디스크 기반의 대규모 GNN 학습시 파티션 예측과 사전 로딩을 통한 I/O 병목 해결 연구

[1]김진우◦, [1]김기현, [1]김유찬, [2]차명훈, [2]김홍연, [3]김종만, [1]김영재
[1]서강대학교 컴퓨터공학과, [2]한국전자통신연구소, [3](주)소테리아

Resolving I/O Bottlenecks through Partition Prediction and Preloading in Large-Scale Disk-based GNN Training

[1]Jinwoo Kim, [1]Kihyun Kim,[1]Yoochan Kim, [2] Myung-Hoon Cha, [2]Hong-Yeon Kim, [3]Jongman Kim,[1]Youngjae Kim
[1]Sogang University, Seoul, South Korea, [2]ETRI, Daejeon, [3]Soteria Inc.

요 약

이 연구는 대규모 디스크 기반 그래프 신경망(GNN) 학습에서 발생하는 I/O 병목 문제를 해결하기 위해 파티션 예측과 사전 로딩이라는 새로운 접근법을 제안한다. 이 방식은 파티션 간의 의존성을 기반으로 데이터를 효율적으로 로딩하여 I/O 병목을 완화하고, 학습 중 GPU의 유휴 시간을 줄이기 위해 필요한 데이터를 미리 로딩한다. 특히, 병렬 파일 시스템(PFS)이 주요 스토리지로 사용되는 고성능 컴퓨팅(HPC) 환경에서 PFS의 속도 한계로 인해 발생하는 디스크 기반의 대규모 GNN 학습의 I/O 병목 문제를 해결하는 데 이 방법이 효과적임을 제시한다. 실험 결과, 제안된 방법은 HPC 환경에서 평균 학습 시간을 2.28배, CPU 실행 시간을 평균 3.97배 단축시키는 성과를 보였다. 또한, 제안하는 사전로딩 기법의 오버헤드가 학습에 영향을 주지 않음을 실험을 통해 확인하였다.

1. 서론

그래프 신경망(GNN)은 복잡한 그래프 데이터에서 패턴을 추출하고 예측하는 데 효과적인 딥러닝 기법으로, 소셜 네트워크 분석, 화학 구조 예측, 추천 시스템 등 다양한 분야에 적용된다[1, 2]. GNN은 그래프의 노드, 엣지, 그리고 노드의 특징 벡터 간의 상호작용을 모델링하여 성능을 향상시킨다.

Deep Graph Library (DGL) [3]와 PyTorch Geometric (PyG) [4] 같은 대표적인 GNN 프레임워크는 호스트 메모리를 사용해 그래프 데이터를 처리한다. 그러나 [표 1]에서 보듯이, 대규모 그래프 데이터셋을 전부 메모리에 적재하기에는 메모리 용량의 한계가 존재해 성능 저하를 초래할 수 있다. 이를 해결하기 위해 대규모 그래프 데이터셋을 파티셔닝 작업으로 분할하여 디스크에 저장하는 방식을 사용하는 디스크 기반 GNN 학습 방식이 도입되었다 [5, 6].

이러한 대규모 데이터를 효과적으로 처리하기 위해 GNN 학습은 고성능 컴퓨팅(HPC) 환경에서 주로 이루어진다. HPC 환경에서는 병렬 파일 시스템(PFS)과 같은 대용량 스토리지를 사용하지만, PFS는 500MB/s 이하의 속도를 제공하는 반면 NVMe SSD는 1TB/s 이상의 속도를 지원하기 때문에, PFS의 느린 로딩 속도는 학습 중 I/O 병목 현상을 악화시키고, 자원의 비효율적 사용을 초래할 수 있다.

본 논문에서는 I/O 병목 문제를 해결하고 디스크 기반 GNN 학습 성능을 최적화하기 위해 파티션 예측 및 사전 로딩(Preloading) 방식을 제안한다. 이 방식은 파티션 간의 연관성을 분석하여, 학습에 필요한 데이터를 미리 예측하고 CPU가 이를 사전에 로드한 뒤, 학습 중 필요한 시점에 GPU로 실시간 전달한다. 이를 통해 학습과 데이터 로딩을 병행하여 발생할 수 있는 대기 시간을 최소화하고, GPU의 유휴 시간을 줄임으로써 자원 활용도를 극대화한다. 실험 결과, 제안된 방법은 평균 학습 시간을 2.28배, CPU 계산 시간을 3.97배 감소시킴으로써 디스크 기반 GNN 학습 성능을 크게 향상시킬 수 있음을 확인하였다.

