Large Scale Graph Mining with G-Miner

Hongzhi Chen, Xiaoxi Wang, Chenghuan Huang, Juncheng Fang, Yifan Hou, Changji Li, James Cheng
Department of Computer Science and Engineering
The Chinese University of Hong Kong
{hzchen,xxwang,chhuang,jcfang6,yfhou,cjli,jcheng}@cse.cuhk.edu.hk

ABSTRACT

This Demo presents G-Miner, a distributed system for graph mining. The take-aways for Demo attendees are: (1) a good understanding of the challenges of various graph mining workloads; (2) useful insights on how to design a good system for graph mining by comparing G-Miner with existing systems on performance, expressiveness and user-friendliness; and (3) how to use G-Miner for interactive graph analytics.

ACM Reference Format:
Hongzhi Chen, Xiaoxi Wang, Chenghuan Huang, Juncheng Fang, Yifan Hou, Changji Li, James Cheng. 2019. Large Scale Graph Mining with G-Miner. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3299869.3320219

1 INTRODUCTION

In recent years, many graph processing systems have been proposed [7]. However, the majority of these systems follow Pregel [2]'s vertex-centric programming framework, which can easily implement a parallel version of algorithms such as PageRank, connected components, and breadth-first search. The common characteristic of these algorithms is that the computation and communication on each vertex are usually light (mostly of linear complexity) in each iteration.

The vertex-centric framework and its respective systems, however, are not suitable for processing graph mining jobs such as community detection, subgraph mining, graph clustering, graphlet counting/listing, graph matching, to name but a few. Graph mining jobs are generally *computation-intensive* and/or *memory-intensive*. Due to the well-known combinatorial explosion problem in the generation of (candidate) subgraphs, the computation and memory overheads in graph mining often grow quickly (at least superlinearly or even exponentially in the worst case). The much heavier workload of graph mining jobs also renders distributed computing a good option for large-scale graph mining; in contrast, McSherry et al. showed that distributed vertex-centric systems have a high COST [3], i.e., the cost needed to outperform a single-threaded implementation is high. Besides the much higher computational complexity, graph mining algorithms are also generally more difficult to implement.

General programming frameworks and systems for large-scale graph mining have been lacking. In our prior work [1], we analyzed the limitations of existing graph mining systems (e.g., NScale [4], Arabesque [5], G-thinker [6]) and proposed a new distributed system, called **G-Miner**. G-Miner follows a new *graph-centric* programming paradigm, in which computation is applied directly on each of the subgraphs that may potentially produce a result of the mining job. Based on this graph-centric paradigm, we develop a *task-oriented* computation framework, which encapsulates the processing of a graph mining job as a stream of independent tasks and streamlines task processing with a novel task-pipeline design. The new system design removes synchronization barrier in existing systems and allows various resources (i.e., CPU, network, disk) to be used concurrently. As a result, the communication and disk I/O costs are hidden inside the higher CPU cost of *computation-intensive* graph mining jobs, while we address the *memory-intensive* problem by buffering tasks on disk as tasks are independent of each other. We showed that G-Miner has a low COST [1].

This Demo plans to show the efficiency of G-Miner for processing various graph mining applications and how G-Miner addresses the limitations of existing systems with its new design. We will visualize the details of G-Miner's internal processing and how various system components work together to resolve typical bottlenecks of distributed graph mining, thereby providing Demo attendees a good understanding of where the challenges of distributed graph mining lie and good insights of how to design and implement an efficient distributed system for general graph mining. Our Demo system will also provide a graphical interface to allow

SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3320219

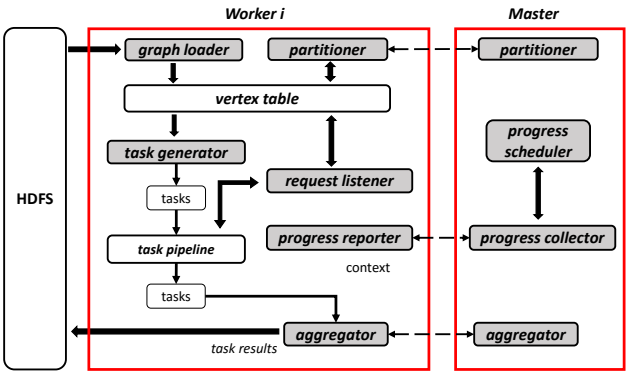

Figure 1: System Architecture

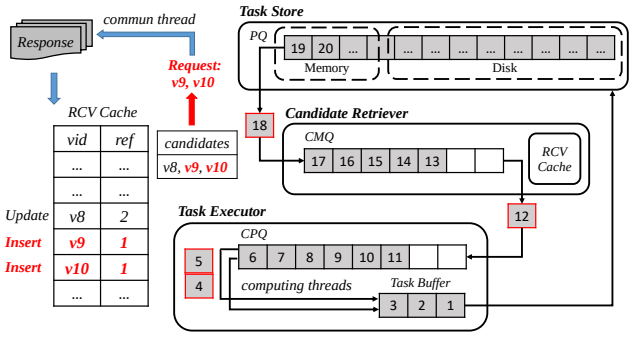

Figure 2: Task-Pipeline

users to interact with G-Miner, so as to help them understand the mining results and conduct further analysis recursively.

