Mapreduce With Parallelizable Reduce

S. *Muthu* Muthukrishnan

Some Premises

- At a deliberately high level, we know the MapReduce system.

Some Premises

- At a deliberately high level, we know the MapReduce system.
  - Parallel. Map and Reduce functions. Used when data is large. Changing system.

Some Premises

- At a deliberately high level, we know the MapReduce system.
  - Parallel. Map and Reduce functions. Used when data is large. Changing system.
- There is nice PRAM theory of parallel algorithms.

Some Premises

- At a deliberately high level, we know the MapReduce system.
  - Parallel. Map and Reduce functions. Used when data is large. Changing system.
- There is nice PRAM theory of parallel algorithms.
  - NC, prefix sums, list ranking, and more.

Some Premises

- At a deliberately high level, we know the MapReduce system.
  - Parallel. Map and Reduce functions. Used when data is large. Changing system.
- There is nice PRAM theory of parallel algorithms.
  - NC, prefix sums, list ranking, and more.
- Goal: Develop a useful theory of MapReduce algorithms.

Some Premises

- At a deliberately high level, we know the MapReduce system.
  - Parallel. Map and Reduce functions. Used when data is large. Changing system.
- There is nice PRAM theory of parallel algorithms.
  - NC, prefix sums, list ranking, and more.
- Goal: Develop a useful theory of MapReduce algorithms.
  - An algorithmus role. Interesting problems, algorithms. Bridge from the other side.

Thoughts Circa 2006

- Prefix sum in $O(1)$ rounds.

Thoughts Circa 2006

- Prefix sum in $O(1)$ rounds.
  - Problem: $A[1, \ldots, n] \Rightarrow PA[1, \cdots, n]$ where $PA[i] = \sum_{j \leq i} A[j]$.

Thoughts Circa 2006

- Prefix sum in $O(1)$ rounds.
  - Problem: $A[1, \ldots, n] \Rightarrow PA[1, \cdots, n]$ where $PA[i] = \sum_{j \leq i} A[j]$.
  - Solution:
    - Assign $A[i\sqrt{n} + 1, \cdots, (i+1)\sqrt{n}]$ to key i.

Thoughts Circa 2006

- Prefix sum in $O(1)$ rounds.
  - Problem: $A[1,\ldots,n] \Rightarrow PA[1,\cdots,n]$ where $PA[i] = \sum_{j\leq i} A[j]$.
  - Solution:
    - Assign $A[i\sqrt{n}+1,\cdots,(i+1)\sqrt{n}]$ to key i.
    - Solve problem on $B[1,\sqrt{n}]$ with one proc, $B[i] = \sum_{i\sqrt{n}+1}^{(i+1)\sqrt{n}} A[j]$. Doable?

Thoughts Circa 2006

- Prefix sum in $O(1)$ rounds.
  - Problem: $A[1, \ldots, n] \Rightarrow PA[1, \cdots, n]$ where $PA[i] = \sum_{j \leq i} A[j]$.
  - Solution:
    - Assign $A[i\sqrt{n} + 1, \cdots, (i+1)\sqrt{n}]$ to key i.
    - Solve problem on $B[1, \sqrt{n}]$ with one proc, $B[i] = \sum_{i\sqrt{n}+1}^{(i+1)\sqrt{n}} A[j]$. Doable?
    - Solve problem for key i with $PB[i-1]$. Doable?

Thoughts Circa 2006

- Prefix sum in $O(1)$ rounds.
  - Problem: $A[1, \ldots, n] \Rightarrow PA[1, \cdots, n]$ where $PA[i] = \sum_{j \leq i} A[j]$.
  - Solution:
    - Assign $A[i\sqrt{n}+1, \cdots, (i+1)\sqrt{n}]$ to key i.
    - Solve problem on $B[1, \sqrt{n}]$ with one proc, $B[i] = \sum_{i\sqrt{n}+1}^{(i+1)\sqrt{n}} A[j]$. Doable?
    - Solve problem for key i with $PB[i-1]$. Doable?
- List ranking in $O(1)$ rounds?
  - Some graph algorithms in $O(1)$ rounds recently.

