SmartSAGE: Training Large-scale Graph Neural Networks using In-Storage Processing Architectures

KAIST

Yunjae Lee, Jinha Chung, and Minsoo Rhu
Research Scope
Emerging ML algorithms?
Graph neural networks (GNNs) widely being utilized in many application domains
Proliferation of GNNs
Active area of research to develop new ML algorithms and ML frameworks for GNNs

* Hamilton et al., "Inductive Representation Learning on Large Graphs", NIPS-2017
* Chen et al., "FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling", ICLR-2018
* Zeng et al., "GraphSAINT: Graph Sampling Based Inductive Learning Method", ICLR-2020
* Wang et al., "Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks", arXiv-2019
* Fey et al., "Fast Graph Representation Learning with PyTorch Geometric", ICLR-2019
Proliferation of GNNs
Active area of research to develop new ML algorithms and ML frameworks for GNNs

Current ML frameworks cannot fully support large-scale GNN training due to their in-memory processing model

* Hamilton et al., "Inductive Representation Learning on Large Graphs", NIPS-2017
* Chen et al., "FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling", ICLR-2018
* Zeng et al., "GraphSAINT: Graph Sampling Based Inductive Learning Method", ICLR-2020
* Wang et al., "Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks", arXiv-2019
* Fey et al., "Fast Graph Representation Learning with PyTorch Geometric", ICLR-2019
Key challenges of GNN training

Memory capacity limitation with scaled-up graph nodes and edges

“\sim100s of billions scale”

Data size
Key challenges of GNN training

Memory capacity limitation with scaled-up graph nodes and edges

ML practitioners must “tune” their algorithms to fit within main memory
System Architecture for “In-memory” Training Model
Existing systems for GNN training

CPU-GPU approach: data preparation on CPU and GNN training on GPU
Existing systems for GNN training

**CPU-GPU** approach: data preparation on CPU and GNN training on GPU
Existing systems for GNN training

**CPU-GPU** approach: data preparation on CPU and GNN training on GPU

- **CPU**
  - Capacity-limited
  - Neighbor edge list

- **Feature table**

- **GPU**
  - GNN training
  - Feature A
    - …
Data preparation on CPU

Goal: generating subgraphs to fit into capacity-limited GPU memory
Data preparation on CPU

#1. Neighbor sampling from the neighbor edge list
Data preparation on CPU

#2. Feature table lookup from the entire feature table
Data preparation on CPU

#2. Feature table lookup from the entire feature table
Data preparation on CPU

#3. CPU-GPU data copy over PCIe
GNN training on GPU
Aggregation and DNN computation

Subgraph generation

CPU

GPU

Subgraph

Feature A

Feature A

CPU-GPU transfer

Aggregated feature

DNNs
Key challenges of GNN training

Facing a dilemma between memory capacity and performance

"~100s of billions scale"

ML practitioners "tune" their algorithms to fit within CPU DIMMs
Key challenges of GNN training

Facing a dilemma between memory capacity and performance

"~100s of billions scale"

ML practitioners “tune” their algorithms to fit within CPU DIMMs

DRAM

NVMs SSD

DRAM, DRAM, DRAM, DRAM

NAND, NAND, NAND
Key challenges of GNN training
Facing a dilemma between memory capacity and performance

Our research question:
How do we “bridge” this wide performance gap?

ML practitioners “tune” their algorithms to fit within CPU DIMMs
System Architecture for "Large-scale" GNN Training
The memory wall for large-scale GNN training
Utilize main memory as a fast cache (OS page cache) via memory-mapped file
Characterization on SSD-based system
Baseline SSD-based system incurs an average 9.8x slowdown vs. in-memory
Characterization on SSD-based system
Baseline SSD-based system incurs an average 9.8x slowdown vs. in-memory
Characterization on SSD-based system
Baseline SSD-based system incurs an average 9.8x slowdown vs. in-memory
Characterization on SSD-based system
Baseline SSD-based system incurs an average 9.8x slowdown vs. in-memory
Characterization on SSD-based system

Neighbor sampling has an irregular dataflow with little compute intensity

1-hop neighbor sampling

2-hop neighbor sampling

"Random" selection
Characterization on SSD-based system
The merits of OS page cache are outweighed by the high latency overheads
SmartSAGE: Training Large-scale Graph Neural Networks using In-Storage Processing (ISP)
Hardware acceleration using ISP
Identifying key intuition of neighbor sampling operator on SSD-based system
Hardware acceleration using ISP
Identifying key intuition of neighbor sampling operator on SSD-based system
Hardware acceleration using ISP
Identifying key intuition of neighbor sampling operator on SSD-based system

1. SSD→CPU transfer of neighbor node IDs in block granularity chunks
Hardware acceleration using ISP

Identifying key intuition of neighbor sampling operator on SSD-based system

1. SSD→CPU transfer of neighbor node IDs in block granularity chunks

2. CPU randomly samples from CPU memory
Hardware acceleration using ISP

Identifying key intuition of neighbor sampling operator on SSD-based system

1. SSD→CPU transfer of neighbor node IDs in block granularity chunks

2. CPU randomly samples from CPU memory

Wastes I/O bandwidth
"In-storage" neighbor sampling
Offloading neighbor sampling operator inside SSD

