Proxy-Guided Load Balancing of Graph Processing Workloads on Heterogeneous Clusters

Shuang Song, Meng Li, Xinnian Zheng, Michael LeBeane, Jee Ho Ryoo, Reena Panda, Andreas Gerstlauer, Lizy K. John
The University of Texas at Austin, Austin, TX, USA
{songshuang1990, meng_li, xzheng1, mlebeane, jr45842, reena.panda, gerstl, ljohn}@utexas.edu

***Abstract*—Big data decision-making techniques take advantage of large-scale data to extract important insights from them. One of the most important classes of such techniques falls in the domain of graph applications, where data segments and their inherent relationships are represented as vertices and edges. Efficiently processing large-scale graphs involves many subtle tradeoffs and is still regarded as an open-ended problem. Furthermore, as modern data centers move towards increased heterogeneity, the traditional assumption of homogeneous environments in current graph processing frameworks is no longer valid. Prior work estimates the graph processing power of heterogeneous machines by simply reading hardware configurations, which leads to suboptimal load balancing.**

In this paper, we propose a profiling methodology leveraging synthetic graphs for capturing a node's computational capability and guiding graph partitioning in heterogeneous environments with minimal overheads. We show that by sampling the execution of applications on synthetic graphs following a *power-law* distribution, the computing capabilities of heterogeneous clusters can be captured accurately (<10% error). Our proxy-guided graph processing system results in a maximum speedup of 1.84x and 1.45x over a default system and prior work, respectively. On average, it achieves 17.9% performance improvement and 14.6% energy reduction as compared to prior heterogeneity-aware work.

I. Introduction

The amount of digital data stored in the world is considered to be around 4.4 zettabytes now and is expected to reach 44 zettabytes before the year 2020 [1]. As data volumes are increasing exponentially, more information is connected to form large graphs that are used in many application domains such as online retail, social applications, and bioinformatics [2]. Meanwhile, the increasing size and complexity of the graph data brings more challenges for the development and optimization of graph processing systems.

Various big data/cloud platforms [3] [4] are available to satisfy users' needs across a range of fields. To guarantee the quality of different services while lowering maintenance and energy cost, data centers deploy a diverse collection of compute nodes ranging from powerful enterprise servers to networks of off-the-shelf commodity parts [5]. Besides requirements on service quality, cost and energy consumption, data centers are continuously upgrading their hardware in a rotating manner for high service availability. These trends lead to the modern data centers being populated with heterogeneous computing resources. For instance, low-cost ARM-based servers are increasingly added to existing x86-based server farms [6] to leverage the low energy consumption.

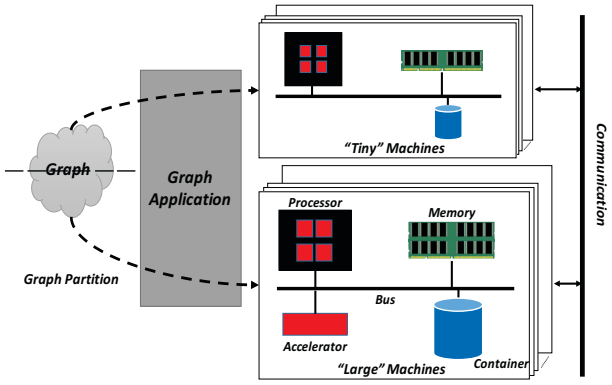


Fig. 1: Uniform graph partition for heterogeneous cluster.

Despite these trends, most cloud computing and graph processing frameworks, like Hadoop [7], and PowerGraph [8], are designed under the assumption that all computing units in the cluster are homogeneous. Since "large" and "tiny" machines coexist in heterogeneous clusters as shown in Figure 1, uniform graph/data partitioning leads to imbalanced loads for the cluster. When given the same amount of data and application, the "tiny" machines in the cluster can severely slow down the overall performance whenever dependencies or the needs of synchronization exists. Such performance degradation has been observed in many prior works [5] [9] [10] [11] [12]. Heterogeneity-aware task scheduling and both dynamic and static load balancing [5] [13] [14] have been proposed to alleviate this performance degradation. Dynamic load balancing [13] is designed to avoid the negative impact of insufficient graph/data partitioning information in the initial stage, where heterogeneity-aware task scheduling [14] can be applied non-invasively on top of load balancing schemes.

Ideally, an optimal load balancing/graph partitioning should correctly distribute the graph data according to each machine's computational capability in the cluster, such that heterogeneous machines can reach the synchronization barrier at the same time. State-of-the-art online graph partitioning work [5] estimates the graph processing speed of different machines solely based on hardware configurations (number of hardware computing slots/threads). However, such estimates cannot capture a machine's graph processing capability correctly. Figure 2 shows that different applications and machines scale differently with increasing computational ability. The dotted line shows the resource-based estimates from prior work [5];

2332-5690/16 $31.00 © 2016 IEEE
DOI 10.1109/ICPP.2016.16

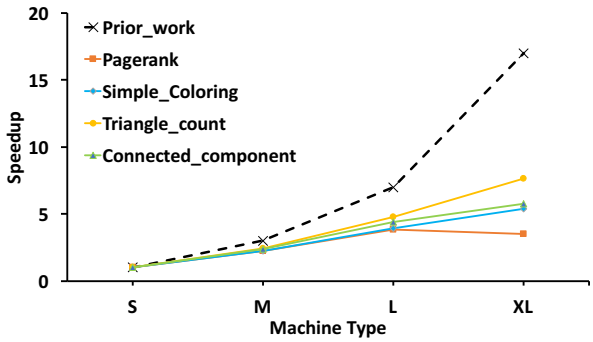


Fig. 2: Speedup estimated by prior work vs. real speedup.

the other lines show the actual scaling of different algorithms, illustrating the diversity of various graph applications.

In order to capture the computing capabilities of heterogeneous machines accurately, profiling is often the most effective methodology. However, computation demands also depend on applications and input graphs. It is difficult to subsample from a natural graph to capture its underlying characteristics, as vertices and edges are not evenly distributed in it. Again, this may lead to inaccurate modeling of machines' capability.

