2016 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing

iGiraph: A Cost-efficient Framework for Processing Large-scale Graphs on Public Clouds

Safiollah Heidari, Rodrigo N. Calheiros and Rajkumar Buyya
Cloud Computing and **D**istributed **S**ystems (CLOUDS) Lab
Department of Computing and Information Systems
The University of Melbourne, Australia
E-mail: sheidari@student.unimelb.edu.au, rbuyya@unimelb.edu.au

Abstract— **Large-scale graph analytics has gained attention during the past few years. As the world is going to be more connected by appearance of new technologies and applications such as social networks, Web portals, mobile devices, Internet of things, etc, a huge amount of data are created and stored every day in the form of graphs consisting of billions of vertices and edges. Many graph processing frameworks have been developed to process these large graphs since Google introduced its graph processing framework called Pregel in 2010. On the other hand, cloud computing which is a new paradigm of computing that overcomes restrictions of traditional problems in computing by enabling some novel technological and economical solutions such as distributed computing, elasticity and pay-as-you-go models has improved service delivery features. In this paper, we present iGiraph, a cost-efficient Pregel-like graph processing framework for processing large-scale graphs on public clouds. iGiraph uses a new dynamic re-partitioning approach based on messaging pattern to minimize the cost of resource utilization on public clouds. We also present the experimental results on the performance and cost effects of our method and compare them with basic Giraph framework. Our results validate that iGiraph remarkably decreases the cost and improves the performance by scaling the number of workers dynamically.**

Keywords-cloud computing; graph processing; partitioning; public clouds; cost-efficient processing; graph analytics

I. INTRODUCTION

As Internet continues to grow, the world is becoming a more connected environment and the number of data resources is increasing beyond what had been predicted before [1]. Amongst various data modeling approaches to store huge data, graphs are widely adopted to model complex relationships among objects. A graph consists of sets of vertices and edges which demonstrate the pairwise relationship between different objects. Many applications and technologies such as social networks, search engines, banking applications, smart phones and mobile devices, computer networks, the semantic web, etc, are modeling and using data in the form of graphs [2]. These applications generate massive amounts of data which are represented by graphs. Facebook [3], for example, has more than one billion users that are considered as the vertices of a huge graph where the relationships between them are considered as the edges of the graph. To gain an insight and discover knowledge from these applications, the graph that represents them should be processed. However, the scale of these graphs poses challenges to their efficient processing [4].

In order to process large graph problems, every solution confronts with some challenges due to the intrinsic properties of graphs. These properties include data-driven computation, unstructured problems, poor locality and high data access to computation ratio [5]. Therefore, graph problems are not well matched with existing processing approaches and usually prevent efficient parallelism. MapReduce [6], for example, which addresses many shortcomings in previous parallel and distributed computing approaches, is not an appropriate solution for large graph processing. This is because first, MapReduce uses a two phased computational model (map and reduce) which is not well suited for iterative characteristic of graph algorithms. Second, its tuple-based approach is poorly suited for most of graph applications [7].

Cloud computing is a new paradigm of computing that has changed software, hardware and datacenters design and implementation. It overcomes restrictions of traditional problems in computing by enabling some novel technological and economical solutions like using distributed computing, elasticity and pay-as-you-go models which make service providers free from previous challenges to deliver services to their customers [8]. Cloud computing presents computing as a utility that users access various services based on their requirements without paying attention to how the services are delivered or where they are hosted. Public cloud computing services, for instance, offer Platform as a Service (PaaS) or Infrastructure as a Service (IaaS) for large distributed processing, are becoming more popular among companies who want to focus on their business instead of being concerned about technical issues. Rapid on-demand compute resource provisioning brings cost scalability based on utilization. As an example, Amazon EC2 provides three cost models for its customers based on their requirements – spot, on-demand and reserved provisioning. Using these commercial services, the customer may choose to pay more to achieve better performance or reliability. So, making a proper decision between using the number of resources the user wants to use and the money that the user wants to pay for the service is an issue while using public clouds.

Some graph processing frameworks such as Surfer [9] and Pregel.Net [10] were developed to support processing large graphs on public clouds, but they have considered some specific issues on these frameworks and do not address the impact of their solutions on the monetary cost of the

978-1-5090-2453-7/16 $31.00 © 2016 IEEE
DOI 10.1109/CCGrid.2016.38

system. For example, Surfer has proposed a graph partitioning method based on network latency and Pregel.Net, which is the .Net-based implementation of Pregel, has analyzed the impact of BSP graph processing models on public clouds using Microsoft Azure. On the other hand, there are some services such as Amazon relational database service (RDS) which is designed for traditional relational databases [11]. It is aimed at facilitating the set-up, operation and scaling a relational database and comes with two reserved and on-demand instance packages. According to a recent report [12], graph databases are getting more and more attentions every day and many companies are going to use this kind of database for their businesses. So, in the near future, public cloud providers will introduce new graph processing services on their infrastructures.

However, current frameworks for graph processing have limitations that hinder their adoption in cloud platforms. First, the majority of them have been designed and tested on cluster environments and not clouds, hence they have not considered monetary optimization, which is a very important factor for service selection on clouds. Second, many graph processing frameworks focus on reducing the operation's execution time, reducing memory utilization, considering task priorities and so on to reduce the cost of processing, but considering a static pool of resources with known size. On the other hand, cloud computing provides high scalability on-demand resources that can help users to perform their tasks using various services. Therefore, a graph processing approach with performance guarantees and optimal cost is a must in a cloud setting.

In this paper, we propose a graph processing framework called iGiraph. It uses a cost-efficient dynamic re-partitioning approach that utilizes network traffic message pattern to reduce the number of virtual machines (workers) during the processing by migrating partitions and vertices to minimize the cost. The new repartitioning method also mitigates network traffic results in faster execution. Our work, iGiraph makes the following **key contributions**:

- iGiraph repartitions the graph dynamically across workers considering network traffic pattern to reduce the communication between compute nodes.
- iGiraph uses high degree vertices concept in partition level, with the convergent level of the algorithms that are running on the system. iGiraph manages the number of compute nodes using a proper combination of these methods.
- While cost is a very critical factor in service selection procedure for any user on a public cloud, iGiraph significantly reduces the cost of processing large-scale graphs with reasonably close runtimes to Giraph by its new approach.

