GraphGen: A Distributed Graph Sample Generation Framework on Industry-Scale Graphs

Yue Jin, Sheng Tian, Yongchao Liu, Chuntao Hong
Ant Group, China

The field of graph computing and graph learning is currently experiencing a significant increase in scale. To illustrate, platforms like Facebook, with approximately 3 billion users, generate trillions of connections through user interactions, relationships, and posts, forming a massive graph with one trillion edges. As time progresses, the scale of the graph will continue to grow as users continue to engage and share content. However, performing graph computing and learning on such large-scale graphs presents considerable challenges. Issues related to data processing, storage, and model training emerge as obstacles to overcome. One widely accepted approach to addressing these challenges is by extracting subgraphs from the large-scale graphs for subsequent graph computing and learning. By extracting subgraphs, the complexity of data processing and training for each iteration can be reduced, leading to much higher efficiency. However, as the size of the graph continues to expand, traditional methods like SQL (input-output bound) for complex subgraph extraction and generation, like k-hop neighbor sampling and long-range random walk sampling, become increasingly inefficient. This paper introduces a novel solution to industry-scale graphs in the form of a distributed graph sample generation framework. This framework employs distributed graph computing techniques to efficiently partition the large-scale graph and generate the required subgraphs in each graph partition in main memory, overcoming the limitations of existing methods. Compared with traditional methods such as SQL, our approach achieves **20× speedup on our industrial graph dataset of Alipay with 530 million nodes and 5 billion edges**.

Graph Sample Generation Framework: The input for our graph sample generation process consists of a base graph G and a set of seed nodes S. These seed nodes represent the nodes of interest, and each subgraph generated originates from a specific seed node. To distribute the workload evenly among distributed machines, we employ a graph partitioning technique to divide the base graph. Each machine is assigned the task of generating its own subgraphs, specifically finding neighbor subgraphs starting from the assigned seed nodes. Furthermore, the generated subgraphs are stored in a key-value store. This store holds essential information about the subgraph, including its topology, as well as the features of its nodes and edges. Once the important subgraphs have been generated, they can be utilized for graph computing and learning tasks. For example, graph learning training tasks involve grouping multiple subgraphs together to form a mini-batched graph data. During each mini-batch iteration, a specific subset of the subgraphs is selected and loaded to train a graph neural network model.

Our current system offers support for a wide range of graph sampling algorithms, encompassing considerable well-established techniques including k-hop neighbor sampling, meta-path sampling, random walk sampling, ego-network sampling, node2vec sampling and more.

Implementation: Our implementation consists of two core components: graph partitioning and graph sampling.

To address graph partitioning, we employ edge partitioning to divide the graph by cutting the vertices. This approach aims to distribute the workload as evenly as possible among the workers. Given that power-law distributions are commonly observed in industrial graph data, edge partitioning proves advantageous in achieving load balancing. Moreover, it helps prevent scenarios where a single worker is burdened with all hot-spots and edges.

In the context of graph sampling, we apply our Exchange-Apply-Scatter (EAS) and MapReduce-on-Graph (MRoG) parallel graph processing models to generate the subgraphs centered around each seed node. To illustrate the process of generating subgraphs, we consider the case of k-hop neighbor sampling. Our input comprises a graph G and a set of seed nodes S. (1) First, we designate each node v in G as *grey* if v belongs to S. (2) Second, we execute the exchange operation for each *grey* node u. Within this step, we gather and mark the neighbors $N(u)$ of u with the index of u. Additionally, we apply and store the neighbor indices within u. (3) Third, we examine each edge in the graph. If both the source and target nodes possess the same seed node index mark, it signifies that the edge originates from a seed node. Consequently, with MRoG, we map each edge and reduce them such that the seed node serves as the destination for collecting the edges belonging to the same subgraph. (4) Finally, for every node w in G, if w is marked as a neighbor of any seed node, we designate w as *grey* and proceed back to the second step, facilitating the exploration of the next level of the subgraph.

The subgraph information is stored utilizing a key-value storage system. Each seed node serves as a unique key, while the corresponding subgraph is stored as the associated value. Simultaneously, we separate the feature information of nodes and edges from the subgraph's topology. This segregation reduces the burden of data storage and parsing, thereby optimizing the overall system.