표 1: 대규모 그래프 데이터셋 크기

Graph	Nodes	Edges	Feat. Dim	Memory (GB)		
				Edges	Feat.	Tot.
Papers100M	111M	1.62B	128	13	57	70
WikiKG90Mv2	91M	601M	100	7	73	80
Mag240M-Cites	122M	1.30B	768	10	375	385
Facebook15	1.4B	1T	100	8k	560	8.5k

2. 배경 지식 및 연구 동기

2.1 디스크 기반 GNN의 데이터 흐름

디스크 기반 GNN의 학습 프로세스는 스토리지를 포함하여 데이터를 처리하는 방식으로, 일반적으로 여섯 단계로 나뉜다 [5, 6, 7]. ① 학습 전 그래프 데이터를 여러 파티션으로 분할하고, 디스크에 저장한다. ② 학습에 필요한 파티션 데이터를 디스크에서 CPU 메모리로 랜덤 선택 방식으로 로드한다. ③ CPU에 로드된 데이터를 GPU로 전송하여 학습을 준비한다. ④ GPU에서 미니배치(mini-batch) 방식으로 모델 학습을 진행하고, 학습된 임베딩 데이터를 업데이트한다. ⑤ GPU에서 업데이트된 노드 임베딩을 CPU 메모리에서 반영하여 업데이트를 완료한다. ⑥ 학습이 종료될 때까지 ①∼⑤를 반복한다.

이 논문은 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No. 2022-0-00498, 거대 학습 모델 초고속 처리를 위한 고효율 AI 컴퓨팅SW 핵심 기술개발)

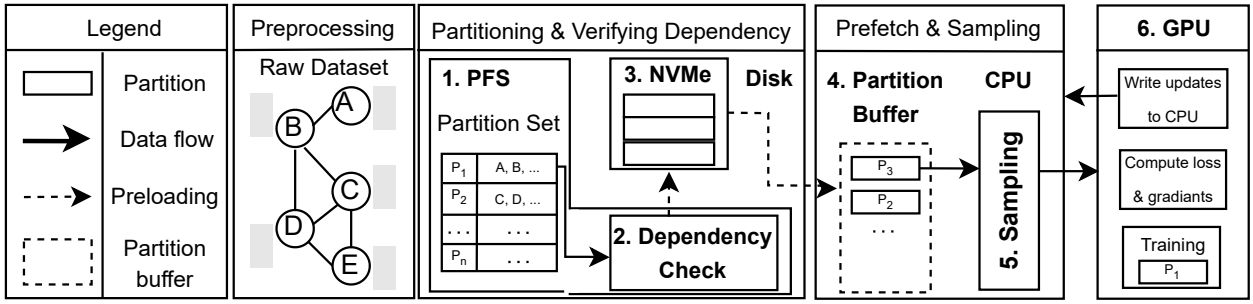

그림 1: HPC환경에서 NVMe에 파티션을 사전로딩하는 아키텍처

2.2 랜덤하게 파티션을 선택하는 방식의 한계

디스크 기반 GNN 프레임워크에서는 대규모 그래프 데이터를 여러 개의 파티션으로 나누어 저장한 후, 학습에 필요한 파티션을 랜덤 선택 방식으로 불러온다. 이로 인해 현재 학습 중인 파티션과 다음에 필요한 파티션 간의 의존성이 충분히 고려되지 않아 필요한 파티션이 제때 준비되지 못할 수 있다. 따라서 GPU는 다음 학습에 필요한 데이터가 메모리에 로드될 때까지 유휴 상태로 대기하게 되며, 결국 자원 활용도가 떨어지고 학습 속도가 저하된다. 특히, 대규모 그래프 데이터셋에서는 노드와 엣지의 수가 방대하여 데이터 로딩에 시간이 더 오래 걸려 병목 현상이 심화된다. 만약 파티션 로딩과 학습이 동시에 진행되지 않으면 자원 사용의 비효율성이 더욱 두드러지며, 결과적으로 전체 학습 성능이 크게 저하된다[7].

