1 Overview

In the last lecture we finished cache-oblivious algorithms and began covering MapReduce. In this lecture we continue with MapReduce.

2 MapReduce Review

Recall that MapReduce has the following components:

- **map** – Takes as input a single `<key, value>` pair and outputs a multiset `{<k_1, v_1>, ..., <k_m, v_m>}`.
- **shuffle** – collects all outputs of map and organizes them by key
- **reduce** – receives all items with the same key, say `<k, (v_1, ..., v_t)>` and outputs a multiset `{<k_1, v'_1>, ..., <k_m, v'_m>}`. This can be repeated for several rounds

And that we aim to fulfill the following goals:

- Use few machines (`<< n`, where `n` is the size of the input)
- Each machine uses `<< n` memory
- Total work is small, i.e. `\sum_{i=1}^{R} (\text{max processing time of machine in round } i) \times (\text{number of machines})`, where `R` is the number of rounds, is small.

3 Minimal Spanning Trees in Dense Graphs

We'll take dense to mean `m = N^{1+c}` with `c` bounded away from 0, where `m, N` are the numbers of edges and vertices respectively.

3.1 Algorithm 1 (by Karloff, Suri, Vassilvitskii [1])

- Pick `k = N^{\frac{c}{2}}` and randomly partition `V` into equal sized sets `V_1, ..., V_k`
- Let `G_{ij} = (V_i \cup V_j, E_{ij})` be the induced graph on `V_i \cup V_j`
• Compute using any sequential algorithm \( M_{ij} = MSF(\mathcal{G}_{ij}) \)

• Send \( H = \bigcup_{i,j} M_{ij} \) to a single reducer and output \( M = MST(H) \).

The number of machines needed is \( k^2 = N^c < N < M^{1-\epsilon} \).

**Claim** w.h.p., we have \( \forall i,j \ |E_{ij}| \leq \tilde{O}(N^{1+\frac{c}{2}}) \) where the tilde hides logarithmic factors.

**Proof** We have \( |E_{ij}| \leq \sum_{v \in V_i} \deg(v) + \sum_{v \in V_j} \deg(v) \). Define \( W_t = \{ v \in V | 2^{t-1} \leq \deg(v) \leq 2^t \} \). For each \( 1 \leq t \leq \log_2 N \) separately, if \( |W_t| \geq N^c \log N \), then it is OK if all of \( W_t \) maps to the graph \( \mathcal{G}_{ij} \). Because no degree is bigger than \( N \), \( W_t \) contributed at most \( N^{1+c/2} \log N \) to the degree sum. So all such \( W_t \) in total contribute \( \ll N^{1+c/2}(\log N)^2 \).

On the other hand, if \( |W_t| \geq N^c \log N \), then \( \mathbb{E}(\text{number of vertices from } W_t \text{ landing in } \mathcal{G}_{ij}) \geq 2 \log N \) (which is \( \frac{2|W_t|}{k} \) in order). Then by a Chernoff bound, we will be close to the expectation (up to a constant factor) with probability \( \frac{1}{\text{poly}(n)} \). Thus we can union bound over \( t = 1, ..., \log N \) and \( i,j = 1, ..., k \) (which is at most \( N^c \log N \) things in total).

The downside of this algorithm is that the total memory across the whole system can be approximately \( m^{2-2\epsilon} \), which is much greater than the space needed to give input \( m \). An advantage is that it needs only two rounds.

There is another algorithm with \( O(N^c) \) machines and \( O(N^{1-\epsilon}) \) memory per machine using \( \lceil \frac{c}{\epsilon} \rceil \) rounds in [2].

### 4 Triangle Counting and Clustering Coefficients (Suri and Vassilvitskii [3])

For Triangle Counting, the input is an undirected graph, and we want to output \( |\{ u < v < w \in V | (u,v), (v,w), (w,u) \in E \}| \). For Clustering Coefficients, for each \( v \in V \) we want to compute \( cc(V) = \frac{|\{ u,w \in \Gamma(v) | (u,w) \in E \}|}{\binom{\deg(v)}{2}} \), where \( \Gamma(v) \) denotes the set of neighbours of \( v \).

#### 4.1 Simple Algorithm

Use three for loops. This runs in time \( O(\sum_{v \in V} \deg(v)^2) \).

#### 4.2 More Clever Algorithm

\[
x \leftarrow 0 \\
\text{for } v \in V \text{ s.t. } \deg(v) \geq \deg(x) \\
\text{for } u \in \Gamma(v) \text{ s.t. } \deg(u) \geq \deg(x) \\
\text{for } w \in \Gamma(v) \text{ s.t. } \deg(w) \geq \deg(x) \\
\text{if } (u,w) \in E 
\]
x ← x + 1

return x

Claim The runtime of the above is $O(m^{\frac{3}{2}})$.

