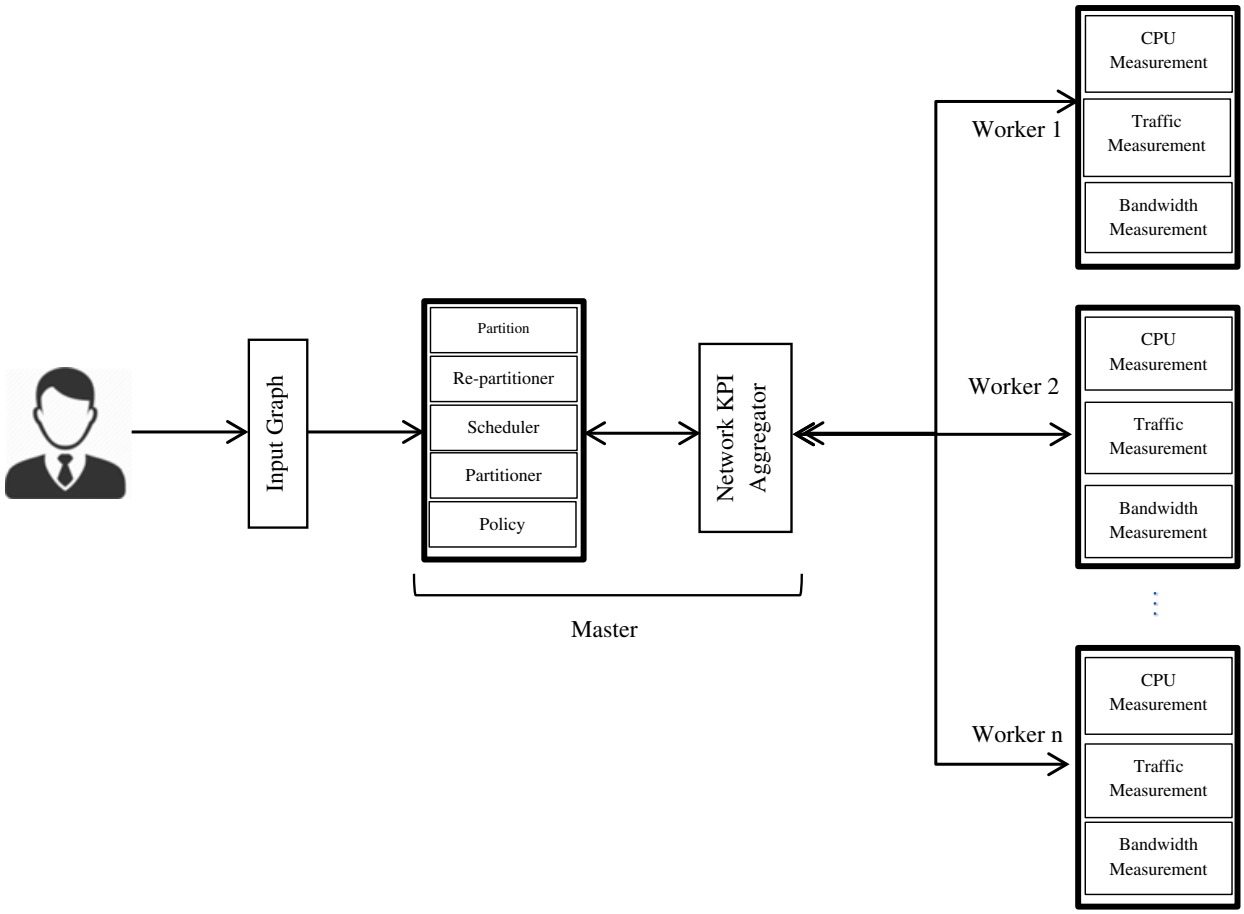

FIGURE 8 System architecture

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 the work of Zhong et al.[29]

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 that it receives 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, users specify their workloads and, based on that, they define what algorithm (bandwidth-aware or computation-aware) they want to be used to process their workload.

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.