3. 파티션 예측 및 사전 로딩

3.1 파티션 예측 및 사전로딩을 통한 GNN의 데이터 흐름

[그림 1]은 PFS와 NVMe를 활용한 HPC 환경에서 대규모 그래프 데이터 학습의 성능 최적화를 위해, PFS에서 NVMe로 다음 학습에 필요한 데이터를 선제적으로 로딩하는 아키텍처이다. ❶ 전처리(Preprocessing) 단계는 기존과 동일하게 그래프를 여러 파티션으로 분할하고, PFS에 저장된다. ❷ [알고리즘 1]을 통해 의존성 검사(Dependency Check)를 하여 현재 학습 중인 파티션과 연관성이 높은 파티션을 식별하고, 다음 학습에 필요한 파티션을 예측한다. 이후 ❸ 에서 선정된 파티션들은 순차적으로 NVMe로 사전로드되며, ❹ 사전로드된 데이터는 Partition Buffer에 프리패칭(Pre-fetching)된다. ❺ Partition Buffer는 이후 CPU가 데이터를 샘플링하여 GPU 학습에 적합한 배치(batch) 형식으로 변환하는 데 사용된다. ❻ GPU 학습 단계에서는 이 배치를 이용해 학습을 진행하고, 임베딩을 업데이트한 후 이를 다시 CPU로 전달한다. 이후, 위 과정들이 학습 종료까지 반복된다. 기존 방식과의 데이터 흐름상 주요 차이점은 현재 학습 중인 파티션과 연관성이 높은 파티션을 식별하여, 다음 학습에 필요한 파티션을 예측하는 것 이다.

3.2 의존성 그래프를 이용한 파티션 예측 사전로딩 알고리즘

먼저 사전 로딩 단계에서 파티션 간 의존성을 분석하기 위해 의존성 그래프 $G_{Dep} = (N_{Dep}, E_{Dep})$를 도입하였다. 원본 그래프 $G = (V, E)$가 주어졌을 때, $N_{Dep} = \{P_0, P_1, ..., P_{n-1}\}$는 파티션의 집합이다. 그리고 두 파티션 P_i와 P_j 간의 간선 $(P_i, P_j) \in E_{Dep}$는 각 파티션에 속한 임의의 두 노드 사이에 연결 관계가 존재할 때 정의된다 (즉, $\exists v \in P_i, u \in P_j : (v, u) \in E$). 이를 기반으로 본 연구에서 제안하는 파티션 예측 및 사전 로딩 기법을 [알고리즘 1]로 표현하였다.

Algorithm 1 Partition Preloading with Dependency Check

Input: Partitions $P = \{P_1, \ldots, P_n\}$, edge sets E_i for each partition P_i
Output: Efficient preloading of partitions for training based on dependency graph

for each $P_i \in P$ **do**
  Define edges $E_i = \{e_{i1}, e_{i2}, \ldots\}$ and nodes N_i associated with P_i
end for
Initialize dependency graph G_{dep} as an empty graph
for each $P_i \in P$ **do**
  for each edge $e_{ik} \in E_i$ connecting nodes u and v **do**
    if node v belongs to partition P_j and $i \neq j$ **then**
      $G_{dep}(P_i, P_j) = \text{True}$
      Add P_j to the dependency list of P_i
    end if
  end for
end for
for each P_i in the training sequence **do**
  Start training on P_i
  for each dependent P_j where $G_{dep}(P_i, P_j) = \text{True}$ **do**
    if P_j not yet loaded to NVMe **then**
      Preload P_j from PFS to NVMe storage
    end if
  end for
end for