The rest of the paper is organized as follow: section 2, explains the basic Apache Giraph framework and its features following by the vertex and algorithm categorization is used for our work. Section 3 gives details about iGiraph solutions. Section 4 shows iGiraph's implementation. Performance evaluation of the system is discussed in section 5 and finally,

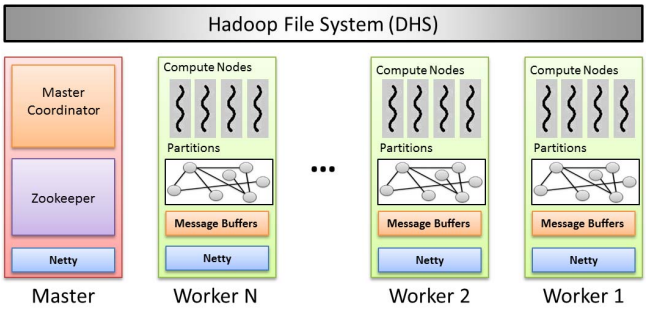

Fig 1. Giraph's Architecture

related works and conclusions and future works are explained in sections 6 and 7, respectively

II. BACKGROUND

In this section, we first introduce Apache Giraph [13] which is the fundamental framework for our system. Then, we explain Bulk Synchronous Parallel (BSP) [14] model following by describing a vertex categorization that is effectively used for our re-partitioning model. Finally, we explain the graph algorithm classification we used in this paper which has a great impact on choosing the right strategy to reduce the cost of the whole system.

A. Giraph

Apache Giraph is an open-source implementation of proprietary Pregel. It is a distributed graph processing framework that uses a set of machines (workers) to process large graph datasets. One of the machines plays the role of master to coordinate with other slave workers. The master is also responsible for global synchronization, error handling, assigning partitions to workers and aggregating aggregator values. Giraph is a Hadoop-based framework that runs workers as map-only jobs and uses Hadoop data file system (HDFS) for data I/O. It also employs Apache ZooKeeper [15] for checkpointing, coordination and failure recovery scheme. Giraph added many features beyond the basic Pregel including sharded aggregators, out-of-core computation, master computation, edge-oriented input and more. Finally, having a growing community of users and developers worldwide, Giraph has become a popular graph processing framework that even big companies such as Facebook are using it to process their huge datasets [16].

Giraph utilizes vertex-centric programming model like Pregel in which each vertex of the graph is identified by a unique ID. Each vertex also has other information such as a vertex value, a set of edges with an edge value for each edge, and a set of messages sent to it. To process a large graph in vertex-centric model, it should be partitioned into smaller parts by a partitioner where each partition is connected to other partitions by cross-edges between them. The partitioner also distributes partitions to a set of worker machines. In Giraph, a partitioner determines which partition a vertex belongs to based on its ID. Giraph uses a default hash function on the vertex ID to partition a graph while other customized partitioners also can be used. To improve the

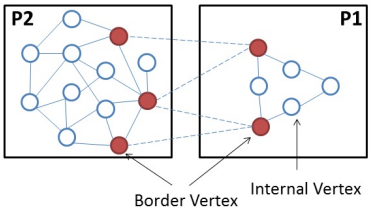

Fig 2. Internal vertices and border vertices

load balancing, the number of partitions is often greater than the number of workers.

Using simple static partitioning methods makes Giraph to run and process various graph algorithms slower and with more costs than other Pregel-like frameworks such as GPS [17] or Giraphx [18]. These systems have shown that using more complicated static partitioning algorithms such as METIS [19], rather than using a simple hash partitioning method, can remarkably improve the performance. GPS for example, uses a combination of a static partitioning algorithm to partition the graph and a dynamic re-partitioning algorithm during the computation to distribute the remaining non-processed vertices to idle workers to reduce the execution time within a superstep. In this paper, we choose the hash partitioning algorithm to start partitioning the graph with, but during the computation we replace that with a dynamic traffic-aware re-partitioning algorithm to reduce the cost of the whole processing operation and improve the performance of the system.

B. *Bulk Synchronous Parallel Model*

Bulk Synchronous Parallel (BSP) is a vertex-centric computational model in which every single vertex of the graph can carry two states of *active* or *inactive*. All vertices are *active* when the computation starts. The processing consists of a series of iterations, called *supersteps*, followed by global synchronization barriers between them. In each iteration, every vertex that is involved in computation, 1) receives its neighbors updated values from previous iteration, 2) the vertex then will be updated by received values, 3) and finally, the vertex sends its updated value to its adjacent vertices that will be available to them in the next superstep and changes its state to *inactive*.

The advantage of using BSP model in Giraph is that all the aforementioned operation is executed by a user-defined *Compute()* function of the *Vertex* class. After all the vertices completed executing *Compute()* function in a superstep, data will be aggregated during the synchronization phase and the messages generated by each vertex will be available to their destinations at the beginning of next superstep. If a vertex does not receive any messages during a superstep, it can deactivate itself by calling *voteToHalt()* function. However, a deactivated vertex can be activated by receiving messages from its neighbors. If there is not any active vertex, the computation is finished.

C. *Internal Vertices and Border Vertices*

A graph G=(V,E) consists of a set of vertices $V=\{v_1, v_2, \ldots, v_n\}$ and a set of edges $E=\{e_1,e_2, \ldots, e_m\}$ where $E \subset V \times V$. In the vertex-centric graph processing approach, the graph is divided into smaller partitions based on vertex divisions so that $P_1 \cup P_2 \cup \ldots \cup P_k = V$ are k partitions of V where $P_i \cap P_j = \varnothing$, $\forall i \neq j$. Therefore, each vertex basically belongs to only one particular partition [20].

