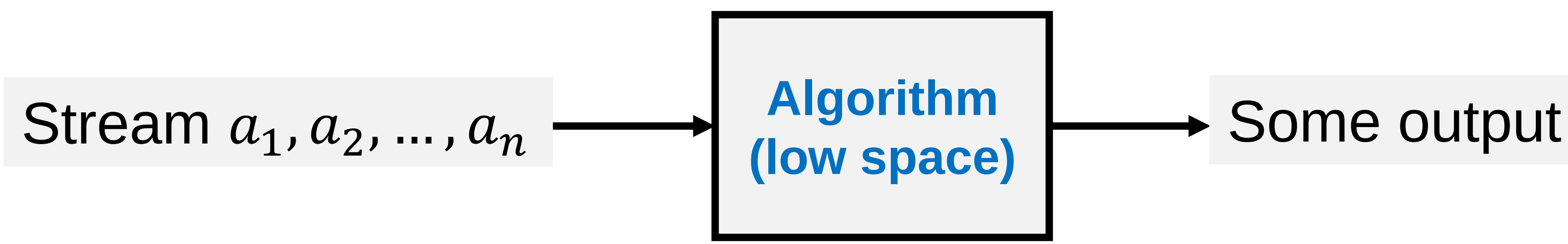
Algorithms for Massive Data Problems

longhuan@sjtu.edu.cn

Massive data problem 1

- The input data is too large to be stored in random access memory.
- Streaming model: data arrive one at a time.
- E.g. $a_1, a_2, \ldots, a_n$ where $a_i \in [1, m]$ and $m = 2^b$ are IP addresses. We would prefer algorithm polynomial in b and $\log n$.



Example: Random sampling 'on the fly'

- A stream $a_1, a_2, \ldots, a_n$.
- To select an index i with probability proportional to the value of a_i.

Sleeping Experts algorithm: (sum of the a_i's seen so far, index i selected with probability $\frac{a_i}{s}$)

- Initially $(s = a_1, i = 1)$
- When the data a_{j+1} comes

  $i \rightarrow j+1$ with probability $\frac{a_{j+1}}{s+a_{j+1}}$;

  $s = s + a_{j+1}$.

Frequency Moments of Data Streams

- A stream $a_1, a_2, \ldots, a_n$, where $a_i \in [1, m]$.
- n, m are very large.
- Goal: determine the number of distinct a_i in the sequence.

Traditional algorithm

- $O(m)$ space, or
- $O(n \log m)$ space

Goal: $O(\log n + \log m)$

- Impossible for deterministic algorithm.
- Trick: randomization and approximation.

Frequency Moments of Data Streams

- A stream $a_1, a_2, \ldots, a_n$, where $a_i \in [1, m]$.
- n, m are very large.
- Goal: determine the number of distinct a_i in the sequence.

Traditional algorithm

- $O(m)$ space, or
- $O(n \log m)$ space

Goal: $O(\log n + \log m)$

- <u>Impossible for deterministic algorithm.</u>
- Trick: randomization and approximation.

Frequency Moments of Data Streams

- A stream $a_1, a_2, \ldots, a_n$, where $a_i \in [1, m]$.
- n, m are very large.
- To determine the number of distinct a_i in the sequence.

✓ Suppose $n \geq m + 1$,

✓ Suppose the Algorithm uses $< m$ bits of memory on all inputs.

There will be a contradiction!

Beat the Lower Bound

The set S of distinct element

- Suppose it's chosen u.a.r. from $\{1, \ldots, m\}$.
- Let min be the minimum element in S.

The expected value of min is: $\frac{m}{|s|+1}$

Thus $|S| \approx \frac{m}{min} - 1$

Keeping track of min in $O(\log m)$ space!

Beat the Lower Bound

In general, the set S is *not* chosen u.a.r.

Hash function:

$$h: \{1,2,\ldots,m\} \to \{0,1,2,\ldots,M-1\}.$$

Keep track of the minimum *hash value*.

Hash function family!

2-Universaly/Pairwise Independent Hash Functions

A set of hash functions

$$H = \{h \mid h: \{1,2, \dots, m\} \to \{0,1,2, \dots, M-1\}\}$$

is 2-universal or pairwise independent if for all x and y in $\{1,2, \dots, m\}$ with $x \neq y$, $h(x)$ and $h(y)$ are

① each equally likely to be any element of $\{0,1,2, \dots, M-1\}$, and

② are statistically independent.

For all w, z :

$$Prob_{h \sim H}\,(h(x) = w \wedge h(y) = z) = \frac{1}{M^2}.$$

2-Universaly/Pairwise Independent Hash Functions

M is a prime number greater than m. For each pair of integers a and b in the range $[0, M-1]$, define a hash function

$$h_{ab}(x) = ax + b \pmod{M}$$

Then $H = \{h_{ab} \mid a, b \in [0, M-1]\}$ is 2-universal.

Distinct Element Counting Algorithm

Let $b_1, b_2, \ldots, b_d$ be the distinct values appearing in input.

