Small and Sweet MapReduce Algorithms

Yufei Tao

Chinese University of Hong Kong

MapReduce

A platform for massive parallel computation.

- First (formal) paper in 2004.
- Now a large collection of algorithms on this platform.

This talk:

Principles in the design and analysis of algorithms in MapReduce (and other similar platforms).

This talk:
Principles in the design and analysis of algorithms in MapReduce (and other similar platforms).

Ultimate goals:

- Small: Simple enough for practical implementation
- Sweet: With non-trivial theoretical guarantees.

Computation Model

This is the starting point before any meaningful analysis.

1. MRC model [SODA'10]
2. BSP-like model [ISSAC'11]

3. Minimality Model [SIGMOD'13]
4. Massively Parallel Computation (MPC) [PODS'13].

Computation Model

This talk:

3. Minimality Model [SIGMOD'13]
    - A very stringent model – suitable for studying "easy" problems.
4. Massively Parallel Computation (MPC) [PODS'13].
    - A more relaxed model – suitable for studying "hard" problems.

Minimality Model

Input and Machines

p = number of machines.
n = number of elements in the input (i.e., dataset)
Assumption: $p \leq \sqrt{\frac{n}{2 \ln n}}$

At the beginning, $O(n/p)$ elements per machine.

- The algorithm has no control over how the elements are initially distributed.

Minimality Model

Superstep

Two phases:

1. Machines message each other.

Minimality Model

Superstep

Two phases:

1. Machines message each other.
2. Machines perform local computation (all operations in the RAM model allowed).

Minimality Model

Superstep

Two phases:

1. Machines message each other.
2. Machines perform local computation (all operations in the RAM model allowed).

Algorithm

An algorithm is a sequence of supersteps.

Minimality Model

Consider a computational problem Π. Suppose that it can be solved in $f(n) = \Omega(n)$ time sequentially in the RAM model.

Minimality Model

Consider a computational problem Π. Suppose that it can be solved in $f(n) = \Omega(n)$ time sequentially in the RAM model.

Π is minimal if it admits an algorithm satisfying all:

1. It has $O(1)$ supersteps.
2. It uses $O(n/p)$ space at all times.
3. Every machine sends $O(n/p)$ words in total.
4. Every machine incurs $O(f(n/p))$ CPU time in total.

The algorithm is called a minimal algorithm for Π.

Minimality Model

Requirement 1: $O(1)$ supersteps.

Minimality Model

Requirement 1: $O(1)$ supersteps.

Requirement 2: $O(n/p)$ space at all times.
Balanced during the entire algorithm.

Minimality Model

Requirement 1: $O(1)$ supersteps.

Requirement 2: $O(n/p)$ space at all times.
Balanced during the entire algorithm.

Requirement 3: $O(n/p)$ words sent and received.
Asymptotically the same time for each machine to look at its portion of the input.

Minimality Model

Requirement 4: $O(f(n/p))$ CPU time for every machine.

- The MapReduce algorithm must improve automatically whenever a faster sequential algorithm is discovered!
  - Implies that the MapReduce algorithm must utilize a sequential algorithm as a black box!

Minimality Model

Requirement 4: $O(f(n/p))$ CPU time for every machine.

- The MapReduce algorithm must improve automatically whenever a faster sequential algorithm is discovered!
  - Implies that the MapReduce algorithm must utilize a sequential algorithm as a black box!

- The best you can do!
  - If you can achieve $o(f(n/p))$ on MapReduce, you can simulate the same algorithm in the RAM model in $o(p \cdot f(n/p))$ time.

Minimal Algorithm for Sorting

At the beginning: n elements on p machines.

At the end: it is possible to label the machines from 1 to p such that all elements on machine i are smaller than those on machine j, for any $i < j$.

Minimal Algorithm for Sorting

At the beginning: n elements on p machines.
At the end: it is possible to label the machines from 1 to p such that all elements on machine i are smaller than those on machine j, for any $i < j$.

> **Remark (illustration of Requirement 4)**
>
> - Comparison-based sort: $\Theta(n \log n)$ sequentially $\Rightarrow$ $O(\frac{n}{p} \log \frac{n}{p})$ MapReduce .
> - Sorting n integers: $O(n \log \log n)$ sequentially $\Rightarrow$ $O(\frac{n}{p} \log \log \frac{n}{p})$ MapReduce.
>
> All by the same algorithm.