2 THE G-MINER SYSTEM

We briefly describe basic concepts and design of G-Miner and its key components. Details can be found in the G-Miner paper [1] and code (https://github.com/yaobaiwei/GMiner).

2.1 What G-Miner Supports and Its API

G-Miner provides a unified programming framework for implementing distributed algorithms for a wide range of graph mining applications including *(1) subgraph/graphlet enumeration (e.g., triangles, cliques, size-k graphlets); (2) subgraph matching (i.e., listing all occurrences of a set of query subgraphs); (3) subgraph finding (e.g., maximum clique finding, densest subgraph finding, etc.); (4) subgraph mining (e.g., frequent graph mining, community detection, correlated subgraph mining, etc.); (5) graph clustering.*

The classic algorithms for solving these mining problems generally follow the pattern that starts from some initial subgraphs (e.g., seed vertices) and then recursively performs an update operation (e.g., grow, prune, split, output) on each intermediate subgraph. G-Miner offers a succinct API that only requires users to implement an *init()* and *update()* function for the specific graph mining task. Details and examples can be found in [1] and will be shown in the Demo.

2.2 Core Concepts

Task Model. We model a graph mining job as a stream of *independent tasks*, where a task is processed in rounds and new tasks may be generated and added to the stream in runtime. A *task* consists of three fields: (1) a *subgraph* g to keep the topology of an intermediate subgraph from which a mining result may be obtained; (2) *candidates* to record the IDs of g's 1-hop neighbors that will be used to update g in the next round; (3) *context* to hold meta-data, e.g., the current round number, the count of matched patterns.

Task Lifetime. A task may have the following four statuses. (1) *active*: being processed by CPU; (2) *inactive*: remote vertices in *candidates* to be pulled through network; (3) *ready*: ready to be processed; (4) *dead*: task completed or pruned.

Task Asynchronism. If a task has no remote vertex required in the current round, it directly continues to the next round without status change. Task processing in G-Miner has no synchronization barrier.

2.3 System Architecture and Components

G-Miner adopts a master-slave shared-nothing architecture, as shown in Figure 1. One node in the cluster serves as the master and others are workers. Each worker W_i loads a piece of graph data P_i from HDFS by the *graph loader*. The *partitioner* in the master communicates with the *partitioner* in each worker to re-distribute the graph data based on a specific partitioning algorithm.

When a mining job starts, the *task generator* in each worker will scan the local vertex table to select the *seed vertices* and then generate one task for each seed. These tasks are fed into the task-pipeline to be executed. An *aggregator* may be used to access the context of each task at the end of each round for global communication and monitoring. Each worker also has a *request listener* to handle requests for vertex pulling or tasks stealing from other workers. To implement *task stealing*, each worker has a *progress reporter* that sends its local progress to the master periodically, while the master uses a *progress collector* to receive the reports for maintaining a global view of the workers' progresses, which is used by the *progress scheduler* to facilitate dynamic migration of tasks from busy workers to idle workers.

Task-Pipeline. Lying at the core of G-Miner is the task-pipeline, which is designed to use CPU, network and disk concurrently in order to achieve good resource utilization: (1) CPU computation for the main mining procedure, (2) network communication to pull candidates from remote machines, and (3) disk writes/reads to buffer intermediate tasks on local disk. This is done by the three main components

in the task-pipeline: *task store*, *candidate retriever* and *task executor*, as illustrated in Figure 2.

The **task store** manages all *inactive* tasks on local disk. Considering different tasks may request the same candidate vertices from remote workers, we apply caching to avoid repeated vertex pulling. To improve the cache hit ratio, we propose a locality-sensitive *task priority queue* (*PQ*) to order tasks by keeping those with common remote candidates near each other. The **candidate retriever** prepares the remote vertices in a task's candidates, by getting it from the *RCV Cache* (which uses a replacement strategy based on the *Reference Counts* of the cached *Vertices*) or pulling via network. When a task has issued its pull requests, it is moved into the *communication queue* (*CMQ*) waiting for the pull responses. Once all remote candidates are obtained, a task changes its status to *ready* and is then inserted into the *computation queue* (*CPQ*) managed by the **task executor**, which consists of a pool of computing threads to process tasks in parallel. If a task becomes *inactive*, it is placed into a *task buffer* to be moved to the task store in batches.