7.6 | Repartitioner

The repartitioning 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, repartitioning happens after each superstep and before the next superstep begins. We will show that, using these components and the new repartitioning 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[30] is the Australian national cloud infrastructure facilities. Medium instances have two-cores with 8GB RAM and 70GB disk including 10GB root dis'k and 60GB ephemeral disk. All the instances are in the same zone and use the same security policies. Since NECTAR does not correlate any price to its infrastructure for research use cases, the prices for VMs are put proportionally based on Amazon Web Service (AWS) on-demand instance costs in Sydney region according to the closest VM configurations as an assumption for this work. Hence, NECTAR m1.medium price is put based on AWS m5.large Linux instance, which costs $0.12 per hour. However, because our experiments are in second scale (instead of hour scale), the prices are being calculated for the entire operation in second scale. Therefore, we charge the machines only based on the number of seconds they were used and do not charge them for one hour because they were used only for few seconds. We also installed NECTAR Ubuntu 14.04 (Trusty) amd64 on each instance. We plugged in our algorithms to 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"** in this 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.

We chose the shortest path and PageRank for communication-bound convergent and nonconvergent algorithms, respectively. Moreover, to show the effectiveness of the distributed processing on large-scale graphs by using our proposed solution, we utilize three real datasets of different sizes, ie, Amazon, YouTube, and Pokec.[31] Properties of these datasets are shown in Table 2.

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

TABLE 2 Datasets' properties

Graph	Vertices	Edges
Amazon (TWEB)	403 394	3 387 388
YouTube Links	1 138 499	4 942 297
Pokec	1 632 803	30 622 564

algorithm still shows Surfer's behavior on the network. We have also compared the results with the original iGiraph paper. 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, original iGiraph, and Surfer on both convergent and nonconvergent 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. Therefore, iGiraph and iGiraph-network-aware remove 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 even more significantly compared to the original iGiraph by using dynamic bandwidth-and-traffic-aware repartitioning and mapping approach on iGiraph-network-aware. This leads to reducing the number of active workers during the processing (Figure 12). 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 the original iGiraph paper because, using the new algorithms in this paper, the number of messages in the network is decreasing too.[4] This also affects the total execution time, as illustrated in Figure 13.

It is also shown that the new mechanism works well on nonconvergent 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, original iGiraph, and Surfer but also the processing has been completed faster using our bandwidth-and-traffic-aware dynamic repartitioning algorithm. Tables 2 and 3 show the cost comparison for different datasets for the shortest path and PageRank algorithms, respectively, on each framework. Tables 3 and Table 4 show the dollar cost of the operations is much less with the proposed techniques in this paper.

The second group of experiments is carried out for computation-intensive scenarios. It is shown that, using computation-aware repartitioning that considers CPU idle time on each worker for mapping, the system performs better compared to Giraph (Figures 16 and 17). 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 the 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 the 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 repartitioning algorithm

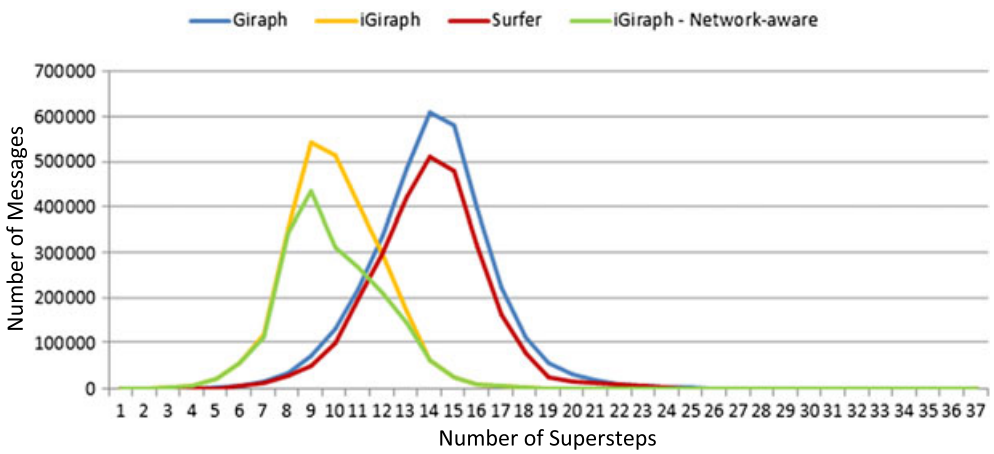

FIGURE 9 Number of network messages transferred between partitions across supersteps for Amazon graph using shortest path algorithm [Colour figure can be viewed at wileyonlinelibrary.com]