① Select h from H,

② $S = \{h(b_1), h(b_2), \ldots, h(b_d)\}$ is a set of d random and pairwise independent values from the set $\{0, 1, 2, \ldots, M-1\}$.

③ $min = \min(S)$

Claim: $d \approx \frac{M}{min}$

M is a prime number greater than m. For each pair of integers a and b in the range $[0, M-1]$, define a hash function

$$h_{ab}(x) = ax + b \pmod{M}$$

Then $H = \{h_{ab} \mid a, b \in [0, M-1]\}$ is 2-universal.

Lemma. With probability at least $\frac{2}{3} - \frac{d}{M}$, we have $\frac{d}{6} \leq \frac{M}{min} \leq 6d$, where min is the smallest element of S.

Number of Occurrences of a Given Element

A stream $a_1, a_2, \ldots, a_n$ where $a_i \in \{0,1\}$. We want to count the number of 1s (m) in this sequence.

We can obviously do this in $\log n$ space. Goal: $\log \log n$.

Strategy: keep a value k such that $2^k \approx m$.

Represent k will need only $\log \log n$ space.

Algorithm.
Initialize $k = 0$
For $i = 1$ to n do
  if $x_i = 0$ then $k = k$;
  if $x_i = 1$ then $k++$ with probability $\frac{1}{2^k}$.
Output $2^k - 1$.

Massive data problem 2

The input is stored in the memory, but because the input is so large, we would need to

① produce a much smaller approximation to it, or

② perform an approximate computation on it in low space.

Example: Matrix Algorithms using Sampling

Algorithms for matrix problems:

- Matrix multiplication
- Low-rank approximation
- SVD
- Compressed representations
- Linear regression
- ……

$O(n^3)$ operations

✓ Pick a random sub-matrix and compute with that.

✓ Pick a subset of columns or rows of the input matrix.

Matrix Multiplication using Sampling

A is an $m \times n$ matrix

B is an $n \times p$ matrix

$A(:,k)$ the k^{th} column of A, a $m \times 1$ matrix;

$B(k,:)$ the k^{th} row of B, a $1 \times p$ matrix.

$$AB = \sum_{k=1}^{n} A(:,k)B(k,:)$$

Sample k from the set $\{1,2,\dots,n\}$ with probability p_k.

Matrix Multiplication using Sampling

A is an $m \times n$ matrix

B is an $n \times p$ matrix

$$AB = \sum_{k=1}^{n} A(:,k)B(k,:)$$

Sample k from the set $\{1,2,\ldots,n\}$ with probability p_k.

Define an associated random matrix variable

$$X = \frac{1}{p_k} A(:,k)B(k,:) \text{ with probability } p_k.$$

$$\boldsymbol{E(X)} = \sum_{k=1}^{n} Prob(z=k)(X=x)$$

$$= \sum_{k=1}^{n} Prob(z=k)\frac{1}{p_k}A(:,k)B(k,:) = \boldsymbol{AB}.$$

Matrix Multiplication using Sampling

Define an associated random matrix variable
$X = \frac{1}{p_k} A(:,k) B(k,:)$ with probability p_k.

$\boldsymbol{E(X) = AB}$

$$Var(X) = \sum_{i=1}^{m} \sum_{j=1}^{p} Var(x_{ij})$$

Minimize
the variance

$$= \sum_{k} \frac{1}{p_k} |A(:,k)|^2 \cdot |B(k,:)|^2 - \|AB\|_F^2$$

$A = B^T$, *length squared sampling*: $\boldsymbol{p_k = \frac{|A(:,k)|^2}{\|A\|_F^2}}$.

Matrix Multiplication using Sampling

Define an associated random matrix variable

$X = \frac{1}{p_k} A(:,k)B(k,:)$ with probability p_k.

Length squared sampling: $\boldsymbol{p_k = \frac{|A(:,k)|^2}{\|A\|_F^2}}$.

Estimate AB:

Do s independent trials. Each trial $i, i = 1,2,\ldots,s$ yields a matrix X_i.

Output $\frac{1}{s}\sum_{i=1}^{s} X_i$

Matrix Multiplication using Sampling

Estimate AB:

Do s independent trials. Each trial $i, i = 1,2,\dots,s$ yields a matrix X_i.

