Mizan: Optimizing Graph Mining in Large Parallel Systems

Zuhair Khayyat[1] Karim Awara[1] Hani Jamjoom[2] Panos Kalnis[1]

[1] King Abdullah University of Science and Technology, Saudi Arabia
[2] IBM Watson Research, NY

ABSTRACT

Extracting information from graphs, from finding shortest paths to complex graph mining, is essential for many applications. Due to the shear size of modern graphs (e.g., social networks), processing must be done on large parallel computing infrastructures (e.g., the cloud). Earlier approaches relied on the MapReduce framework, which was proved inadequate for graph algorithms. More recently, the message passing model (e.g., Pregel) has emerged. Although the Pregel model has many advantages, it is agnostic to the graph properties and the architecture of the underlying computing infrastructure, leading to suboptimal performance.

In this paper, we propose *Mizan*, a layer between the users' code and the computing infrastructure. Mizan considers the structure of the input graph and the architecture of the infrastructure in order to: (i) decide whether it is beneficial to generate a near-optimal partitioning of the graph in a preprocessing step, and (ii) choose between typical point-to-point message passing and a novel approach that puts computing nodes in a virtual overlay ring. We deployed Mizan on a small local Linux cluster, on the cloud (256 virtual machines in Amazon EC2), and on an IBM Blue Gene/P supercomputer (1024 CPUs). We show that Mizan executes common algorithms on very large graphs 1-2 orders of magnitude faster than MapReduce-based implementations and up to one order of magnitude faster than implementations relying on Pregel-like hash-based graph partitioning.

1. INTRODUCTION

Graphs model complex relationships among objects in a variety of applications, such as social networks (such as Facebook and Twitter), the Internet web, biology (e.g., protein-protein interaction), and public health (e.g., spread of epidemics). Many of these applications require very large graphs, containing hundreds of millions to billions of edges. Because of their shear size, processing such graphs is very expensive and must be performed on large parallel computing infrastructures. This applies to complex graph mining operators (e.g., finding frequent subgraphs) as well as to simpler operators (e.g., calculating the PageRank or finding single-source shortest paths).

A variety of parallel systems are used for processing graphs, ranging from large proprietary infrastructures (e.g., Yahoo! M45 supercomputer) to general purpose Linux clusters, where cloud computing (e.g., Amazon's EC2) has recently emerged as a cost-effective alternative. Unfortunately, it is extremely difficult to implement efficient and scalable parallel graph algorithms, especially when scheduling issues and failures are taken into account.

To alleviate this problem, several frameworks have been developed. HADI [10], PEGASUS [11] and X-RIME [24] utilize the MapReduce paradigm to implement common graph mining operators. MapReduce is easy to program and takes care of scheduling and failures. However, MapReduce is not efficient for graph algorithms because it is a functional programming model that requires transferring the status of the entire graph between iterations. Recently, the Pregel [16] programming paradigm has been proposed to minimize this problem. Pregel is based on message passing: at each iteration, every graph node computes its status independently and either sends messages to other nodes or becomes inactive. These messages are received in the next iteration. The process continues until all nodes are inactive. Graph algorithms can be implemented more efficiently in Pregel than MapReduce because, between iterations, Pregel transfers only the necessary messages for status updates instead of the entire graph status.

Despite its advantages, Pregel has two limitations. First, it is agnostic to the properties of the input graph. Consider a power-law graph and the PageRank operator. By default, Pregel will use a hash function to partition the graph nodes among *processing elements* (PEs[1]). Because the graph is power-law, some PEs will be assigned high-degree nodes and will receive exponentially more messages than the others, leading to workload imbalance. Even if Pregel attempts to reshuffle the data, with high probability, many high-degree nodes will reside in different PEs than their neighbors. This will cause significant network communication overhead due to the exchange of messages. Second, Pregel is agnostic to the architecture of the underlying computing infrastructure. Consider again the PageRank operator, but with a more uniform graph. Each node will send the same message to many others, residing across different PEs. Because of the topology of the communication network, point-to-point communication between several pairs of PEs may be routed through the same physical link. Consequently, the same message will be transmitted several times, overloading the physical link.

Motivated by these limitations, we developed *Mizan*. As shown in Figure 1, Mizan is a layer between the implementation of the graph processing algorithm and the physical com-

[1]We use the term *processing element* (PE) for the computing nodes of the physical infrastructure to avoid confusion with the graph nodes.

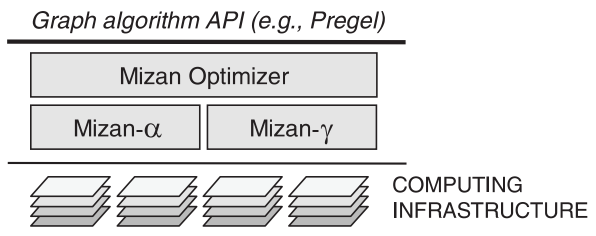


Figure 1: Mizan overview

puting infrastructure. Mizan minimizes the execution time of the graph algorithm by (*i*) appropriate partitioning of graph nodes over PEs, and (*ii*) appropriate implementation of the message passing mechanism.

The first component of Mizan is the optimizer. It gathers statistics from the input and estimates whether the graph is close to power-law. There are two cases. The first case is when the graph resembles the power-law distribution. Mizan-α is invoked. Mizan-α runs a minimum-cut algorithm to partition the graph in as many partitions as the available PEs. Processing then continues in a Pregel-like fashion, with point-to-point message passing. Intuitively, by minimizing the edges that cross different PEs, most message passing is done locally within each PE using fast in-memory operations, whereas the physical network is accessed only for edges between different PEs. Although the minimum-cut algorithm is not new, other systems have avoided it because of its high computational cost. However, we show that if the optimizer's decision is correct, the execution of the user's code is much faster. Therefore, the total time of finding the minimum-cut and executing the user's code is almost twice as fast in the worst case, and up to 9 times faster, when compared to hash-based partitioning.

The second case is when the graph distribution is far from power-law. The overhead of the minimum-cut partitioning cannot be justified by the potential gains. In this case, the optimizer invokes Mizan-γ, which works with any random graph partitioning. Mizan-γ constructs a virtual overlay ring between the PEs. Every message is transmitted only once, must go around the entire ring, and can be used by any PE. Intuitively, Mizan-γ minimizes the replicas of a message transmitted through a physical link; the trade-off is increased latency. We show that, if the optimizer's decision is correct, Mizan-γ is up to one order of magnitude faster compared to point-to-point message passing.

