4.1 Outline

1. Matrix Vector Multiply ($Av$)

2. PageRank
   - on MapReduce
   - on RDD’s / Spark

4.2 Matrix Vector Multiplication on MapReduce

We have a sparse matrix $A$ stored in the form $<i, j, a_{ij}>$, where $i, j$ are the row and column indices and a vector $v$ stored as $<j, v_j>$. We wish to compute $Av$.

For the following algorithm, we assume $v$ is small enough to fit into the memory of the mapper.

Algorithm 1 Matrix Vector Multiplication on MapReduce

1: function MAP($<i, j, a_{ij}>$)
2:   Emit($i, a_{ij}v[j]$)
3: end function
4: function REDUCE(key, values)
5:   ret ← 0
6:   for val ∈ values do
7:     ret ← ret + val
8:   end for
9:   Emit(key, ret)
10: end function

4.3 PageRank

For a graph $G$ with $n$ nodes, we define the transition matrix $Q = D^{-1}A$, where $A \in \mathbb{R}^{n \times n}$ is the adjacency matrix and $D \in \mathbb{R}^{n \times n}$ is a diagonal matrix composed of the outgoing edges from each node.

We use Power Iteration to estimate importance values for webpages as $v^{(k+1)} = v^{(k)}Q$, where $v \in \mathbb{R}^n$ is a row vector, and $k$ is the number of iterations. We set $v^{(0)} = 1$, a vector with each element equaling one.
Using $Q$ as the probability distribution for random walks is a problem when $G$ contains dead-ends, i.e. “sink” nodes (nodes 2 and 7 in Figure 1). We introduce the idea of random teleports. With probability $\alpha$, the random walker can teleport to a random webpage or continue walking with probability $1 - \alpha$ where $0 < \alpha < 1$. Then we have a new matrix:

$$P = (1 - \alpha)Q + \alpha \Lambda$$

where

$$\Lambda = \begin{pmatrix}
\cdots & \lambda & \cdots \\
\cdots & \lambda & \cdots \\
\cdots & \cdots & \cdots \\
\cdots & \lambda & \cdots \\
\end{pmatrix}_{n \times n}$$

and $\alpha \in \mathbb{R}^n$ is composed of the probability distribution of teleporting to a webpage.

The Power Iteration applies again: $\pi^{(k+1)} = \pi^{(k)}Q$.

**Theorem 4.1**

$$\|\pi - v^{(k)}\|_2 \leq e^{-ak}$$

for some constant $a > 0$.

According to 4.1, for $n = 10^9$, around 9 iterations are enough to get correct ranking.

4.3.1 PageRank on MapReduce

$P$ is stored as $<i, \{(j, P_{ij})\}>$, where $\sum_j P_{ij} = 1, \forall i \in [1, n]$.

$v$ is stored as $<i, v^{(k)}_i>$. We use a two-step algorithm:

**Step 1:**
Annotate $P_i$ with $v_i$, i.e. Emit $<i, v_i, \{(j, P_{ij})\}>$.

**Step 2:**
Algorithm 2 PageRank Computation on MapReduce, Step 2

1: function MAP(\( < i, v_i, \{ (j, P_{ij}) \} > \))
2: \hspace{1em} for (\( j, P_{ij} \)) \in \text{links} do
3: \hspace{2em} Emit(\( j, P_{ij}v_i^{(k)} \))
4: \hspace{1em} end for
5: end function

6: function REDUCE(key,values)
7: \hspace{1em} v_i^{(k+1)} = \sum_{v \in \text{values}} v
8: \hspace{1em} Emit (\( i, v_i^{(k+1)} \))
9: end function