An internal vertex is a vertex that all its adjacent vertices are inside the same partition as this particular vertex is. So, the messages coming out from an internal vertex only flow within the partition. On the other side, a border vertex is a vertex that at least one of its neighbors is placed in another partition. Hence, a border vertex's outgoing messages need to be sent to at least one different partition than the partition this particular vertex belongs to. Passing messages between partitions leads to increasing network traffic which results in longer execution time, inefficient resource utilization and higher costs in turn. Internal vertices and border vertices are shown in figure 2. One of the approaches for avoiding message passing side effects is to partition the graph in a way that reduces the number of border vertices and cross-edges between partitions so that the number of messages passing between partitions will be reduced.

D. *Graph Algorithms*

Different research use different classification of algorithms. For example, one may classify graph algorithms into traversal algorithms, graph aggregation algorithms, random walk algorithms and so on, while another one classifies them as global queries and targeted queries [21]. In this paper we use our own classification which categorizes graph algorithms based on their behavior in network traffic making and generating messages during the processing. We classify algorithms into two groups as follow:

- Non-Convergent Algorithms: Non-convergent algorithms are the algorithms that generate almost the same number of messages during processing. They complete the processing by passing the same number of messages in the last superstep as the number of messages they passed during first supersteps. So, the number of messages are generated using these applications never tends to become zero. PageRank [22], for instance, is a non-convergent algorithm.
- Convergent Algorithms: In contrast to non-convergent algorithms, the number of messages are generated using convergent algorithms tend to fall down to zero by the end of processing operations. Computing shortest paths [23] and connected components [24] algorithms are among convergent algorithms.

Here, we give a brief explanation of the algorithms we use from each category.

1) *PageRank*

PageRank is an algorithm which is used to measure the significance of website pages. PageRank works by measuring the number of links (hyperlinks) to a page to specify an importance estimation of a website. The more

important the page is, the more links it receives from other pages. PageRank does not rank a website as a whole, but is assessed by each page exclusively. The PageRank of page P_i does not impress the PageRank of a typical page P uniformly because of different weights that each page has. The summation of weighted PageRanks of all pages P_i then is multiplied by an alleviation factor 'd' that usually is set between 0 and 1. PageRank is also a non-convergent algorithm according to above classification because it produces the same number of messages in each superstep during a processing operation.

2) Connected Components

A connected component algorithm finds different sub-graphs of a particular graph in which there is a path between any two vertices and that is not connected to any further vertices in the super-graph. We use HCC that starts with having all vertices in an initial active state. Each vertex starts computing by considering its ID as its component ID and update this component ID when it receives a smaller component ID. The vertex then propagates the updated value to its adjacent vertices. Connected component is a convergent algorithm because the number of passing messages between vertices tends to fall down to zero as the states of vertices change to inactive until the end of computation.

3) Single Source Shortest Paths

The shortest path in graph theory is the problem of discovering a path between two nodes such that the summation of the weights of its edge components is minimized. This is a well-known problem in graph theory and there are different approaches and applications applying various solutions to various problems in this field.

Single source shortest path (SSSP) problem is one derivation of the main shortest path problem. This problem needs to find a shortest path between a single source node and all other vertices in the graph. In this algorithm, each vertex initializes its value (distance) to INF (∞), while the source node put 0 as its distance. INF is larger than any possible path from the source node in the graph. In the first superstep, only the source node updates its neighbors; in the next superstep, the updated neighbors will send messages to their own neighbors and so on. The algorithm completes when there is no more updates happening and the states of vertices also changed to inactive. So, SSSP is a convergent algorithm according to the aforementioned definition.

E. Graph Processing Challenges on Clouds

A large-scale graph processing operation that includes a series of iterations to process a graph usually causes considerable overheads due to its large memory consumption, CPU utilization, error handling, etc. Accordingly, various frameworks are proposed to optimize and improve the performance of graph processing operations. Although many of these frameworks offer specified scalability improvements on high performance clusters with fast interconnections, their performance on cloud environments in which some critical factors such as service cost is determinative, is less studied. So, there are not many works that considered monetary optimizations. Besides, many existing frameworks consider memory utilization, runtime reduction, tasks prioritization and so on by using constant number of resources. So, they are not utilizing clouds elasticity and scalability that are important characteristics of cloud environments and can have significant impact on monetary costs. Our work is scoped to reduce the monetary cost of processing large-scale graphs on public clouds by proposing a Pregel-like framework.

III. IGIRAPH

A. Motivation

iGiraph utilizes a distributed architecture on top of Hadoop and uses its distributed file system for data I/O. It is a Pregel-like graph processing system which means it employs vertex-centric processing solutions to process a graph and follows Pregel-like systems' behaviors. The problem with many of existing graph processing systems, particularly Pregel-like frameworks, is that although they propose methods to run the processing faster and improve the performance of the system, resource utilization and monetary cost factors are less studied. Nonetheless, cost is a crucial factor for every business that wants to use public cloud infrastructure. As cloud providers are using pay-as-you-go models for the services they are providing, considering the factors that have impacts on the cost of the services is very important for customers to choose the right services. There are many factors that influence the whole processing costs in a cloud environment including:

- Execution time: The longer the operation takes, the more user has to pay.
- Resource costs: Every resource has its own price. So, choosing the right number of machines with the right size can make huge differences.
- Communication: Sending and receiving data in a cloud environment is not free hence reducing the cost of communication for each operation is vital.
- Storage: Storing data could also become costly specially, for big data related services.

One of the most important parts of a graph processing system is the partitioning method that is used to partition and distribute data across the workers. Choosing between various static partitioning methods or between static and dynamic partitioning approaches can affect the system performance and cost. iGiraph uses a dynamic graph re-partitioning method which considers the main cost factors and improves the processing performance.

B. iGiraph's Dynamic Re-partitioning Approach

iGiraph's repartitioning algorithm uses the concept of high degree vertices in partition level and merges the partitions to reduce the number of cross-edges between them by migrating partitions from one worker to another. During

this process, some workers (resources) gradually become empty and can be released to decrease the cost of resource utilization.

In many real world graphs only a few number of nodes contains a large fraction of all the edges in the graph [17]. These vertices are known as *high degree* vertices. While the number of edges connecting to a vertex states the degree of that vertex, a high degree vertex has much more connected edges compared to majority of the vertices in a graph. For example, in a social network, a singer, an actor or celebrities can have millions of followers in comparison with the average of tens or hundreds of friends and followers for an ordinary user.