In this paper, we present a methodology that uses synthetic *power-law* proxy graphs' profiling to guide the graph partitioning for heterogeneous clusters. The specific contributions of our work can be summarized as follows:

1) We demonstrate that synthetic graphs following *power-law* distributions can be used as proxy graphs to measure a machine's graph processing capability in heterogeneous data centers. We define a Computation Capability Ratio (CCR) metric to represent the computing units' diverse and application-specific processing speeds in a heterogeneous cluster. Compared to state-of-the-art work [5], we reduce the heterogeneity estimation error from 108% to 8% with negligible overhead, as we only need to generate synthetic graphs once to cover real world graphs with a wide distribution range. Furthermore, profiling only needs to be done once for each reusable application in the heterogeneous cluster.
2) We illustrate CCR-guided graph partitioning performance and energy benefits in three cases. We achieve maximum and average speedups of 1.45x and 1.16x in a heterogeneous cluster formed by two Amazon EC2 nodes, for which prior work [5] cannot achieve any benefits. For a cluster constructed from nodes with different numbers of cores, we show a maximum and average speedup of 1.67x and 1.45x, which is 17.7% better than prior work. As "tiny" servers start to emerge as mainstream computing resources, we configure a cluster with machines that operate at different frequency ranges. In this case, our approach achieves an average speedup of 1.58x while prior work only sees a 1.37x speedup. Overall, we reduce energy by 25%, where as prior work only saves 10.4%.
3) Besides runtime and energy improvements, our methodology also shows advantages in measuring a machine's cost efficiency within commercial cloud computing platforms. It is difficult to select the right machines that provide high performance with reasonable cost simply by reading their hardware configuration information. Synthetic graph profiling can help to quantify the cost efficiency of formed clusters, or select the nodes with better cost efficiency for graph related work.

The rest of the paper is organized as follows. Section II defines our heterogeneity metric CCR and discuss how we incorporate it into the heterogeneity-aware graph partitioning algorithms. Section III explains how to generate representative synthetic *power-law* graphs and our profiling methodology in detail. Section IV describes our experimental setup, including machines, graphs and applications used in the experiments. We evaluate the CCR estimation accuracy of synthetic proxy graphs, demonstrate the performance and energy improvement for three heterogeneous clusters, and discuss the cost efficiency projection in Section V. We review the related state-of-the-art work in Section VI and conclude our paper in Section VII.

II. Heterogeneity-aware Graph Partitioning

In this section, we define our heterogeneity-aware CCR metric and discuss its incorporation into the heterogeneity-aware graph partitioning algorithms proposed in [5].

A. Definition of CCR

CCR is used to characterize the cluster's heterogeneity. It represents the application-specific relations between graph processing speeds of different types of machines in the cluster. Formally, for a given application i and machine j, $CCR_{i,j}$ is defined as follow:

$$CCR_{i,j} = \frac{\max(t_{i,j})\ \forall j}{t_{i,j}}, \tag{1}$$

where $max(t_{i,j})$ denotes the execution time of the slowest machine in the cluster. Several factors can affect CCR, such as the heterogeneity of the cluster, the degree distribution of synthetic graphs, and the graph applications themselves. Graph size is a trivial factor, since it only affects the magnitude of execution time while not reflecting the relative speedups in a heterogeneous cluster. A cluster's heterogeneity is the main component impacting CCR, as it is the variation on computing resources that determines the graph processing speeds and maximum parallelisms for graph applications. As shown in Figure 2, graph applications exhibit diverse scaling on the executing machines depending on the level of available parallelisms. The Pagerank application saturates at the "Medium" machine case. However, the other three applications still gain benefits from the increased number of hardware threads. The degree distribution also impacts the CCR in the way that denser graphs require more computation power and hence result in more speedup on fast machines. Our profiling process can completely cover these three important factors and generate accurate CCRs to guide the heterogeneity-aware graph partitioning algorithms and thus achieve load balancing.

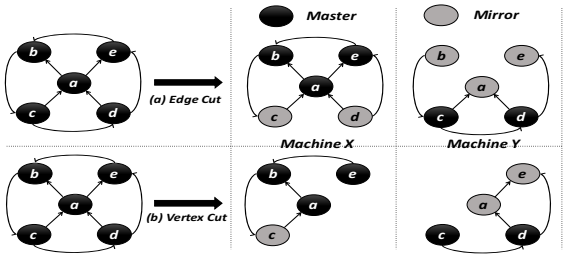


Fig. 3: Illustration of edge cuts vs. vertex cuts.

B. Vertex Cuts Algorithms

Graph partitioning algorithms can choose either edge cuts or vertex cuts, as shown in Figure 3. Vertex cuts split the vertices by assigning edges to different machines. Graphs with non-negligible amount of high-degree vertices prefer the vertex cut methods, as this can reduce the amount of replications of graph segments (defined as mirrors). Random Hash, Oblivious, and Grid are three partitioning algorithms using vertex cuts only. In this paper, we adopt the heterogeneity-aware algorithms proposed in [5] and extend the algorithms by employing a CCR-based partitioning.

1) Heterogeneity-aware Random Hash: The Random Hash is the baseline algorithm developed in [8] and reused in all other partitioning methods described in this section. To assign an edge, a random hash of edge e is computed and used as the index of the machine to assign the edge to. As shown in Figure 4, each machine has the same probability of receiving an incoming edge. In order to embed heterogeneity information, we extend the algorithm to weigh machines differently, such that the probability of generating indexes for each machine strictly follows the CCR.

2) Heterogeneity-aware Oblivious: The Oblivious partitioning algorithm is designed to enhance data locality by partitioning based on the history of edge assignments. The Original Oblivious algorithm is based on several heuristics that aim to assign an edge accounting for the load situation and the assignment of source and target vertices. To enable heterogeneity-aware partitioning, we follow similar ideas as in random hashing to assign machines different weights based on the CCR. Besides considering the load situation, this allows weights of different machines to be incorporated to guide the assignment of each edge. Note that the heuristics combined with CCR-guided weight assignment do not guarantee an exact balance in accordance with CCR.

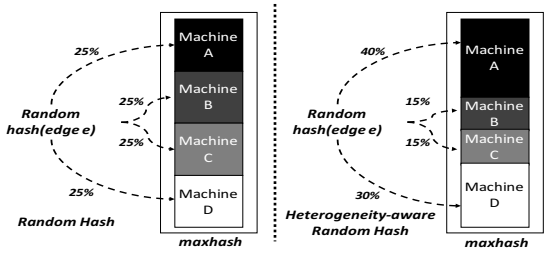


Fig. 4: Random Hash vs. heterogeneity-aware Random Hash.

3) Heterogeneity-aware Grid: The Grid method is designed to limit the communication overheads by constraining the number of candidate machines for each assignment. The number of machines in the cluster has to be a square number, as they are used to form a square matrix grid as displayed in Figure 5. A shard is defined as a row or column of machines in this context. Similar to the concept of heterogeneous Random Hash, each shard has its weight, which is determined from the weights of machines in the shard. Differently, every vertex is hashed to a shard instead of single machine. For each edge, two selected shards corresponding to the source and target vertices generate an intersection. Considering the current edge distribution and the edge placements suggested by CCR, each machine in the intersection receives a score. The edge will be allocated to the machine with the maximum score.

