LLAMA: Efficient graph analytics using Large Multiversioned Arrays

Presenters:
Bryan Wilder, Ethan Kripke, Zach Dietz
Background
Big-Data Graph Analytics Applications

Big-Data Graph Analytics Applications

FriendBook

Don't Not Not Be Not Evil™
Design Considerations

Compressed Sparse Row

Compressed Sparse Bitmaps

Adjacency Lists
Design Considerations

Compressed Sparse Row

- Mutability
- Multi-versioning
- In-Memory vs. Out of Memory
- Overhead
Compressed Sparse Row (CSR)

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
[4] \\
[2, 8, 1] \\
[] \\
[1, 3]
\end{align*}
\]
Compressed Sparse Row (CSR)

<table>
<thead>
<tr>
<th>Value</th>
<th>Cum. NNZ</th>
<th>Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0]</td>
<td>[1]</td>
<td>[4]</td>
</tr>
<tr>
<td>4</td>
<td>[1, 1]</td>
<td>[2]</td>
<td>[2]</td>
</tr>
<tr>
<td>0</td>
<td>[4, 1]</td>
<td>[3]</td>
<td>[8]</td>
</tr>
<tr>
<td>0</td>
<td>[4, 2]</td>
<td>[3]</td>
<td>[0]</td>
</tr>
<tr>
<td>0</td>
<td>[4, 3]</td>
<td>[3]</td>
<td>[0]</td>
</tr>
<tr>
<td>0</td>
<td>[4, 4]</td>
<td>[3]</td>
<td>[0]</td>
</tr>
<tr>
<td>0</td>
<td>[4, 5]</td>
<td>[3]</td>
<td>[0]</td>
</tr>
</tbody>
</table>

(implicit: row)
Compressed Sparse Row (CSR)

\[ \begin{array}{ccc}
0 & 1 & 1 \\
0 & 0 & 1 \\
1 & 0 & 1 \\
\end{array} \]

\( \text{NNZ} \)

\[ [0, 2, 3, 5] \]

\( \text{Col} \)

\[ [1, 2, 2, 0, 2] \]
Compressed Sparse Row (CSR)

Vertex Table

Edge Table

\[
\begin{bmatrix}
0 & 1 & 1 \\
0 & 0 & 1 \\
1 & 0 & 1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0, 2, 3, 5 \\
1, 2, 2, 0, 2 \\
\end{bmatrix}
\]
Implementation
**Vertex Table**

<table>
<thead>
<tr>
<th>Snapshot</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex 0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Vertex 1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Vertex 2</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Vertex 3</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

**Edge Table**

- **Edge Property**
  - Blue
  - Red
  - Green
  - Blue

- **Vertex Property**
  - Tall
  - Short
  - Skinny
  - Short

**Indirection Array**

- Page 0: Snapshot 0
  - Vertex 0: Offset: 0, Length: 2
  - Vertex 1: Offset: 2, Length: 1
  - Vertex 2: Offset: 3, Length: 0
  - Vertex 3: Offset: 3, Length: 1

- Page 1: Snapshot 0
  - Vertex 0: Offset: 0, Length: 2
  - Vertex 1: Offset: 2, Length: 1
  - Vertex 2: Offset: 3, Length: 0
  - Vertex 3: Offset: 3, Length: 1

**Indirection Array**

- Page 0: 0-1
- Page 1: 2-3

**Multiple of file system & virtual mem page size**

Multiple of file system & virtual mem page size
Indirection Table

Vertex Table

Edge Table

Vertex Table

Snapshot 0 | 0 | 1 | 2 | 3
Page 0

Snapshot 1 | 0 | 1
Page 0'

Continuation: NONE

Continuation: Snapshot 0, Offset 2

Rule for merging?
Using LLAMA

Exposes three operations to user:
   Iterate over vertices
   Select a specific vertex
   Iterate over neighbors of a vertex

Contrast: does **not** enforce sequential access pattern

( Unlike GraphChi or X-stream)
Experimental Evaluation
Tasks

PageRank computation

Breadth-first search

Triangle counting
Tasks

PageRank computation
Breadth-first search
Triangle counting

Datasets

Large synthetic graphs (R-MAT)
Large, publicly available social graphs
(Twitter, Livejournal)
Tasks

PageRank computation
Breadth-first search
Triangle counting

Datasets

Large synthetic graphs (R-MAT)
Large, publicly available social graphs (Twitter, Livejournal)

In memory vs out of memory
PageRank, in-memory

![Bar chart comparing runtime (s) for LLAMA, GraphLab, GreenMarl, GraphChi, and X-Stream in memory and out of memory.](image)

- LLAMA
- GraphLab
- GreenMarl
- GraphChi
- X-Stream

**In memory**

**Out of memory**
PageRank, out-of-memory
Runtime breakdown

Enormous variation!

LLAMA minimizes overhead
How is this possible? What features of LLAMA’s design essentially eliminate overhead like buffer management?
Overhead of snapshots

GraphChi (for reference)

LLAMA

Merges are IO-bound

PageRank

Time to merge snapshots
Scalability in number of cores (PageRank)
Experimental evaluation

Compare to **in-memory** and **out-of-memory** systems

(\texttt{GreenMarl} and \texttt{GraphLab}, \texttt{GraphChi} and \texttt{X-stream})

Evaluate **multi-version** support

How much overhead from snapshots?

Evaluate **scalability** on different datasets
Experimental takeaways

LLAMA performs well both in and out of memory

Significantly reduces existing overhead

Introduces manageable additional overhead
Thank you!
How can we store variable length properties?

Can we create a hybrid between a row- and column-store with the storage of properties?