COMS E6998-9: Algorithmic Techniques for Massive Data — Dec. 3, 2015

Lecture 24 — MapReduce Algorithms

Instructor: *Alex Andoni* — Scribes: *Kui Tang*

1 Introduction

We review the computational model of MapReduce algorithms. Our model is more general than the original Google MapReduce framework: we do not constrain the functional forms of computation (e.g decomposition into local map and associative reduce functions) but only in terms of space and communication constraints.

We have M machines with S space per machine, and $MS \approx O(\text{input size})$. This means we cannot replicate data too much. We have n inputs and $O(n)$ outputs, but $S \ll n$, i.e. neither input nor output can fit on one machine.

Local computation is cheap but communication (shuffling) is very expensive, so we bound the number of rounds. In particular, we want rounds $R = O(1)$ for $S \leq n^\delta$, for instance $S > \sqrt{n}$ where $S > M$. We want $O(S)$ in-communication (how much input each node requires) per round, and ideally linear. (You cannot have more communication than you have space.)

This model is the culmination of the bulk-synchronous parallel, the original MapReduce, and massively parallel computing (MPC) frameworks. MR can simulate PRAM algorithms in $R = O(\text{parallel time})$. But this often takes a logarithmic slowdown, because to collect n items of data in a tree of constant degree s, we need a depth of $O(\log_s n)$, which is also how many rounds we must take. In particular, we need $\widetilde{\Omega}(\log n)$ on CRCW PRAMs.

1.1 Sorting

Also called the Terasort problem — sorting a terabyte of numbers.

Suppose $S = O(n^{2/3})$ and $M = O(n^{1/3})$. The entire input of n numbers can be sorted in just three rounds of communication. How?

- Each machine picks some distinguished elements with probability $n^{1/2}/n$. This means $\Theta(n^{1/2})$ elements will be selected.
- **All machines send the distinguished elements to machine #1.** (This fits, since $n^{1/2} < n^{2/3}$.)
- Machine #1 chooses approximately equidistant pivots and assign a range to each machine. (For a particular pivot, its range is the set of numbers which are closer to it than to any other pivot.)
- **Machine #1 sends to pivots to each machines.**
  - Note that each range will have $O(n^{1/2}/M) = O(n^{1/2}/n^{1/3}) = O(n^{2/3})$ elements.
- **Each machine sends all elements in range i to machine #i.**

- Each machine $\#i$ sorts its received elements locally.

Each bolded line indicates a round of communication.

1.2 Connectivity

Consider searching for a *certificate* of connectivity, e.g. a spanning tree. We call the size of a spanning tree the solution size. When $S \gg$ solution size, then we say the problem is dense. We filter input until it fits on a machine, and with $S = n^{1+\delta}$, we can do it in $O(1/\delta)$ rounds.

The sparse case $S \ll$ solution size is harder. We assume $S = \sqrt{n}$. This is hard, and still open to do s-t connectivity in $\ll O(\log n)$ rounds.

We consider an easier problem: geometric graphs, which are implicit graphs on n points in $\mathbb{R}^d$. The weight (cost) on each edge is the euclidean distance in $\mathbb{R}^d$. We assume a geometric graph is fully connected, connectivity is meaningless. Instead, we seek a minimum-cost spanning tree. This is equivalent to single-linkage agglomerative hierarchical clustering.

Theorem 1. *We can get a $(1+\epsilon)$ approximation in low dimensional ($\mathbb{R}^d$) space in a constant number of rounds $R = (\log_s n)^{O(1)}$ for MST if $S \le e^{-O(d)}$ and EMD if $S \ge n^{o(1)}$ for constant ϵ, d.*

We sketch the proof (pun intended) in the following section.

1.3 Solve and Sketch

A general framework for solving problems in linear spaces is to partition the space hierarchically in a "nice way," which can be represented in a tree. We then compute bottom-up according to the tree.

Assume each node is one machine (which could be reused). Each node computes a pseudo-solution for its local view and then sketches this pseudo-solution using a small space, and then sends the sketch to its parent. We assume each pseudo-solution uses all of the space S on its machine, so a machine cannot simply send the exact pseudo-solution up to its parent, since the parent would lack space to receive all pseudo-solutions from all of its children.

We use quad trees for partitioning, local MST for the local pseudo-solutions, and one representative in the partition as the sketch.

One problem is that quad trees can cut MST edges, which is irrevocable (since we must greedily solve the local problems). And, we may choose a wrong representative.

So instead of quad trees, we use a randomly shifted grid with side length Δ and take an $\epsilon\Delta$-net called N.[1] The points in N are our entry/exit portals to the cell. That is, if d is the distance on $\mathbb{R}^d$, then

$$d'(p,q) = \begin{cases} d(p,q) & p,q \text{ in same grid cell} \\ d(p,p') + d(p',q') + d(q',q) & p,q \text{ in different cells} \end{cases}$$

where p' and q' are the nearest neighbors to p and q respectively in N. We call the operation of finding p' and q' *snapping to the $\epsilon\Delta$ net.* Think of the points in N as airports, so to go from the points in one region to another, you must first go to the airport.

