Nimble Algorithms for Cloud Computing

Ravi Kannan, Santosh Vempala and David Woodruff
Cloud computing

Data is distributed arbitrarily on many servers

Parallel algorithms: time
Streaming algorithms: sublinear space

Cloud Complexity: time, space and communication

[Cormode-Muthukrishnan-Ye 2008]

**Nimble algorithm**: polynomial time/space (as usual) and sublinear (ideally polylog) communication between servers.
Cloud vs Streaming

- Streaming algorithms make small “sketches” of data
- Nimble algorithms must communicate small “sketches”
- Are they equivalent?

Simple observation:
- Communication in cloud = $O(\text{memory in streaming})$
  [Daume-Philips-Saha-Venkatasubramanian12]
- Is cloud computing more powerful?
Basic Problems on large data sets

- Frequency moments
- Counting copies of subgraphs (homomorphisms)
- Low-rank approximation
- Clustering

- ...
- Matchings
- Flows
- Linear programs
Streaming Lower Bounds

Frequency moments: Given a vector of frequencies \( f = (f \downarrow 1, f \downarrow 2, ..., f \downarrow n) \) presented as a set of increments, estimate \( \| f \| \downarrow k = \sum i \uparrow \downarrow f \downarrow i \uparrow k \) to relative error \( \epsilon \).

[Alon-Matias-Szegedy99, Indyk-Woodruff05]:
\( \theta (n \uparrow 1 - 2/k) \) space \( (k = 1, 2 \) by random projection\)

Counting homomorphisms: Estimate \#triangles, \#\( C \downarrow 4 \uparrow \), \#\( K \downarrow r, r \) ... in a large graph \( G \).
\( \Omega (n \uparrow 2) \) space lower bounds in streaming.
Streaming Lower Bounds

Low-rank approximation: Given $n \times d$ matrix $A$, find $A'$ of rank $k$ s.t.

$$
\|A - A'\|_{F} \leq (1 + \epsilon) \|A - A'^{k}\|_{F}
$$

[Clarkson-Woodruff09]

Any streaming algorithm needs $\Omega((n+d)k \log nd)$ space.
Frequency moments in the cloud

- Lower bound via multi-player set disjointness.

- $t$ players have sets $S_1, S_2, ..., S_t$, subsets of $[n]$

- Problem: determine if sets are disjoint or have one element in common.

- Thm: Communication needed = $\Omega(n/t \log t)$ bits.
Frequency moments in the cloud

Thm. Communication needed to determine set disjointness of $t$ sets is $\Omega\left(n/t \log t\right)$ bits.

Consider $s$ sets being either
(i) completely disjoint or (ii) with one common element
(each set is on one server)

Then $k$'th frequency moment is either $n$ or $n-1+s^k$

Suppose we have a factor 2 approximation for the $k$'th moment. With $s^k = n+1$, then we can distinguish these cases. Therefore, communication needed is $\Omega\left(s^k-1\right)$. 
Frequency moments in the cloud

Thm. [Kannan-V.-Woodruff13]
Estimating k’th frequency moment on s servers takes $O\left(\frac{s^k}{\epsilon^2}\right)$ words of communication, with $O(b+\log n)$ bits per word.

- Lower bound is $s^k-1$
- Previous bound: $s^k-1 \left(\frac{\log n}{\epsilon}\right)^{O(k)}$ [Woodruff-Zhang12]
- Streaming space complexity is $n^{1-2/k}$

Main idea of algorithm: sample elements within a server according to higher moments.
Warm-up: 2 servers, third moment

Goal: estimate $\sum_i (u_i + v_i)^3$

1. Estimate $\sum_i u_i^3$
2. Sample $j$ w.p. $p_j = u_j^3 / \sum_i u_i^3$; announce
3. Second server computes $x = u_j^2 v_j / p_j$
4. Average over many samples.

$E(X) = \sum_i u_i^2 v_i$
Warm-up: 2 servers, third moment

**Goal: estimate** \( \sum_i v_i^3 \sum_j u_j^3 \) \( \sum_i u_i v_i \)

\[
p_j = u_j^3 / \sum_i u_i^3 \quad X = u_j^2 v_j / p_j \quad E(X) = \sum_i u_i^2 v_i
\]

\[Var(X) \leq \sum_i v_i > 0 \quad (u_i^2 v_i^3) \quad \leq \sum_i u_i^3 \quad \sum_i u_i v_i^2 \]

\[\leq (\sum_i u_i^3 + v_i^3) \quad \]

