CIS 700:
"algorithms for Big Data"

Lecture 10:
Massively Parallel Algorithms

Slides at http://grigory.us/big-data-class.html

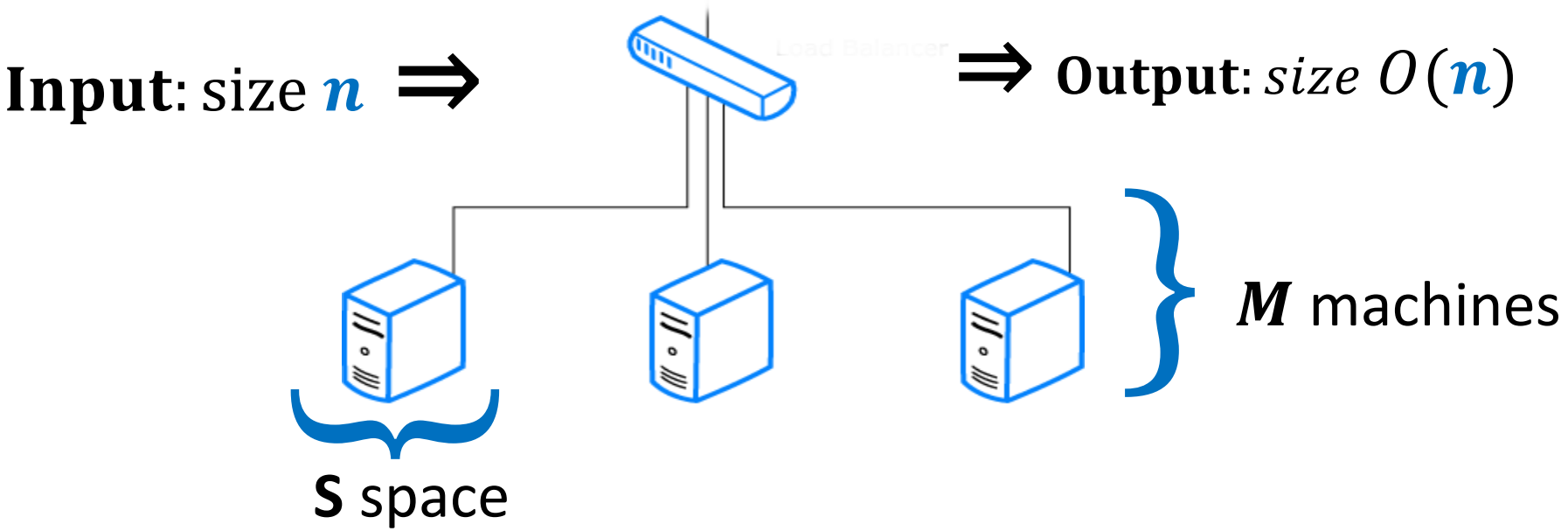
Grigory Yaroslavtsev

http://grigory.us

Penn
University of Pennsylvania

Computational Model

- **Input**: size **n**
- $\boldsymbol{M}$ machines, space $\boldsymbol{S}$ on each ($\boldsymbol{S} = \boldsymbol{n}^{\alpha}$, $0 < \alpha < 1$)
  - Constant overhead in total space: $\boldsymbol{M} \cdot \boldsymbol{S} = O(\boldsymbol{n})$
- **Output**: solution to a problem (often size O($\boldsymbol{n}$))
  - Doesn't fit on a single machine ($\boldsymbol{S} \ll \boldsymbol{n}$)