Although Mizan resembles traditional query optimization in relational databases, there is an important difference. A typical DBMS contains several implementations of an operator (e.g., nested-loop and hash-based join); the user requests the specific operator (e.g., join) and the optimizer chooses one of the available implementations. In our case, the user must provide the code for the graph processing algorithm, whereas Mizan decides the distribution of the data and message passing mechanism.

Mizan supports the Bulk Synchronous Parallel model [23]. It is, thus, as versatile as Pregel, but Pregel is not a prerequisite. Mizan exposes its own API, which can be accessed directly and provides to the user finer control over the inter- and intra-PE operations. Mizan supports a variety of parallel computing infrastructures. We have deployed it successfully on a small in-house Linux cluster, on Amazon EC2 with hundreds of virtual machines, and on a large IBM Blue Gene/P supercomputer with thousands of computing nodes.

Our contributions are:

- We propose Mizan, a framework that supports the parallel implementation of graph processing algorithms, such as shortest-paths, PageRank, diameter estimation, and complex graph mining.
- We implement: (*i*) Mizan-α, a point-to-point message passing mechanism with optimized partitioning, suitable for power-law-like graphs; (*ii*) Mizan-γ, an overlay ring message passing mechanism, optimized for non-power-law graphs; and (*iii*) an optimizer to decide between Mizan-α and Mizan-γ.
- We deploy Mizan on a local Linux cluster (16 machines), on a cloud environment (Amazon EC2, 256 virtual machines) and on a supercomputer (IBM Blue Gene/P, 1024 CPUs).
- We experiment with large real and synthetic graphs. We demonstrate that Mizan is orders of magnitude faster than MapReduce based implementations, and up to 1 order of magnitude faster than implementations based on Pregel-style hash-based partitioning.

The rest of the paper is organized as follows. Section 2 presents the required background. Section 3 describes our system, Mizan, followed by case studies in Section 4. Section 5 contains the experimental evaluation. Section 6 presents the related work and Section 7 concludes the paper.

2. BACKGROUND

This section presents two common approaches for implementing large scale parallel graph algorithms: the MapReduce framework and the Message Passing model.

2.1 MapReduce for Graphs

The MapReduce [7] framework allows the implementation of large scale parallel algorithms and has been used extensively for graph algorithms. HADI [10], for instance, implements graph diameter estimation on top of Hadoop [1]. PEGASUS [11] also uses Hadoop to implement a distributed generalization of the block matrix vector multiplication algorithm (GIM-V). On top of GIM-V, PEGASUS implements a variety of graph mining algorithms, including PageRank and diameter estimation.

Figure 2 shows a typical trace of PageRank on MapReduce for a graph with 5 nodes. It is an iterative algorithm, which stops when the calculated PageRank values converge. Each PageRank iteration requires two MapReduce phases. Phase-1 reads the structure of the graph (i.e., $\langle source, destination\rangle$ pairs), together with the initial PageRank values of the nodes in the form $\langle nodeID, value\rangle$. The input is mapped to reducers using a simple hash function $H(nodeID) = \lfloor(nodeID - 1)/2\rfloor$. The output contains the initial $\langle nodeID, value\rangle$ pairs and the transfer of information. for instance $\langle 3, f(v_2)\rangle$ corresponds to edge $3 \rightarrow 2$. The output is written on the network file system (HDFS for Hadoop), and is read again by the mappers in phase-2. The second phase groups together the partial values for each node and the reducers compute the new PageRank values. The output is written on the HDFS and is used as input for the next iteration of PageRank.

The example demonstrates several drawbacks of using MapReduce for graph algorithms:

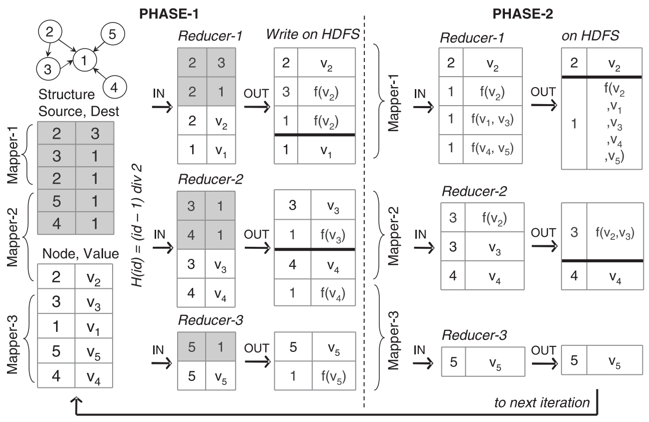


Figure 2: PageRank on MapReduce with 3 mappers and 3 reducers. Each PageRank iteration requires two MapReduce phases.

Disk intensive. The input of each MapReduce job must be read from the disk and the output must be written on disk. This is inefficient especially if deployed in the cloud, where many virtual machines share a disk, either locally or through a shared network channel.

Iterative computation model. The previous problem is amplified by the fact that many graph algorithms require many iterations and each iteration can be translated to multiple MapReduce phases. Moreover, the status of the entire graph, together with the graph structure must be transferred between iterations.

Poor data locality. Data is scattered and recombined frequently. For example, at the end of phase-1 node 1 is scattered across three machines, but in phase-2, it is recombined to one reducer. This is a recurring cost since it is repeated in every iteration of the graph algorithm.

Graph agnostic. Typically, graph algorithms propagate information along the graph edges. Therefore, high-degree nodes tend to send or receive most of the information. MapReduce does not take this into account, leading to workload imbalance, especially for graphs resembling power-law. For example, reducer-1 in phase-2 is assigned much more work compared to reducer-3, causing the entire system to delay.

2.2 Message Passing Model: Pregel

Pregel [16] is a framework designed specifically for implementing graph algorithms on large parallel computing infrastructures. Pregel is inspired by the Bulk Synchronous Parallel model [23], where algorithms are executed in multiple stages separated by synchronization barriers and information is propagated by sending messages between stages. In Pregel, each graph node corresponds to an object that can receive messages, execute a local algorithm and send messages. Initially, all nodes are active. At iteration k, each node u_i receives the messages that have been sent to it at the previous iteration $k-1$, performs local computations and may send messages to other nodes; these messages will reach their destination at iteration $k+1$. u_i may then change its status to inactive. The process is repeated for more iterations, until all nodes become inactive.