Superstep 1

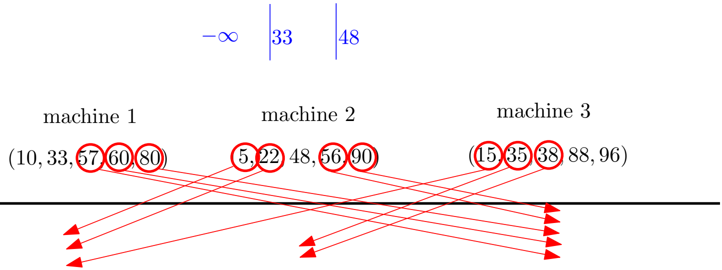
- **Phase 1:** Each machine samples each object with probability $\rho = \frac{p}{n} \ln(nt)$ independently, and sends the samples to all other machines.
- **Phase 2:** Each machine sorts the samples received, and identify the split elements that partition the list into t segments evenly.
  - Machine i will get all the elements in the i-th segment at the end.

machine 1	machine 2	machine 3
$(10, 33, 57, 60, 80)$	$(5, 22, 48, 56, 90)$	$(15, 35, 38, 88, 96)$
$(5, \boxed{33}, 38, \boxed{48}, 80, 88)$	$(5, \boxed{33}, 38, \boxed{48}, 80, 88)$	$(5, \boxed{33}, 38, \boxed{48}, 80, 88)$

Minimal Algorithm for Sorting

Superstep 2

- **Phase 1:** Each machines sends the elements in the i-th segment to machine i.
- **Phase 2:** Machine i sorts the elements in the i-th segment.



Minimal Algorithm for Sorting

Theorem: The above algorithm is minimal with probability at least $1 - O(1/n)$.

Minimality $\Rightarrow$ MPC

For certain problems, if $O(1)$ supersteps are required, it is impossible to ensure that each machine sends $O(n/p)$ words only.
$\Rightarrow$ No minimal algorithms.

Minimality $\Rightarrow$ MPC

For certain problems, if $O(1)$ supersteps are required, it is impossible to ensure that each machine sends $O(n/p)$ words only.
$\Rightarrow$ No minimal algorithms.

Phrased differently, if $O(1)$ supersteps are required, we must allow a machine to send more than $O(n/p)$ words – but how much more?
$\Rightarrow$ The MPC model.

MPC Model

Input and Machines

p = number of machines.
n = number of elements in the input (i.e., dataset)
Assumption: $p \leq n^{1/c}$ for some constant $c > 1$.

At the beginning, $O(n/p)$ elements per machine.

- The algorithm has no control over how the elements are initially distributed.

MPC Model

Superstep

Two phases:

1. Machines message each other.
2. Machines perform local computation (all operations in the RAM model allowed).

Algorithm

An algorithm is a sequence of supersteps.

MPC Model

An algorithm is required to finish in $O(1)$ supersteps.

MPC Model

An algorithm is required to finish in $O(1)$ supersteps.

Communication cost of one machine $=$
Total number of words it communicated (sent and received combined)

MPC Model

An algorithm is required to finish in $O(1)$ supersteps.

Communication cost of one machine $=$
Total number of words it communicated (sent and received combined)

Load of the algorithm $=$
Communication cost of the most expensive machine

MPC Model

An algorithm is required to finish in $O(1)$ supersteps.

Communication cost of one machine $=$
Total number of words it communicated (sent and received combined)

Load of the algorithm $=$
Communication cost of the most expensive machine

Aim: Minimize load.

MPC Model

An algorithm is required to finish in $O(1)$ supersteps.

Communication cost of one machine =
Total number of words it communicated (sent and received combined)

Load of the algorithm =
Communication cost of the most expensive machine

Aim: Minimize load.

Remark 1: CPU time for free.

MPC Model

An algorithm is required to finish in $O(1)$ supersteps.

Communication cost of one machine $=$
Total number of words it communicated (sent and received combined)

Load of the algorithm $=$
Communication cost of the most expensive machine

Aim: Minimize load.

