Parallel Computing
And MapReduce

Howard Karloff (AT & T Labs)
Siddharth Suri (Yahoo! Research)
Sergei Vassilvitskii (Yahoo! Research)
What is MapReduce?

MapReduce is a distributed computing paradigm that’s here now
- Designed for 10,000+ node clusters
- Very popular for processing large datasets
- Processing over 20 petabytes per day [Google, Jan 2008]
- But virtually NO analysis of MapReduce algorithms
What is MapReduce?

MapReduce is a distributed computing paradigm that’s here now
- Designed for 10,000+ node clusters
- Very popular for processing large datasets
- Processing over 20 petabytes per day [Google, Jan 2008]
- But virtually NO analysis of MapReduce algorithms

Logically:
- Treat the data as a set of <Key, Value> pairs.
- Map:
  - Take a <Key, Value> pair: output a list of new <Key, Value> pairs
  - Each pair processed individually (parallelization!)
- Reduce:
  - Take all of the values associated with the same <Key>
  - Output a new set of values for this Key
  - Each reducer can be processed in parallel!
Map Reduce Example

Input: A text document: “Call me Ishmael. Some years ago... “
Output: Histogram of word frequencies
Input: A text document: “Call me Ishmael. Some years ago... “
Output: Histogram of word frequencies

Tuple representation:
<Word, Position>
Map Reduce Example

Input: A text document: “Call me Ishmael. Some years ago... “
Output: Histogram of word frequencies

Tuple representation:
<Word, Position>

Map:
Replace Position with 1.
Map Reduce Example

Input: A text document: “Call me Ishmael. Some years ago... “
Output: Histogram of word frequencies

Tuple representation:

<Word, Position>

Map:
Replace Position with 1.
Map Reduce Example

Input: A text document: “Call me Ishmael. Some years ago... “
Output: Histogram of word frequencies

Tuple representation:

\[ (\text{Word}, \text{Position}) \]

Map:

Replace Position with 1.
Map Reduce Example

Input: A text document: “Call me Ishmael. Some years ago...”
Output: Histogram of word frequencies

Tuple representation:

\(<\text{Word}, \text{Position}>\)

Map:

Replace Position with 1.
Map Reduce Example

Input: A text document: “Call me Ishmael. Some years ago...“
Output: Histogram of word frequencies

Tuple representation:

\(<\text{Word}, \text{Position}>\)

Map:

Replace Position with 1.

Reduce: Sum up the values for each key.
Map Reduce Example

Input: A text document: “Call me Ishmael. Some years ago... “
Output: Histogram of word frequencies

Tuple representation:
<Word, Position>

Map:
Replace Position with 1.

Reduce: Sum up the values for each key.
Map Reduce Example

Input: A text document: “Call me Ishmael. Some years ago...“
Output: Histogram of word frequencies

Tuple representation:
<Word, Position>

Map:
Replace Position with 1.

Reduce: Sum up the values for each key.
Map Reduce Example

Input: A text document: “Call me Ishmael. Some years ago...“
Output: Histogram of word frequencies

Tuple representation:
<Word, Position>

Map:
Replace Position with 1.

Reduce: Sum up the values for each key.
Modeling Map Reduce

Map Reduce Class (MRC):

Three Guiding Principles

- [Space] Bounded memory per machine
- [Time] Small number of rounds
- [Machines] Bounded number of machines
Modeling Map Reduce

Map Reduce Class (MRC):

Three Guiding Principles

• Input of size \( n \)
  
  - [Space] Bounded memory per machine
    • Cannot fit all of input onto one machine
    • Memory per machine \( n^{1-\epsilon} \)
  
  - [Time] Small number of rounds
    • Strive for constant, but OK with \( \log^{O(1)} n \)
    • Polynomial time per machine (No streaming constraints)
  
  - [Machines] Bounded number of machines
    • Substantially sublinear number of machines
    • Total \( n^{1-\epsilon} \)
Theorem: Any NC algorithm using at most $n^{2-\epsilon}$ processors and at most $n^{2-\epsilon}$ memory can be simulated in MRC.

Instant computational results for MRC:
- Matrix inversion [Csanky’s Algorithm]
- Matrix Multiplication & APSP
- Topologically sorting a (dense) graph
- ... 

But the simulation does not exploit full power of MR
- Each reducer can do sequential computation
Going beyond PRAM Algorithms

How to find an MST in two rounds:

Given a graph: \( G = (V, E) \)

Partition the vertex set (randomly) into \( n^{2/3} \) groups \( V = \{V_1, \ldots, V_{n^{2/3}}\} \)

Send the graph \( G[V_i \cup V_j] \) to reducer \((i, j)\)

Compute the MST \( M_{ij} = \text{MST}(G[V_i \cup V_j]) \) on each reducer

Finally, combine the results on one machine, compute \( M^* = \text{MST}(\cup_{ij} M_{ij}) \)

Theorem: \( M^* = \text{MST}(G) \)

- Sequential computation on each reducer (compute many smaller MSTs in parallel)
- Interleaving sequential and parallel computations
Conclusion

MapReduce is a readily available parallel computing infrastructure
- Google, Yahoo, Facebook, Berkeley, Cornell, CMU, ...

“The beauty of MapReduce is that any programmer can understand it, and its power comes from being able to harness thousands of computers behind that simple interface” [David Patterson]

One-off efforts using MapReduce for computation:
- MapReducing EM [Das et al., WWW 07]
- MapReduce for Machine Learning [Chu et al., NIPS 06]

Theory:
- Streaming MapReduce [Feldman et al., SODA 08]