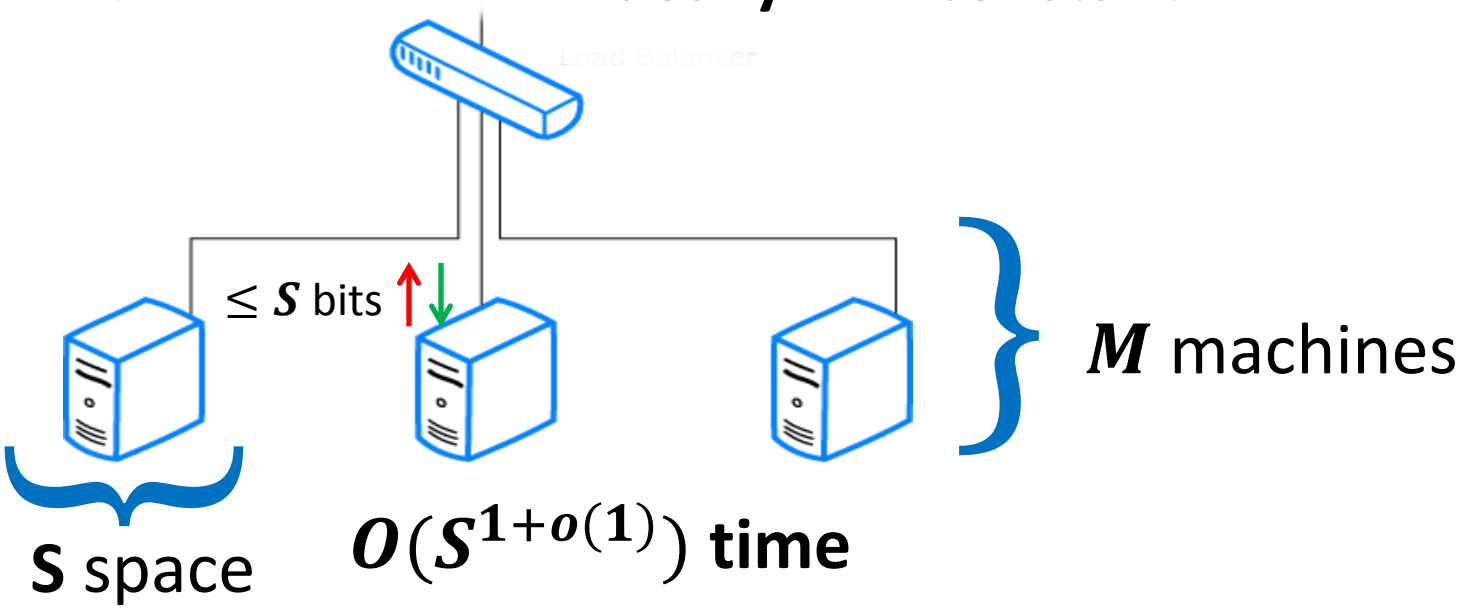


Computational Model

- Computation/Communication in $\boldsymbol{R}$ rounds:
  - Every machine performs a **near-linear time** computation => Total running time $O(\boldsymbol{n^{1+o(1)}R})$
  - Every machine **sends/receives at most $\boldsymbol{S}$ bits** of information => Total communication $O(\boldsymbol{nR})$.

Goal: Minimize $\boldsymbol{R}$.

Ideally: $\boldsymbol{R}$ = constant.

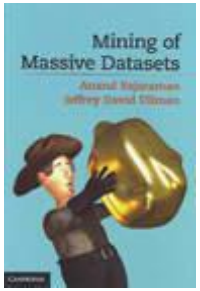


MapReduce-style computations

YAHOO! Google

What I won't discuss today

- PRAMs (**shared memory**, multiple processors) (see e.g. [Karloff, Suri, Vassilvitskii'10])
  - Computing XOR requires $\widetilde{\Omega}(\log n)$ rounds in CRCW PRAM
  - Can be done in $O(\log_s n)$ rounds of MapReduce
- Pregel-style systems, Distributed Hash Tables (see e.g. Ashish Goel's class notes and papers)
- Lower-level implementation details (see e.g. Rajaraman-Leskovec-Ullman book)



Models of parallel computation

- **Bulk-Synchronous Parallel Model** (BSP) **[Valiant,90]**

  Pro: Most general, generalizes all other models

  Con: Many parameters, hard to design algorithms

- **Massive Parallel Computation** [Feldman-Muthukrishnan-Sidiropoulos-Stein-Svitkina'07, Karloff-Suri-Vassilvitskii'10, **Goodrich-Sitchinava-Zhang'11, ..., Beame, Koutris, Suciu'13**]

  Pros:

  - Inspired by **modern** systems (Hadoop, MapReduce, Dryad, …)
  - Few parameters, **simple** to design algorithms
  - **New algorithmic ideas**, robust to the exact model specification
  - **# Rounds** is an information-theoretic measure => can prove unconditional lower bounds
  - Between **linear sketching** and **streaming with sorting**