High degree vertices can play an important role in causing network traffic and delaying the execution time specially when they are placed as border vertices in partitions or close to border vertices. That is, putting high degree vertices as close as possible to their neighbors can significantly improve the network and system performance. Figure 3 shows the importance of this issue. In Figure 3.a vertex *v* from partition P1 is connected to many vertices in P2 results in huge network traffic while passing messages between two partitions and therefore delays the run-time and increases the cost. But as is shown in Figure 3.b, moving *v* to P2 can remarkably reduce the cross-edges between two partitions.

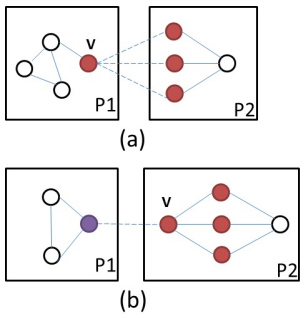

Fig 3. The role of high degree border vertices in reducing network traffic

iGiraph uses high degree vertex concept in partition level not vertex level. It means that as there are vertices with higher degree than other vertices in the graph, there are also partitions that send or receive more messages than other partitions in the graph of system workers. In order to store the information about which partition has sent or received more messages, iGiraph uses two separate lists. One stores the number of outgoing messages from each partition and the other, stores the number of incoming messages to each partition. We also define α, which is a threshold that is an average value for the number of messages that are transferring between each pairs of partitions. α is defined as follows:

$$\alpha = \frac{\sum_{i=1}^{N_P} \sum_{j=1}^{n} N_m(P_j, P_{j+1})}{N_P}, \qquad (1)$$

In the above formula, $N_m(P_j,P_{j+1})$ shows the number of messages between partition *j* and partition *j+1*, N_P shows the number of partitions that are involved in each superstep and *n* is calculated based on the number of partitions to show the number of pairs in each superstep. This formulation is calculated between each supersteps in iGiraph. According to this:

$$n = \begin{cases} N_P \times \left\lceil \frac{N_P}{2} \right\rceil & \text{If } N_p \text{ is odd} \\ (N_P - 1) \times \left\lceil \frac{N_P}{2} \right\rceil & \text{If } N_p \text{ is even} \end{cases} \qquad (2)$$

If the number of messages received by a partition is equal or greater than α, then that partition is a potential candidate for migration, otherwise the program looks at the number of outgoing messages at that partition to see if it can host vertices from other partitions or merge with them. Using factor α alone, border vertices can migrate between partitions.

Although α is a determinative factor to specify which partitions are suitable for migration and merging, there are other important factors that can influence the final decision as well. One factor is the number of total messages transferred between all partitions in a particular superstep compared to the number of total messages transferred between all partitions in previous superstep. Merging (not migration) only can occur if this proportion is decreasing. As long as the number of messages is growing during the processing, no merging will happen.

Another factor that determines whether partitions can merge is the size of partitions and workers' capacities. As the processing continues, for convergent algorithm such as connected components and shortest path, the vertices that complete the computation change their states to *inactive*. So, instead of keeping these vertices in the memory until the end of processing operation, iGiraph deletes them temporarily from memory to provide room for partition merging. On the other side, if a removed vertex is invoked during the computation, iGiraph can bring it back to the memory. So, before merging two partitions, the system checks if the destination worker has enough space or not.

When all above conditions are true, then migrating a partition from one worker to another worker to merge it with the other partition is possible. This has influences on the total cost of the service. For example, according to Figure 4, partition P1 is a high degree partition, which means it has the greatest number of incoming messages among other partitions, and is placed on worker W1. Partition P2 which is placed on worker W2 has sent the greatest number of messages to P1, P3 is in the second place after P2, P4 is next and so on. In addition, total number of transferred messages in current superstep (i+1) is less than transferred messages in previous superstep (i) and the workers have sufficient memory after removing inactive vertices. At this time, P1 will merge with P2 until there is free space on W2. Additional vertices will be migrated to W3 and so on. A load balancer balances the number of vertices in each partition on remaining workers.

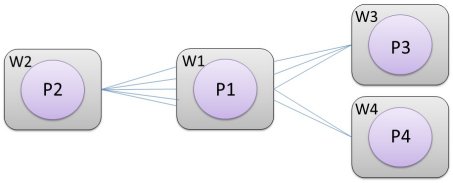

Fig 4. Worker W2 has sent more messages to W1 than other workers

According to our experiment results, using the proposed re-partitioning algorithm for convergent applications can reduce the cost of resource utilization while the execution time is close to Giraph's experiment results or with only a bit of increasing in some cases, but still do not affect the whole results.

For non-convergent applications, iGiraph does not merge the partitions. So, the number of workers will remain the same from beginning of the processing to the end. Instead, only border vertices from high degree partitions will be migrated to reduce the cross-edges between partitions. In this case, the total average number of transferred messages is mitigated which leads to faster execution compared to Giraph.

IV. IGIRAPH IMPLEMENTATION

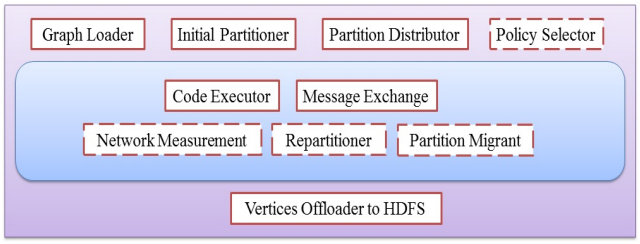

Fig 5. System architecture and components

Figure 5 shows the iGiraph's system architecture and components added to basic Giraph. The components that are surrounded by simple lines are basic Giraph's that are used in iGiraph too. The components that are surrounded by dashed lines are the components which are added to the basic framework. Like Giraph, data is loaded and stored on HDFS. Then, an initial partitioner function will partition the graph and prepare the partitions for being distributed across workers. In this paper we use only a simple hash function as initial partitioner. The hash function partitioning method is proved that results in worst performance compared to other complicated initial partitioning methods. Hence, we want to reach a better performance using this approach to show that our method can work very well even in this case. In the next step, partitions will be distributed across workers. The policy selector selects the appropriate computation method based on the type of application. For example, if the algorithm is convergent it enables partition migration. Code executer is the main *Compute()* function that executes the algorithm on each active vertex. After that, according to the number of messages transferred between partitions during the superstep, a network measurement component will determine which partitions have sent or received messages in a descent order. Then the repartitioner chooses vertices or partitions to migrate or merge according to the policy is selected. This will be done by the partitions migrant. This process will continue until all the vertices in the graph change their states to inactive and there is no more vertices to be computed. Finally, the results will be written back to HDFS.

