GraphGen+: Advancing Distributed Subgraph Generation and Graph Learning On Industrial Graphs

Yue Jin, Yongchao Liu, Chuntao Hong
Ant Group, China
{jinyue.jy,yongchao.ly,chuntao.hct}@antgroup.com

Graph-based computations are crucial in a wide range of applications, where graphs can scale to trillions of edges. To enable efficient training on such large graphs, mini-batch subgraph sampling is commonly used, which allows training without loading the entire graph into memory. However, existing solutions face significant trade-offs: online subgraph generation, as seen in frameworks like DGL and PyG, is limited to a single machine, resulting in severe performance bottlenecks, while offline precomputed subgraphs, as in GraphGen, improve sampling efficiency but introduce large storage overhead and high I/O costs during training. To address these challenges, we propose **GraphGen+**, an integrated framework that synchronizes distributed subgraph generation with in-memory graph learning, eliminating the need for external storage while significantly improving efficiency. GraphGen+ achieves a **27×** speedup in subgraph generation compared to conventional SQL-like methods and a **1.3×** speedup over GraphGen, supporting training on 1 million nodes per iteration and removing the overhead associated with precomputed subgraphs, making it a scalable and practical solution for industry-scale graph learning.

1 Introduction

Graph-based computations are essential for many applications. As real-world graphs expand in size, encompassing billions or even trillions of edges [1], efficiently processing and learning from these massive graphs has become increasingly important. To enable efficient training on such large graphs, mini-batch subgraph sampling is commonly used, which allows training without loading the entire graph into memory. However, existing solutions face significant trade-offs. Frameworks like DGL and PyG are designed for online subgraph generation, where subgraphs are sampled dynamically during training, but limited to single-machine environments, making them inefficient for large-scale graphs. The sampling and training processes are constrained by the memory and computational capabilities of a single machine, leading to high overhead and poor scalability. Therefore, they can only handle graphs with up to a billion edges.

In contrast, frameworks such as AGL [6] and GraphGen [3] utilize MapReduce [2] for offline subgraph generation, significantly improving sampling efficiency. By distributing the subgraph extraction process across multiple machines, GraphGen addresses the computational bottleneck present in single-machine frameworks. However, this approach still faces a major limitation: it requires substantial storage to precompute and store the subgraphs. This is especially problematic for large graphs, as storing these precomputed subgraphs demands significant disk space and incurs high I/O costs. Furthermore, the need to read and write subgraphs from local or network disk storage introduces significant delays, slowing down the training process. On the other hand, AGL utilizes a node-centric MapReduce paradigm, which serially processes neighbor collection when high-degree nodes occur, creating performance bottlenecks.

To overcome these limitations, we propose **GraphGen+**, a novel high-performance framework that integrates distributed subgraph generation with in-memory graph learning. By synchronizing subgraph generation and in-memory training, GraphGen+ eliminates the need for pre-computed subgraphs and external storage, improving both storage efficiency and system performance. Moreover, GraphGen+ employs an edge-centric MapReduce approach, enabling parallel neighbor collection even in the presence of high-degree nodes. The experimental results show that GraphGen+ achieves a 27× speedup in subgraph generation compared to traditional SQL-like methods and a 1.3× speedup over GraphGen. In addition, it supports training at 1 million nodes per iteration, making it a scalable and practical solution for large-scale graph learning tasks.

In summary, we present GraphGen+, a high-performance framework that is oriented to industry-scale graph learning: ① Integrates distributed subgraph generation with in-memory graph learning, enabling high-performance graph learning on large-scale graphs. ② Optimizes subgraph distribution across workers using a load-balancing strategy, ensuring efficient parallel execution and minimizing resource underutilization. ③ Reduces storage overhead by eliminating the need for external storage, improving both speed and scalability. ④ Achieves a 27× speedup in subgraph generation compared to traditional SQL-like methods and a 1.3× speedup over GraphGen, making it a scalable and practical solution for large-scale graph learning tasks.

2 Methods

GraphGen+ integrates distributed subgraph generation with in-memory graph learning to optimize large-scale graph processing. The method proceeds as follows: (1) **Graph Partitioning:** The input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is partitioned across multiple worker nodes, enabling parallel computation and