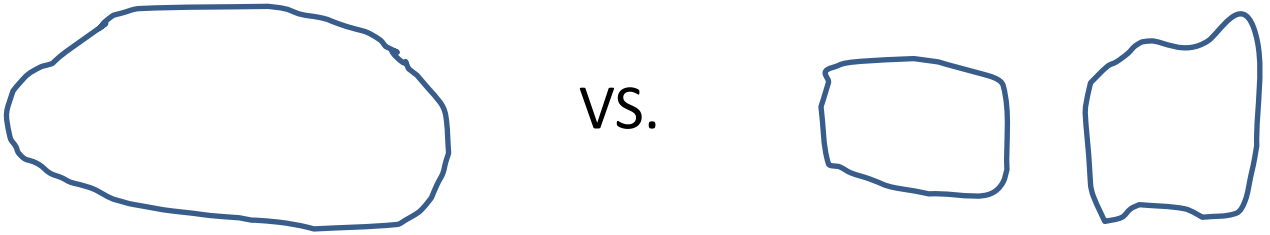
Sorting: Terasort

- Sorting $\boldsymbol{n}$ keys on $\boldsymbol{M = O(n^{1-\alpha})}$ machines
  - Would like to partition keys uniformly into blocks: first $\boldsymbol{n/M}$, second $\boldsymbol{n/M}$, etc.
  - Sort the keys locally on each machine
- Build an approximate histogram:
  - Each machine takes a sample of size $\boldsymbol{s}$
  - All $\boldsymbol{M * s \leq S = n^{\alpha}}$ samples are sorted locally
  - Blocks are computed based on the samples
- By Chernoff bound $\mathbf{M} * \boldsymbol{s} = O\left(\frac{log\, \boldsymbol{n}}{\boldsymbol{\epsilon^2}}\right)$ samples suffice to compute al block sizes with $\pm\epsilon\boldsymbol{n}$ error
- Take $\epsilon = \frac{\boldsymbol{n^{\alpha-1}}}{2}$: error $\mathrm{O}(\boldsymbol{S})$; $\mathbf{M} * \boldsymbol{s} = \widetilde{O}(\boldsymbol{n^{2-2\alpha}}) = \boldsymbol{O(M^2)} \leq \boldsymbol{O(n^{\alpha})}$ for $\alpha \geq 2/3$

Algorithms for Graphs

- **Dense graphs** vs. sparse graphs
  - **Dense**: $S \gg |V|$
    - Linear sketching: one round
    - "Filtering" (Output fits on a single machine) [Karloff, Suri Vassilvitskii, SODA'10; Ene, Im, Moseley, KDD'11; Lattanzi, Moseley, Suri, Vassilvitskii, SPAA'11; Suri, Vassilvitskii, WWW'11]
  - Sparse: $S \ll |V|$ (or $S \ll$ solution size)

    Sparse graph problems appear hard (**Big open question**: connectivity in $o(\log n)$ rounds?)



Algorithm for Connectivity

- Version of Boruvka's algorithm
- Repeat $O(\log n)$ times:
  - Each component chooses a neighboring component
  - All pairs of chosen components get merged
- How to avoid **chaining**?
- If the graph of components is bipartite and only one side gets to choose then no chaining
- **Randomly** assign components to the sides

Algorithm for Connectivity: Setup

Data: **N** edges of an undirected graph.

Notation:

- For $v \in V$ let $\pi(v)$ be its id in the data
- $\Gamma(S) \equiv$ set of neighbors of a subset of vertices S⊆V.

Labels:

- Algorithms assigns a label $\ell(v)$ to each v.
- Let $L_v \subseteq V$ be the set of vertices with the label $\ell(v)$ (invariant: subset of the connected component containing v).

Active vertices:

- Some vertices will be called **active**.
- Every set L_v will have exactly one active vertex.

Algorithm for Connectivity

- Mark every vertex as **active** and let $\ell(v) = \pi(v)$.
- For phases $i = 1,2, \ldots, O(\log N)$ do:
  - Call each **active** vertex a **leader** with probability 1/2. If v is a **leader**, mark all vertices in L_v as **leaders**.
  - For every **active** **non-leader** vertex w, find the smallest **leader**(with respect to π) vertex $w^\star \in \Gamma(L_w)$.
  - If $w^\star$ is not empty, mark w **passive** and relabel each vertex with label w by $w^\star$.
- Output the set of CCs, where vertices having the same label according to ℓ are in the same component.

Algorithm for Connectivity: Analysis

- If $\ell(u) = \ell(v)$ then u and v are in the same CC.
- Unique labels w.h.p after $O(\log N)$ phases.
- For every CC # active vertices reduces by a constant factor in every phase.
  - Half of the active vertices declared as non-leaders.
  - Fix an active **non-leader** vertex $\boldsymbol{v}$.
  - If at least two different labels in the CC of v then there is an edge $(\boldsymbol{v'}, \boldsymbol{u})$ such that $\ell(\boldsymbol{v}) = \ell(\boldsymbol{v'})$ and $\ell(\boldsymbol{v'}) \neq \ell(\boldsymbol{u})$.
  - $\boldsymbol{u}$ marked as a **leader** with probability 1/2; in expectation half of the active non-leader vertices will change their label.
  - Overall, expect 1/4 of labels to disappear.
  - By Chernoff after $O(\log N)$ phases # of active labels in every connected component will drop to one w.h.p.

Algorithm for Connectivity: Implementation Details

- Distributed data structure of size $O(|V|)$ to maintain labels, ids, leader/non-leader status, etc.
  - O(1) rounds per stage to update the data structure
- Edges stored locally with all auxiliary info
  - Between stages: use distributed data structure to update local info on edges
- For every **active non-leader** vertex w, find the smallest **leader** (w.r.t π) vertex $w^\star \in \Gamma(L_w)$
  - Each (**non-leader, leader**) edges sends an update to the distributed data structure
- Much faster with Distributed Hash Table Service (DHT) [Kiveris, Lattanzi, Mirrokni, Rastogi, Vassilvitskii'14]

Applications

- Using same reductions as in streaming:
  - Bipartiteness
  - k-connectivity
  - Cut-sparsification