3 DEMONSTRATION PLANS

Objectives. The objectives of this demo are to show SIGMOD attendees: (1) the superior efficiency and scalability of G-Miner, as well as its expressiveness, compared with two state-of-the-art graph mining systems, Arabesque [5] and G-thinker [6], and two popular vertex-centric systems, Giraph and GraphX, on various categories of graph mining algorithms; (2) real-time display of G-Miner's runtime system status and resource utilization, in order to demonstrate how various components of G-Miner interact with each other to achieve superior performance and address the bottlenecks of existing systems; and (3) an intuitive visualization of the in-process mining results and an interface for interaction with G-Miner to help users understand the mining results.

Set-up. The back-end engine of G-Miner will be deployed and run on cloud or a remote cluster (e.g., the cluster used for the experimental evaluation of G-Miner in [1]). The front-end interface of G-Miner will be run on a laptop and support user interaction with Demo attendees. We plan to use the six real-world graph datasets in [1].

3.1 System Comparison

This part of the Demo justifies why G-Miner is preferred over the state-of-the-art graph mining systems and other popular graph processing systems regarding to its performance. In addition, for a system to be useful, one should also show that it is expressive and easy to use. To this end, we will show Demo attendees G-Miner's user-friendly API, and how succinctly and intuitively we can implement various categories of applications, including typical graph mining

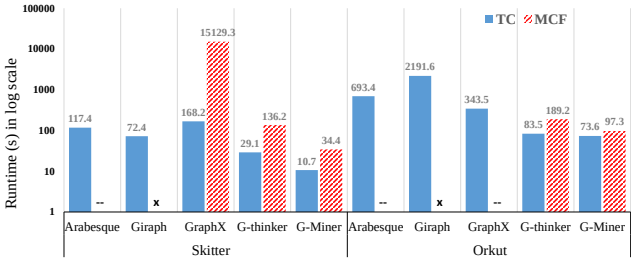

Figure 3: Performance results of TC and MCF ("-": >24 hours; "x": job failed due to OOM)

algorithms such as *Triangle Counting* (**TC**), *Maximum Clique Finding* (**MCF**), *Graph Matching* (**GM**), *Community Detection* (**CD**), and *Graph Clustering* (**GC**). Meanwhile, we will also explain the difficulties in using the APIs of existing systems to implement these algorithms, through this process Demo attendees will also see why graph mining workloads are generally much heavier than typical workloads of vertex-centric systems (as we have also discussed in Section 1).

The above algorithms are listed in ascending order of their computational complexity. We will show the performance benefits G-Miner has over existing systems by showing how their performance bottlenecks are addressed by G-Miner's new system design. In particular, for algorithms with higher complexity, G-Miner's performance advantages become more obvious. For example, Figure 3 shows that G-Miner is significantly faster than some other systems we compared in [1] for processing TC and MCF on the two datasets *Skitter* and *Orkut*, and the performance gap widens considerably for the heavier MCF workload. While more detailed performance comparison results were reported in [1], this Demo aims to explain (with visualized details) why other systems are inefficient in processing graph mining workloads, by comparing their design with that of G-Miner.

3.2 The Anatomy of G-Miner

In this part of the Demo, we will show a detailed view of where the superior performance of G-Miner comes from, by showing its key design idea and how various system components work together. The take-aways for Demo attendees are a good understanding of the difficulties in processing large-scale graph mining workloads and insights on how to build an efficient and scalable graph mining system.

To this end, we implemented a *runtime information monitor* (*RIM*) for G-Miner, which allows Demo attendees to configure and interact with G-Miner. Attendees may manually configure system parameters (e.g., cache size, thread-pool size, the sizes of various queues) for various components through RIM's control panel, so that they can easily observe the performance benefits brought by various system components and optimization techniques, and their respective