C. Mixed Cut Algorithms

Compared to vertex cuts, edge cuts shown in Figure 3 can significantly reduce mirrors for graphs with huge amount of low-degree vertices and few high-degree vertices. Different from all three algorithms discussed in Section II-B, mixed cut algorithms, including Hybrid and Ginger partitioning schemes proposed in [15], take advantage of both vertex and edge cuts.

1) Heterogeneity-aware Hybrid and Ginger: Hybrid and Ginger use two-phase methods to accomplish the partitioning. In the first phase, edge cuts are used to partition the graph. All edges are assigned to nodes based on the random hashes of target vertices. After the first pass, all in-degree edges of vertices with a small amount of edges are grouped with target vertices and no mirrors would be created. As the entire graph has been scanned through in the first phase, the total number of edges of each vertex can be easily obtained. In the second phase, all vertices with a large amount of in-degree edges (higher than a certain threshold) are randomly re-assigned by hashing their source vertices. For high-degree vertices, the number of mirrors are constrained by the number of partitions rather than the degree of vertices.

Ginger is a heuristic version of Hybrid, which was proposed by Fennel [16]. For high-degree vertices, it operates the same as Hybrid. For low-degree vertices, Ginger uses reassignment to achieve minimal replication in the second round. The reassignment of vertex v must satisfy equation 2 below.

$$score(v,i) > score(v,j), \forall j \in cluster, \tag{2}$$

where $score[v,i] = \left|N(v) \cap V_p\right| - \gamma * b(p)$ is the *score* function. V_p denotes v in machine p and $N(v)$ represents the number of neighboring vertices of v. $b(p)$ is a balance function to express the cost of assigning v to machine p, which considers both vertices and edges located on machine p.

The way of modifying the first pass and second pass (for high-degree vertices only) to be heterogeneity-aware is exactly the same as in the Random Hash method described above. A heterogeneity factor $1 - CCR_p$ is incorporated into the score calculation formula such that a fast machine has a smaller factor to gain a better score. The function $score.max()$ returns the machine ID with the maximum score in the list.

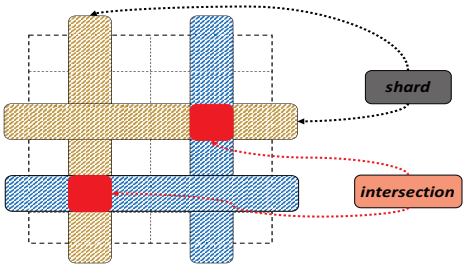


Fig. 5: Illustration of Machine Grid and Shards

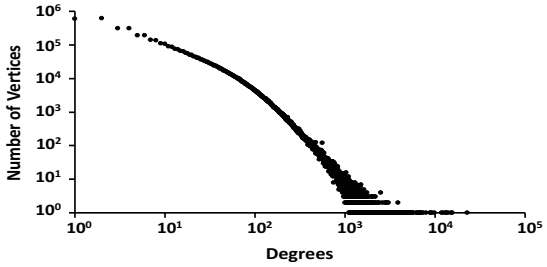


Fig. 6: Friendster Graph [20] following Power-law distribution.

III. Methodology

Our approach consists of utilizing synthetic proxy graphs following *power-law* distributions for profiling and CCR estimations. First, we give an overview of *power-law* distribution and describe how we generate synthetic proxy graphs. Secondly, we demonstrate the profiling process for the heterogeneous cluster using proxies. Lastly, we illustrate the execution flow of the graph processing framework with integrated CCRs.

A. Synthetic Power-Law Graph Generation

This section reviews the *power-law* distribution and demonstrates the algorithm that is used to generate synthetic proxy graphs following *power-law* distributions. A numerical procedure is used to compute the parameter α in the power-law distribution for real graphs. As we will show, the parameter can be used to tune the distribution/density of synthetic graphs to form samples with better coverage.

1) Power-law Distribution: It has been widely observed that most natural graphs follow *power-law* distributions [8] [17] [18] [19]. In our proposed methodology, we therefore generate synthetic *power-law* proxy graphs to characterize a machine's graph processing speed. A *power-law* distribution is a functional relationship between two objects in statistics, where one object varies as a power of another. A graph is defined to follow the *power-law* distribution if the distribution of the degree d of a vertex follows:

$$P(d) \propto d^{-\alpha}, \tag{3}$$

where the exponent α is the positive constant that controls the degree distribution. For instance, a high degree d leads to smaller probability *P(d)*, which results in a fewer amount of vertices with high degrees in the graph. Similarly, small values of the exponent α induce high graph density, where a small number of vertices have extremely high degrees, as shown in Figure 6 obtained from Friendster social network graph [20].

2) Synthetic Graph Generation: We implement a simple graph generator that can quickly produce graphs following *power-law* distributions. Since the performance of most graph applications is highly dependent on input graph distribution and sparsity, generated synthetic proxy graphs and real graphs need to follow similar distributions to achieve accurate profiling. However, it is impossible to use real graphs for profiling and CCR generation, as it is too expensive to profile the cluster once receiving a new graph. Furthermore, it is difficult to form a comprehensive sample graph set by randomly selecting natural graphs. However, these difficulties can easily be avoided by synthetic graphs. Note that having similar distributions does not guarantee the capability to predict real execution time. However, as we will show in experimental results, it is sufficient to detect heterogeneous machines' graph processing capabilities.

Our synthetic graph generator is illustrated in Algorithm 1. It takes the number of vertices N and α parameter as inputs. Based on distribution factor α, the probability of each vertex is calculated and associated with the number of degrees that will be generated later. Then, the probability density function (pdf) will be transformed into a cumulative density function (cdf). The total number of degrees of any vertex is generated by the cdf function. All the connected vertices are produced by a random hash. If directional edges are needed, the order of $edge(u,v)$ could be understood as the graph having an edge from u to v and vice verse. To omit self-loops, a condition check on vertex u being unequal to vertex v is added in the process, if necessary. The overhead of generating synthetic graphs depends on the graph size and distribution. Generating three deployed proxies took 67 seconds in total.