Remark 1: CPU time for free.
Remark 2: Every minimal algorithm must have load $O(n/p)$.

MPC Model

An algorithm is required to finish in $O(1)$ supersteps.

Communication cost of one machine $=$
Total number of words it communicated (sent and received combined)

Load of the algorithm $=$
Communication cost of the most expensive machine

Aim: Minimize load.

Remark 1: CPU time for free.
Remark 2: Every minimal algorithm must have load $O(n/p)$.
Remark 3: Any problem admits an algorithm of load $O(n)$.

MPC Model

The MPC model has largely been used to study problems where the output size can be much larger than n, e.g., joins.

MPC Model

The MPC model has largely been used to study problems where the output size can be much larger than n, e.g., joins.

Focus of this talk: **Cartesian Product**

MPC Model

The MPC model has largely been used to study problems where the output size can be much larger than n, e.g., joins.

Focus of this talk: **Cartesian Product**

R: a set of n **red** elements
S: a set of n **blue** elements

At the beginning: The $2n$ elements are stored on the p machines (each $O(n/p)$ elements).
At the end: Each element in $R \times S$ must appear on some machine.

MPC Lower Bound for Cartesian Product

To warm up, we will prove that, when $p = 2$, any algorithm solving the problem must have load $\Omega(n)$.

$\Rightarrow$ Implication: No better algorithm than simply sending everything over to the other machine.

MPC Lower Bound for Cartesian Product

For simplicity, assume each machine has n elements at the beginning.

Suppose that an algorithm solves the problem with load L.

MPC Lower Bound for Cartesian Product

For simplicity, assume each machine has n elements at the beginning.

Suppose that an algorithm solves the problem with load L.
$\Rightarrow$ Machine 1 sees $n + L$ elements overall.

MPC Lower Bound for Cartesian Product

For simplicity, assume each machine has n elements at the beginning.

Suppose that an algorithm solves the problem with load L.
$\Rightarrow$ Machine 1 sees $n + L$ elements overall.
$\Rightarrow$ It can produce at most

$$\left(\frac{n+L}{2}\right)^2$$

pairs

MPC Lower Bound for Cartesian Product

For simplicity, assume each machine has n elements at the beginning.

Suppose that an algorithm solves the problem with load L.
$\Rightarrow$ Machine 1 sees $n + L$ elements overall.
$\Rightarrow$ It can produce at most

$$\left(\frac{n+L}{2}\right)^2$$

pairs
$\Rightarrow$ Same for machine 2.

MPC Lower Bound for Cartesian Product

For simplicity, assume each machine has n elements at the beginning.

Suppose that an algorithm solves the problem with load L.
$\Rightarrow$ Machine 1 sees $n + L$ elements overall.
$\Rightarrow$ It can produce at most

$$\left(\frac{n+L}{2}\right)^2$$

pairs
$\Rightarrow$ Same for machine 2.
$\Rightarrow$

$$2\left(\frac{n+L}{2}\right)^2 \geq n^2$$

MPC Lower Bound for Cartesian Product

For simplicity, assume each machine has n elements at the beginning.

Suppose that an algorithm solves the problem with load L.
⇒ Machine 1 sees $n + L$ elements overall.
⇒ It can produce at most

$$\left(\frac{n+L}{2}\right)^2$$

pairs
⇒ Same for machine 2.
⇒

$$2\left(\frac{n+L}{2}\right)^2 \geq n^2$$

which solves to $L \geq \frac{2-\sqrt{2}}{2\sqrt{2}} n$.

MPC Lower Bound for Cartesian Product

Next, we will extend the algorithm to prove that, any cartesian product algorithm must have load $\Omega(n/\sqrt{p})$.

$\Rightarrow$ Implication: No minimal algorithm!

MPC Lower Bound for Cartesian Product

Suppose that an algorithm has load L.

MPC Lower Bound for Cartesian Product

Suppose that an algorithm has load L.
$\Rightarrow$ Every machine can receive at most L elements.

MPC Lower Bound for Cartesian Product

Suppose that an algorithm has load L.
$\Rightarrow$ Every machine can receive at most L elements.
$\Rightarrow$ In total, the machine sees $c \cdot n/p + L$ elements during the entire algorithm.

