Counting Triangles &
The Curse of the Last Reducer

Siddharth Suri
Sergei Vassilvitskii
Yahoo! Research
Why Count Triangles?
Why Count Triangles?

Clustering Coefficient:

Given an undirected graph \( G = (V, E) \)

\[
cc(v) = \text{fraction of } v's \text{ neighbors who are neighbors themselves}
\]

\[
= \frac{|\{(u, w) \in E | u \in \Gamma(v) \land w \in \Gamma(v)\}|}{\binom{d_v}{2}}
\]
Clustering Coefficient:

Given an undirected graph $G = (V, E)$

\[ cc(v) = \text{fraction of } v\text{'s neighbors who are neighbors themselves} \]

\[ = \frac{|\{(u, w) \in E | u \in \Gamma(v) \land w \in \Gamma(v)\}|}{\binom{d_v}{2}} \]

- $cc(\bullet) = N/A$
- $cc(\circ) = 1/3$
- $cc(\text{red}) = 1$
- $cc(\text{black}) = 1$
Why Count Triangles?

Clustering Coefficient:

Given an undirected graph $G = (V, E)$

$cc(v) = \text{fraction of } v\text{'s neighbors who are neighbors themselves}$

$= \frac{|\{(u, w) \in E | u \in \Gamma(v) \land w \in \Gamma(v)\}|}{\binom{d_v}{2}} = \frac{\#\Delta s \text{ incident on } v}{\binom{d_v}{2}}$

$cc (\blue) = \text{N/A}$

$cc (\red) = \frac{1}{3}$

$cc (\green) = 1$

$cc (\black) = 1$
Why Clustering Coefficient?

Captures how tight-knit the network is around a node.

\[ \text{cc (red node) } = 0.5 \]

\[ \text{cc (blue node) } = 0.1 \]
Why Clustering Coefficient?

Captures how tight-knit the network is around a node.

\[ \text{cc (red) } = 0.5 \]  
\[ \text{cc (blue) } = 0.1 \]

Network Cohesion:
- Tightly knit communities foster more trust, social norms. [Coleman ’88, Portes ’88]

Structural Holes:
- Individuals benefit from bridging [Burt ’04, ’07]
Why MapReduce?

De facto standard for parallel computation on large data
- Widely used at: Yahoo!, Google, Facebook,
- Also at: New York Times, Amazon.com, Match.com, ...
- Commodity hardware
- Reliable infrastructure

- Data continues to outpace available RAM!
How to Count Triangles

Sequential Version:

```python
foreach v in V
    foreach u, w in Adjacency(v)
        if (u, w) in E
            Triangles[v]++
```

```
Triangles[v]=0
```
How to Count Triangles

Sequential Version:

\[
\text{foreach } v \text{ in } V \\
\text{foreach } u,w \text{ in } \text{Adjacency}(v) \\
\quad \text{if } (u,w) \text{ in } E \\
\quad \text{Triangles}[v]++
\]

Triangles[v]=1
Sequential Version:

```python
foreach v in V
  foreach u,w in Adjacency(v)
    if (u,w) in E
      Triangles[v]++
```

Triangles[v] = 1
How to Count Triangles

Sequential Version:

foreach v in V
    foreach u,w in Adjacency(v)
        if (u,w) in E
            Triangles[v]++

Running time: $\sum_{v\in V} d_v^2$

Even for sparse graphs can be quadratic if one vertex has high degree.
Parallel Version

Parallelize the edge checking phase
Parallelize the edge checking phase

- Map 1: For each $v$ send $(v, \Gamma(v))$ to single machine.
- Reduce 1: Input: $(v; \Gamma(v))$
  Output: all 2 paths $\langle (v_1, v_2); u \rangle$ where $v_1, v_2 \in \Gamma(u)$
  
  \[(\text{\#}, \text{\#}); \text{\#} \quad (\text{\#}, \text{\#}); \text{\#} \quad (\text{\#}, \text{\#}); \text{\#} \]
Parallel Version

Parallelize the edge checking phase

- Map 1: For each \( v \) send \( (v, \Gamma(v)) \) to single machine.
- Reduce 1: Input: \( \langle v; \Gamma(v) \rangle \)
  Output: all 2 paths \( \langle (v_1, v_2); u \rangle \) where \( v_1, v_2 \in \Gamma(u) \)
  \((\bullet, \bullet); \color{red}{\bullet} \) \quad \((\bullet, \color{green}{\bullet}); \color{red}{\bullet} \) \quad \((\color{green}{\bullet}, \bullet); \color{red}{\bullet} \)

- Map 2: Send \( \langle (v_1, v_2); u \rangle \) and \( \langle (v_1, v_2); \$ \rangle \) for \( (v_1, v_2) \in E \) to same machine.
- Reduce 2: input: \( \langle (v, w); u_1, u_2, \ldots, u_k, \$ \rangle \)
  Output: if \$ \ part of the input, then: \( u_i = u_i + \frac{1}{3} \)
  \((\color{green}{\bullet}, \bullet); \color{red}{\bullet}, \$ \) \quad \rightarrow \quad \color{red}{\bullet} + \frac{1}{3} \quad \color{green}{\bullet} + \frac{1}{3} \quad \bullet + \frac{1}{3}
  \((\bullet, \bullet); \color{red}{\bullet} \) \quad \rightarrow
How much parallelization can we achieve?