V. PERFORMANCE EVALUATION

A. Experimental Setup

We chose shortest path and connected components algorithms among convergent applications and PageRank among non-convergent applications for our experiment. We also use three real datasets [25] of varying sizes: Amazon, YouTube and Pokec which is a Slovak social network.

TABLE I. EVALUATION DATASETS AND THEIR PROPERTIES [25]

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

We use *m1.medium* NECTAR VM instances for all partition worker roles. NECTAR is Australian national cloud infrastructure facilities [26]. 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 use Apache Hadoop version 0.20.203.0 and Apache Giraph version 1.1.0 with its checkpointing characteristic turned off. All experiments run using 16 instances where one takes the master role and others are set up as workers.

B. Evaluation and Results

First, we investigate the impact of our proposed approach on convergent algorithms and compare the results with basic Giraph. Then, we investigate non-convergent PageRank algorithm on both frameworks.

1) Evaluation of Convergent Algorithms

Figure 6 and 7 show the results of comparison experiments between Giraph and iGiraph on Amazon and Pokec datasets respectively. Considering that the size of every network message is the same in all experiments, here the computation can converge faster using iGiraph while the number of messages passing through network is reduced significantly. In Figure 7, after using factor α the number of messages increased a bit at first superstep, but noticeably decreased after that and still shows significant network message reduction compared to Giraph.

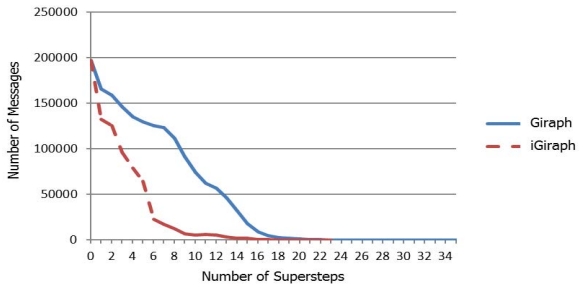

Fig 6. Number of network messages transferred between partitions across supersteps for the Amazon graph using connected components algorithm

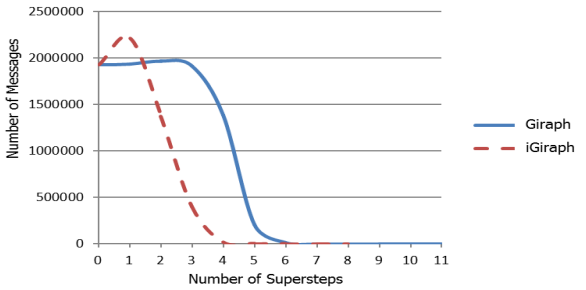

Fig 7. Number of network messages transferred between partitions across supersteps for the Pokec graph using connected components algorithm

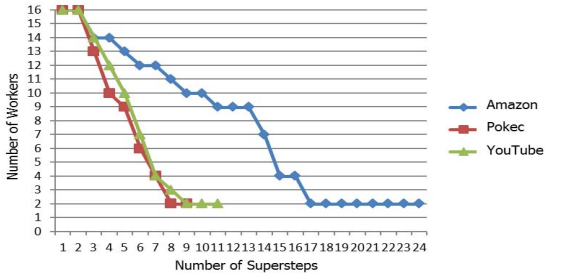

Fig 8. Number of machines varying during supesteps while running connected component algorithms on different datasets on iGiraph

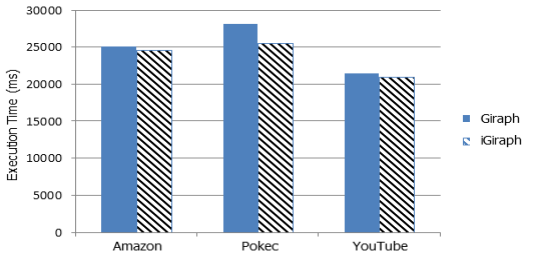

Fig 9. Total time taken to perform connected components algorithm

In contrast to Giraph in which the number of workers is kept intact during the whole operation, iGiraph releases compute nodes as the graph get converged. That is because by keeping only active vertices for the operation and doing repartitioning between each supersteps, less computation resources are required to continue the processing. We observed that by removing inactive vertices after each superstep, we could merge more partitions to use the capacities of each worker's memory efficiently. So, the more partition merge, the more resources can be freed which results in more money saving. But this claim only can be true when we consider both resource reduction and execution time together.

$$Cost_{final} = \sum_{i=1}^{n} \left(P(VM_i) \times T_{total}(VM_i) \right) \quad (3)$$

According to the above formulation, total cost of using resources on a cloud environment is equal to the summation of the price of each resource $P(VM_i)$ multiplied by total time of using that resource $T_{total}(VM_i)$. To calculate the final cost for the whole processing operation beside reducing the number of resources, we need to measure the system run-time too. Note that although data transfer also has impact on the final cost calculation, we have not considered that here, but we will take it into consideration for our future works. Figure 9 shows the execution time for processing aforementioned datasets using connected components algorithm. It shows that in addition to decreasing the cost of resource utilization, the run time for the operation is also reduced. Therefore, according to formula 3, the total cost of the operation falls down too.