Proof We break into cases. First, suppose $\deg(v) < t$. Then
\[
\sum_{v \in V, \deg(v) < t} \deg(v)^2 \leq \max_{v \in V, \deg(v) < t} \sum_{v \in V} \leq t \cdot 2m = 2mt
\]

Next, suppose $\deg(v) \geq t$. There are at most $2m^t$ such vertices. The total runtime is at most $(\frac{2m^t}{t})^3 + 2mt$, so setting $t = \sqrt{m}$ proved the claim.

We can implement the above algorithm in MapReduce, with the following properties:

- No reducer gets more than $O(\sqrt{m})$ items
- The total work done is $O(m^{\frac{3}{2}})$
- The number of rounds is 2

4.3 Algorithm 2

Let $\rho$ be some parameter.
Map: The input is $< (u, v); 1 >$. $h : V \rightarrow [\rho]$  
\[
i \leftarrow h(u) \
j \leftarrow h(v) 
\]
for $a \in [\rho]$  
for $b \in [\rho], b > a$  
for $c \in [\rho], c > b$  
if $\{i, j\} \subseteq \{a, b, c\}$  
emit $<(a, b, c); (u, v)>$

Reduce: The input is $< (i, j, k); E_{ijk} \subseteq E >$.  
use any algorithm to enumerate the triangles in $E_{ijk}$
for each triangle $(u, v, w)$  
\[
z \leftarrow 1 \
z \leftarrow (\frac{h(u)}{2}) + h(u)(\rho - h(u) - 1) + (\rho - h(u) - 1)\frac{1}{2}
\]
else if $|\{h(u), h(v), h(w)\}| = 2$  
z $\leftarrow \rho - 2$ output $<(u, v, w), \frac{1}{2}>$

This algorithm is doing the following: Randomly partition $V$ into $V_1, ..., V_\rho$ and define $V_{ijk} = V_i \cup V_j \cup V_k$. Define $E_{ijk}$ to be the edges induced by $V_{ijk}$. Let $G_{ijk} = (V_{ijk}, E_{ijk})$. Each $V_{ijk}$ has size approximately $\frac{3N}{\rho}$, and $|E_{ijk}|$ is expected to be $O(\frac{N^2}{\rho^2})$. 


5 Frequency Moments

We’re given $<x_1>, ..., <n;x_n>$ where $x_i \in [N]$ and we want to compute

$$F_k = \sum_{i=1}^{N} (\text{number of occurences of } i)^k.$$ 

5.1 Naive Approach

Map $<i;x_i>$ to $<x_i;1>$. Reduce outputs (number of $x_i$’s)$^k$. Finally, aggregate. Unfortunately, if all $x_i$’s are the same, one machine needs $\Omega(n)$ memort.

5.2 Better Approach

Say a function $f$ on sets $S$ is MR-parallelizable if there are $g, h$ such that for all partitions $T_1, ..., T_k$ of $S$ we have $f(S) = h(g(T_1), ..., g(T_k))$ and we can communicate descriptions of $g, h$ efficiently (say in $O(\log n)$ bits).

Claim Given a universe $U$ of size $n$ and $S_1, ..., S_k \subseteq U$ (which may overlap) with $k \leq n^{2-\epsilon}$ and $\sum_{i=1}^{k} |S_i| \leq n^{2-c\epsilon}$, and MR-parallelizable functions $f_1, ..., f_k$, we may compute $f(S_1), ..., f(S_k)$ using $O(n^{1-\epsilon})$ reducers each with $O(n^{1-\epsilon})$ memory.

Proof Sketch Break up the $M = n^{1-\epsilon}$ reducers into $t = n^{1-2\epsilon}$ blocks of equal size $B = n^\epsilon$. For each $i \in [k]$ we has $i$ to some value in $[t]$. Then for each item in $S_i$, we process it using a random reducer in that block. Next round, we aggregate over the block.

For frequency moments, $U$ is just the set of input pairs. $x_i \in [N]$, so for each $\ell \in [N] S_{\ell} = \{\text{indices } i \text{ with } x_i = \ell\}$. Set $f(S_{\ell}) = |S_{\ell}|^k$, $h(i_1, ..., i_m) = \left(\sum_{j=1}^{m} i_j\right)^k$ and $g(T_j) = |T_j|$.

References

