CS 229r: Algorithms for Big Data Fall 2015

Lecture 26 — December 1, 2015

Prof. Jelani Nelson *Scribe: Zezhou Liu*

1 Overview

Final project presentations on Thursday. Jelani has sent out emails about the logistics: 8 minute presentation with 1-2 minutes for questions. Groups can split up the work however they want (one person can make the slides and the other presents, etc.)

This time, we want to cover algorithms in a distributed environment, maybe 1M machines. How to do this efficiently? Work starts as early as ('76 Csanky)[1], with a survey by ('88, Eppstein, Galil) on PRAM.

PRAM: ignore communication (shared memory across all processors) and synchronization. The goal is to understand time vs. number of processors. However, shared memory is unrealistic, and somewhere in late 80s/early 90s, PRAM wasn't studied much anymore because you had to consider communication and syncrhonization.

Bulk Synch Parallel (BSP) [BSP]: You should explicitly worry about communication and synchronization. BSP today: Apache, Hama, Google, Pragel.

Today: MapReduce [DG08]. This system was built and used at Google to deal with massively parallel jobs. Hadoop is an open-source version. This system is used by Google, Facebook, Amazon, ...

How do we work with these systems and build efficient algorithms on these models?

2 MapReduce

In this model, data items are each $\langle key, value\rangle$ pairs. Computation is broken up into rounds:

Round:

- Map: Each item is processed by some *map* function, and emits a set of new $\langle key, value\rangle$ pairs.
- Shuffle: This step is oblivious to the programmer. All items emitted in the map phase are grouped by key, and items with the same key are sent to the same reducer.
- Reducer: Receives $\langle k, v_1, v_2, ..., v_3\rangle$ and emits new set of items.

Goals:

- Few number of rounds
- Want to use $<< n$ mem per reducer.

- Want $<< n^2$ total mem used.
- Small total work (# machines * max(work per machine))
- Small parallel work (What max(time per machine)?)

3 MapReduce Problems

Example: Sorting.

Theorem 1. *TeraSort (O'Malley, '08) [2]. Sorts arbitrary and comparable elements (although the original was on bounded integers and used tricks to speed that up).*

Input elements are $\langle i; A[i] \rangle$, where A is an unsorted array A[1..n].

Say we want to use p-machines in parallel. So we want to divide it into the smallest n/p elements, next smallest n/p elements, etc. so we can send the jth n/p elements to machine j to be sorted.

Two-round algorithm:

Round 1: Sample $T = log(p)/\varepsilon^2)$

```
def map1 $\langle i, A[i] \rangle$:
    emit $\langle i\%p, A[i] \rangle$
    w.p T/n:
        for j = 0...p:
            emit $\langle j, A[i] \rangle$
```

First partition the values, and then for some random samples, send to some reducer.

```
def reduce1 $\langle j; X \rangle$:
    B are the (i, A[i]), and S is set of sampled elements
    S ← sort(S)
    for each (i, x) ∈ B:
        find some r ∈ $0, ...p - 1$ that x should map to in relation to S, and emit $\langle r; (i, x) \rangle$
```

Round 2:

```
def map2: identity
def reduce2 $\langle j, B \rangle$:
    Sort B by 2nd element (recall B is (i, x))
    write output to j.out
```

At this point, each machine has their own sorted list, so we just need to concatenate the sorted lists from each machine.

Analysis: First, I have to choose epsilon. There are two things I want to balance: 1) Each reducer is getting set of size n/p, but also getting some additional sampled elements $log(p)/\varepsilon^2$. So want $log(p)/\varepsilon^2 \leq n/p$. 2) Chernoff bound says: if we sample $T = log(1/delta) * C/\varepsilon^2$ elements and look at αth smallest element in sample, its rank in actual sorted order of A is $\alpha n + -\varepsilon n$ w.p. $1 - \delta$. Union bound says that with high probability, each reducer in the second round gets $\leq n/p + \varepsilon n$ items to sort. (set $\varepsilon = 1/p \rightarrow T = log(p)\ p^2$)

Going back, we want $p^2 lg(p) \leq n/p$, so $p = n^{1/3}/lg(n) \rightarrow$ with high probability each of the p machines needs to sort $\leq O(n/p) = O(n^{2/3} lg(n))$.