Pregel solves some of MapReduce's drawbacks. First, it is less disk-intensive because the graph is loaded in the main memory of the PEs and only the final result is written back to the disk.[2] Second, it replaces the iterative computation model with the message passing model. Therefore, instead of transferring the entire status of the graph, only updates are propagated between iterations. Third, data locality is improved because all information of a specific node is kept in the same PE; moving information is performed only if necessary for load balancing.

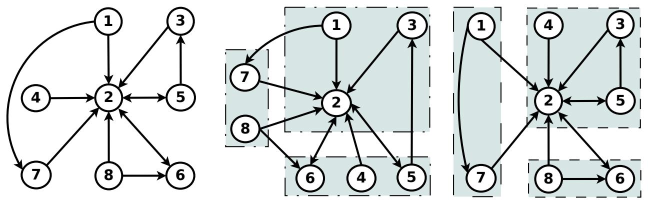


Figure 3: Different graph partitionings: (a) Input graph; (b) Hash-based, 8 edges cross PEs; (c) Mizan-α, only 4 edges cross PEs.

Similar to MapReduce, Pregel is graph agnostic. By default, it uses a hash function that is based on node IDs to distribute graph nodes to PEs, ignoring the structure of the graph. This significantly affects the cost of the message passing model. In the following section, we will discuss how our system, Mizan, utilizes the structure of the graph to decide the data placement and the message passing mechanism.

3. MIZAN

Mizan is similar to Pregel in the sense that it uses message passing and implements an instance of the Bulk Synchronous Parallel model. Mizan exposes an API with four functions: (i) `initialize` assigns initial values to the nodes and edges; (ii) `compute` executes local computation for each graph node; (iii) `sendMessage` is used by a node u_i to send a message to a node u_j; (iv) `testTerminate` is the user-defined condition for the termination of the graph algorithm. Since Mizan follows a Bulk Synchronous Parallel model, there are synchronization barriers between iterations; therefore a message sent at iteration k is received at iteration $k+1$.

Mizan can be used autonomously, as an alternative to Pregel. However, Mizan complements the message passing model by adding awareness of the graph structure, and the architecture of the underlying message passing mechanism. Therefore, Mizan corresponds to a layer between the graph algorithm (which may be expressed in Pregel) and the computing infrastructure (see Figure 1). This section describes Mizan's three main components: (i) Mizan-α, which distributes graph nodes to PEs in a near-optimal way, follows point-to-point message passing, and is suitable for power-law-like graphs; (ii) Mizan-γ, which supports random assignment of nodes to PEs, implements a novel message-passing approach using a virtual overlay ring, and is suitable for non-power-law graphs; (iii) the optimizer that decides between Mizan-α and Mizan-γ guided by the graph properties and knowledge of the message passing mechanism.

3.1 Mizan Alpha

Pregel-like systems assign graph nodes to PEs using a hash function $H(nodeID)$. Typical graph algorithms propa-

[2]Checkpoints can also be written on disk for failure recovery.

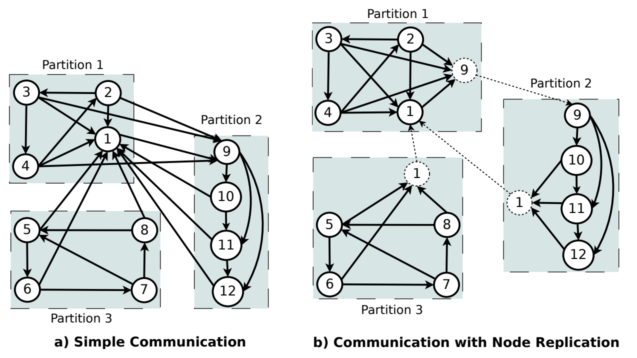


Figure 4: Node replication reduces cost. (a) Original partitioning. (b) Nodes u_1 and u_9 are replicated.

gate information along graph edges. Assuming roughly uniform distribution of node degrees, hash-based partitioning results in balanced workload among PEs. However, this assumption does not hold for many real graphs. Social networks, for instance, are usually power-law graphs. In such cases, hash-based partitioning results in significant communication because of many edges crossing PEs. A number of studies [3, 9, 22] have shown that communication cost is the dominant factor affecting scalability in cloud environments. This explains the performance deterioration for power-law graphs with hash-based partitioning.

3.1.1 Minimum-cut Partitioning

Motivated by the problem mentioned above, Mizan-α employs an m-way minimum-cut algorithm to partition the graph in m subgraphs, where m is the number of PEs, such that the number of edges that cross partitions is minimized. An example is shown in Figure 3, where assuming 3 PEs, hash-based partitioning ($H(nodeID) = \lfloor (nodeID - 1)/3 \rfloor$) results in 8 edges crossing PEs. Mizan-α, on the other hand, generates better partitioning with only 4 edges crossing PEs. Edges that are inside a single PE do not introduce significant message passing overhead, because messages are delivered with simple in-memory operations. Only those edges that cross PEs need to access the communication network. Therefore, by minimizing such edges, Mizan-α achieves significant performance improvement.

Computing the optimal minimum-cut is NP-hard, so we employ METIS [13], a heuristic multi-level graph partitioning algorithm. Intuitively, METIS groups strongly connected nodes into a single subgraph. The algorithm works in one or more stages. Each stage reduces the size of the graph by collapsing vertices and edges, partitions the smaller graph, then maps back and refines this partition of the original graph. In our system, we use the parallel version of the algorithm, called ParMETIS [12].

3.1.2 Node Replication

Although the minimum-cut partitioning minimizes the number of edges that cross PEs, there still exist problematic hub nodes. Consider the partitioning in Figure 4(a), where nodes u_1 and u_9 are hubs. Irrespective of which partition these nodes belong to, there will be a large number of edges crossing PEs. Mizan-α solves this problem by replicating such nodes to multiple PEs. In Figure 4(b), node u_1 is replicated to partitions 2 and 3, whereas node u_9 is replicated to partition 1. Therefore, the number of edges that cross PEs

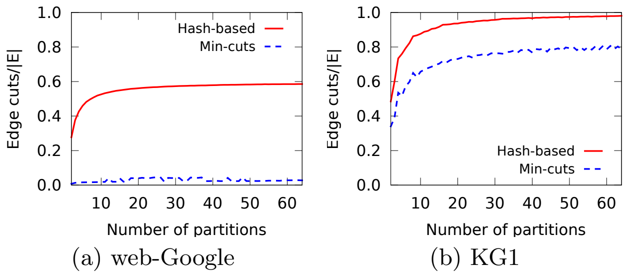


Figure 5: Minimum-cut versus hash-based partitioning. The gain is much higher for (a), which is a power-law graph, when compared to the random graph in (b).