MPC Lower Bound for Cartesian Product

Suppose that an algorithm has load L.
$\Rightarrow$ Every machine can receive at most L elements.
$\Rightarrow$ In total, the machine sees $c \cdot n/p + L$ elements during the entire algorithm.
$\Rightarrow$ The machine can produce at most

$$\left(\frac{c \cdot n/p + L}{2}\right)^2$$

pairs.

MPC Lower Bound for Cartesian Product

Suppose that an algorithm has load L.
$\Rightarrow$ Every machine can receive at most L elements.
$\Rightarrow$ In total, the machine sees $c \cdot n/p + L$ elements during the entire algorithm.
$\Rightarrow$ The machine can produce at most

$$\left(\frac{c \cdot n/p + L}{2}\right)^2$$

pairs.
$\Rightarrow$ Putting together p machines means

$$p \cdot \left(\frac{c \cdot n/p + L}{2}\right)^2 \quad \geq \quad n^2$$

MPC Lower Bound for Cartesian Product

Suppose that an algorithm has load L.
$\Rightarrow$ Every machine can receive at most L elements.
$\Rightarrow$ In total, the machine sees $c \cdot n/p + L$ elements during the entire algorithm.
$\Rightarrow$ The machine can produce at most

$$\left(\frac{c \cdot n/p + L}{2}\right)^2$$

pairs.
$\Rightarrow$ Putting together p machines means

$$p \cdot \left(\frac{c \cdot n/p + L}{2}\right)^2 \geq n^2$$

which solves to $L = \Omega(n/\sqrt{p})$.

MPC Lower Bound for Pairwise Natural Join

Pairwise Natural Join

$R = (A, B)$: a table of n tuples
$S = (B, C)$: a table of n tuples

At the beginning: The $2n$ tuples are stored on the p machines.
At the end: Each element in $R \bowtie S$ must appear on some machine.

MPC Lower Bound for Pairwise Natural Join

Pairwise Natural Join

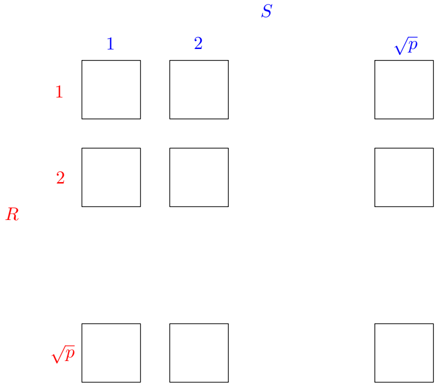
$R = (A, B)$: a table of n tuples
$S = (B, C)$: a table of n tuples

At the beginning: The $2n$ tuples are stored on the p machines.
At the end: Each element in $R \bowtie S$ must appear on some machine.

LOWER BOUND: Load $\Omega(n/\sqrt{p})$!

MPC Cartesian Product

An optimal algorithm—named cube—with load $O(n/\sqrt{p})$ is known. The algorithm actually computes the cartesian product.



MPC Model: Understood Problems

Problems whose communication complexities (i.e., load) have been well understood:

- Enumeration of constant-sized subgraphs.
- Restricted natural joins.
- Linear programming.
- ...

Summary

Principles in design and analysis of MapReduce algorithms

- Minimality
- MPC

Summary

Principles in design and analysis of MapReduce algorithms

- Minimality
- MPC

Future work 1: Re-examine all problems!

Summary

Principles in design and analysis of MapReduce algorithms

- Minimality
- MPC

Future work 1: Re-examine all problems!
Future work 2: Lower bound for problems with small result sizes!

- From the theory of communication complexity, it is easy to prove that when $p = 2$, $\Omega(n)$ communication is necessary for detecting whether a natural join has an empty result!

Summary

Principles in design and analysis of MapReduce algorithms

- Minimality
- MPC

Future work 1: Re-examine all problems!
Future work 2: Lower bound for problems with small result sizes!

- From the theory of communication complexity, it is easy to prove that when $p = 2$, $\Omega(n)$ communication is necessary for detecting whether a natural join has an empty result!

Future work 3: $o(n/p)$-load algorithms!