So, \( O(1/\varepsilon^2) \) samples suffice.
Many servers, k’th moment

**Goal:** \[ \sum_{i=1}^{n} \left( \sum_{j=1}^{d} f_{ij} \right)^{k} \]

\[ = \sum_{i} \sum_{v_{1}, \ldots, v_{d}} \binom{k}{v_{1}, v_{2}, \ldots, v_{d}} \prod_{j} f_{ij}^{v_{j}} \]

\[ \sum v_{j} = k \]

\[ = \sum_{v_{1}, \ldots, v_{d}} \binom{k}{v_{1}, \ldots, v_{d}} \prod_{i} \sum_{j} f_{ij}^{v_{j}} \]
Many servers, k’th moment

\[
\text{Goal}: \quad \sum_{i} f_{i1}^{r_1} \cdots f_{im}^{r_m} \quad \left( \sum_{j} r_j = k \right)
\]

Each server j:

- Sample i w. prob \( p_{\downarrow i} = \frac{f_{\downarrow ij} \uparrow k}{\sum_{t} f_{\downarrow tj} \uparrow k} \) according to k’th moment.
- Every j’ sends \( f_{\downarrow ij} \uparrow \) if j’ < j and \( f_{\downarrow ij} \uparrow < f_{\downarrow ij} \) or j’ > j and \( f_{\downarrow ij} \uparrow \leq f_{\downarrow ij} \)
- Server j computes \( X_{\downarrow i} = \prod_{j=1}^{s} f_{\downarrow ij} \downarrow r_{\downarrow j} / p_{\downarrow i} \)
Many servers, k’th moment

Each server j:
- Sample i w. prob \( p_i = \frac{f_{ij}}{\sum t f_{tj}} \) according to k’th moment.
- Every j’ sends \( f'_{ij} \) if j’ < j and \( f'_{ij} < f_{ij} \)
  or j’ > j and \( f'_{ij} \leq f_{ij} \)
- Server j computes \( X_i = \prod_{j=1}^s f_{ij} r_j / p_i \)

Lemma. \( E(X) = \sum R_j \prod f_{ij} r_j \) and \( Var(X) \leq (\sum f_{ij} k)^2 \)

Theorem follows as there are \( <s^k \) terms in total.
Counting homomorphisms

- How many copies of graph $H$ in large graph $G$?

- E.g., $H = \text{triangle, 4-cycle, complete bipartite etc.}$

- Linear lower bounds for counting 4-cycles, triangles.

- We assume an (arbitrary) partition of the vertices among servers.
Counting homomorphisms

- To count number of paths of length 2, in a graph with degrees $d_{\downarrow 1}$, $d_{\downarrow 2}$, ..., $d_{\downarrow n}$, we need:

  $$t(K_{\downarrow 1,2}, G) = \sum_{i=1}^{\uparrow n} n^{d_{\downarrow i}/2}$$

  This is a polynomial in frequency moments!

- #stars is $t(K_{\downarrow 1,r}, G) = \sum_{i=1}^{\uparrow n} n^{d_{\downarrow i}/r}$

- #C4's: let $d_{\downarrow ij}$ is the number of common neighbors of $i$ and $j$. Then,

  $$t(C_{\downarrow 4}, G) = \sum_{i=1}^{\uparrow n} n^{d_{\downarrow ij}/2}$$

- #K_{\downarrow a,b}: let $d_{\downarrow S}$ be the number of common neighbors of a set of vertices $S$. Then,

  $$t(K_{\downarrow a,b}, G) = \sum_{S \subseteq V, |S| = a} n^{d_{\downarrow S}/b}$$
Low-rank approximation

Given \( n \times d \) matrix \( A \) partitioned arbitrarily as
\[
A = A_{\downarrow 1} + A_{\downarrow 2} + \ldots + A_{\downarrow s}
\]
among \( s \) servers, find \( A \) of rank \( k \) s.t.
\[
\|A - A\|_F \leq (1 + \epsilon) \text{OPT}.
\]

To avoid linear communication, on each server \( t \), we leave a matrix \( A_{\downarrow t} \), s.t.
\[
A = A_{\downarrow 1} + A_{\downarrow 2} + \ldots + A_{\downarrow s}
\]
and is of rank \( k \).