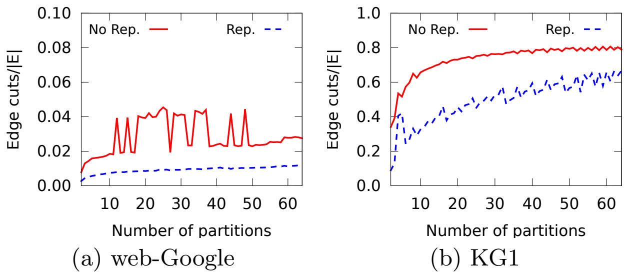


Figure 6: Node replication reduces significantly the number of edges that cross PEs, both for (a) power-law and (b) random graphs.

decreases from 10 to only 3.

If the user function (i.e., the implementation of the `compute` API function) is distributive (e.g., min, max, sum, PageRank, graph diameter, etc) or algebraic (e.g., average), partial computation can be done in the replicas. This maximizes the advantage of replication. On the other hand, if the function is holistic (e.g., median) the replica acts only as communication aggregator that packs all related messages to one super-message. The user sets a flag to indicate if partial computation is permitted.

Intuitively, node replication is similar to Pregel's combiners. However, for the case of distributive or algebraic functions, Pregel's combiners require the user to know the actual placement of nodes to PEs and perform the replication manually. Mizan-α automates this in a transparent way.

3.1.3 Discussion

The trade-off in using the minimum-cut algorithm is its high computational cost. The benefit depends on the properties of the graph. In Figure 5, we show the number of edges that cross PEs over the total number of edges for a power-law (web-Google) and a random (KG1) graph.[3] For web-Google, minimum-cut reduces the number of edges that cross PEs by roughly 95% compared to hash-based partitioning. For KG1, on the other hand, the gain is about 20%. In Section 5, we will show that if the graph resembles the power-law distribution, then the total cost of computing the minimum-cut and executing the user's code is lower than using hash-based partitioning

Node replication, on the other hand, is always beneficial. The computational overhead of replication is very low. At the same time, as shown in Figure 6, the benefit is up to 50%

[3]Refer to Section 5 for details of the experimental setup.

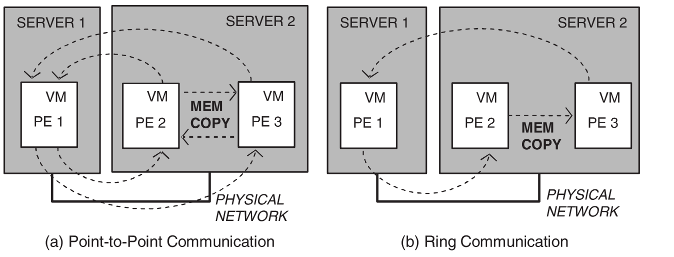


Figure 7: Two physical servers with three virtual machines in EC2. (a) Point-to-point. (b) Ring ($PE_1 \rightarrow PE_2 \rightarrow PE_3 \rightarrow PE_1$) communication.

for web-Google and 29% for KG1. The experiment assumes that the minimum-cut algorithm has already been applied. The "saw-teeth" shape in Figure 6(a) is due to the randomization step of the METIS heuristic, which results in variability when the number of partitions changes. Interestingly, node replication smoothes the variability.

3.2 Mizan Gamma

Mizan-α significantly decreases the execution time for power-law graphs. However, if the graph is not power-law, Mizan-α is slower than Pregel-like systems with hash-based partitioning, because of the cost of the minimum-cut algorithm. Motivated by this, we developed Mizan-γ, a system that supports any random partitioning of the data, thus eliminating the cost of pre-processing.

3.2.1 Overlay Ring for Message Passing

Observe that, if random partitioning is coupled with a typical point-to-point message passing mechanism, the communication cost will overwhelm the system [3,9,22]. To address this problem, Mizan-γ implements a novel message passing approach, based on an overlay ring. Assume there are m processing elements, $PE_1 \dots PE_m$. Mizan-γ arranges them into a virtual overlay ring $PE_1 \rightarrow PE_2 \rightarrow \dots \rightarrow PE_{m-1} \rightarrow PE_m \rightarrow PE_1$, such that each PE receives messages only from its predecessor and sends messages only to its successor. Each message M is inserted in the ring and travels a full round along the ring. M does not have a specific PE destination. Instead, M pass through all PEs and it is used by those PEs that include graph nodes that need M.

Consider the example of Figure 7, which depicts 3 PEs in the cloud, each corresponding to a virtual machine (VM). In cloud environments, it is common to host multiple VMs in the same physical server; in our example there are two physical servers. Assume PE_1 needs to send the same message M to PE_2 and PE_3, and consider the point-to-point mechanism in Figure 7(a). PE_1 sends two copies of M that travel through the same physical link and arrive at server 2. If all 3 PEs need to communicate with each other, then six messages will be sent (2 per PE). For server 1, its *outgoing* physical link will be shared across the two messages. For server 2, two of the messages ($PE_2 \rightarrow PE_3$ and $PE_3 \rightarrow PE_2$) will be passed through local memory copies (by the underlying hypervisor) and the other two messages ($PE_2 \rightarrow PE_1$ and $PE_3 \rightarrow PE_1$) will be sent through the physical link. Now consider the ring topology $PE_1 \rightarrow PE_2 \rightarrow PE_3 \rightarrow PE_1$ in Figure 7(b). PE_1 sends M to PE_2 through the physical link. PE_2 uses M locally and also forwards it to PE_3. This too will be through memory copies. Finally, $PE_3 \rightarrow PE_1$ will send a message on the outgoing link of server 2. Overall, a ring mechanism pipelines messages in a predictable way, while improving the sharing of physical links across PEs.

The overlay ring reduces the cost in three ways: (i) It reduces the number of messages sent over physical links. In the worst case, a poorly constructed ring sends messages proportional to the number of PEs on a physical machine, where as a point-to-point mechanism sends N-1 messages (where N is the number of PEs in the system); (ii) It requires only two communication threads (to predecessor and successor) per PE. Traditional point-to-point approaches require multiple threads to handle communication by multiple PEs, which results in high CPU cost, especially in large scale deployments with a lot of PEs; (iii) Since each message M travels around the entire ring, M does not need to specify the destination PE, enabling the dynamic redistribution of graph nodes to PEs without overhead of bookkeeping.