- Generate all the paths to check in parallel
- The running time becomes $\max_{v \in V} d_v^2$
Data skew

How much parallelization can we achieve?
- Generate all the paths to check in parallel
- The running time becomes \( \max_{v \in V} d_v^2 \)

Naive parallelization does not help with data skew
- Some nodes will have very high degree
- Example. 3.2 Million followers, must generate 10 Trillion (10^{13}) potential edges to check.
- Even if generating 100M edges per second, 100K seconds ~ 27 hours.
“Just 5 more minutes”

Running the naive algorithm on LiveJournal Graph

- 80% of reducers done after 5 min
- 99% done after 35 min
Adapting the Algorithm

Approach 1: Dealing with skew directly
- Currently every triangle counted 3 times (once per vertex)
- Running time quadratic in the degree of the vertex
- Idea: Count each once, from the perspective of lowest degree vertex
- Does this heuristic work?
Adapting the Algorithm

Approach 1: Dealing with skew directly
- currently every triangle counted 3 times (once per vertex)
- Running time quadratic in the degree of the vertex
- Idea: Count each once, from the perspective of lowest degree vertex
- Does this heuristic work?

Approach 2: Divide & Conquer
- Equally divide the graph between machines
- But any edge partition will be bound to miss triangles
- Divide into overlapping subgraphs, account for the overlap
Sequential Version [Schank ’07]:

```plaintext
foreach v in V
    foreach u,w in Adjacency(v)
        if deg(u) > deg(v) && deg(w) > deg(v)
            if (u,w) in E
                Triangles[v]++
```

![Diagram of a graph with a triangle and other connections]
Does it make a difference?

Distribution of Reducer Completion Times

Runtime (minutes)

Number ofReducers

0.95 1.00 1.05 1.10 1.15 1.20 1.25 1.30
Dealing with Skew

Why does it help?

- Partition nodes into two groups:
  - Low: $L = \{ v : d_v \leq \sqrt{m} \}$
  - High: $H = \{ v : d_v > \sqrt{m} \}$

- There are at most $n$ low nodes; each produces at most $O(m)$ paths
- There are at most $2\sqrt{m}$ high nodes
  - Each produces paths to other high nodes: $O(m)$ paths per node
Why does it help?
- Partition nodes into two groups:
  • Low: \( \mathcal{L} = \{v : d_v \leq \sqrt{m}\} \)
  • High: \( \mathcal{H} = \{v : d_v > \sqrt{m}\} \)
- There are at most \( n \) low nodes; each produces at most \( O(m) \) paths
- There are at most \( 2\sqrt{m} \) high nodes
  • Each produces paths to other high nodes: \( O(m) \) paths per node
- These two are identical!
- Therefore, no mapper can produce substantially more work than others.
- Total work is \( O(m^{3/2}) \), which is optimal
Approach 2: Graph Split

Partitioning the nodes:
- Previous algorithm shows one way to achieve better parallelization
- But what if even $O(m)$ is too much. Is it possible to divide input into smaller chunks?

Graph Split Algorithm:
- Partition vertices into $p$ equal sized groups $V_1, V_2, \ldots, V_p$.
- Consider all possible triples $(V_i, V_j, V_k)$ and the induced subgraph:
  \[ G_{ijk} = G[V_i \cup V_j \cup V_k] \]
- Compute the triangles on each $G_{ijk}$ separately.
Approach 2: Graph Split

Some Triangles present in multiple subgraphs:

- $V_i$, $V_j$, $V_k$ in $p$ subgraphs
- in $p^2$ subgraphs
- in $p-2$ subgraphs
- in 1 subgraph

Can count exactly how many subgraphs each triangle will be in.
Analysis:
- Each subgraph has $O(m/p^2)$ edges in expectation.
- Very balanced running times
Approach 2: Graph Split

Analysis:

- Very balanced running times
- $p$ controls memory needed per machine
Approach 2: Graph Split

Analysis:

- Very balanced running times
- $p$ controls memory needed per machine
- Total work: $p^3 \cdot O\left(\frac{m}{p^2}\right)^{3/2} = O\left(\frac{m^{3/2}}{p}\right)$, independent of $p$
Approach 2: Graph Split

Analysis:
- Very balanced running times
- $p$ controls memory needed per machine
- Total work: $p^3 \cdot O((m/p^2)^{3/2}) = O(m^{3/2})$, independent of $p$

Input too big: paging

Shuffle time increases with duplication
Overall

Naive Parallelization Doesn’t help with Data Skew
Related Work

• Tsourakakis et al. [09]:
  - Count global number of triangles by estimating the trace of the cube of the matrix
  - Don’t specifically deal with skew, obtain high probability approximations.

• Becchetti et al. [08]
  - Approximate the number of triangles per node
  - Use multiple passes to obtain a better and better approximation
Conclusions

Think about data skew.... and avoid the curse
Conclusions

Think about data skew.... and avoid the curse

- Get programs to run faster
Conclusions

Think about data skew.... and avoid the curse
- Get programs to run faster
- Publish more papers
Conclusions

Think about data skew.... and avoid the curse
- Get programs to run faster
- Publish more papers
- Get more sleep
Conclusions

Think about data skew.... and avoid the curse

- Get programs to run faster
- Publish more papers
- Get more sleep
- ..
Conclusions

Think about data skew.... and avoid the curse

- Get programs to run faster
- Publish more papers
- Get more sleep
- ..
- The possibilities are endless!
Thank You