Output $\frac{1}{s}\sum_{i=1}^{s} X_i$

$$\frac{1}{s}\sum_{i=1}^{s} X_i = \frac{1}{s}\left(\frac{A(:,k_1)B(k_1,:)}{p_{k_1}} + \frac{A(:,k_2)B(k_2,:)}{p_{k_2}} + \cdots + \frac{A(:,k_s)B(k_s,:)}{p_{k_s}}\right)$$

$$= \left(\frac{A(:,k_1)}{\sqrt{sp_{k_1}}}, \frac{A(:,k_2)}{\sqrt{sp_{k_2}}}, \dots, \frac{A(:,k_s)}{\sqrt{sp_{k_s}}}\right) \cdot \left(\frac{B(k_1,:)}{\sqrt{sp_{k_1}}}, \frac{B(k_2,:)}{\sqrt{sp_{k_2}}}, \dots, \frac{B(k_s,:)}{\sqrt{sp_{k_s}}}\right)$$

$$= C_{m\times s} \cdot R_{s\times p}$$

Matrix Multiplication using Sampling

Estimate AB:
Do s independent trials. Each trial $i, i = 1,2,\dots,s$ yields a matrix X_i.
Output $\frac{1}{s}\sum_{i=1}^{s} X_i$

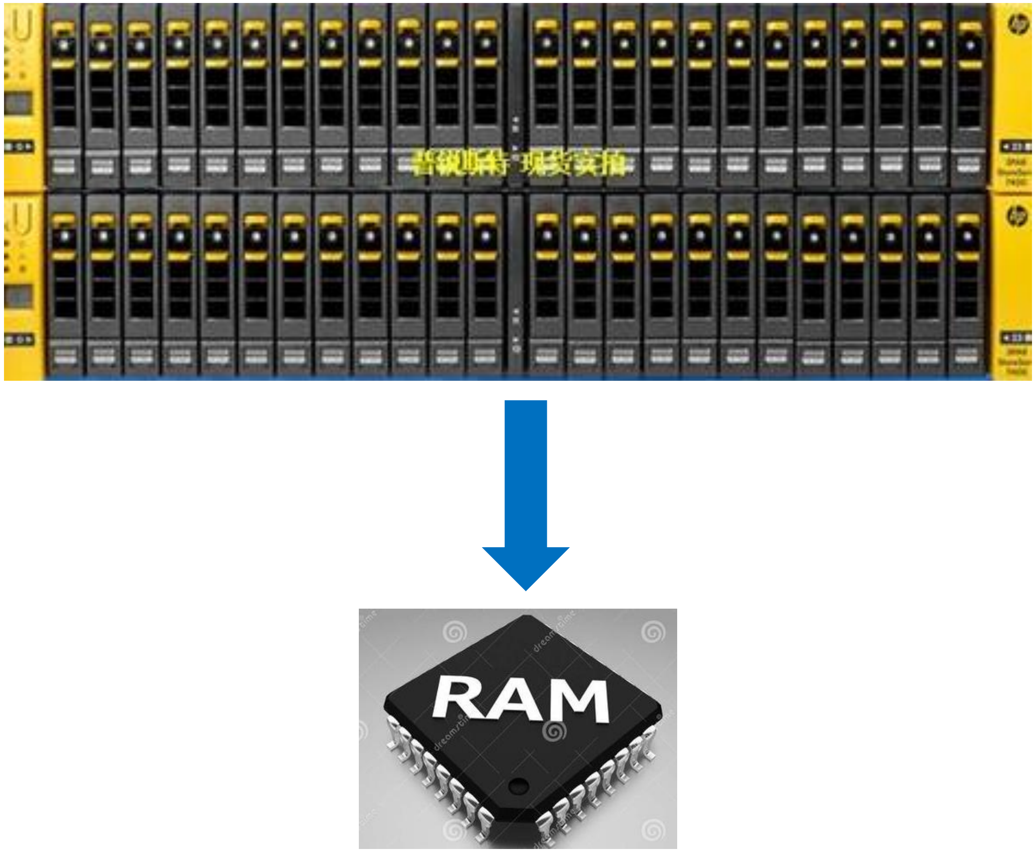
$$AB \approx C_{m\times s} \cdot R_{s\times p} = \left(\frac{A(:,k_1)}{\sqrt{sp_{k_1}}}, \frac{A(:,k_2)}{\sqrt{sp_{k_2}}}, \cdots, \frac{A(:,k_s)}{\sqrt{sp_{k_s}}}\right) \cdot \left(\frac{B(k_1,:)}{\sqrt{sp_{k_1}}}, \frac{B(k_2,:)}{\sqrt{sp_{k_2}}}, \cdots, \frac{B(k_s,:)}{\sqrt{sp_{k_s}}}\right)$$

Theorem. Suppose A is an $m \times n$ matrix and B is an $n \times p$ matrix. The product AB can be estimated by CR, where C is an $m \times s$ matrix consisting of s columns of A picked according to length-squared distribution and scaled as above, R is the $s \times p$ matrix as above. Then the error is bounded by

$$E(\|AB - CR\|_F^2) \le \frac{\|A\|_F^2 \|B\|_F^2}{s}$$

Thus to ensure $E(\|AB - CR\|_F^2) \le \epsilon^2 \|A\|_F^2 \|B\|_F^2$, it suffices to make s greater than or equal to $\frac{1}{\epsilon^2}$. If ϵ is $\Omega(1)$, so is $s \in O(1)$, then the multiplication CR can be carried out in time $O(mp)$.

Implementing Length Squared Sampling



Sketch of a Large Matrix

- Interpolative approximation
- SVD