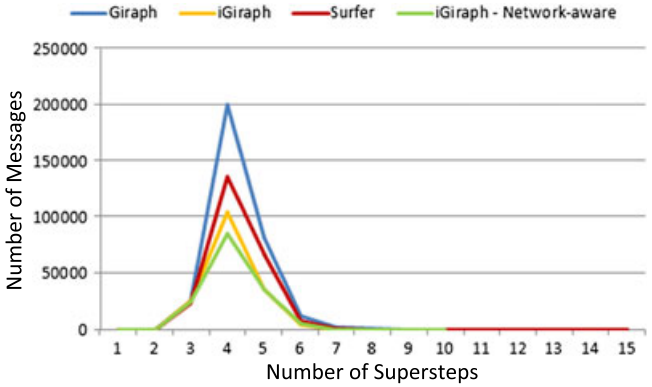

FIGURE 10 Number of network messages transferred between partitions across supersteps for YouTube graph using shortest path algorithm [Colour figure can be viewed at wileyonlinelibrary.com]

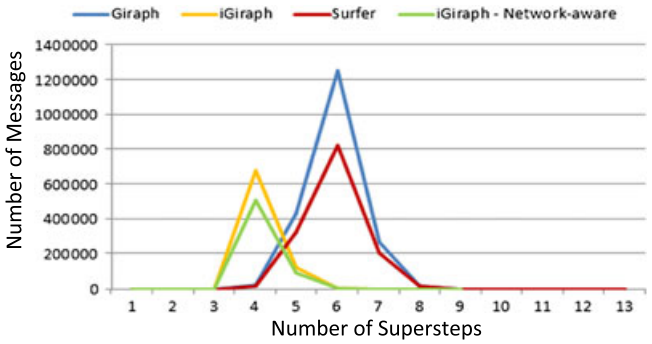

FIGURE 11 Number of network messages transferred between partitions across supersteps for Pokec graph using shortest path algorithm [Colour figure can be viewed at wileyonlinelibrary.com]

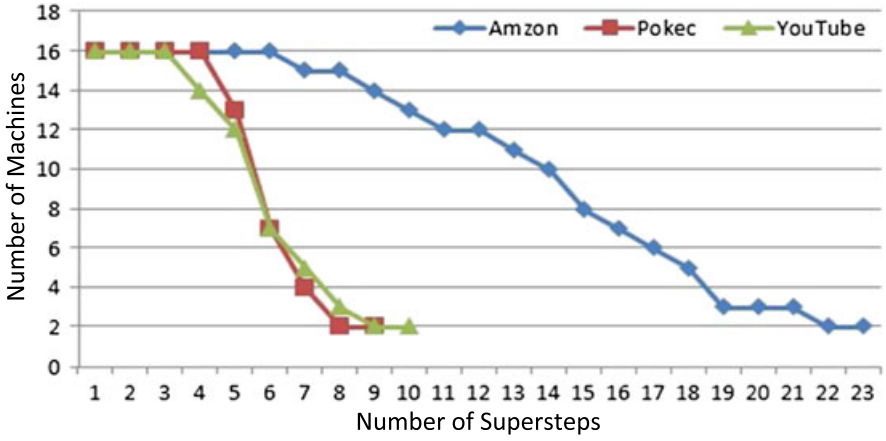

FIGURE 12 Number of machines varying during supersteps while running shortest path algorithm on different datasets [Colour figure can be viewed at wileyonlinelibrary.com]

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 the computation information on the network.

As shown in Figure 18, again, the number of machines has noticeably decreased in iGiraph-network-aware using the computation-aware dynamic repartitioning approach for a convergent algorithm such as the shortest path algorithm. Therefore, processing the graph on iGiraph-network-aware is much cheaper than doing so on Giraph (Table 5). The same results have been obtained for the nonconvergent algorithm PageRank. It shows that our proposed mechanism has

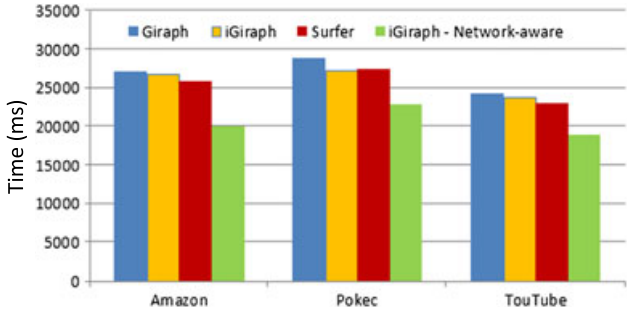

FIGURE 13 Total time taken to perform shortest path algorithm [Colour figure can be viewed at wileyonlinelibrary.com]