3.2.2 Discussion

The trade-off of Mizan-γ is latency because a message M may need to traverse the entire ring before reaching its destination. If many PEs need M, as is the case for non-power-law graphs, then the total cost saving is more than the cost introduced by the increased latency. If, however, few PEs need M (e.g., power-law graphs), then Mizan-γ is not appropriate because latency dominates the cost.

The available hardware infrastructure also affects the performance of Mizan-γ. In cloud environments, Mizan-γ reduces the number of required messages on the underlying network fabric. In other architectures the potential benefit can be much higher. For example, if there is hardware support for efficient multicast or broadcast operations, Mizan-γ can use them to further reduce the communication cost. As another example, we deployed Mizan-γ on an IBM Blue Gene/P supercomputer, which has 3-dimensional torus network topology. A link in Mizan's virtual ring corresponds to an individual physical link in Blue Gene/P, therefore network congestion is minimized. Although this is an exotic architecture, mainstream systems nowadays have similar capabilities. For instance, in the market there are affordable network cards (e.g., Infiniband) that support remote direct memory access (RDMA) and can be used to implement efficiently the ring topology.

3.3 Mizan Optimizer

From the previous discussion it is clear that Mizan-α is more suitable for graphs that are roughly power-law, whereas Mizan-γ is more appropriate for other graph types. Therefore, it is important to distinguish whether the input graph is power-law.

A random sample x that follows the power-law has probability distribution $p(x) \propto x^{-\alpha}, \forall x \geq x_{min}$, where α and x_{min} are constants. To decide if an input graph follows power-law distribution, α and x_{min} are estimated for the edge degree distribution and tested against the theoretical behavior of a power-law distribution with similar α and x_{min} values. This problem is known as empirical data fitting, and generates a p-value to represent the fitting goodness of the empirical data on the theoretical distribution.

We developed an optimizer to select between Mizan-α and Mizan-γ by following the work of Clauset et al. [6] for power-law empirical data fitting. The optimizer works as follows. First, it reads a random sample of the graph nodes. Then

```
Node initialize(node graphNode)
  Assign Random Bitmask to node

Map compute(List inputEdge, Map curData)
  Map newData
  for (all edges in inputEdge)
    Apply bitwise-or and save result in newData
  return newData;

void sendMessage(List inputEdge,Map oldData)
  send bitmask to all neighbours

boolean testTerminate(Map curData, Map newData)
  if (curData != newData) then return false
  else return true;
```

Listing 1: Diameter Estimation in Mizan

it calculates the edge degree (i.e., $inDegree + outDegree$) distribution for the input sample and represents it as a set of probabilities. The probability for each distinct edge degree distribution is calculated by the number of its occurrences in the sample data over the total count of the sampled data: $|degree_i|/|x_{samples}|$. After that, x_{min} is estimated by solving the optimization problem of maximizing:

$$D_x = max_{x \geq x_{min}} |S(x) - P(x)|, \tag{1}$$

where D_x is the distance between the input data $S(x)$ and the theoretical power-law model $P(x)$. This is known as Kolmogorov-Smirnov statistic [21]. For each estimated x_{min}, $P(x)$ is generated by estimating the parameter α using maximum likelihood estimation [2] such that:

$$\alpha = 1 + \frac{n}{\sum_{i=1}^{n} \ln \frac{x_i}{x_{min}}}. \tag{2}$$

The final step is using the best estimated x_{min} to quantify the fitting goodness, represented as p-value, to the theoretical power-law distribution. This is done by generating a fairly large number of synthetic random power-law samples y_m that follow the estimated x_{min} and α, and measure their distances $D_{y_m} = |S(y_m) - P(y_m)|$. The p-value is calculated as the fraction of times that D_{y_i} is greater than D_x. If the resulted p-value is lower than 10% or unreasonably too high (above 90%),[4] the sampled graph is ruled out of having power-law distribution.

4. CASE STUDIES

Recall from Section 3 that Mizan's API exposes four functions: `initialize`, `compute`, `sendMessage` and `testTerminate`. Mizan is a generic framework and can support any graph algorithm that can be expressed in the message passing model, from simple shortest paths to complex graph mining. In the following, we will present two examples that demonstrate the use of Mizan's API to implement two popular graph algorithms: diameter estimation and PageRank.

Diameter Estimation. The algorithm finds the longest shortest path between any two nodes in a graph. We implemented the algorithm by Palmer et al. [20] that computes an estimation of the true diameter using bitmasks to implement probabilistic counting. Let $G = (N, E)$ be the graph, where N and E are the sets on nodes and edges, respectively. The algorithm starts by randomly generating k bitmasks for each node $u \in N$. At each iteration, each pair of nodes u and v connected by an edge $e = (u, v)$ exchange their bitmasks b and apply a bitwise-or operator, such that:

$$b_u^1 = b_u^0 (bitwiseOR) \{b_v^0 | e = (u, v) \in E\}.$$

The algorithm terminates if during iteration d no bitmask changes. The complexity is $\mathcal{O}(d(|E|+|N|))$ and the estimated diameter is d. The pseudocode is shown in Listing 1.

[4]Very high p-value can be indicative that the graph follows another distribution that exhibits similar characteristics to power-law distributions.

```
Node initialize(node graphNode)
  assign 1/totalNodes to node.value
  assing 1/totalNodeOutdegree to edge.weight

Map compute(List inputEdge, Map curData)
  Map newData
  for (all edges in inputEdge)
    Apply src.value += inputEdge.value and store 0
      0in newData
  return newData;

void sendMessage(List inputEdge,Map oldData)
  send oldData * inputEdge.weight to all 0
    0neighbours

boolean testTerminate(Map curData, Map newData)
  Apply random factor on newData
  if ((curData - newData) > error) return false
  else return true;
```

Listing 2: Pseudocode for PageRank in Mizan

PageRank. We implement PageRank [19] using the power iterative method. The algorithm uses matrix-vector multiplications, which calculate the eigenvalues of the graph's adjacency matrix at each iteration. Formally, each iteration calculates:

$$v^{(k+1)} = cA^t v^k + (1 - c)/|N|,$$

where v^k is the eigenvector of iteration k, c is the damping factor used for normalization, and A is a row normalized adjacency matrix. The algorithm terminates when the PageRank values of all nodes change by less that *error* during an iteration. The pseudocode is shown in Listing 2.