Similar to previous evaluations for connected component algorithm, we repeated experiments using shortest path algorithm for both Giraph and iGiraph. From the network traffic point of view, the difference between shortest path and connected component is that the former starts with passing a few number of messages at the beginning of computation and gradually increases until reach a maximum and then starts converging, but connected component starts with passing great number of messages hence it immediately starts the convergence process. Figure 10 shows the results of a comparison experiment between Giraph and iGiraph on Amazon dataset using connected components algorithm. It takes 37 supersteps for this process to be completed on Giraph while it converges around superstep 23 using iGiraph. This is because in contrast to Giraph in which the number of messages starts falling down from superstep 14, using factor α, this happens to iGiraph after superstep 8. From this point onwards in iGiraph, three conditions for partition merging are provided and according to Figure 13 it can be seen that the number of active workers are decreasing. The results for Pokec and YouTube are shown in Figure 11 and 12, respectively.

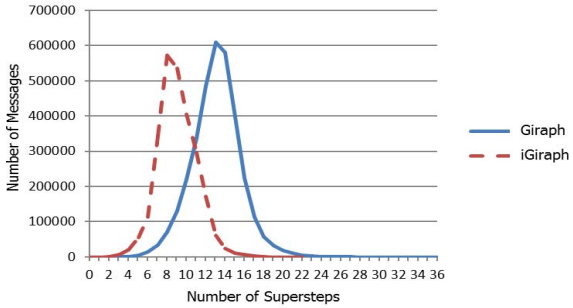

Fig 10. Number of network messages transferred between partitions across supersteps for the Amazon graph using shortest path algorithm

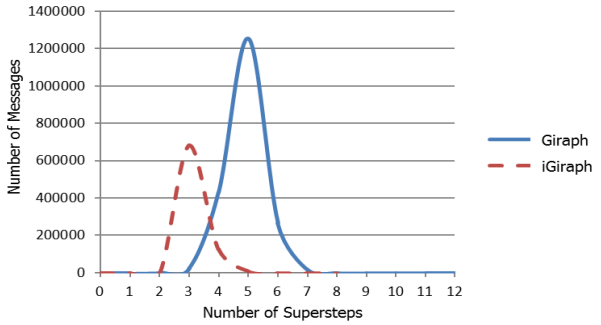

Fig 11. Number of network messages transferred between partitions across supersteps for the Pokec graph using shortest path algorithm

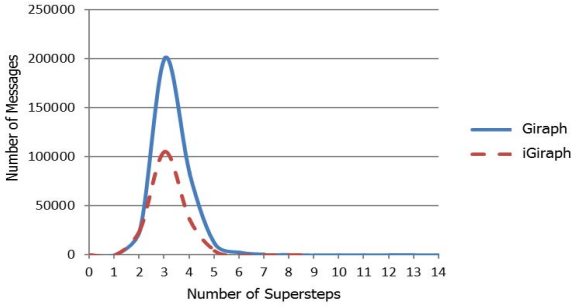

Fig 12. Number of network messages transferred between partitions across supersteps for the YouTube graph using shortest path algorithm

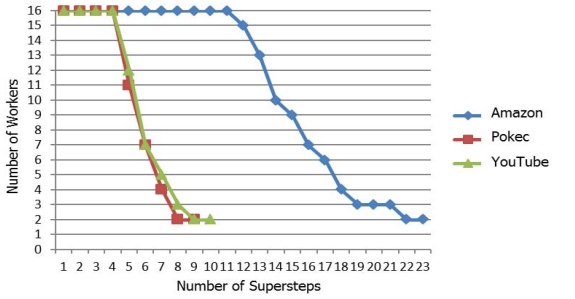

Fig 13. Number of machines varying during supesteps while running connected component algorithms on different datasets on iGiraph

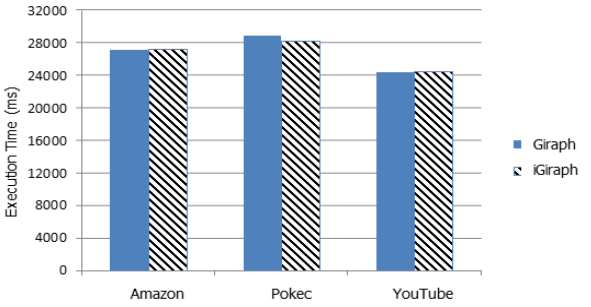

Fig 14. Total time taken to perform shortest path algorithm

The above figure shows that the time taken to complete shortest path algorithm on 16 machines using iGiraph is not significantly different than Giraph. As a result, considering total execution time and decreasing number of active workers in each experiment, iGiraph is more cost-effective than Giraph for convergent algorithms on public clouds.

2) Evaluation of Non-Convergent Algorithms

Processing non-convergent algorithms such as PageRank shows that the number of messages generated in each superstep is almost the same as other supersteps during the whole processing. In PageRank for example, vertices always update their neighbors during the computation hence as long as the number of vertices is the same, the number of messages is also the same. But it is still possible to reduce the network messages by using α factor. α determines the partitions that receive more messages through network than the other partitions (high degree partitions). Then, to balance the messaging pattern, iGiraph selects a number of border vertices from high degree partitions to relocate based on the aforementioned algorithm in section 4. After relocating the vertices, a load balancer method will balance the number of vertices in each partition. It can be seen that the average number of network messages falls down a bit in iGiraph results in faster computation. Figure 15 shows the average number of network messages in both Giraph and iGiraph. The total execution time for each experiment also can be seen in figure 16. According to these figures, although we did not decrease the number of workers like what was done for convergent algorithms, total runtime of the system decreased because there are few messages passing through network compared to Giraph.

