Graph Algorithms & Iteration on Map-Reduce

CompSci 590.04
Instructor: Ashwin Machanavajjhala
Recap: Map-Reduce

\[
\begin{align*}
\text{map} & \quad (k1, v1) \rightarrow \text{list}(k2, v2); \\
\text{reduce} & \quad (k2, \text{list}(v2)) \rightarrow \text{list}(k3, v3).
\end{align*}
\]
This Class

• Graph Processing

• Iterative-aware Map Reduce
GRAPH PROCESSING
Graph Algorithms

• Diameter Estimation
  – Length of the longest shortest path in the graph

• Connected Components
  – Undirected s-t connectivity (USTCON): check whether two nodes are connected.

• PageRank
  – Calculate importance of nodes in a graph

• Random Walks with Restarts
  – Similarity function that encodes proximity of nodes in a graph
Connected Components

• What is an efficient algorithm for computing the connected components in a graph?
HCC [Kang et al ICDM ‘09]

• Each node’s label $l(v)$ is initialized to itself
• In each iteration
  $l(v) = \min \{l(v), \min_{y \in \text{neigh}(v)} l(y)\}$

• $O(d)$ iterations ($d = \text{diameter of the graph}$)
  $O(|V| + |E|)$ communication per iteration
GIM-V

• Generalized Iterative Matrix-Vector Multiplication

Connected Components
• Let $c^h$ denote the component-id of a vertex in iteration $h$

• $c^{h+1} = M_xGc^h$
  - $c^{\text{new}}[i] = \min_j(m[i,j] \times c^h[j])$
  - $c^{h+1}[i] = \min(c^h[i], c^{\text{new}}[i])$

• Keep iterating till $c^{h+1} = c^h$. 

Step 1: Generate $m[i,j] \times c[j]$
Step 2: Aggregate to find the min for each node

Lecture 13 : 590.04 Fall 15
GIM-V and Page Rank

\[ p = (cE^T + (1 - c)U)p \]

- \( p^{\text{next}} = M x_G \ p^{\text{cur}} \)
- \( p^{\text{next}}[i] = (1-c)/n + \sum_j (c \times m[i,j] \times p^{\text{cur}}[j]) \)
GIM-V BL

- We assumed each edge in the graph is represented using a different row.
- Can speed up processing if each row represents a bxb sub matrix

\[
\begin{align*}
B_{0,0} & \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix} & \quad B_{0,1} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\
B_{1,0} & \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix} & \quad B_{0,2} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}
\end{align*}
\]
Connected Components

- Iterative Matrix Vector products need $O(d)$ map reduce steps to find the connected components in a graph.

- Diameter of a graph can be large.
  - $> 20$ for many real world graphs.

- Each map reduce step requires writing data to disk + remotely reading data from disk (I/O + communication)

- Can we find connected components using a smaller number of iterations?
Hash-to-all

• Maintain a cluster at each node
  – Current estimate of connected component

• Initialize cluster(v) = Neighbors(v) U {v}

• Each node sends its cluster to all nodes in the cluster
  – Map: (v, C(v)) \rightarrow \{(u, C(v))\} for all u in C(v)

• Union all the clusters sent to a node v
  – Reduce: (u, \{C1, C2, ..., Ck\}) \rightarrow (u, C1 U C2 U ... U Ck)
Hash-to-all

- Number of rounds = log d
  - Proof?

- Communication per round = O(n|V| + |E|)
  - Each node is replicated at most n times, where n is the maximum size of a connected component.
Hash-to-Min

- Each node $v$ maintains a cluster $C(v)$ which is initialized to $\{v\} \cup \text{Neighbors}(v)$

- In each iteration

  **Map:**
  
  $v_{\text{min}} = \min \{C(v)\}$
  
  Send $C(v)$ to $v_{\text{min}}$
  
  Send $v_{\text{min}}$ to nodes in $C(v)$

  **Reduce:**
  
  $C(v)$ is the union of all incoming clusters
Hash-to-Min

- Each node \( v \) maintains a cluster \( C(v) \) which is initialized to \( \{v\} \cup \text{Neighbors}(v) \)

- In each iteration

Map:
\[
\begin{align*}
\nu_{\text{min}} &= \min \{ C(v) \} \\
\text{Send } C(v) \text{ to } \nu_{\text{min}} \\
\text{Send } \nu_{\text{min}} \text{ to nodes in } C(v)
\end{align*}
\]