5. EXPERIMENTAL EVALUATION

We implemented a version of Mizan for Linux clusters and cloud environments; Mizan-α is implemented in Java and Mizan-γ in C++. We also ported Mizan-γ to supercomputer environments that support MPI. We compared Mizan against PEGASUS[5], which is implemented in Java on top of Hadoop. We also implemented a striped-down Pregel-like system that excludes dynamic load balancing and advanced features such as topology mutation.

We run our experiments on Amazon's EC2 Cloud using up to 256 large EC2 instances.[6] We also used an IBM Blue Gene/P supercomputer with 16,384 PowerPC-450 CPUs, each with 4 cores at 850MHz and 4GB RAM; we accessed up to 1,024 CPUs concurrently. We downloaded real datasets from the Stanford Network Analysis Project.[7] We also generated synthetic datasets using the Kronecker [14] generator that models the structure of real life networks. Our datasets contain both power-law and non-power-law graphs. The details are shown in Table 1.

[5]`http://www.cs.cmu.edu/~pegasus`
[6]Type: Large; CPU: 4CU, Memory: 7.5GB; Network: Gigabit Ethernet; Storage: EBS; OS: 64-bit Red Hat Linux
[7]`http://snap.stanford.edu`

G(N,E)	|N|	|E|	p-val	Opt
KG1	1,048,576	5,360,368	2%	γ
KG2m1	2,097,152	2,097,152	58.4%	α
KG2m2	2,097,152	2,866,928	28%	α
KG2m3	2,097,152	3,901,320	19%	α
KG2m4	2,097,152	5,285,328	6.7%	γ
KG2m5	2,097,152	11,632,000	0%	γ
KG2m6	2,097,152	29,960,082	0%	γ
KG2m7	2,097,152	74,082,437	0%	γ
KG16me1	1,048,576	17,161,558	0%	γ
KG16me2	2,097,152	17,233,107	0%	γ
KG16me3	4,194,304	17,112,694	0.3%	γ
KG16me4	8,388,608	17,119,151	7.1%	γ
KG16me5	16,777,216	17,184,533	7.2%	(γ)
cit-Patents	3,774,768	16,518,948	21.6%	α
LiveJournal	4,847,571	68,993,773	0%	γ
twitter tweets	1,370,080	2,714,598	34.3%	α
web-Google	875,713	5,105,039	17.7%	α
wiki-Talk	2,394,385	5,021,410	95.7%	(γ)

Table 1: Datasets. N, E denote nodes and edges, respectively. Graphs with prefix `KG` are synthetic. p-val is calculated by the optimizer and Opt is the optimizer's decision; parentheses denote wrong decision.

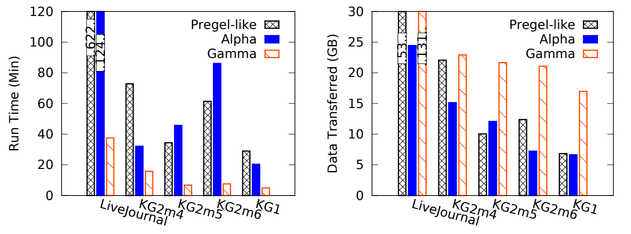
5.1 Experiments on Amazon EC2

The first set of experiments were run on 8 large Amazon EC2 instances. We execute the diameter estimation algorithm on all graphs in Table 1. Due to space considerations, we only report the most representative cases.

Non-power-law graphs. Figure 8(a) shows the total execution time for several non-power-law graphs. As expected, Mizan-γ is much faster than both Mizan-α and the Pregel-like system. Interestingly Mizan-α is in some cases slower than Pregel-like. This is due to the overhead of the minimum-cut partitioning and the fact that, for non-power-law graphs, the quality of the partitioning is low. Notice that Pregel-like is extremely slow for the LiveJournal input. This graph is problematic because it creates a very imbalanced hash partitioning; therefore, one machine is overloaded and slows down the entire system. Recall that, in contrast to Google's implementation, our Pregel-like system does not include dynamic load balancing. We believe that, with Google's implementation, Pregel will perform much better on this input. We also run experiments with PEGASUS, but the results are excluded because they were from 15% to 50% worse than both Mizan versions. Figure 8(b) shows the amount of data transferred among PEs. This includes the aggregate traffic flowing in all VMs.[8] Mizan-γ always transfers much more data than the other systems. However, due to the ring topology, the communication is pipelined (see Section 3.2) so the execution time remains low.

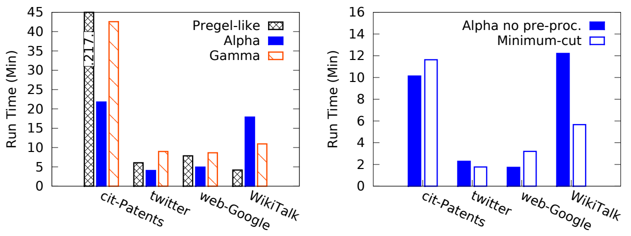
Power-law graphs. Figure 9(a) shows the total execution time for several power-law graphs. Similarly, PEGASUS was 2 to 17 times slower than any of the other system, so its results are excluded. As expected, Mizan-α is better in most of the cases. The exception is the wikiTalk, where Pregel-like is better than both Mizan versions. This graph is almost power-law;[9] thus, it is not suitable for Mizan-γ. At the same time, the expensive partitioning of Mizan-α does not pay-off. Therefore, the best approach is to use point-to-point message passing without minimum-cut partitioning. However, the current version of Mizan's optimizer does not have this option. We are currently working on an improved optimizer to tackle such cases. In Figure 9(b), we focus on Mizan-α and show separately the amount of time required for the minimum-cut partitioning and the actual time to run the user's code. It is clear that computing the minimum-cut is expensive. If the partitioning is available from a previous step (e.g., when running multiple graph mining algorithms on the same graph), then the benefit of Mizan-α is greater.

[8]We cannot differentiate between traffic flowing through the underlying physical network and those between VMs on the same physical host.