SIRCOCCO Challenge

- Problem: Given graph $G = (V, E)$, count the number of triangles.[1]

[1] For ex, see. Fast Counting of Triangles in Large Real Networks without counting: Algorithms and Laws, ICDM 08, by C. Tsourakakis.

SIROCCO Challenge

- Problem: Given graph $G = (V, E)$, count the number of triangles.[1]
- Solution:
  - For each edge (u, v), generate a tuple $(u, v, 0)$.
  - For each vertex v and for each pair of neighbors x, z of v, generate a tuple $(x, z, 1)$.
  - Presence of both 0 and 1 tuple for an edge is a triangle.

[1] For ex, see. Fast Counting of Triangles in Large Real Networks without counting: Algorithms and Laws, ICDM 08, by C. Tsourakakis.

SIROCCO Challenge

- Problem: Given graph $G = (V, E)$, count the number of triangles.[1]
- Solution:
  - For each edge (u, v), generate a tuple $(u, v, 0)$.
  - For each vertex v and for each pair of neighbors x, z of v, generate a tuple $(x, z, 1)$.
  - Presence of both 0 and 1 tuple for an edge is a triangle.
- Solution: The number of triangles is $\frac{\sum_i \lambda_i^3}{6}$ where λ_i are eigenvalues of adjacency matrix A of G in sorted order.
  - A_{ii}^3 is the number of triangles involving i.
  - The trace is 6 times the number of triangles.
  - If λ is eigenvalue of A, ie., $Ax = \lambda x$, then λ^3 is eigenvalue of A^3.
  - In practice, computing top few eigenvalues suffices.

[1] For ex, see. Fast Counting of Triangles in Large Real Networks without counting: Algorithms and Laws, ICDM 08, by C. Tsourakakis.

Eigenvalue Estimation

A is a $n \times n$ real valued matrix.

- Lanczos method.

Eigenvalue Estimation

A is a $n \times n$ real valued matrix.

- Lanczos method.
- Sketches. Ar for pseudo random $n \times d$ vector r, $d << n$. Will $O(nd)$ sketch fit into one machine?

Special Case

Motivation: Logs processing.

```
x = inputrecord;
x-squared = x * x;
aggregator: table sum;
emit aggregator <- x-squared;
```

MUD Algorithm $m = (\Phi, \oplus, \eta)$.

- Local function $\Phi : \Sigma \to Q$ maps input item to a message.
- Aggregator $\oplus : Q \times Q \to Q$ maps two messages to a single message.
- Post-processing operator $\eta : Q \to \Sigma$ produces the final output, applying $m_{\mathcal{T}}(\mathbf{x})$.
- Computes a function f if $\eta(m_{\mathcal{T}}(\cdot)) = f$ for all trees $\mathcal{T}$.

MUD Examples

$$\boxed{\begin{array}{l}\Phi(x) = \langle x, x\rangle \\ \oplus(\langle a_1, b_1\rangle, \langle a_2, b_2\rangle) = \langle \min(a_1, a_2), \max(b_1, b_2)\rangle \\ \eta(\langle a, b\rangle) = b - a\end{array}}$$

Figure: mud algorithm for computing the total span (left)

MUD Examples

$$\Phi(x) = \langle x, h(x), 1\rangle$$

$$\oplus(\langle a_1, h(a_1), c_1\rangle, \langle a_2, h(a_2), c_2\rangle) = \begin{cases} \langle a_i, h(a_i), c_i\rangle & \text{if } h(a_i) < h(a_j) \\ \langle a_1, h(a_1), c_1 + c_2\rangle & \text{otherwise} \end{cases}$$

$$\eta(\langle a, b, c\rangle) = a \text{ if } c = 1$$

Figure: Mud algorithms for computing a uniform random sample of the unique items in a set (right). Here h is an approximate minwise hash function.

Streaming

- streaming algorithm $s = (\sigma, \eta)$.
- operator $\sigma : Q \times \Sigma \to Q$
- $\eta : Q \to \Sigma$ converts the final state to the output.
- On input $\mathbf{x} \in \Sigma^n$, the streaming algorithm computes $f = \eta(s^0(\mathbf{x}))$, where 0 is the starting state, and $s^q(\mathbf{x}) = \sigma(\sigma(\ldots\sigma(\sigma(q, x_1), x_2), \ldots, x_{k-1}), x_k)$.
- Communication complexity is $\log|Q|$

MUD vs Streaming

- For a mud algorithm $m = (\Phi, \oplus, \eta)$, there is a streaming algorithm $s = (\sigma, \eta)$ of the same complexity with same output, by setting $\sigma(q, x) = \oplus(q, \Phi(x))$.