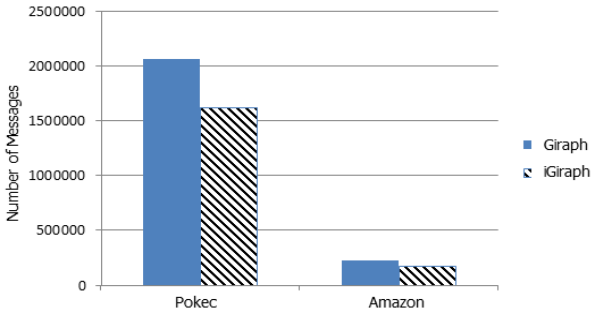

Fig 15. The average number of network messages in each experiment

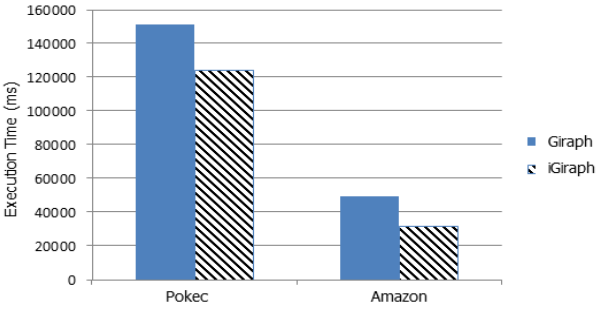

Fig 16. Total time taken to perform PageRank algorithm

VI. Related Work

According to The National Research Council of the National Academies of the United States [27], graph processing is one of the seven computational giants of massive data analysis. Google's Pregel [28] is the first graph processing framework in the literature that uses a bulk synchronous parallel (BSP) model [14] for graph computation based on a vertex-centric approach. Public implementations of this framework include Giraph [13], GoldenOrb [29], Apache Hama [30], etc. These frameworks are developed based on distributed architectures in which usually one machine acts as the master and one or several other machines act as workers. In the master-worker approach, the input graph is split into partitions and each partition assigns to a worker to process it. Many of graph processing frameworks use a simple hash function for partitioning the graph. However, such simple partitioning leads to huge network traffic in a graph processing task that consequently affects the system performance. To improve the partitioning efficiency, various approaches are proposed in different frameworks [31] [32] [33]. While most graph processing systems offer some specified improvements on HPC clusters with fast interconnects, their conduct on virtualized commodity hardware which is provided by cloud computing paradigm and is accessible to a wider population of users is less investigated [10].

Frameworks designed to process large-scale graphs based on Pregel are called Pregel-like frameworks. They are designed based on distributed architecture on high performance computing systems such as distributed clusters. Although graph processing systems created to overcome previous large data processing solutions such as MapReduce, some of distributed frameworks use series of MapReduce jobs iteratively. Giraph [13] and Surfer [9] are examples of these systems. Other features of Pregel-like frameworks include using bulk synchronous parallel (BSP), message passing communication method and global synchronization barrier between supersteps. However, systems such as GraphLab [34] provide asynchronous computations. Since iGiraph is a Pregel-like system and developed based on Giraph, it contains all of these specifications with some additional features such as dynamic repartitioning and cost minimization. There are many non-Pregel graph processing frameworks developed on distributed architecture. Among these frameworks are Trinity [35] and Presto [36].

GPS [23] is the most similar to our work. It has an optimization called LALP (large adjacency-list partitioning) by which stores high degree vertices and use the list to send one message, instead of thousands for instance, to the partitions are containing those vertices. After the message gets to the destination, it will be replicated thousands times to the message queues of each vertex in its outgoing neighbors list. Instead of storing the list of vertices, iGiraph stores two lists of the number of outgoing and incoming messages from/to each partition that show which partitions are sending or receiving more messages. These lists are noticeably smaller than GPS's adjacency lists.

Another difference between our system and GPS is that high degree vertices in GPS are defined by the programmer, but in iGiraph, the decisions about migrating the partitions are making based on an automatic formula. In GPS, the programmer specifies a parameter τ. If the number of outgoing messages for any vertex is more than τ, it will be considered as high degree. Here, selecting the right value for τ is very important and can directly affect the system's performance.

There are previous studies on the performance effects of different partitionings of graphs on other systems. The main challenge in partitioning a graph is to find how to partition the data to gain better vertex or edge cuts with considering the simplicity of computation. Pregel, Giraph and GraphLab partition the graph by cutting the edges while PowerGraph [37] and X-Stream [38] cut vertices for partitioning. From another point of view, the majority of graph processing frameworks only use static partitioning approaches that means they only partition the graph once before the processing starts or they do it once during the computation. On the other hand, some frameworks such as GPS use dynamic repartitioning approach that allows them to repartition the graph multiple times during the computation based on some pre-defined features to achieve better performance.

VII. Conclusions and Future Work

Huge amount of data is created and stored in the form of graphs every day. In this paper, we presented iGiraph, a Pregel-like system developed based on Giraph for processing large-scale graphs on public clouds. iGiraph uses a new repartitioning method to reduce the number of messages passing through network by decreasing the number of cross-edges between partitions. It utilizes high degree concept in partition level for both convergent and non-convergent types of algorithms. iGiraph also considers processing large graphs as a service on public clouds. Therefore, it reduces the cost of resource utilization by decreasing the number of workers that are using for the operation and executes the applications within a period which is reasonably close to Giraph's time.

We plan to extend iGiraph to use other critical network factors such as network bandwidth and topology, and study the impacts of these factors on system performance. We also want to use other graph partitioning methods such as METIS instead of a simple hash partitioning approach to see how effective are those methods in iGiraph. To investigate the graph processing as a service (GPaaS) more, we will study the factors that affect quality of services for large graph processing services on cloud environment as well.

Acknowledgments

We want to thank NECTAR research cloud for their support through providing infrastructures for this research. We also thank Amir Vahid Dastjerdi, Adel Nadjaran Toosi and Chenhao Qu for their comments on improving this work. This work is partially supported by ARC Future Fellowship grant.

REFERENCES

[1] C. Snijders, U. Matzat and U.-D. Reips, "Big Data: Big gaps of knowledge in the field of Internet," *International Journal of Internet Science,* vol. 7, no. 1, pp. 1-5, 2012

[2] R. Sedgewick and K. Wayne, Algorithms (4th Edition), Upper Saddle River, NJ: Addison-Wesley Professional, 2011

[3] M. Prigg, "Facebook hits one billion users in a single day: Mark Zuckerberg reveals one in seven people on earth used the social network on Monday," [Online]. Available: http://www.dailymail.co.uk/sciencetech/article-3213456/Facebook-s-billion-user-day-Mark-Zuckerberg-reveals-one-seven-people-EARTH-used-social-network-Monday.html. [Accessed 01 09 2015].