Algorithm 1 Synthetitc Graph Generator

```
1:  procedure GRAPH GENERATION
2:      for i ≤ N do
3:          pdf[i] = i^(-α)
4:      end for
5:      cdf = transform(pdf)
6:      hash = constant_value
7:      for u ≤ N do
8:          degree = multinomial(cdf)
9:          for d ≤ degree do
10:             v = (u + hash) mod N
11:             output_edge(u, v)
12:         end for
13:     end for
14: end procedure
```

3) Numerical Method for Computing α: In order to artificially generate power-law graphs for performance sampling, the parameter α that determines the sparsity level of the underlying graphs is essential. To precisely generate representative synthetic proxy graphs, we need to explore the distribution diversity of real graphs. In this section, we

describe a numerical procedure for computing the tunable parameter α of an existing natural graph with only the number of vertices and edges given. From the *power-law* distribution in Equation 3, we note that a characterization of the *power-law* distribution does not explicitly show the normalization constant. For the purpose of estimating α, it is convenient to work with the following characterization

$$P(d) = \frac{d^{-\alpha}}{\sum_{i=1}^{i=D} i^{-\alpha}}, \tag{4}$$

where D denotes the total number of degrees. We compute the first moment of the discrete random variable d as follow,

$$E[d] = \sum_{d=1}^{d=D} dP(d) = \sum_{d=1}^{d=D} \frac{d^{-\alpha+1}}{\sum_{i=1}^{i=D} i^{-\alpha}}. \tag{5}$$

Let $\mathscr{E}$ and $\mathscr{V}$ denote the sets of edges and vertices in a graph, respectively. We can approximate the average degree of a graph $E[d]$ empirically as follow,

$$E[d] = \frac{|\mathscr{E}|}{|\mathscr{V}|}, \tag{6}$$

where $|\mathscr{X}|$ denotes the cardinality of the set $\mathscr{X}$. Since the total number of edges and vertices of the input graph is given, we thus compute α by equating (5) with (6). Thus, we can express α as the root of the following function,

$$F(\alpha) = \sum_{d=1}^{d=D} \frac{d^{-\alpha+1}}{\sum_{i=1}^{i=D} i^{-\alpha}} - \frac{|\mathscr{E}|}{|\mathscr{V}|} = 0. \tag{7}$$

We then apply a standard Newton method for solving the root of the equation $F(\alpha) = 0$. Once α is computed, we can feed it into the synthetic graph generator. Normally, generating several synthetic proxy graphs with different α is a one-time procedure, which covers a wide range of real graphs, as most natural graphs follow *power-law* distribution with α parameters varying only within a limited range (from 1.9 to 2.4). However, in order to verify the coverage of generated synthetic graphs, we can calculate the α of each natural input graph. If its α is beyond the covered range, an additional synthetic graph can be generated and added to the current set. Our α computing process is extremely quick (less than 1ms), and the overhead is negligible.

TABLE I: Amazon Virtual Machine [3] and Local Physical Machine Configurations

Name	HW Threads	Computing Threads	Cost Rate	Type
c4.xlarge	4	2	$0.209/hour	Virtual
c4.2xlarge	8	6	$0.419/hour	Virtual
m4.2xlarge	8	6	$0.479/hour	Virtual
r3.2xlarge	8	6	$0.665/hour	Virtual
c4.4xlarge	16	14	$0.838/hour	Virtual
c4.8xlarge	36	34	$1.675/hour	Virtual
Xeon_Server_S	4	2	N/A	Physical
Xeon_Server_L	12	10	N/A	Physical

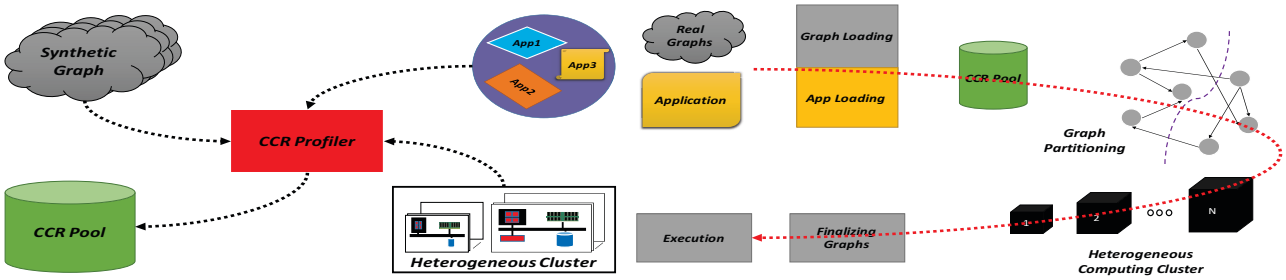
B. Profiling for Heterogeneous Cluster

The main idea of graph partitioning in heterogeneous environments, is to distribute input graphs onto different machines proportional to their CCRs. To accurately generate CCRs, we have to cover all the impacting factors, such as the heterogeneous machines, graph applications, and the distributions of the graphs. To do so, we need to profile graph applications executing in the heterogeneous cluster using synthetic graphs with diverse distributions. As shown in Figure 7a, we take the generated synthetic graphs as inputs and combine them with each graph application to form independent profiling sets. It is necessary to profile each application because graph applications are naturally diverse, as demonstrated in Figure 2. This implies that a single profiling set is not enough to cover all application characteristics. Moreover, our application-specific profiling methodology provides more flexibility, as any special-purpose application can be sampled and fit into our flow.

For a given heterogeneous cluster, we have to classify machines into different groups and select only one machine from each group, in order to minimize the profiling overhead. For instance, if the heterogeneous cluster is formed by Amazon EC2 virtual nodes [3], all *C4.xlarge* machines within the deployed cluster should be treated as one group, but only one of them needs to be profiled. After grouping, each profiling set is executed on one machine from each group in parallel. The purpose of running profiling sets on machines individually is that each machine's graph computation power can be captured without communication interference. Minimizing communication overheads for distributed graph frameworks is beyond the scope of this paper and is considered for future work.

After the parallel profiling process, the runtime of each machine group can be obtained. This runtime information is used to compute the speedup among machines. The CCR for the application is created from this speedup data. For example, if machine A runs profiling set X two times faster than the baseline machine B, the CCR for these two machines on profiling set X is $2:1$. After the profiling process finishes, each application's CCR will be collected into a CCR pool for future use. CCR profiling is a one-time offline process. The CCR pool needs to be updated whenever computing resources in the heterogeneous cluster change. However, re-profiling is only required if new machines types are deployed or machine characteristics otherwise change. Varying the cluster composition among existing machines does not require CCR updates. Given its low overhead, dynamic changes in resources can be captured by running the profiler and updating the CCR pool online at regular intervals. The benefits of our profiling method will be discussed in the Section V.

In contrast to our methodology, prior work [5] proposed to simply read a machine's hardware configuration (number of virtual cores) for estimating node computing capabilities. For example, the CCR between machine A with four hardware threads and machine B with eight hardware threads is $1:3$ $(4-2:8-2)$, as two logical cores on each node are reserved



(a) Illustration of profiling for CCR generation.

(b) Flow of the modified PowerGraph framework.

Fig. 7: Synthetic graphs profiling and heterogeneity-aware graph processing flow.

for communication. The inaccuracy caused by this naive estimation has been shown in Figure 2. Compared to this prior work, the overhead of our purposed profiling method may seems costly. However, each profiling set only needs to be executed once on a small subset of machines in the cluster. All generated CCR information is reusable over future executions, as graph applications are often reused to analyze dozens of different real world graphs.