(a) Processing Time (b) Network Traffic

Figure 8: Non-power-law graphs, using 8 EC2 instances



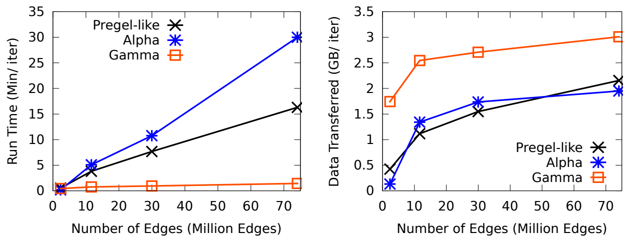
(a) Processing Time (b) Mizan-α vs Minimum-cut

Figure 9: Power-law graphs, using 8 EC2 instances

Mizan optimizer. Recall from Section 3.3 that the optimizer calculates the p-value for the input graph. If the p-value is between 10% and 90%, then the optimizer chooses Mizan-α, else it chooses Mizan-γ. The range has been suggested in the bibliography after empirical evaluation. The p-values and the corresponding decisions of the optimizer for all our graphs are shown in Table 1. We run both versions of Mizan with all inputs and confirmed that the optimizer was correct for all but two graphs (enclosed in parentheses in the table): (*i*) KG16me5 has p-value less than 10%, so it was processed by Mizan-γ, but in reality it is close to power-law; (*ii*) wikiTalk is a special case that does not belong in either category. As discussed earlier, our optimizer cannot handle such cases.

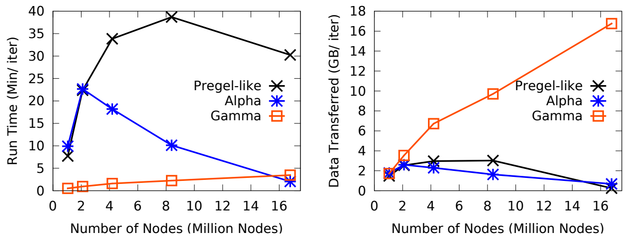
Variable number of edges. In this experiment, we fix the number of graph nodes to 2M and vary the number of edges. We used graphs KG2m{1,5,6,7}. Based on their p-values (Table 1), the first one is power-law, whereas the rest are not. Recall that we are using diameter estimation as the user's code. Because the graphs are different, the estimat-

[9]The graph fails the power-law test described in Section 3.3, where its p-value is 95% (higher than the 90% maximum acceptance threshold).



(a) Processing Time (b) Network Traffic

Figure 10: Graphs KG2m{1,5,6,7}. |N|=2M. 8 EC2 instances. Results are average per iteration.



(a) Processing Time (b) Network Traffic

Figure 11: Graphs KG16me{1,2,3,4,5}. |E|=17M. 8 EC2 instances. Results are average per iteration.

ed diameter and consequently the number of iterations will defer. For fair comparisons, we report the average time per iteration. The results are shown in Figure 10(a). Mizan-γ is fairly unaffected by the number of edges, whereas the execution times for the other two methods increase linearly. Interestingly, Mizan-α is slower than Pregel-like. To explain this, Figure 10(b) shows the amount of transferred data, which is roughly the same for Mizan-α and Pregel-like. This means that the minimum-cut partitioning did not provide any benefit, but Mizan-α had to pay the extra cost for the partitioning.

Variable number of nodes. We perform a similar experiment, but this time we fix the number of edges to 17M and vary the number of nodes. We used graphs KG16me{1,2,3,4,5} and report the average time per iteration. The results are shown in Figure 11. According to their p-values (Table 1), the smaller graphs are non-power-law, but as the number of nodes increases, the distribution becomes more power-law-like. This explains the fact that Mizan-α becomes faster with increasing number of nodes. For the same reason, Mizan-γ becomes slower when the number of nodes increases.

5.2 Scalability in the Cloud (Speed-up)

In the next experiment, we test the strong scalability (also known as speed-up) of Mizan on the cloud. We employ 2 to 16 instances of Amazon's EC2 and use the citPatents graph. Based on its p-value, the graph is power-law; therefore, Mizan-α is invoked. We test two different user algorithms: diameter estimation and PageRank. A significant difference between these algorithms is the amount of data they must process. In our implementation, each node in the diameter estimation algorithm is assigned 32 times more data compared to a PageRank node. The results are shown in Figure 12. As expected, diameter estimation benefits more when more processors are available. There is, however, one more aspect. The minimum-cut partitioning considers the graph as undirected, but in reality, the graph is directed. Moreover, the diameter estimation algorithm sends messages to the opposite direction of PageRank's messages. Therefore, the same partitioning results in a different number of messages crossing PEs, which explains why diameter estimation is faster than PageRank even though messages are 32 times larger.

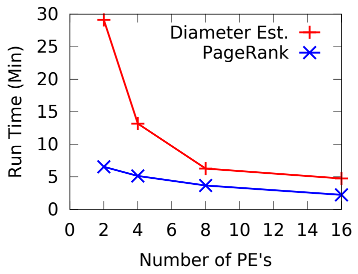
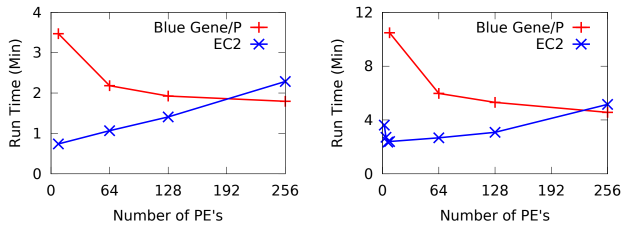


Figure 12: Strong scalability; 1 to 16 EC2 instances; citPatents graph; Mizan-α running diameter estimation and PageRank.



(a) KG2m5 (b) LiveJournal

Figure 13: Strong scalability; 2 to 256 compute nodes in BG/P; Mizan-γ running diameter estimation.

5.3 Large-scale Supercomputer Runs

In the final set of experiments, we demonstrate the performance difference when using two different computing infrastructures. We compare the Amazon's EC2 cloud infrastructure against an IBM Blue Gene/P (BG/P) supercomputer. BG/P is built around a very fast 3-dimensional network. When compared to a traditional x86 cluster, the architecture of BG/P favors using a larger number of slower compute nodes (PowerPC at 850MHz) that can take advantage of its high bisectional bandwidth. We used a synthetic (KG2m5) and a real (LiveJournal) graph. Based on their p-values, both are not power-law, so Mizan-γ is invoked. The results are shown in Figure 13. When the number of processing elements (PEs) is small, the EC2 environment is faster because the CPUs are roughly 3 times faster. However, when the number of PEs increases the physical links become the bottleneck in EC2, both in terms of bandwidth and latency, leading to slower execution. The BG/P, on the other hand, has a more efficient network, so performance is improved when the number of PEs increase. After roughly 200 PEs, BG/P becomes faster than EC2. We did not have access to more than 256 PEs in EC2, but in BG/P we managed to scale up to 1,024 PEs. Note that although a supercomputer is out of reach for most people, we believe that similar results can be achieved with affordable modern network cards that support remote direct memory access (RDMA).