4 Min Spanning Tree (MST)

In the case where it is not too sparse, with $m = \#edges = n^{1+c}$ for some $c > 0$. We want memory per machine to be $<< m(m^{1-\delta})$. Algorithm by (Karloff, Suri, Vassilivitskii, SODA '10)[3].

For each pair $(i, j) \in [k]^2$:

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

MapReduce:

Round1:

```
def map1 $\langle (u, v); NULL \rangle$:
    emit $\langle h(u), h(v); (u, v) \rangle$
    if h(u) == h(v) = i:
        for each j = 1, ..., k
            emit $\langle (i, j); (u, v) \rangle$
def reduce1 $\langle (i, j), E_{i,j} \rangle$:
    let $e_1, ..., e_t$ be MSF of $E_{ij}$
    for a = 1..t:
        emit$\langle \$, (e_a, (i, j)) \rangle$
```

This is emitting all the edges of $M_{i,j}$.

def map2: identity

def reduce2: Take all previous edges and call MST of this set of edges.

Analysis: (memory) - mem per machine round 2: $\leq k^2 \cdot O(n/k) = O(kn) = O(n^{1+c/2}) << m = n^{1+c}$ - mem per machine round 1: $\max_i, j\ |E_{i,j}|$. The expected size of $E_{i,j} = \sum_{p \in E} \mathbb{P}(\text{endpoints of e in } Vi \cup Vj) = 2m/k$

For Chernoff, we have that $\mathbb{E}_{i,j} \leq \sum_{v \in V} \nVdash_{h(u) \in \{i,j\}} \cdot deg(v)$. You can do better than this, as shown in KSV '10: partition V according to degrees. $V_t = \{v : 2^{t-1} \leq deg(v) < 2^t\}$. This means with high probability $\max_{i,j} |E_{i,j}| = O(n^{1+c/2})$

Downside: Total memory needed $\sim k^2 \cdot n^{1+c/2} = n^{1+3c/2} = n^{2+c/2} = m^{2-\varepsilon}$ (Lattanzi et all, SPAA '11) with O(n^c) machines, O($n^{1-\varepsilon}$) mem per machine, and ceil(c/ε) rounds.

5 Triangle Counting

For Triangle Counting, the input is an undirected graph, and we want to output $|\{u < v < w \in V | (u,v), (v,w), (w,u) \in E\}|$. With no parallelism, there is a simple $O(n^3)$ algorithm using 3 nested for-loops by looping over all (u, v, w). You can even get $m^{3/2}$. MapReduce can implement this with $\sqrt(m)$ machines.

```
$x \leftarrow 0$
  for $v \in V$
    for $u \in \Gamma(v)$ s.t. deg($u$) $\geq$ deg($v$)
      for $w \in \Gamma(v)$ s.t. deg($w$) $\geq$ deg($v$)
        if $(u, w) \in E$
          $x \leftarrow x + 1$
return $x/2$
```

We can implement the above algorithm in MapReduce, with the 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

5.1 MapReduce algorithm

```
Round 1:
def map1: The input is $< (v, u); \varnothing >$.
    if $u \succ v$ : emit $\langle v, u \rangle$
def reduce1: The input is $< v; S \subseteq T(v) >$.
    for $< u, w > \in S$:
        emit $< u, w >$ if is an edge.
Round 2:
def map2:
    if input is $< v, (u, w) >$:
        emit $< (u, w); v >$
    if input is $< (u, w), \varnothing) >$:
            emit $< (u, w), \$ >$
def reduce2 $< (u, w); S >$:
        if $\$ \in S$: then
            for each $v \in S$ s.t. $v \neq \$$:
                emit $< v, 1 >$
```

References

[1] Csanky, L., Fast parallel matrix inversion algorithms, *SIAM J. Computing* 5, 1976,

[BSP] Leslie G. Valiant. 1990. A bridging model for parallel computation. *Commun. ACM 33*, 8 (August 1990), 103-111. DOI=10.1145/79173.79181 http://doi.acm.org/10.1145/79173.79181

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[2] O'Malley, O. TeraByte sort on Apache Hadoop, 2008;

[3] Howard J. Karloff, Siddharth Suri, Sergei Vassilvitskii. A Model of Computation for MapReduce. *SODA 2010*: 938-948.

[4] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, Sergei Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. *SPAA 2011*: 85-94.

[5] Siddharth Suri, Sergei Vassilvitskii. Counting triangles and the curse of the last reducer *WWW 2011*: 607-614.