C. Graph Processing Flow

We evaluate our proposed methodology using the PowerGraph [8] framework. Since the profiling work is done completely offline, our scheme is independent of the underlying setup and can be equally applied to other distributed graph processing frameworks. Figure 7b shows the execution flow of our current platform. Generally, graph processing inputs are the application, the graph, and other graph-related information, such as number of edges/vertices and the format. The framework first loads input graph files and the application. Then, based on the application, one corresponding CCR set would be picked from the pool, which is pre-generated by the offline profiling process described in Section III-B. Based on the application specific CCR and user selected partitioning algorithm (heterogeneity-aware partitioning algorithms described in Section II), the graph partitioner splits the graph into multiple chunks and distributes them onto nodes in the cluster accordingly. After the partitioning phase, the framework needs to finalize the graph by constructing the connections among machines, to achieve point-to-point communication and synchronization during execution. The last step of the flow is the application execution.

IV. Experiment Setup

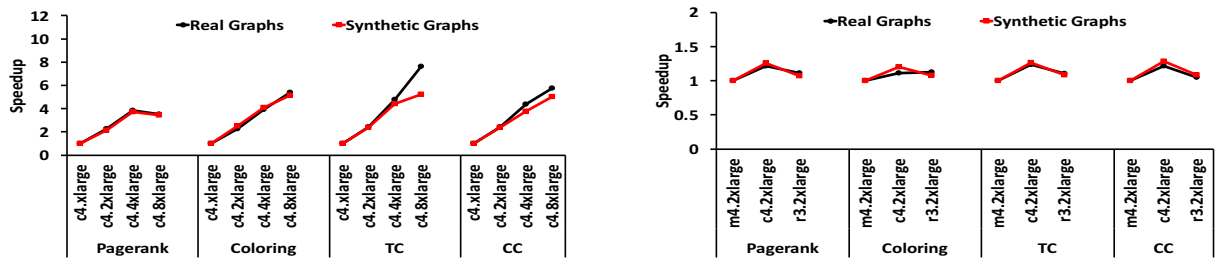
This section introduces our experimental setup, including the heterogeneous machine configurations, data sets, and graph applications used in the evaluation. As shown in Table I, we deploy both Amazon EC2 Virtual nodes [3] and local physical servers. We use EC2 virtual machines from three popular categories in our experiments, including C type (computation optimized), M type (general purpose) and R type (memory optimized). Due to the high heterogeneity provided by Amazon virtual computers, we verify the accuracy of our methodology among different machines under the same category and among machines with the same number of hardware threads across different categories. Performance studies are done in two parts. First, we compare application runtime achieved by our methodology to the performance achieved by the state-of-the-art work [5] using the cluster formed by *m4.2xlarge* and *c4.2xlarge*. These two types of machines have the same number of computing threads but behave differently. However, such performance differences will be ignored by prior work [5], which only differentiates machines by looking at the number of computing threads. Additionally, a performance comparison for a cluster consisting of machines with different amounts of hardware compute units is done on a local servers as this allows us to monitor the energy savings, which is not provided on Amazon EC2 platform [3]. Moreover, local servers provide us an opportunity to manipulate processor frequency ranges to emulate "mini" servers (such as ARM processors with lower compute capabilities or operating at lower frequencies) populated with modern data centers. This allows us to study and project a setup in future data centers future heterogeneous system designs.

Our cost study is entirely performed on Amazon, as it is one of the most popular cloud service providers and lists all the price information publicly [3]. All physical nodes on local servers have Intel Xeon E5 processors, and are connected via high-speed router. All performance information is measured and reported by the PowerGraph [8] platform. The processor and memory energy consumption data is recorded using Intel RAPL counters read by the Linux GNU perf toolset. For our studies, we select four real world graphs and generate three synthetic proxy graphs as shown in Table II. The size of the graph data varies from 40 Megabytes to over 1 Gigabyte with diverse edge density.

We selected four popular graph applications from machine learning and data mining (MLDM) fields. Those applications are briefly described as follows:

TABLE II: Real world graphs [20] and synthetic graphs.

Name	Vertices	Edges	Footprint	Alpha α
amazon	403,394	3,387,388	46MB	2.004
citation	3,774,768	16,518,948	268MB	2.169
social_network	4,847,571	68,993,773	1.1GB	1.950
wiki	2,394,385	5,021,410	64MB	2.478
SyntheticGraph_one	3,200,000	42,011,049	4.2GB	1.95
SyntheticGraph_two	3,200,000	15,962,962	1.5GB	2.1
SyntheticGraph_three	3,200,000	7,061,587	760MB	2.4



(a) Machines with different amount of computing threads from Amazon EC2 computing optimized category.

(b) Machines with same amount of computing threads from three different Amazon EC2 categories.

Fig. 8: Comparison of CCR acquired from real world graphs and synthetic graphs.

Pagerank: The Pagerank algorithm [21] is a method to measure the importance of web pages based on their link connected. Its main use is to compute a ranking for every website in the world. This algorithm is defined as:

$$PR(u) = \frac{1-d}{N} + d \sum_{v \in B_u} \frac{PR(v)}{L(v)}. \quad (8)$$

Here, d is the dumping factor and N is the total number of pages. B_u is the set of pages. $L(v)$ represents the number of outbound links on page v.

Coloring: The Coloring application is a special case of graph labeling. It attempts to color the vertices with different colors such that no two connected vertices share the same color. In PowerGraph, this application is implemented to color directed graphs, and count the total number of colors in use.

Connected Component (CC): The Connected Component algorithm is designed to count fully connected subgraphs in which any two vertices are connected by a path. The algorithm counts connected components in a given graph, as well as the number of vertices and edges in each connected component.

Triangle Count (TC): Each graph triangle is a complete subgraph formed by three vertices. The Triangle Count application counts the total number of triangles in a given graph, as well as the number of triangles for each vertex. The number of triangles of a vertex indicates the graph connectivity around that vertex. The application implemented in PowerGraph maintains a list of neighbors for each vertex in a hash set. It counts the number of intersections of vertex u's and vertex v's neighbor sets for every edge (u,v).