Claim 2. *All distances are preserved by up to $1+4\epsilon$ in expectation.*

[1] A $\epsilon\Delta$-net N is a subset of $\mathbb{R}^d$ such that for $p \in \mathbb{R}^d$, there exists a $q \in N$ such that $\|p-q\| \le \delta$.

Proof. Let D denote the event that p and q are in different cells. We have $\delta = P[D] \leq 2\|p-q\|/\Delta$ by construction of the randomly shifted grid. In this case, $d(p,p') \leq \epsilon\Delta$ so that $d'(p,q) = 2\epsilon\Delta + \|p-q\|$. In the other case, $d(p,q) = \|p-q\|$. Putting it together, we have

$$\begin{aligned} E\left[d'(p,q) - \|p-q\|\right] &= P[D](2\epsilon\Delta + \|p+q\|) + (1-P[D])\|p+q\| - \|p+q\| \\ &= 2\epsilon\Delta P[D] \\ &\leq 4\epsilon\|p+q\| \end{aligned}$$

□

1.3.1 Algorithm

Assume $d = 2$ and that the entire point set is in a cube of size $n^{2/3} \times n^{2/3}$ and $S \gg n^{2/3}$.

Partition by *two* levels of randomly-shifted grid of size $\Delta = n^{1/3}$, such that each local node has size $\Delta \times \Delta < S$. For the pseudo-solution, we run Kruskal's algorithm locally for edges up to length $\epsilon\Delta$.[2] To sketch a pseudo-solution, we snap points to $\epsilon^2\Delta$ net N_2 and store the connectivity of the net points. Being a 2-dimensional $\epsilon^2\Delta$ net, we have at most $O(1/\epsilon^4)$ points. Storing connectivity just means storing one bit for each point denoting whether it belongs to the MST, so it takes $O(1/\epsilon^4)$ space.

Claim 3. *The above algorithm is equivalent to running Kruskal's algorithm on the distance d' up to a $1+O(\epsilon)$ approximation.*

Proof. The distance between two cells is at least $\epsilon\Delta$ (or else we would not have placed a net point there). So if we run Kruskal's locally inside each cell up to distance $\epsilon\Delta$, we ensure that the optimal tree in each cell will not contain any points in adjacent cells. But snapping to the $\epsilon^2\Delta$-net points introduces a $1+2\epsilon$ factor error, since all distances are at least $\epsilon\Delta$. Putting this together with the results of the previous claim, we have $E[\text{cost of MST}] \leq (1+O(\epsilon))\,\text{MST}_{opt}$. □

What about improving the local runtime? Kruskal's algorithms with exotic data structures are near-linear in the number of edges. But we have fully connected graphs, so these are not good enough. We need to use approximate Kruskal's algorithms.

The solution is distributed across the machines. Each machine stores some edges, and the top machine stores the leftovers.

2 Wrap-up

We review the types of algorithms and problems we've covered in this class.

First we discussed streaming algorithms. We covered frequency moments and heavy hitters and graph algorithms. But we haven't discussed algorithms for lists, like median/quantile selection, longest increasing sequence. There are nice algorithms here. Neither have we discussed geometric objects. What we just did in MR we could also have done in streaming.

Linear sketches are very powerful when they exist. The sketching function $S(a+b) = S(a)+S(b)$ makes the problem trivially decomposable. For computing ℓ_1 and ℓ_2 norms, we can get a $1+\epsilon$ approximation in constant space. We also covered dimensional reduction (Johnson-Lindenstrauss) and fast JL (using

[2] Kruskal's algorithm: Connect points with the shortest edge that does not introduce a cycle.

the fast Fourier transform). They are also useful to speed up classical algorithms, in numerical linear algebra. Compressive sensing is also a form of linear sketch which is very wide and powerful.

For nearest neighbors, a basic technique is to apply sketching. But LSH and data-dependent LSH are even better. We can also embed complicated spaces into simpler ones. What we didn't cover in class is NNS for general metrics. The complexity depends on a measure of "intrinsic dimension."

Property testing, or sampling, is an even more restrictive setting. We've covered testing distributions of uniformity and identity (to another distribution). We can also compute entropy or support. There are also *instance-optimal* algorithms, which essentially adapts the algorithm as samples come in to get better bounds for the particular instance. We also covered property testing for graph connectivity. For dense graphs, there is a regularity lemma, which is really hard. To test for ϵ-farness, we have bounds of the form of a tower of $2^{2^{\cdots^2}}$ of height $1/\epsilon$, so reasonably the proofs would also be very hard.

Finally we talked about the MR model, which is the topic of this lecture. The important constraint is an upper bound on the in-communication complexity.