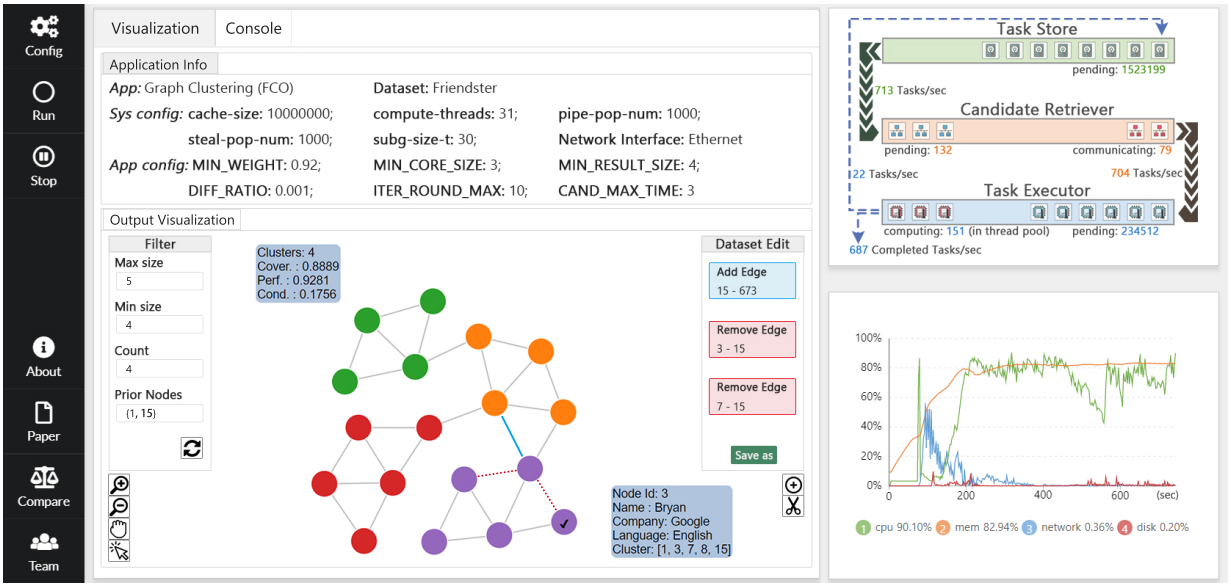

Figure 4: A screenshot of G-Miner's RIM interface (best viewed in color)

trade-offs (if any). RIM will also display a detailed view of G-Miner's runtime info (e.g., job progress, in-process results, resource utilization), in real time (a snapshot at current time is depicted in Figure 4). RIM also has a runtime display panel to illustrate how various components (e.g., task store, candidate retriever, task executor) in G-Miner interact with each other and how tasks of different statuses flow in the task-pipeline. In particular, the panel shows the number of specific tasks being processing and queued in each component of the task-pipeline, as well as the task flow rate between any two connecting components, to help attendees better understand the mining procedure of G-Miner and the combinatorial explosive nature of graph mining applications. The utilization of various resources (e.g., CPU, memory, network, disk) by G-Miner will show attendees how CPU-intensive and memory-intensive a graph mining job is, and how G-Miner hides the network cost and disk I/O inside the CPU cost.

In addition, the display panel also visualizes the in-process result of the current ongoing graph mining job, e.g., for MCF (or GM), we can show the currently found maximum clique (or matched patterns), including its size and topology.

3.3 Interactive Graph Mining w/ G-Miner

The Demo system also supports interactive graph mining. For example, for GC, users may remove any vertex from, and/or add/remove any edge to/from, a found cluster through our GUI, and then observe visually how the cluster changes, which is useful to understand the importance of certain vertices and their connection to others. Other detailed information will also be presented, including metrics that measure the quality of clusters (e.g., Conductance, Coverage), as shown in Figure 4. Similar interactions may also apply to other mining applications, e.g., CD and MCF. For GM, we also allow users to interactively find matching patterns and their locations in the graph, zoom in and out to explore their neighborhoods. For graphlet counting/listing, in addition to all the statistics about the counts of various graphlets, we can also display their visual occurrences in the graph, how they connect to and overlap with each other.

Acknowledgments. We thank the reviewers for their valuable comments. This work was supported in part by ITF 6904945 from the Government of the Hong Kong, and GRF 14208318 & 14222816 from the Hong Kong RGC.

REFERENCES

[1] Hongzhi Chen, Miao Liu, Yunjian Zhao, Xiao Yan, Da Yan, and James Cheng. 2018. G-Miner: an efficient task-oriented graph mining system. In *Proceedings of the Thirteenth EuroSys Conference*. ACM, 32.

[2] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *SIGMOD*. ACM, 135–146.

[3] Frank McSherry, Michael Isard, and Derek Gordon Murray. 2015. Scalability! But at what COST?. In *HotOS*.

[4] Abdul Quamar, Amol Deshpande, and Jimmy Lin. 2016. NScale: neighborhood-centric large-scale graph analytics in the cloud. *The VLDB Journal* 25, 2 (2016), 125–150.

[5] Carlos HC Teixeira, Alexandre J Fonseca, Marco Serafini, Georgos Siganos, Mohammed J Zaki, and Ashraf Aboulnaga. 2015. Arabesque: a system for distributed graph mining. In *SOSP*. ACM, 425–440.

[6] Da Yan, Hongzhi Chen, James Cheng, M Tamer Özsu, Qizhen Zhang, and John Lui. 2017. G-thinker: big graph mining made easier and faster. *arXiv preprint arXiv:1709.03110* (2017).

[7] Da Yan, Yuanyuan Tian, and James Cheng. 2017. *Systems for Big Graph Analytics*. Springer.