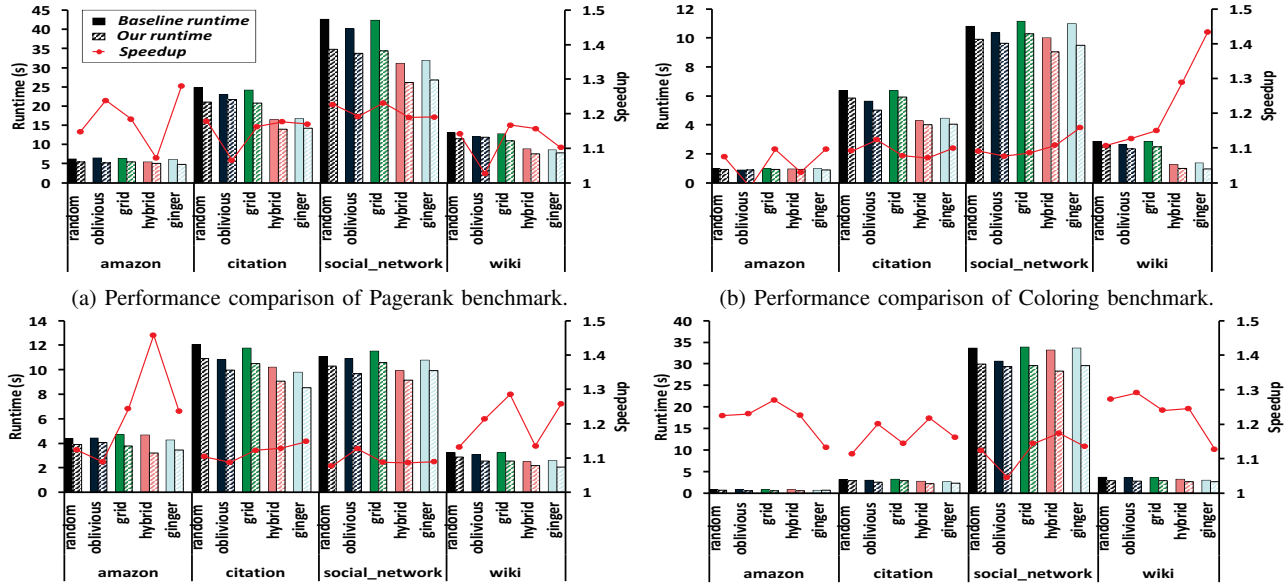
V. Evaluation

In the following section, we illustrate the accuracy of using synthetic *power-law* graphs to measure machine's computational capabilities in a heterogeneous cluster. We further demonstrate the benefits brought by our profiling-aided methodology for heterogeneous clusters in terms of performance and energy. We show the comparison on Amazon EC2 virtual nodes and local servers, the latter including energy data. Furthermore, we deploy local servers to form a more complicated projected case by manipulating processor frequency ranges. The baseline we chose to compare against is the performance and energy brought by state-of-the-art methodology proposed in [5], which uses the number of cores in a machine to estimate the performance differences in a heterogeneous cluster. Lastly, we discuss another benefit of profiling the synthetic *power-law* proxy graphs on the commercial cloud computing platform, which is indeed useful to cloud service users for graph related work.

A. Profiling Accuracy

In order to partition a graph strictly following the machine's processing capability, CCRs captured by our profiling methodology using synthetic graphs need to be very precise. They should not only reflect the performance differences among all types of machines in the cluster, but also distinguish the performance variances brought by diverse graph applications. To verify the accuracy, we examine our proposed methodology with synthetic graphs in different situations. First, as shown in Figure 8a, we compare performance of four MLDM graph applications on four *c4* machines from the computation-optimized domain on Amazon EC2. As can be seen, Coloring and Connected Component have nearly linear performance improvements from the "smallest" *xlarge* machine to the "biggest" *8xlarge* machine. Differently, Pagerank has a unique saturation point between machine *4xlarge* and *8xlarge*, which is accurately captured by our synthetic graphs. Uniquely, Triangle Count has a very sharp speedup increase when going from *4xlarge* to *8xlarge*. As we mentioned before, graph applications are very diverse. Using hardware configuration data alone, it is almost impossible to capture all the differences among machines and applications. For Pagerank and Coloring application, the speedup estimated using synthetic graphs closely matches the speedup obtained when running with real graphs. The only mismatching is the *8xlarge* case of the Triangle Count application, where real graphs show a 7.6x performance improvement over baseline whereas synthetic graphs only estimates a speedup of 5.3x. Overall, our synthetic graphs and profiling methodology achieve 92% accuracy for different types of machine from the same domain. By contrast, estimating performance using the number of cores leads to 108% error on average.

As cloud applications are very diverse in terms of performance and energy consumption, cloud service providers usually offer a number of machines from different categories that are aimed at satisfying users' needs while maintaining low energy bill. The Amazon EC2 platform offers general purpose computing units, as well as machines optimized for



(a) Performance comparison of Pagerank benchmark.

(b) Performance comparison of Coloring benchmark.

(c) Performance comparison of Connected Component benchmark.

(d) Performance comparison of Triangle Count benchmark.

Fig. 9: Performance comparison between prior work and CCR-guided partitioning on Amazon virtual platform.

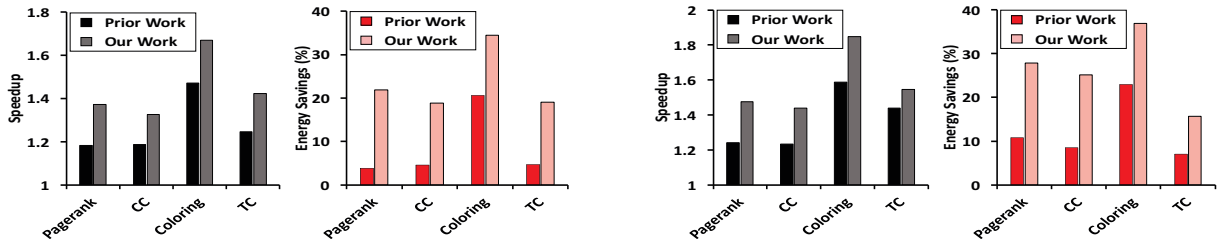
computation or memory. We evaluate our work across three domains to explore this heterogeneity trend. The machines *m4.2xlarge*, *c4.2xlarge* and *r3.2xlarge* have the same hardware configuration in terms of computing threads and network speed (if the graph does not exceed the memory capacity, different memory sizes will not affect the graph execution). However, as Figure 8b shows, these three machines actually have diverging behaviors. Even though the differences among these three machines are relatively small, our synthetic graphs can still precisely capture them. Compared to a baseline machine *m4.2xlarge*, *c4.2xlarge* speedups are about 1.2x on average whereas *r3.2xlarge* achieve an average of 1.1x speedup. As shown in Figure 8b, the CCRs obtained using synthetic graphs almost perfectly match real world graphs across all applications, with an average of 96% accuracy.

B. Heterogeneous Cluster

This section demonstrates the performance and energy improvements brought by our CCR-guided graph partitioning. We examine the benefits under three different cases. First, we show execution time improvement in a heterogeneous cluster formed by machines with the same number of computing threads, which would be considered as homogeneous by prior work [5]. Secondly, to replicate the experiments in prior work [5] while measuring both runtime and energy consumption, we perform the comparison on a local cluster composed of machines with different numbers of threads running in the same range of frequency. Lastly, to project the future situation in heterogeneous data centers, we configured our cluster to be made up of machines with different number of cores operating at different frequency ranges.