How to compute these matrices?
Low-rank approximation in the cloud

Thm. [KVW13]. Low-rank approximation of $n \times d$ matrix $A$ partitioned arbitrarily among $s$ servers takes $O^* (skd)$ communication.
Warm-up: row partition

- Full matrix $A$ is $n \times d$ with $n \gg d$.
- Each server $j$ has a subset of rows $A_{\downarrow j}$

- Computes $A_{\downarrow j}^\top A_{\downarrow j}$ and sends to server 1.
- Server 1 computes $B = \sum_{j=1}^{s} A_{\downarrow j}^\top A_{\downarrow j}$ and announces $V$, the top $k$ eigenvectors of $B$.
- Now each server $j$ can compute $A_{\downarrow j} VV^\top$.

- Total communication $= O(sd^2)$.
Low-rank approximation: arbitrary partition

- To extend this to arbitrary partitions, we use limited-independence random projection.

- Subspace embedding: matrix $P$ of size $O(d/\epsilon^2) \times n$ s.t. for any $x \in \mathbb{R}^d$, $\|PAx\| = (1 \pm \epsilon)\|Ax\|$.

- Agree on projection $P$ via a random seed
- Each server computes $PA_\downarrow t$, sends to server 1.
- Server 1 computes $PA = \sum t \uparrow \bigwedge PA_\downarrow t$ and its top $k$ right singular vectors $V$.
- Project rows of $A$ to $V$.

- Total communication = $O(sd/\epsilon^2)$.
Low-rank approximation: arbitrary partition

- Agree on projection \( P \) via a random seed
- Each server computes \( PA\downarrow t \), sends to server 1.
- Server 1 computes \( PA = \sum t\uparrow \overline{\times} PA\downarrow t \) and its top \( k \) right singular vectors \( V \).
- Project rows of \( A \) to \( V \).

Thm. \( \|A - AVV^T\| \leq (1 + O(\epsilon))OPT \).

Pf. Extend \( V \) to a basis \( v\downarrow 1, v\downarrow 2, \ldots, v\downarrow d \). Then,

\[
\|A - AVV^T\| \downarrow F^2 = \sum_{i=k+1}^d \|Av_i\|_2^2 \leq (1+\epsilon)\sum_{i=k+1}^d \|PAv_i\|_2^2 \leq (1+\epsilon)\sum_{i=k+1}^d \|Pu_i\|_2^2 = (1+O(\epsilon))OPT\downarrow 2.
\]
Low-rank approximation in the cloud

To improve to $O(\text{skd})$, we use a subspace embedding up front, and observe that $O(k)$-wise independence suffices for the random projection matrix.

- Agree on $O(k/\epsilon) \times n$ matrix $S$ and $O(k/\epsilon^2) \times n$ matrix $P$.
- Each server computes $SA_{\downarrow t}$ and sends to server 1.
- $S1$ computes $SA=\sum t_{\uparrow}SA_{\downarrow t}$ and an orthonormal basis $U^T$ for its row space.
- Apply previous algorithm to $AU$. 
K-means clustering

- Find a set of k centers $c_1, c_2, ..., c_k$ that minimize
  $\sum_{i \in S} \min_{j=1}^{k} \|A_i - c_j\|_2$

- A near-optimal (i.e. $1 + \epsilon$) solution could be very different!

- So, cannot project up front to reduce dimension and approximately preserve distances.
K-means clustering

- **Kannan-Kumar condition:**
  - Every pair of cluster centers are \( f(k) \) standard deviations apart.
  - “variance”: maximum over 1-d projections, of the average squared distance of a point to its center.
    (e.g. for Gaussian mixtures, max directional variance)

- **Thm. [Kannan-Kumar10].** Under this condition, projection to the top \( k \) principal components followed by the k-means iteration starting at an approximately optimal set of centers finds a nearly correct clustering.

- Finds centers close to the optimal ones, so that the induced clustering is same for most point.
K-means clustering in the cloud

- Points (rows) are partitioned among servers
- Low-rank approximation to project to SVD space.

- How to find a good starting set of centers?
- Need a constant-factor approximation.

- Thm [Chen]. There exists a small subset ("core") s.t. the k-means value of this set (weighted) is within a constant factor of the k-means value of the full set of points (for any set of centers!).
- Chen’s algorithm can be made nimble.

**Thm.** K-means clustering in the cloud achieves the Kannan-Kumar guarantee with $O(d^2 + k^4)$ communication on $s = O(1)$ servers.
Cloud computing: What problems have nimble algorithms?

- Approximate flow/matching?
- Linear programs
- Which graph properties/parameters can be checked/estimated in the cloud? (e.g., planarity? expansion? small diameter?)
- Other Optimization/Clustering/
- Learning problems
  [Balcan-Blum-Fine-Mansour12, Daume-Philips-Saha-Venkatasubramaian12]
- Random partition of data?
- Connection to property testing?