4. 실험 결과 및 분석

4.1 실험 환경

실험은 [표 2]에 정리된 실험환경을 가진 뉴런(Neuron) 고성능 컴퓨팅 시스템(HPC)에서 수행되었으며, 메모리 용량을 데이터셋 크기의 80%로 제한하여 메모리에서 데이터셋 전체를 처리하지 못하는 조건을 설정하였다. 또한, 데이터가 NVMe용량을 초과할 경우, 추가 데이터를 로딩하기 위해 학습이 완료된 파티션 데이터를 삭제하는 전략을 설정했다. 파티션 예측과 사전 로딩 기법이 다양한 데이터셋 크기와 특성에서 성능을 어떻게 향상시키는지 측정하기 위해 다양한 크기(Small, Medium, Large)와 특징(Knowledge Graph, Social)을 가진 데이터셋들을 사용하였다.

표 2: 실험환경

Processor	AMD EPYC 7543 / 2.80GHz (32-core) / 2 socket
Main Memory	1TB DDR4 Memory
GPU	Tesla A100 80 GB
PFS	HPE ClusterStor E1000 Lustre 2.12.6
SSD	NVMe SSD 30 TB

표 3: 실험에 사용된 데이터셋

Dataset	type	# of nodes	# of edges	Size (GB)
Reddit	KG	232K	11M	0.33
LiveJournal	social	4M	68M	1.9
Twitter	social	66M	1.75B	33.2
OGBN-100M	KG	111M	1.62B	142

표 4: 데이터 로딩 위치에서의 평균 학습 시간 비교 (Minutes)

구 분	Reddit	LiveJournal	Twitter	Papers100M
PFS	1.51	7.58	23.4	80.4
NVMe	0.54	3.15	11.4	35.4
PFS+Preloading	0.55	3.21	12.6	37.8

[표 3]은 사용된 데이터셋의 타입과 크기를 나타낸다.

4.2 결과 및 분석

파티션 사전 로딩의 성능을 검증하기 위해 PFS, NVMe, 그리고 PFS에서 NVMe로 Preloading하는 세 가지 데이터 로딩 위치에 따른 성능을 실험하였다. 그 결과로 [그림 2]에서 볼 수 있듯이, PFS+Preloading 방식을 사용했을 때 학습 시간이 PFS 대비 평균 2.28배 감소하는 결과를 보였다. 또한, 학습 중 다음 파티션을 미리 로딩함으로써 CPU 실행 시간도 평균 3.97배 단축되었다. 그러나 상대적으로 많은 엣지를 포함하고 있어 파티셔닝 과정이 오래 걸리는 Twitter 데이터셋의 경우, CPU 실행 시간 감소 폭이 2.75배로 다른 세 데이터셋의 평균인 4.38배와 큰 차이가 있었다.

[표 4]는 데이터 로딩 위치에 따른 평균 학습 시간을 비교한 표이다. Papers100M 데이터셋을 기준으로 PFS와 NVMe의 평균 학습 시간 차이는 44%인 반면, PFS+Preloading과 NVMe의 평균 학습 시간 차이는 8.25%로 나타났다. 이를 통해 PFS에 Preloading을 적용했을 때 성능이 크게 향상된 것을 확인할 수 있었다.

[그림 3]에서 확인할 수 있듯이, 세 가지 GNN 모델 (GraphSage, GCN, GAT)을 social 타입 LiveJournal 데이터셋 [그림 3 (a)]과 같은 타입이지만 파티셔닝 과정이 오래 걸리는 Twitter 데이터셋 [그림 3 (b)]에 적용한 결과, GPU 학습 시간이 Preloading 시간보다 길어 Preloading 오버헤드가 GPU 학습 성능에 영향을 미치지 않음을 확인하였다. 이는 Preloading 전략이 I/O 병목을 효과적으로 완화하면서 GPU 자원을 최대한 활용하도록 해주었음을 보여준다.

