Dynamic Graph Sampling Service for Real-time GNN Inference at Scale

Jie Sun[†] Li Su[‡] Wenting Shen[‡] Zichao Zhang[‡] Zuocheng Shi[†] Jingbo Xu[‡]
Yong Li[‡] Wenyuan Yu[‡] Zeke Wang[†] Fei Wu[†] Jingren Zhou[‡]

{jiesun, shizuocheng, wangzeke, wufei}@zju.edu.cn[†]
{lisu.sl, wenting.swt, houbai.zzc, xujingbo.xjb, jiufeng.ly, wenyuan.ywy, jingren.zhou}@alibaba-inc.com[‡]
Zhejiang University[†] Alibaba Group[‡]

Abstract

Graph neural networks(GNNs) learn graph vertex representations by aggregating multi-hop neighbor information. Industrial applications often adopt mini-batch training to scale out GNNs on large-scale graphs, where neighbor sampling is used during both model training and inference. Since the structure and attributes of real-world graphs often change dynamically, it is imperative that the inferred vertex representation can accurately reflect these updates.

GNN inference services, such as real-time recommendation systems, often require stable millisecond-level latency SLO. However, meeting this requirement can be challenging due to highly concurrent inference requests and dynamic graph updates. Firstly, multi-hop sampling can introduce high time complexity, as sampling a vertex often requires traversing all its neighbors. Secondly, as the graphs in industrial settings often exceed the single-machine memory, graphs are either persisted in disk or partitioned and stored in-memory in a distributed cluster. Both approaches will incur significant I/O overheads during graph sampling. Thirdly, the computation required for sampling different vertices can vary significantly due to the inherent skewness in real-world graphs, which will lead to unstable latency performance among concurrent inference requests. Given these observations, existing approaches, e.g., using graph databases for storage and graph sampling, cannot fulfill the performance SLOs of real-time GNN inference services.

In this work, we propose Dynamic Graph Sampling Service (DGS), which aims to address the challenges associated with graph sampling in real-time GNN inference on dynamic graphs. Our key insight is that the GNN inference service is query-aware: given a GNN model, the graph sampling query for both training and inference is fixed. With this observation, we propose an event-driven pre-sampling mechanism in DGS. Driven by the graph updates, sample caches of vertices are dynamically updated using reservoir sampling following the specified query. In specific, DGS decomposes a k-hop sampling query into k one-hop sampling queries. For each one-hop query, when a graph update of a relevant vertex (e.g., an edge sourcing from this vertex with a specified vertex label) arrives, the one-hop sampling results of this vertex will be updated accordingly. The k-hop sampling result of an inference request can be constructed via a fixed number of point look-ups in the cached one-hop sampling results.

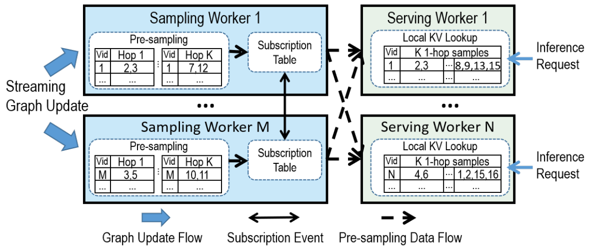

Figure 1: Architecture of DGS.

The overall architecture of DGS is depicted in Figure 1. DGS mainly consists of two types of components: sampling workers and serving workers. The input graph updates are partitioned according to the key (e.g., vertex IDs) range. Each sampling worker is responsible for a specific partition: conducting pre-sampling for the one-hop sampling queries and transmitting the results to the serving workers. Each serving worker caches the sampling results of k one-hop queries received from sampling workers and serves the inference requests for a partition of vertices in the graph. The sampling and serving workers can scale independently to cope with workload fluctuations of graph updates and inference requests. To minimize the latency in generating complete k-hop sampling results, DGS sends all k-hop sampling results of vertex v_i to the serving worker that handles v_i's inference request, such that generating the complete graph sampling for an inference request only requires accessing local caches on a single serving worker. To achieve this, every sampling worker maintains a subscription table for each one-hop query recording the list of serving workers that subscribe to the one-hop query results. E.g., either adding or deleting vertex v_j from the first-hop samples of v_i triggers a message recording this event sent to the sampling server that holds the partition containing v_j, and the subscription information of v_j will be updated correspondingly. With this design, DGS can achieve a very stable latency performance under highly concurrent inference workloads.

Experiments on real Alibaba e-commerce datasets show that DGS can maintain the P99 latency of inference requests (of a two-hop random sampling query) within $20ms$ milliseconds, and process around $20,000$ requests per second in each serving worker. The throughput of update ingestion of a single sampling worker reaches 109MB/s and can scale out linearly.

This work was supported by Alibaba Group through Alibaba Innovative Research Program.