CS 744: MARIUS

Shivaram Venkataraman
Fall 2022
- Midterm grades out! → 3pm today?
- Regrade requests: In-person (strongly preferred)
  - Thu: After class, Roger’s OH
  - Mon: Shivaram’s OH, Roger’s OH
  - Tue: After class
- Course Project: Check in by Nov 23th
One page document that includes the following:

- What have you done so far
- Any challenges that you have faced so far
- Your timeline (from now till end of the semester)
- Things you need help from the course staff
- Any other comments/remarks

Are you on track or need help?
EXAMPLE: LINK PREDICTION

Task: Predict potential connections in a social network

friend recommendation

Typed edge

Machine learning?

vector representation → embedding

Find K-nearest neighbors

image, label

... 

Learn embeddings
* Capture "graph structure"

[0.25, 0.45, 0.30]

d = 3 / d = 50 or 100

[0.15, 0.85, 0.92]
Score function

Capture structure of the graph given source, destination embedding

Loss function

Maximize score for edges in graph
Minimize for others (negative edges)

\[ \mathcal{L} = \sum_{e \in G} \sum_{e' \in S_e} \max(f(e) - f(e') + \lambda, 0) \]

If two vertices are "close" 
Distance \rightarrow \text{Euclidean cosine similarity}

\text{Distance function}
TRAINING ALGORITHM

SGD/AdaGrad optimizer

Sample positive, negative edges

Access source, dest embeddings for each edge in batch

Model = n x d matrix → Sparse updates to the model

1 epoch training = all the edges in the graph

for i in range(num_batches)

B = getBatchEdges(i)

E = getEmbeddingParams(B)

G = computeGrad(E, B)

updateEmbeddingParams(G)

Sparse updates to the model

n x d matrix

n vertices

d - dimension

initial random

gradient

compute the score

so that you can

lots of random accesses to the model

 lots of

7
CHALLENGE: LARGE GRAPHS

Large graphs → Large model sizes

Example

3 Billion vertices, $d = 400$
Model size $= 3 \text{ billion} \times 400 \times 4 = 4.8 \text{ TB}!$

Need to scale beyond GPU memory, CPU memory!

Reset $\sim 50 - 100 \text{ MB}$
BERT $\sim 500 - 1 \text{ GB}$
CHALLENGE: DATA MOVEMENT

DGL-KE: Sample edges, embeddings from CPU memory → random access, PCIe for every iteration

Pytorch-BigGraph: Partition embeddings so that one partition fits on GPU memory. Load sequentially

One epoch on the Freebase86m knowledge graph
MARIUS

I/O efficient system for learning graph embeddings

Marius Design
- Pipelined training
- Partition ordering
PIPELINED TRAINING

5 stage pipeline

While $b_0$ is being transferred
While $b_2$ is being loaded

If $b_0$ has any embeddings it is in common with $b_{25}$ then load will be stale!

Pipeline is as fast as slowest stage
- Queues provide back pressure

→ Bounded staleness, sparse updates
OUT OF MEMORY TRAINING

Key idea: Maintain a cache of partitions in CPU memory

Questions
Order of partition traversal?
How to perform eviction?

Partitions in Buffer

Partitions on disk

Eviction

$\Theta_0 \Theta_1 \Theta_2 \Theta_3 \Theta_4 \Theta_5$

$c = 3$

$\Theta_0 \Theta_1 \Theta_2 \Theta_3 \Theta_4 \Theta_5$

$p = 6$

$\rightarrow$ Embedding Table

$\rightarrow$ CPU Memory
BETA ORDERING

**Goal:** Min number of disk swaps

Initialize cache with c partitions

Swap in partition that leads to highest number of unseen pairs

Achieved by fixing c-1 partitions and swap remaining in any order

*Intuition*

\[
\begin{array}{ccc}
0 & 1 & 2 \\
0 & 1 & 3 \\
0 & 1 & 4 \\
0 & 1 & 5 \\
\end{array}
\]

\[
\begin{array}{ccc}
0 & 1 & 2 \\
0 & 1 & 3 \\
0 & 1 & 4 \\
0 & 1 & 5 \\
\end{array}
\]

\[
\begin{array}{ccc}
0 & 1 & 2 \\
0 & 1 & 3 \\
0 & 1 & 4 \\
0 & 1 & 5 \\
\end{array}
\]

\[
\begin{array}{ccc}
0 & 1 & 2 \\
0 & 1 & 3 \\
0 & 1 & 4 \\
0 & 1 & 5 \\
\end{array}
\]

*Cache contents*

- very fast to get embeddings for edges with src/dst in \{0, 1, 2\}

*At every swap*

- Keep c-1 fixed and cycle through rest

- Reinitialize cache and repeat
Graph Embeddings: Learn embeddings from graph data for ML

Marius: Efficient single-machine training
  Pipelining to use CPU, GPU
  Partition buffer, BETA ordering
DISCUSSION

https://forms.gle/uNAKsPsZp56CclVz9
How does the partitioning scheme used in this paper differ from partitioning schemes used in PowerGraph and why?

Marius → batch of edges → get src/dest embedding → sparse access into key access

Power graph → Page Rank → nbr vertices → minimize number of remote reads for nbrs

→ Multiple machines

1-GPU

disk
<table>
<thead>
<tr>
<th>System</th>
<th>Deployment</th>
<th>Epoch Time (s)</th>
<th>Per Epoch Cost ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marius</td>
<td>1-GPU</td>
<td>727</td>
<td>0.61</td>
</tr>
<tr>
<td>DGL-KE</td>
<td>2-GPUs</td>
<td>1068</td>
<td>1.81</td>
</tr>
<tr>
<td>DGL-KE</td>
<td>4-GPUs</td>
<td>542</td>
<td>1.84</td>
</tr>
<tr>
<td>DGL-KE</td>
<td>8-GPUs</td>
<td>277</td>
<td>1.88</td>
</tr>
<tr>
<td>DGL-KE</td>
<td>Distributed</td>
<td>1622</td>
<td>2.22</td>
</tr>
<tr>
<td>PBG</td>
<td>1-GPU</td>
<td>3060</td>
<td>2.6</td>
</tr>
<tr>
<td>PBG</td>
<td>2-GPUs</td>
<td>1400</td>
<td>2.38</td>
</tr>
<tr>
<td>PBG</td>
<td>4-GPUs</td>
<td>515</td>
<td>1.75</td>
</tr>
<tr>
<td>PBG</td>
<td>8-GPUs</td>
<td>419</td>
<td>2.84</td>
</tr>
<tr>
<td>PBG</td>
<td>Distributed</td>
<td>1474</td>
<td>2.02</td>
</tr>
</tbody>
</table>

If you use 8 GPUs, you get more memory to fit the embeddings, more compute speedup, and better cost/perform with higher utilization and also expensive. Multiple CPU machines work better.

Not the fastest but lowest per epoch cost.
What are some shortcomings of Marius? What could the authors do to further improve the system?

- BETA ordering
- take into account differences
- load-aware scheduling
- latency is higher with pipelining?
NEXT STEPS

Next class: Distributed GNNs
Project check-ins by Nov 23th