1) Case 1: We first perform a comparison on a small, fixed heterogeneous cluster formed by *m4.2xlarge* and *c4.2xlarge*. We use four representative natural graphs accompanied by four MLDM applications. The bars and left axis of Figure 9 demonstrates the application runtimes of different applications for different graphs and partitioning algorithms. The Pagerank algorithm illustrated in Figure 9a shows an average of 1.17x speedup across all graphs and partitioning algorithms. Oblivious has the minimal runtime improvement for most graphs in Pagerank, as it is a greedy algorithm that is designed to achieve an even load balance. Coloring has the least performance improvement among all four applications. It is only 1.12x faster than the baseline. This minimal speedup is limited by two factors: the 8% estimation error on the *c4.2xlarge* and the asynchronous execution manner of the Coloring application. The Connected Component algorithm has the maximum speedup of 1.45x using the hybrid algorithm on the amazon graph, while its average runtime reduction is around 14%. A large application diversity can be seen in Connected Component, as the citation graph behaves very differently compared to its performance in other applications. Triangle Count shows the most runtime benefits from CCR-guided graph partitioning. It outperforms the baseline by 1.19x on average. Overall, among all applications and graphs, hybrid and ginger partitioning algorithms achieve better performance than the other three algorithms.

2) Case 2: Similar to experiments performed in prior work [5], we form a heterogeneous cluster by deploying machines with different numbers of cores. A small cluster contains two machines, one having 4 computing threads and the other 12. The application CCRs for this cluster are generally around $1:3.5$, which means that the fast machine will be overloaded if we partition graphs based on the number of cores. Compared to the default PowerGraph setup, Figure 10a shows that the prior work achieves a 1.27x speedup across all applications, which is quite similar to the results provided in [5]. However,



(a) Local cluster with same frequency range.

(b) Local cluster with different frequency ranges.

Fig. 10: Performance and energy improvements on local clusters.

the energy savings are insignificant, as the powerful machine is overloaded to run longer that it has to. Therefore, energy savings are limited. Our proposed methodology speeds up the default system by as much as 1.67x and 1.45x on average, which is 17.7% better than prior work. Moreover, since we load the right amount of data on the fast machine, we achieve an average of 23.6% energy savings (compared to 8.4% using the approach from prior work).

3) Case 3: Since more "tiny" ARM-like servers are being deployed in modern data centers, we construct a cluster with machines operating at two frequency domains to emulate such future heterogeneous environments. The fast machine has 12 cores and runs at a maximum of 2.5Ghz, while the little machine only has 4 cores with a maximum frequency of 1.8Ghz. Not surprisingly, the applications' CCRs change substantially. The Pagerank, Connect Component, and Coloring CCRs all become more than 1 : 6. Different from the significant changes in other three applications, Triangle Count's CCR increases from 1 : 3.1 to 1 : 4.5, and it becomes quite similar to the partition ratio suggested by the number of hardware threads. Therefore, we can observe in Figure 10b that both the runtime improvement and energy reduction of this application is similar to what prior work achieves. As the heterogeneity of the cluster increases, our scheme achieves better speedup and energy savings as compared to Case 2. On average, the graph executions achieve 1.58x speedup and 26.4% energy savings over the default system. This presents an average of 20.2% speedup and 14.1% energy reduction over the prior work.

C. Cost Efficiency Projection

For users of cloud computing services, cost is a primary consideration. Other than the performance and energy improvements achieved in a heterogeneous cluster, profiling the synthetic graphs can also offer an accurate overview of the cost efficiency of different machines. Figure 11 plots the Pareto space of each individual machine's performance and cost on four applications. All cost and speedup information is generated by profiling synthetic graphs, and the accuracy of using synthetic graphs has been discussed in Section V-A.

There are many metrics that can be used to evaluate cost efficiency, such as total cost of ownership (TCO) and cost per throughput/performance. Similarly, we deploy the cost per task to define a machine's efficiency. The cost per task is defined as the product of task runtimes and a machine's hourly rate (as shown in Table I). As we can see, machines of similar type are

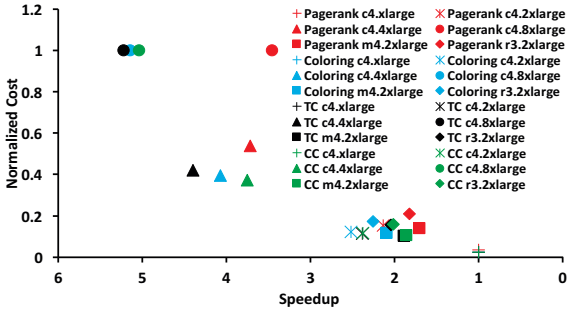


Fig. 11: Cost and performance pareto graph of different computing nodes and different graph applications.

clustered in the Figure 11. All *2xlarge* machines (from three different domains) are grouped together with around 2x around speedup and 0.2x cost, which means none of them demonstrate their "advertised" specialty for graph applications. Within the computation-optimized domain, we can see *8xlarge* being the most expensive machine for graph workloads, which is a result of the high charge rate and relatively low performance. The *4xlarge* and *2xlarge* saves 60% and 80% cost and provides 4x and 2x speedup, which should be considered as reasonable candidates for graph applications to satisfy both aspects. Without profiling using synthetic graphs, users would have no insights about the machines provided by cloud services or the machines they may have already deployed.

VI. Related Work

Other than the PowerGraph [8] framework we used, distributed graphlab [22], PGX.D [23], Pregel [24] and Giraph [25] also target graph applications in distributed systems. Different from these, Graphchi [26], Graphlab [27], GPSA [28] target improvements in graph processing performance on a single node. Guo et al. [29] and Han et al. [30] performed comprehensive studies on the strength and weakness of these graph processing frameworks.

Besides the studies on graph platforms, a few papers attempt to address the data center heterogeneity for graph workloads. Semih [31] used dynamic load balancing technology in their Graph Processing System (GPS) to alleviate the negative effect of node-level heterogeneity. Similarly, Mizan (Pregel-like system) [13] was designed to reduce the performance degradation in a heterogeneous environment by runtime monitoring and vertex migrating. LeBeane et al. [5] optimized the existing graph framework by ingressing the data in a

heterogeneous way. However, their inaccurate estimation of a machine's graph processing capability leads to an imbalanced situation and results in suboptimal performance improvements. No prior work has ever used synthetic graphs for profiling in a heterogeneous environment to guide graph ingress. Other than the graph processing frameworks, Hadoop [7] framework has also discovered the influence of data center heterogeneity and attempted to exploit it. Most works were implemented on the MapReduce [32] programming model. LATE [11] is one of the earliest works to alleviate the performance effects of "slow" stragglers. Tarazu [9] is another work that improves MapReduce performance for heterogeneous hardware by using communication aware load balancing and task scheduling.