MUD vs Streaming

- For a mud algorithm $m = (\Phi, \oplus, \eta)$, there is a streaming algorithm $s = (\sigma, \eta)$ of the same complexity with same output, by setting $\sigma(q, x) = \oplus(q, \Phi(x))$.
- Central question: Can MUD simulate streaming?

MUD vs Streaming

- For a mud algorithm $m = (\Phi, \oplus, \eta)$, there is a streaming algorithm $s = (\sigma, \eta)$ of the same complexity with same output, by setting $\sigma(q, x) = \oplus(q, \Phi(x))$.
- Central question: Can MUD simulate streaming?
  - Count the occurrences of the first odd number on the stream.

MUD vs Streaming

- For a mud algorithm $m = (\Phi, \oplus, \eta)$, there is a streaming algorithm $s = (\sigma, \eta)$ of the same complexity with same output, by setting $\sigma(q, x) = \oplus(q, \Phi(x))$.
- Central question: Can MUD simulate streaming?
  - Count the occurrences of the first odd number on the stream.
  - Symmetric problems? Symmetric index problem.

$$\begin{aligned} S &= (\mathsf{a}, 1, x_1, p), (\mathsf{a}, 2, x_2, p), \ldots, (\mathsf{a}, 2, x_n, p), \\ &\quad (\mathsf{b}, 1, y_1, q), (\mathsf{b}, 2, y_2, q), \ldots, (\mathsf{b}, 2, y_n, q). \end{aligned}$$

  Additionally, we have $x_q = y_p$. Compute function $f(S) = x_q$.

MUD vs Streaming

For any symmetric function $f : \Sigma^n \to \Sigma$ computed by a $g(n)$-space, $c(n)$-communication streaming algorithm (σ, η), with $g(n) = \Omega(\log n)$ and $c(n) = \Omega(\log n)$,

MUD vs Streaming

For any symmetric function $f : \Sigma^n \to \Sigma$ computed by a $g(n)$-space, $c(n)$-communication streaming algorithm (σ, η), with $g(n) = \Omega(\log n)$ and $c(n) = \Omega(\log n)$,

there exists a $O(c(n))$-communication, $O(g^2(n))$-space mud algorithm $(\Phi, \oplus, \eta)$ that also computes f.

MUD vs Streaming: 2 parties

- $\mathbf{x}_A$ and $\mathbf{x}_B$ are partitions of the input sequence $\mathbf{x}$ sent to Alice and Bob.

MUD vs Streaming: 2 parties

- $\mathbf{x}_A$ and $\mathbf{x}_B$ are partitions of the input sequence $\mathbf{x}$ sent to Alice and Bob.
- Alice runs the streaming algorithm on her input sequence to produce the state $q_A = s^0(\mathbf{x}_A)$, and sends this to Carol. Similarly, Bob sends $q_B = s^0(\mathbf{x}_B)$ to Carol.

MUD vs Streaming: 2 parties

- $\mathbf{x}_A$ and $\mathbf{x}_B$ are partitions of the input sequence $\mathbf{x}$ sent to Alice and Bob.
- Alice runs the streaming algorithm on her input sequence to produce the state $q_A = s^0(\mathbf{x}_A)$, and sends this to Carol. Similarly, Bob sends $q_B = s^0(\mathbf{x}_B)$ to Carol.
- Carol receives the states q_A and q_B, which contain the sizes n_A and n_B of the input sequences $\mathbf{x}_A$ and $\mathbf{x}_B$, and needs to calculate $f = s^0(\mathbf{x}_A || \mathbf{x}_B)$.

2 Parties Communication

- Carol finds sequences $\mathbf{x}'_A$ and $\mathbf{x}'_B$ of length n_A and n_B such that $q_A = s^0(\mathbf{x}'_A)$ and $q_B = s^0(\mathbf{x}'_B)$.