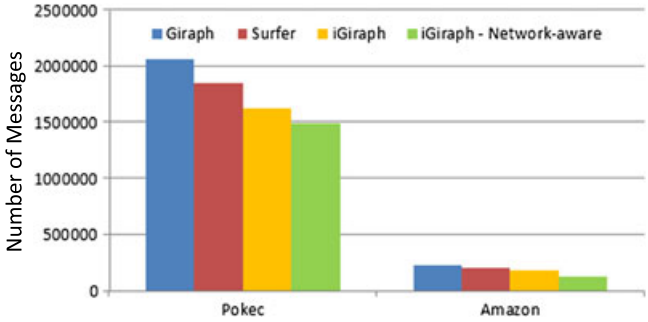

FIGURE 14 The average number of network messages in each superstep [Colour figure can be viewed at wileyonlinelibrary.com]

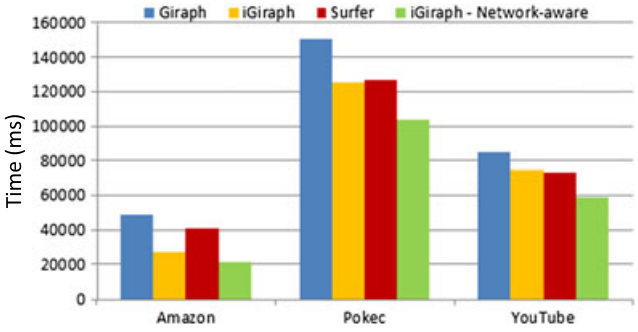

FIGURE 15 Total time taken to perform the PageRank algorithm [Colour figure can be viewed at wileyonlinelibrary.com]

reduced the average number of messages in the network while completing the computation faster (Figure 19). Table 5 and Table 6 show that the dollar cost of the operations is much less with the proposed techniques in this paper.

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. In addition, 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 the work of Han et al,[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)*.

TABLE 3 Processing cost for single-source-shortest-path on different frameworks

Dataset	Giraph	Surfer	iGiraph	iGiraph-network-aware
Amazon	$0.0140	$0.0130	$0.0096	$0.0079
YouTube	$0.0125	$0.0120	$0.0082	$0.0067
Pokec	$0.0145	$0.0140	$0.0099	$0.0071

TABLE 4 Processing cost for PageRank on different frameworks

Dataset	Giraph	Surfer	iGiraph	iGiraph-network-aware
Amazon	$0.0250	$0.0210	$0.0140	$0.0110
YouTube	$0.0430	$0.0370	$0.0380	$0.0295
Pokec	$0.0760	$0.0640	$0.0630	$0.0525

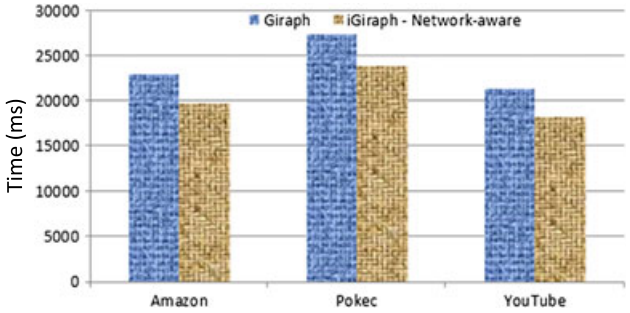

FIGURE 16 Total time taken to perform shortest path algorithm [Colour figure can be viewed at wileyonlinelibrary.com]

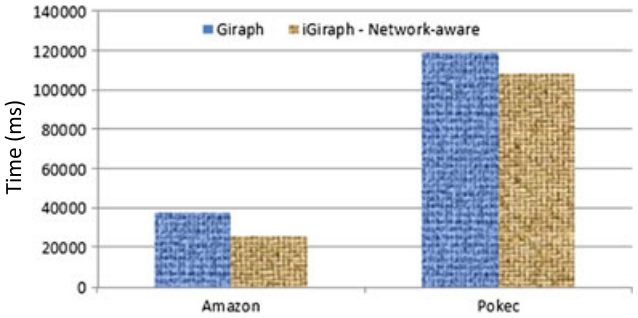

FIGURE 17 Total time taken to perform PageRank algorithm [Colour figure can be viewed at wileyonlinelibrary.com]