5. 결론

본 연구에서는 디스크 기반 GNN 학습에서 다음에 필요한 파티션을 예측하고 이를 사전에 로딩하여 학습 성능을 최적화하는 방법을 제안하였다. 실험 결과, 파티션 사전 로딩을 통해 CPU 실행 시간이 크게 감소함에 따라 전체 평균 학습 시간이 유의미하게 단축되었다. 특히, GPU 학습 시간이 Preloading 시간보다 길어, Preloading 오버헤드가 GPU 학습 시간에 가려지며 전체 성능에

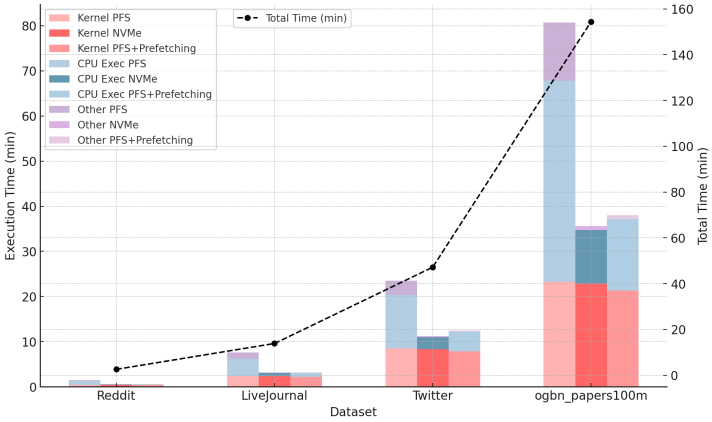

그림 2: 서로 다른 크기와 특성을 가진 데이터셋에서 데이터 로딩 방법과 위치에 따른 평균 학습 시간 비교 (epochs=5)

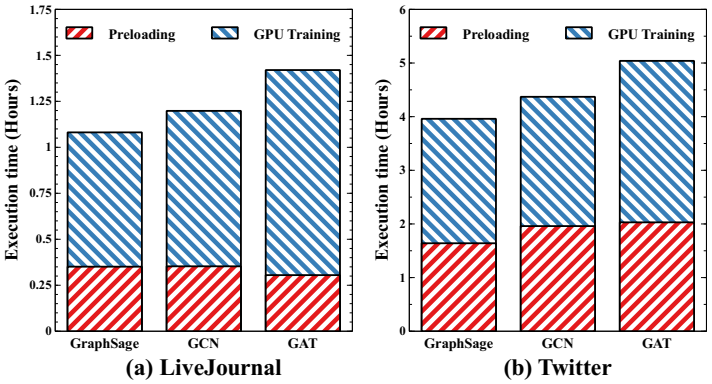

그림 3: Preloading과 GPU 학습의 Overlap 실험 결과(epochs=20)

부정적인 영향을 미치지 않는다는 점을 확인하였다. 향후 연구에서는 다양한 GNN 모델과 데이터셋에 대한 실험을 확장하여 일반화된 성능을 검증하고, 파티션 로딩과 GPU 학습 간의 동시성을 극대화할 수 있는 방안을 탐구할 예정이다.

참고 문헌

[1] I. F. Ilyas, T. Rekatsinas, V. Konda, J. Pound, X. Qi, and M. Soliman, "Saga: A platform for continuous construction and serving of knowledge at scale," in *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, p. 2259–2272, 2022.

[2] S. Maass, C. Min, S. Kashyap, W. Kang, M. Kumar, and T. Kim, "Mosaic: Processing a trillion-edge graph on a single machine," EuroSys '17, p. 527–543, 2017.

[3] M. Y. Wang, "Deep graph library: Towards efficient and scalable deep learning on graphs," in *ICLR workshop on representation learning on graphs and manifolds*, 2019.

[4] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[5] J. Mohoney, R. Waleffe, H. Xu, T. Rekatsinas, and S. Venkataraman, "Marius: Learning massive graph embeddings on a single machine," in *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, pp. 533–549, 2021.

[6] R. Waleffe, J. Mohoney, T. Rekatsinas, and S. Venkataraman, "Mariusgnn: Resource-efficient out-of-core training of graph neural networks," in *Proceedings of the Eighteenth European Conference on Computer Systems*, pp. 144–161, 2023.

[7] Y. Li, T.-Y. Yang, M.-C. Yang, Z. Shen, and B. Li, "Celeritas: Out-of-core based unsupervised graph neural network via cross-layer computing 2024," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 91–107, 2024.