Reduce:
\( C(v) \) is the union of all incoming clusters

<table>
<thead>
<tr>
<th>( v )</th>
<th>( C(v) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,2</td>
</tr>
<tr>
<td>2</td>
<td>1,2,3,4</td>
</tr>
<tr>
<td>3</td>
<td>2,3</td>
</tr>
<tr>
<td>4</td>
<td>2,4,5</td>
</tr>
<tr>
<td>5</td>
<td>4,5,6</td>
</tr>
<tr>
<td>6</td>
<td>5,6</td>
</tr>
</tbody>
</table>
Hash-to-Min

- Each node \( v \) maintains a cluster \( C(v) \) which is initialized to \( \{v\} \cup \text{Neighbors}(v) \)

- In each iteration

  Map:
  \[
  v_{\text{min}} = \min \{C(v)\}
  \]
  
  Send \( C(v) \) to \( v_{\text{min}} \)
  
  Send \( v_{\text{min}} \) to nodes in \( C(v) \)

  Reduce:
  \( C(v) \) is the union of all incoming clusters

<table>
<thead>
<tr>
<th>( v )</th>
<th>( C(v) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,2,3,4</td>
</tr>
<tr>
<td>2</td>
<td>1,2,3,4,5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1,4,5,6</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

Lecture 13 : 590.04 Fall 15
Hash-to-Min

- Each node \(v\) maintains a cluster \(C(v)\) which is initialized to \(\{v\} \cup \text{Neighbors}(v)\)

- In each iteration
  
  **Map:**
  
  \[v_{\text{min}} = \min \{C(v)\}\]
  
  Send \(C(v)\) to \(v_{\text{min}}\)
  
  Send \(v_{\text{min}}\) to nodes in \(C(v)\)

  **Reduce:**
  
  \(C(v)\) is the union of all incoming clusters

<table>
<thead>
<tr>
<th>(v)</th>
<th>(C(v))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,2,3,4,5,6</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>
Hash-to-Min

• In the end, cluster of vertex with minimum id contains the entire connected component. Cluster of other vertices in the component is a singleton having the minimum vertex.

• Communication cost: Assuming a random assignment of ids to vertices, expected communication cost is $O(k(|V| + |E|))$ in iteration $k$.

• Number of iterations: ???
  – On a path graph: $4 \log n$
  – In a general graph: Can be as big as $d$
Leader Algorithm

- Let $\pi$ be an arbitrary total order over the vertices.
- Begin with $l(v) = v$, and all nodes active

In each iteration:
- Let $C(v)$ be the connected component containing $v$
- Let $\Gamma(v)$ be the neighbors of $C(v)$ that are not in $C(v)$
- Call each active node a leader with probability $\frac{1}{2}$
- For each active non-leader $w$, find $w^* = \min(\Gamma(w))$
- If $w^*$ is not empty and $l(w^*)$ is a leader, then mark $w$ as passive, and relabel each node with label $w$ by $l(w^*)$
Correctness

- If at any point of time two nodes s and t have the same label, then they are connected in G.

- Consider an iteration, when \( l(s) \neq l(t) \) before the iteration, but \( l(s) = l(t) \) after.
- This means, \( l(s) = w \) (non-leader node), \( l(t) = w^* \)
- By induction, s is connected to all nodes in \( \Gamma(w) \), t is connected to all nodes in \( \Gamma(w^*) \), and w is connected to \( w^* \).
- Therefore, s and t are connected.
Number of Iterations

• Every connected component has a unique label after $O(\log N)$ rounds with high probability

• Suppose there is some connected component with two active labels.

• An active label $w$ survives an iteration if:
  1. $w$ is marked a leader
  2. $w$ is not marked a leader and $l(w^*)$ is not marked a leader

• Hence, in every iteration, the expected number of active labels reduces by $\frac{1}{4}$. 
ITERATION AWARE MAP-REDUCE
Iterative Computations

PageRank:

\[
\text{do } \quad p^{\text{next}} = (cM + (1-c) U)p^{\text{cur}} \\
\text{while}(p^{\text{next}} \neq p^{\text{cur}})
\]

- Loops are not supported in Map-Reduce
  - Need to encode iteration in the launching script
- M is a loop invariant. But needs to written to disk and read from disk in every step.
- M may not be co-located with mappers and reducers running the iterative computation.
HaLoop

• Iterative Programs

\[ R_{i+1} = R_0 \cup (R_i \bowtie L) \]

Initial Relation

Invariant Relation
Loop aware task scheduling

- Inter-Iteration Locality
- Caching and Indexing of invariant tables
Summary

• No native support for iteration in Map-Reduce
  – Each iteration writes/reads data from disk leading to overheads

• Many graph algorithms need iterative computation
  – Need to design algorithms that can minimize number of iterations

• New frameworks that minimize overheads by caching invariant tables in the iterative computation
  – HaLoop