2 Parties Communication

- Carol finds sequences $\mathbf{x}'_A$ and $\mathbf{x}'_B$ of length n_A and n_B such that $q_A = s^0(\mathbf{x}'_A)$ and $q_B = s^0(\mathbf{x}'_B)$.
- Carol then outputs $\eta(s^0(\mathbf{x}'_A \cdot \mathbf{x}'_B))$.

$$
\begin{aligned}
\eta(s^0(\mathbf{x}'_A \cdot \mathbf{x}'_B)) &= \eta(s^0(\mathbf{x}_A \cdot \mathbf{x}'_B)) \\
&= \eta(s^0(\mathbf{x}'_B \cdot \mathbf{x}_A)) \\
&= \eta(s^0(\mathbf{x}_B \cdot \mathbf{x}_A)) \\
&= \eta(s^0(\mathbf{x}_A \cdot \mathbf{x}_B)) \\
&= f(\mathbf{x}_A \cdot \mathbf{x}_B) \\
&= f(\mathbf{x}).
\end{aligned}
$$

Space Efficient 2 Party Communication

- Non-deterministic simulation:

Space Efficient 2 Party Communication

- Non-deterministic simulation:
  - First, guess the symbols of $\mathbf{x}'_A$ one at a time, simulating the streaming algorithm $s^0(\mathbf{x}'_A)$ on the guess.

Space Efficient 2 Party Communication

- Non-deterministic simulation:
  - First, guess the symbols of $\mathbf{x}'_A$ one at a time, simulating the streaming algorithm $s^0(\mathbf{x}'_A)$ on the guess. If after n_A guessed symbols we have $s^0(\mathbf{x}'_A) \neq q_A$, reject this branch.

Space Efficient 2 Party Communication

- Non-deterministic simulation:
  - First, guess the symbols of $\mathbf{x}'_A$ one at a time, simulating the streaming algorithm $s^0(\mathbf{x}'_A)$ on the guess. If after n_A guessed symbols we have $s^0(\mathbf{x}'_A) \neq q_A$, reject this branch. Then, guess the symbols of $\mathbf{x}'_B$, simulating (in parallel) $s^0(\mathbf{x}'_B)$ and $s^{q_A}(\mathbf{x}'_B)$.

Space Efficient 2 Party Communication

- Non-deterministic simulation:
  - First, guess the symbols of $\mathbf{x}'_A$ one at a time, simulating the streaming algorithm $s^0(\mathbf{x}'_A)$ on the guess. If after n_A guessed symbols we have $s^0(\mathbf{x}'_A) \neq q_A$, reject this branch. Then, guess the symbols of $\mathbf{x}'_B$, simulating (in parallel) $s^0(\mathbf{x}'_B)$ and $s^{q_A}(\mathbf{x}'_B)$. If after n_B steps we have $s^0(\mathbf{x}'_B) \neq q_B$, reject this branch; otherwise, output $q_C = s^{q_A}(\mathbf{x}'_B)$.
  - This procedure is a non-deterministic, $O(g(n))$-space algorithm for computing a valid q_C.
- By Savitch's theorem, it follows that there is a deterministic, $g^2(n)$-space algorithm.
- Simulation time is superpolynomial.

Proof

- Finish the proof for arbitrary computation tree inductively.
- Extends to streaming algorithms for approximating f that work by computing some other function g exactly over the stream, for example, sketch-based algorithms that maintain $c_i = \langle \mathbf{x}, \mathbf{v}_i \rangle$ where $\mathbf{x}$ is the input vector and some $\mathbf{v}_i$. Public randomness.
- Doesn't extend to randomized algorithms with private randomness, partial functions, etc.

Multiple Keys

- Any N-processor, M-memory, T-time EREW-PRAM algorithm which has a $\log(N+M)$-bit word in every memory location,
  can be simulated by a $O(T)$-round, $(N+M)$-key mud algorithm with communication complexity $O(\log(N+M))$ bits per key.
- In particular, any problem in class NC has a $\text{polylog}(n)$-round, $\text{poly}(n)$-key mud algorithm with communication complexity $O(\log(n))$ bits per key.

Concluding Remarks

- Jon Feldman, S. Muthukrishnan, Anastasios Sidiropoulos, Clifford Stein, Zoya Svitkina: On distributing symmetric streaming computations. SODA 2008: 710-719