6. RELATED WORK

A number of research efforts have focused on speeding up graph mining algorithms [15]. Manaskasemsak and Rungsawang [17] implemented a simple parallel PageRank and showed that by distributing the load over 8 machines and skipping communication on some iterations, the speed of the algorithm can be improved at the expense of accuracy. Bradley et al. [4] improved the performance of parallel MPI-based PageRank by using a hypergraph-based partitioning approach with one and two dimensional decomposition.

Hadoop [1] is an open source implementation of MapReduce. Hadoop-based graph mining parallelization has been adopted in both X-RIME [24], a social network aware data mining application and PEGASUS [11], a library of graph mining algorithms for very large graphs. However, due to the limitations of Hadoop, the structure of the graph can affect severely the performance.

A lot of work been done to improve the MapReduce [7] framework, such as Dremel [18], HaLoop [5] and iHadoop [8]. Dremel is a framework for interactive analysis of web-scale datasets which supports various analysis tasks for Google products. It is based on column-based storage systems, which reduces the amount of data read from disk significantly compared to MapReduce. On the other hand, HaLoop [5] and iHadoop [8] improve MapReduce by reducing the overhead of successive MapReduce tasks when processing complex applications. Pregel [16], on the other hand, uses the message passing model, which is well-suited for many graph mining algorithms. Our approach, Mizan, follows Pregel's model but improves performance by considering the structure of the input graph and the architecture of the underlying computing infrastructure. Mizan uses MPI and multilevel partitioning (based on ParMETIS [12]) to improve the performance of large graph mining algorithms.

7. CONCLUSION

In this work, we developed Mizan, a framework for supporting graph mining algorithms in large parallel computing infrastructures. Mizan is a layer between the user's code and a variety of computing infrastructures, such as Linux clusters, cloud environments and supercomputers. Mizan examines the graph structure and decides the placement of the data and the low level message passing mechanism transparently to the user's code. We deployed Mizan on 256 virtual machines in Amazon's EC2 and 1,024 CPUs on an IBM Blue Gene/P supercomputer. Compared to existing approaches, Mizan executes the user code up to 50% faster for power-law graphs and up to 500% faster for other graphs. Currently, we are working on an improved optimizer. Also, we are planning to support the dynamic case, where the exchanged messages do not depend only on the static structure of the graph, but on conditions that are modified during runtime.

8. REFERENCES

[1] Hadoop MapReduce Programming Model, Documentation and Implementation. http://hadoop.apache.org/mapreduce.

[2] J. Aldrich. R. A. Fisher and the Making of Maximum Likelihood 1912-1922. *Statistical Science*, 12(3):162–176, 1997.

[3] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards Predictable Datacenter Networks. In *Proc. of the ACM SIGCOMM*, pages 242–253, NY, NY, 2011.

[4] J. T. Bradley, D. V. Jager, W. J. Knottenbelt, and A. Trifunovic. Hypergraph Partitioning for Faster Parallel PageRank Computation. In *Lecture Notes in Computer Science 3670*, pages 155–171. Springer, 2005.

[5] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst. HaLoop: Efficient Iterative Data Processing on Large Clusters. *PVLDB*, 3:285–296, September 2010.

[6] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-Law Distributions in Empirical Data. *SIAM Review*, 51(4):661–703, 2009.

[7] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. of OSDI*, pages 137–150, 2004.

[8] E. Elnikety, T. Elsayed, and H. E. Ramadan. iHadoop: Asynchronous Iterations for MapReduce. In *Proceedings of CloudCom*, November 2011.

[9] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In *Proceedings of IEEE CloudCom*, 2010.

[10] U. Kang, C. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec. HADI: Fast Diameter Estimation and Mining in Massive Graphs with Hadoop. *ACM Trasactions on Knowledge Discovery from Data (TKDD)*, 2011.

[11] U. Kang, C. E. Tsourakakis, and C. Faloutsos. PEGASUS: A Peta-Scale Graph Mining System Implementation and Observations. In *Proc. of IEEE ICDM*, 2009.

[12] G. Karypis and V. Kumar. Parallel Multilevel k-Way Partitioning Scheme for Irregular Graphs. In *Proc. of the ACM/IEEE Conference on Supercomputing*, 1996.

[13] G. Karypis and V. Kumar. Multilevel k-Way Partitioning Scheme for Irregular Graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[14] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani. Kronecker Graphs: An Approach to Modeling Networks. *Journal of Machine Learning Research*, 11:985–1042, Feb. 2010.

[15] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. W. Berry. Challenges in Parallel Graph Processing. *Parallel Processing Letter*, pages 5–20, 2007.

[16] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A System for Large-scale Graph Processing. In *Proc. of ACM SIGMOD*, pages 135–146, 2010.

[17] B. Manaskasemsak and A. Rungsawang. Parallel PageRank Computation on a Gigabit PC Cluster. In *Proc of Int. Conf. on Advanced Information Networking and Applications*, 2004.

[18] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: Interactive Analysis of Web-scale Datasets. *Communications of ACM*, 54:114–123, June 2011.

[19] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. http://ilpubs.stanford.edu:8090/422/, 2001.

[20] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. ANF: A Fast and Scalable Tool for Data Mining in Massive Graphs. In *Proc. of the ACM SIGKDD*, pages 81–90, 2002.

[21] A. N. Pettitt and M. A. Stephens. The Kolmogorov - Smirnov Goodness-of-Fit Statistic with Discrete and Grouped Data. *Technometrics*, 19(2):205–210, 1977.

[22] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *PVLDB*, 3:460–471, 2010.

[23] L. G. Valiant. A bridging Model for Parallel Computation. *Communications of the ACM*, 33:103–111, August 1990.

[24] W. Xue, J. Shi, and B. Yang. X-RIME: Cloud-Based Large Scale Social Network Analysis. In *Proc. of IEEE Int. Conf. on Services Computing*, pages 506 –513, 2010.