reducing memory burdens on individual worker nodes. This step is executed on the central node. (2) **Load-Balanced Subgraph Mapping:** The central node creates a balance table to map subgraphs to workers' memory, ensuring that the final subgraph for each seed node is evenly distributed among workers. This distribution aims to maintain a balanced workload across all workers. The algorithm for creating the balance table allocates seed nodes to a list of workers sequentially. If there are leftover seed nodes that do not fill an entire cycle of the list, they are discarded. This approach ensures that the workload remains balanced, adhering to the principles of distributed data parallel graph learning training. (3) **Distributed Subgraph Generation:** Using a MapReduce-based approach, subgraph extraction is carried out by assigning each edge to the corresponding worker based on its seed node. The workers then generate subgraphs by collecting edges connected to their assigned seed nodes. To address hot nodes and mitigate load imbalance, we employ a Tree-Reduction method. This method helps distribute the workload more evenly among workers, reducing the impact of synchronization overhead. However, performance can still be affected by bandwidth limitations and worker locality issues, which we aim to optimize in future work. (4) **In-Memory Graph Learning:** After subgraphs are generated, they are immediately loaded into memory for in-memory graph learning, eliminating storage and I/O overhead, which enhances training efficiency. Finally, subgraph generation and training are conducted concurrently, with workers generating new subgraphs during training and immediately loading them into memory for further processing. At the end of each iteration, gradient synchronization between all workers is performed to ensure the convergence of the model. This step is executed on worker nodes.

The entire workflow is summarized in Algorithm 1, which provides a detailed description of the partitioning, mapping, extraction, and training steps involved.

Algorithm 1: GraphGen+ Workflow

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V}$ and edges $\mathcal{E}$,
Seed nodes $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$,
Worker nodes $\mathcal{W} = \{w_1, w_2, \ldots, w_m\}$
Output: Trained graph model with optimized subgraph generation and workload distribution

Step 1: Graph Partitioning
Distribute $\mathcal{G}$ across $\mathcal{W}$ using a distributed partitioning strategy;
Step 2: Load-Balanced Subgraph Mapping
Initialize iterator $it \leftarrow$ begin of $\mathcal{S}$;
Compute $max\_i \leftarrow \lfloor |\mathcal{S}|/|\mathcal{W}| \rfloor \times |\mathcal{W}|$;
for $i \leftarrow 0$ **to** $max\_i - 1$ **do**
    if it = *end of* $\mathcal{S}$ **then**
        break;
    end
    Assign $\mathcal{M}[it] \leftarrow \mathcal{W}[i \bmod |\mathcal{W}|]$;
    Increment it;
end
Step 3: Distributed Subgraph Generation
foreach *edge* $E \in \mathcal{E}$ **do**
    Identify seed node S containing E;
    Query *subgraphMap* to determine worker ID W storing S;
    Append E to subgraph $Graph(S)$ on worker W;
end
Step 4: In-Memory Graph Learning
while *training is not complete* **do**
    foreach *worker* $W \in \mathcal{W}$ *in parallel* **do**
        Fetch next available subgraph $Graph(S)$;
        Train model on $Graph(S)$;
        Synchronize gradients across workers;
    end
end

3 Preliminary Results

We evaluate GraphGen+ on a graph with 530 million nodes and 5 billion edges, using a GCN [4] model for mini-batch training. Our experiments are conducted on a 256-node Docker cluster, where each container is allocated 8 CPU cores and 16GB of memory. For subgraph sampling, we use a 2-hop neighborhood expansion strategy, selecting 40 neighbors in the first hop and 20 neighbors in the second hop for each seed node. Subgraph generation is completed in 3 minutes, processing 5.9 million nodes per second, which represents a 27× speedup over traditional SQL-like methods and 1.3× speedup over GraphGen.

The 1.3× speedup is primarily attributed to the Load-Balanced Subgraph Mapping, which ensures balanced workload among workers, and the Tree-Reduction technique that addresses hot node load issues. Our system is capable of training on up to 1 million nodes per iteration, a limit derived from experimental settings that showcase efficient concurrent subgraph generation and training at this scale.

4 Conclusion

GraphGen+ integrates distributed subgraph generation with in-memory graph learning, achieving a **27×** speedup in subgraph generation and eliminating the need for external storage. Our approach is implemented in our graph intelligent computing system [5] for in production environment and offers a scalable, efficient solution for industrial-scale graph learning, making it well-suited for real-world applications.

References

[1] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. 2015. One trillion edges: Graph processing at facebook-scale. In *VLDB*.

[2] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[3] Yue Jin, Sheng Tian, Yongchao Liu, and Chuntao Hong. 2024. GraphGen: a distributed graph sample generation framework on industry-scale graphs. In *EuroSys 2024 (poster track)*.

[4] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[5] Yongchao Liu, Houyi Li, Guowei Zhang, Xintan Zeng, Yongyong Li, Bin Huang, Peng Zhang, Zhao Li, Xiaowei Zhu, Changhua He, and Wenguang Chen. 2023. GraphTheta: A Distributed Graph Neural Network Learning System With Flexible Training Strategy. arXiv:2104.10569

[6] Dalong Zhang, Xin Huang, Ziqi Liu, Jun Zhou, Zhiyang Hu, Xianzheng Song, Zhibang Ge, Lin Wang, Zhiqiang Zhang, and Yuan Qi. 2020. AGL: a scalable system for industrial-purpose graph machine learning. In *VLDB*.