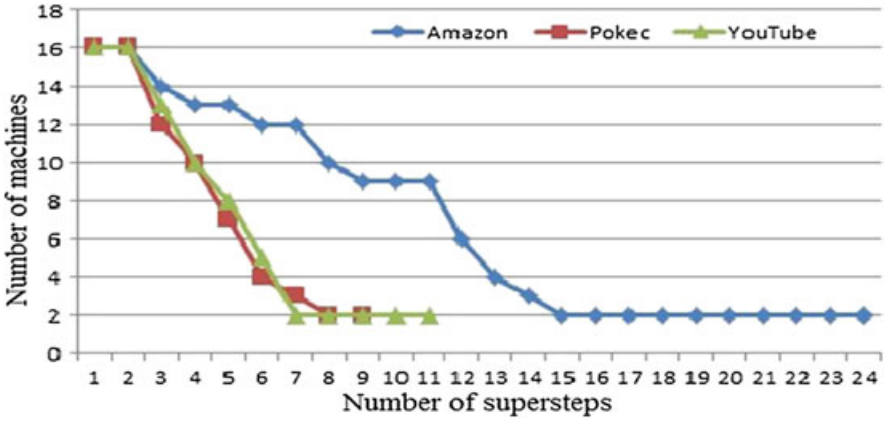

FIGURE 18 Number of machines varying during supesteps while running shortest path algorithms on different datasets [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 5 Processing cost for single-source-shortest-path on different frameworks

Dataset	Giraph	iGiraph-network-aware
Amazon	$0.0115	$0.0055
YouTube	$0.0110	$0.0042
Pokec	$0.0140	$0.0068

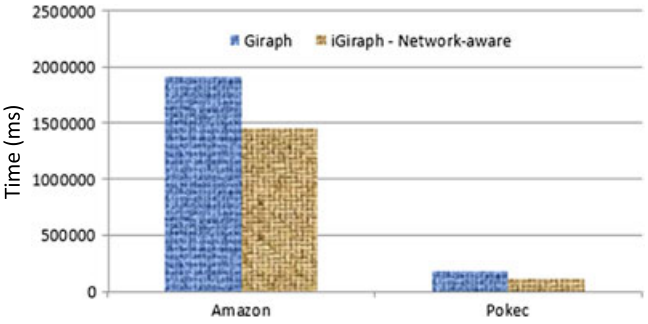

FIGURE 19 The average number of network messages in each experiment [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 6 Processing cost for PageRank on different frameworks

Dataset	Giraph	iGiraph-network-aware
Amazon	$0.0195	$0.0130
Pokec	$0.0600	$0.0545

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 nonconvergent algorithms and is using a repartitioning 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, whereas iGiraph-network-aware uses a random approach. METIS and ParMETIS have been shown to give better partitioning results than random partitioning. Therefore, 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 compared 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 the 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 repartitioning 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 the famous Giraph.

As future work, we plan to consider other different scenarios such as processing graphs using different types of machine configurations (eg 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 the Australian Research Council (ARC) Discovery Project (DP160102414). 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.

ORCID

Safiollah Heidari http://orcid.org/0000-0002-9392-8144
Rajkumar Buyya http://orcid.org/0000-0001-9754-6496

REFERENCES

1. Page L, Brin S, Motwani R, Winograd T. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report. Stanford, CA: Stanford InfoLab; 1998.
2. Zha Z-J, Mei T, Wang J, Wang Z, Hua X-S. Graph-based semi-supervised learning with multiple labels. *Int J Geogr Inf Sci*. 2009;20(2):97-103.
3. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. In: Proceedings of Sixth Symposium on Operating Systems Design and Implementation (OSDI '04); 2004; San Francisco, CA.
4. Heidari S, Calheiros RN, Buyya R. iGiraph: a cost-efficient framework for processing large-scale graphs on public clouds. In: Proceedings of 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '16); 2016; Cartagena, Colombia.
5. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. *Futur Gener Comput Syst*. 2009;25(6):599-616.
6. Chen R, Weng X, He B, Yang M. Large graph processing in the cloud. In: Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD '10); 2010; Indianapolis, IN.
7. Redekopp M, Simmhan Y, Prasanna VK. Optimizations and analysis of BSP graph processing models on public clouds. In: Proceedings of IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS'13); 2013; Boston, MA.
8. Zhang M, Zhuo Y, Wang C, et al. GraphP: reducing communication for PIM-based graph processing with efficient data partition. In: Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'18); 2018; Vienna, Austria.
9. Song L, Zhuo Y, Qian X, Li H, Chen Y. GraphR: accelerating graph processing using ReRAM. In: Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'18); 2018; Vienna, Austria.
10. Lumsdaine A, Gregor D, Hendrickson B, Berry J. Challenges in parallel graph processing. *Parallel Process Lett*. 2007;17(1):5-20.
11. Siek J, Lee L-Q, Lumsdaine A. *The Boost Graph Library: User Guide and Reference Manual*. Boston, MA: Addison-Wesley; 2002.
12. Gregor D, Lumsdaine A. The parallel BGL: a generic library for distributed graph computations. In: Proceedings of Parallel Object-Oriented Scientific Computing (POOSC); 2005; San Diego, CA.
13. Coulouris G, Dollimore J, Kindberg T, Blair G. *Distributed Systems: Concepts and Design*. 5th ed. Boston, MA: Addison-Wesley; 2012.