VII. Conclusion

Graph processing applications are emerging as an extremely important class of workloads during the era of big data. As the heterogeneity of modern data centers continue to increase due to the requirements of low energy consumption, diverse service types, and high service availability, understanding the computing capability of heterogeneous nodes becomes essential to maximize the performance and minimize the energy consumption. We illustrate that profiling synthetic proxy graphs on a heterogeneous cluster can estimate its machines' computing capabilities with an average of 92% accuracy. With our proposed methodology, the proxy-guided heterogeneity-aware graph processing system achieves a maximum speedup of 1.84x and 1.45x over a default system and prior work [5], respectively. Compared to prior work, we improve the default system's performance by an average of 17.9% with 14.6% less energy consumption on average.

VIII. Acknowledgments

This work was partially supported by Semiconductor Research Corporation Task ID 2504, and National Science Foundation grant CCF-1337393. The authors would also like to thank Amazon for their donation of the EC2 computing resources used in this work. Any opinions, findings, conclusions, or recommendations are those of the authors and do not necessarily reflect the views of these funding agencies.

References

[1] C. Baru, M. Bhandarkar, R. Nambiar, *et al.*, "Setting the direction for big data benchmark standards," in *Selected Topics in Performance Evaluation and Benchmarking*, pp. 197–208, Springer, 2013.

[2] K. Ammar and M. T. Özsu, "Wgb: Towards a universal graph benchmark," in *Advancing Big Data Benchmarks*, pp. 58–72, Springer, 2014.

[3] "Amazon EC2." http://aws.amazon.com/ec2. Accessed: 04-16-2015.

[4] "Microsoft azure." https://azure.microsoft.com. Accessed: 02-01-2010.

[5] M. LeBeane, S. Song, R. Panda, *et al.*, "Data partitioning strategies for graph workloads on heterogeneous clusters," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 56:1–56:12, ACM, 2015.

[6] "Paypal deploys arm servers in data centers." http://www.datacenterknowledge.com/. Accessed: 04-29-2015.

[7] "Apache hadoop." https://hadoop.apache.org/. Accessed: 08-11-2015.

[8] J. E. Gonzalez, Y. Low, H. Gu, *et al.*, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 17–30, USENIX Association, 2012.

[9] F. Ahmad, S. T. Chakradhar, A. Raghunathan, *et al.*, "Tarazu: Optimizing mapreduce on heterogeneous clusters," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 61–74, ACM, 2012.

[10] Z. Fadika, E. Dede, J. Hartog, *et al.*, "Marla: Mapreduce for heterogeneous clusters," in *International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 49–56, IEEE, 2012.

[11] M. Zaharia, A. Konwinski, A. D. Joseph, *et al.*, "Improving mapreduce performance in heterogeneous environments," in *Conference on Operating Systems Design and Implementation (OSDI)*, pp. 29–42, USENIX Association, 2008.

[12] J. Xie, S. Yin, X. Ruan, *et al.*, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in *International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pp. 1–9, IEEE, 2010.

[13] Z. Khayyat, K. Awara, A. Alonazi, *et al.*, "Mizan: A system for dynamic load balancing in large-scale graph processing," in *European Conference on Computer Systems (EuroSys)*, pp. 169–182, ACM, 2013.

[14] S. Sanyal, A. Jain, S. Das, and R. Biswas, "A hierarchical and distributed approach for mapping large applications to heterogeneous grids using genetic algorithms," in *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*, pp. 496–499, Dec 2003.

[15] R. Chen, J. Shi, Y. Chen, and H. Chen, "Powerlyra: Differentiated graph computation and partitioning on skewed graphs," in *EuroSys*, Apr. 2015.

[16] C. Tsourakakis, C. Gkantsidis, B. Radunovic, *et al.*, "Fennel: Streaming graph partitioning for massive scale graphs," in *International conference on Web search and data mining*, pp. 333–342, ACM, 2014.

[17] U. Brandes and T. Erlebach, *Network Analysis*. Theoretical Computer Science and General Issues, Springer-Verlag Berlin Heidelberg, 2005.

[18] D. Chakrabarti and C. Faloutsos, "Graph mining: Laws, generators, and algorithms," *ACM Computing Surveys (CSUR)*, vol. 38, no. 1, p. 2, 2006.

[19] J. Yan, G. Tan, and N. Sun, "Study on partitioning real-world directed graphs of skewed degree distribution," in *International Conference on Parallel Processing (ICPP)*, pp. 699–708, IEEE, 2015.

[20] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." http://snap.stanford.edu/data. Accessed: 04-16-2015.

[21] L. Page, S. Brin, R. Motwani, *et al.*, "The pagerank citation ranking: Bringing order to the web.," Technical Report 1999-66, Stanford InfoLab, 1999.

[22] Y. Low, D. Bickson, J. Gonzalez, *et al.*, "Distributed graphlab: A framework for machine learning and data mining in the cloud," *Proc. VLDB Endow.*, vol. 5, pp. 716–727, Apr. 2012.

[23] S. Hong, S. Depner, T. Manhardt, *et al.*, "Pgx.d: A fast distributed graph processing engine," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 58:1–58:12, ACM, 2015.

[24] G. Malewicz, M. H. Austern, A. J. Bik, *et al.*, "Pregel: A system for large-scale graph processing," in *International Conference on Management of Data (SIGMOD)*, pp. 135–146, ACM, 2010.

[25] C. Avery, "Giraph: Large-scale graph processing infrastructure on hadoop," *Proceedings of the Hadoop Summit. Santa Clara*, 2011.

[26] A. Kyrola, G. Blelloch, and C. Guestrin, "Graphchi: Large-scale graph computation on just a pc," in *Conference on Operating Systems Design and Implementation (OSDI)*, pp. 31–46, USENIX Association, 2012.

[27] Y. Low, J. E. Gonzalez, A. Kyrola, *et al.*, "Graphlab: A new framework for parallel machine learning," *UAI*, pp. 340–349, 2010.

[28] J. Sun, D. Zhou, H. Chen, *et al.*, "Gpsa: A graph processing system with actors," in *International Conference on Parallel Processing (ICPP)*, IEEE, 2015.

[29] Y. Guo, M. Biczak, A. L. Varbanescu, *et al.*, "How well do graph-processing platforms perform? an empirical performance evaluation and analysis," in *International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 395–404, IEEE, 2014.

[30] M. Han, K. Daudjee, K. Ammar, *et al.*, "An experimental comparison of pregel-like graph processing systems," *Proc. VLDB Endow.*, vol. 7, pp. 1047–1058, Aug. 2014.

[31] S. Salihoglu and J. Widom, "Gps: A graph processing system," in *International Conference on Scientific and Statistical Database Management (SSDBM)*, pp. 22:1–22:12, ACM, 2013.

[32] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.