1. Conduct in-storage neighbor sampling directly

2. SSD→CPU transfer of sampled node IDs

"Amplified effective throughput"
**Two types of ISP architecture**

FPGA-based vs. firmware-based computational storage device

---

**FPGA-based**

- CPU
- Memory
- System interconnect (PCIe)
- **Integrated device**
  - PCIe switch
  - SSD
  - FPGA

**Peer-to-peer (P2P)**

**Firmware-based**

- CPU
- Memory
- System interconnect (PCIe)
- **Embedded core**
  - DRAM
  - NAND
Two types of ISP architecture

FPGA-based vs. firmware-based computational storage device

FPGA-based

Firmware-based

1. SSD
2. FPGA
3. PCIe switch

CPU → Memory

System interconnect (PCIE)

Embedded core

DRAM

NAND
Two types of ISP architecture

FPGA-based vs. “firmware-based” computational storage device

FPGA-based

Firmware-based

System interconnect (PCIe)

Embedded core

DRAM

NAND

P2P overhead overkills benefits of ISP acceleration
Software architecture of SmartSAGE
Software/hardware co-design based on our ISP architecture
Software architecture of SmartSAGE

Software/hardware co-design based on our ISP architecture
Software architecture of SmartSAGE
The baseline SSD-based system suffers from high latency overheads

Significant misses due to irregular dataflow
"Latency-optimized" runtime and host driver
Bypass the opportunistic OS page cache

Diagram:
- Mmap-based
  - Application
  - OS page cache
- Direct I/O-based
  - Application
  - Buffer
- NVMe device driver
- SmartSAGE
  - Embedded core
  - DRAM
  - NAND
"Latency-optimized" runtime and host driver

Coalesce multiple I/O commands within a given subgraph
"Latency-optimized" runtime and host driver

Coalesce multiple I/O commands within a given subgraph

- Mmap-based
  - Application
    - OS page cache
  - NVMe device driver

- Direct I/O-based
  - Application
    - Buffer
  - SmartSAGE driver

- SmartSAGE
  - Embedded core
  - DRAM
  - NAND

A single "encapsulated" NVMe command

Direct I/O (as-is)
- I/O req for node 1
- I/O req for node 7
- I/O req for node 10

Coalesced I/O req
Evaluation
Evaluation methodology
Implementation details and graph datasets

- Hardware/software platform
  - Cosmos+ OpenSSD platform
  - PyTorch Geometric framework

- Public graph datasets
  - Reddit, Movielens, Amazon, OGBN-papers100M, Protein-protein interaction
Evaluation methodology
System configurations explored

- Four system design points
  - SSD (mmap): baseline
  - SmartSAGE (SW): direct I/O without ISP
  - SmartSAGE (HW/SW): direct I/O with ISP + I/O command coalescing
  - DRAM: upper bound design point
Latency breakdown

Normalized to the baseline SSD-based system
Latency breakdown

Normalized to the baseline SSD-based system
Latency breakdown
Reducing neighbor sampling latency by bypassing OS page cache (average 3x)

|          | Neighbor sampling | Feature lookup | CPU-to-GPU transfer | GNN training | Else |
|----------|-------------------|----------------|---------------------|--------------|------|
| SSD (mmap) | 1.00             | 0.00           | 0.00                | 0.00         | 0.00 |
| SmartSAGE (HW/SW) | 0.80       | 0.20           | 0.00                | 0.00         | 0.00 |
| DRAM     | 0.60             | 0.40           | 0.00                | 0.00         | 0.00 |

Latency breakdown (normalized)
Latency breakdown
Further performance improvement (average 1.5x) due to ISP acceleration
Latency breakdown
Still average 2.7x slowdown vs. in-memory due to low ISP compute power
Latency breakdown

Still average 2.7x slowdown vs. in-memory due to low ISP compute power

OpenSSD’s wimpy embedded cores get *overwhelmed* in delivering sufficient levels of ISP compute power
Alternative ISP system configuration
A hypothetical design point with higher ISP compute power than OpenSSD

- SmartSAGE (oracle): contains ISP-purposed embedded cores like NGD system’s Newport*
  - Newport contains a quad core ARMv8 Cortex-A53 solely dedicated for ISP purposes

* Do et al., "Cost-Effective, Energy-Efficient, and Scalable Storage Computing for Large-Scale AI Applications", ACM Transactions on Storage (2020)
Latency breakdown

SmartSAGE (oracle) achieves only average 1.3x slowdown vs. in-memory
SmartSAGE: Training Large-scale Graph Neural Networks using In-Storage Processing Architectures

A **detailed** characterization on the **data-intensive** data preparation stage of large-scale GNN training

Explores the **viability** of exploiting NVMe SSDs as a substitute for capacity-limited in-memory GNN training model

Achieves more than “**3.5x**” performance improvement over the baseline SSD-based system
Questions?