14. Malewicz G, Austern MH, Bik AJC, et al. Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data; 2010; Indianapolis, IN.
15. Valiant LG. A bridging model for parallel computation. *Commun ACM*. 1990;33(8):103-111.
16. Apache Giraph. [Online]. Available: http://giraph.apache.org/
17. Apache Hama. [Online]. Available: https://hama.apache.org/. Accessed January 7, 2016.
18. Sagharichian M, Naderi H, Haghjoo M. ExPregel: a new computational model for large-scale graph processing. *Concurrency Computat Pract Exper*. 2015;27(17):4954-4969.
19. Salihoglu S, Widom J. GPS: a graph processing system. In: Proceedings of 25th International Conference on Scientific and Statistical Database Management (SSDBM); 2013; Baltimore, MD.
20. Low Y, Gonzalez J, Kyrola A, Bickson D, Guestrin C, Hellerstein J. GraphLab: a new framework for parallel machine learning. In: Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI 2010); 2010; Catalina Island, CA.
21. Ousterhout K, Rasti R, Ratnasamy S, Shenker S, Chun B-G. Making sense of performance in data analytics frameworks. In: Proceeding of the 12th USENIX Symposium on Networked Systems Design and Implementation; 2015; Oakland, CA.
22. Gonzalez JE, Xin RS, Dave A, Crankshaw D, Franklin MJ, Stoica I. GraphX: graph processing in a distributed dataflow framework. In: Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14); 2014; Broomfield, CO.
23. McSherry F. The impact of fast networks on graph analytics, part 1. [Online]. Available: https://github.com/frankmcsherry/blog/blob/master/posts/2015-07-08.md. Accessed January 7, 2016.
24. Ching A, Edunov S, Kabiljo M, Logothetis D, Muthukrishnan S. One trillion edges: graph processing at Facebook-scale. *Proc VLDB Endow*. 2015;8(12):1804-1815.
25. Apache Hadoop. Apache. [Online]. Available: http://hadoop.apache.org/. Accessed January 7, 2016.
26. Kang U, Tong H, Sun J, Lin C-Y, Faloutsos C. GBASE: an efficient analysis platform for large graphs. *VLDB J*. 2012;21(5):637-650.
27. Han W-S, Lee S, Park K, et al. TurboGraph: a fast parallel graph engine handling billion-scale graphs in a single PC. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13); 2013; Chicago, IL.
28. Tian Y, Balmin A, Corsten SA, Tatikonda S, McPherson J. From "think like a vertex" to "think like a graph". *Proc VLDB Endow*. 2013;7(3):193-204.
29. Zhong J, He B, Gong Y. Network performance aware MPI collective communication operations in the cloud. *IEEE Trans Parallel Distrib Syst*. 2015;26(11):3079-3089.
30. NECTAR Cloud. [Online]. Available: http://nectar.org.au/research-cloud/. Accessed January 7, 2016.
31. Kunegis J. KONECT - the Koblenz network collection. In: Proceedings of the 22nd International Conference on World Wide Web; 2013; Rio de Janeiro, Brazil.

How to cite this article: Heidari S, Buyya R. Cost-efficient and network-aware dynamic repartitioning-based algorithms for scheduling large-scale graphs in cloud computing environments. *Softw Pract Exper*. 2018;48:2174–2192. https://doi.org/10.1002/spe.2623