[4] F. Pellegrini , "Current challenges in parallel graph partitioning", *Comptes Rendus Mécanique,* vol. 339, no. 2-3, pp. 90-95, 2011

[5] A. Lumsdaine, D. Gregor, B. Hendrickson and J. Berry, "Challenges in Parallel Graph Processing", *Parallel Processing Letters,* vol. 17, no. 1, pp. 5-20, 2007

[6] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", *Proceedings of Sixth Symposium on Operating Systems Design and Implementation*, San Francisco, California, USA , 2004

[7] F. N. Afrati, A. Das Sarma, S. Salihoglu and J. D. Ullman, "Vision Paper: Towards an Understanding of the Limits of Map-Reduce Computation", *Proceedings of Cloud Futures 2012 Workshop*, Berkeley, California, USA, 2012

[8] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility", *Future Generation Computer Systems,* vol. 25, no. 6, pp. 599-616, 2009

[9] R. Chen, X. Weng, B. He and M. Yang, "Large Graph Processing in the Cloud", *Proceedings of ACM SIGMOD International Conference on Management of data*, Indianapolis, Indiana, USA, 2010

[10] M. Redekopp, Y.Simmhan, V. Parasanna, :Optimizations and Analysis of BSP Graph Processing Models on Public Clouds", *Proceedings of 27th IEEE International Symposium on Parallel and Distributed Processing,* Boston, MA, 2013

[11] "Amazon Relational Database Service (RDS)", [Online], Available: https://aws.amazon.com/rds/

[12] "DB-Engines Ranking Per Database Model Category", [Online]. Available: http://db-engines.com/en/ranking_categories.

[13] "Apache Giraph", [Online]. Available: http://giraph.apache.org/

[14] L. G. Valiant, "A bridging model for parallel computation", *Communications of the ACM,* vol. 33, no. 8, pp. 103-111, 1990

[15] "Apache ZooKeeper", [Online]. Available: https://zookeeper.apache.org/

[16] A, Ching, "Scaling Apache Giraph to a Trillion Edges", 2013, [Online], Availavle: https://www.facebook.com/notes/facebook-engineering/scaling-apache-giraph-to-a-trillion-edges/10151617006153920

[17] S. Salihoglu and J. Widom, "GPS: A Graph Processing System", *Proceedings of 25th International Conference on Scientific and Statistical Database Management*, Baltimore, Maryland, 2013

[18] S. Tasci and M. Demirbas, "Giraphx: Parallel Yet Serializable Large-Scale Graph Processing", *Proceedings of 19th international conference on Parallel Processing (Euro-Par'13)*, Aachen, Germany, 2013

[19] G. Karypis and V. Kumar, "Multilevel Graph Partitioning Schemes", *Proceedings of The International Conference on Parallel Processing*, Raleigh, NC, US, 1995

[20] Y. Tian, A. Balmin, S. Andreas Corsten, S. Tatikond and J. McPherson, "From "Think Like a Vertex" to "Think Like a Graph"", *Proceedings of the VLDB Endowment,* vol. 7, no. 3, pp. 193-204, 2013

[21] U. Kang, H. Tong, J. Sun, C.-Y. Lin and C. Faloutsos, "GBASE: A Scalable and General Graph Management System", *Proceedings of 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, San Diego, California, 2011

[22] L. Page, S. Brin, R. Motwani and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web", Stanford InfoLab, 1998

[23] P. Roy, "A new memetic algorithm with GA crossover technique to solve Single Source Shortest Path (SSSP) problem", *Proceedings of 2014 Annual IEEE India Conference*, Pune, India, 2014

[24] S. Salihoglu and J. Widom, "Optimizing Graph Algorithms on Pregel-like Systems", *VLDB Endowment,* vol. 7, no. 7, pp. 577-588, 2014

[25] J. Kunegis, "KONECT - The Koblenz Network Collection", *Proceedings of International.Web Observatory Workshop*, 2013.

[26] "NECTAR Cloud", [Online], Availavle: http://nectar.org.au/research-cloud/

[27] Committee on the Analysis of Massive, Committee on Applied and Theoretical Statistics, Board on Mathematical Sciences and Their Applications, Division on Engineering and Physical Sciences and National Research Council, *Frontiers in Massive Data Analysis*, The National Academies Press, 2013

[28] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser and G. Czajkowski, "Pregel: A System for Large-Scale Graph Processing", *Proceedings of The 2010 ACM SIGMOD International Conference on Management of Data*, 2010

[29] L. Cao , "GoldenOrb", 2011. [Online]. Available: https://github.com/jzachr/goldenorb. [Accessed 25 July 2015]

[30] "Apache Hama", [Online], Available: https://hama.apache.org/

[31] U. Elsner, Static and Dynamic Graph Partitioning. A Comparative Study of Existing Algorithms, Berlin, Germany: Logos Verlag Berlin, 2002

[32] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders and C. Schulz, "Recent Advances in Graph Partitioning", arXiv preprint arXiv:1311.3144, 2013

[33] D. A. Bader, H. Meyerhenke, P. Sanders and D. Wagner, Graph Partitioning and Graph Clustering, Atlanta, GA, US: American Mathematical Society, 2013

[34] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin and J. M. Hellerstein, "Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud", *VLDB Endowment,* vol. 5, no. 8, pp. 716-727,2012

[35] B. Shao, H. Wang and Y. Li, "Trinity: A Distributed Graph Engine on a Memory Cloud", *Proceedings of ACM SIGMOD International Conference on Management of Data*, New York, USA,2013

[36] S. Venkataraman, E. Bodzsar, I. Roy, A. AuYoung and R. S. Schreiber, "Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices", *Proceedings of 8th ACM European Conference on Computer Systems*, Prague, Czech Republic, 2013

[37] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson and C. Guestrin, "PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs", *Proceedings of 10th USENIX conference on Operating Systems Design and Implementation*, Hollywood, CA, 2012.

[38] A. Roy, I. Mihailovic and W. Zwaenepoel, "X-Stream: Edge-centric Graph Processing using Streaming Partitions", *Proceedings of 24th ACM Symposium on Operating Systems Principles*, Farmington, USA, 2013.