Map Reduce Algorithms

Barna Saha

Acknowledgement: Majority of the slides are taken from
Sergei Vassilivski's tutorial on MapReduce

A Sense of Scale

At web scales...

- Mail: Billions of messages per day
- Search: Billions of searches per day
- Social: Billions of relationships

...even the simple questions get hard

- What are the most popular search queries?
- How long is the shortest path between two friends?
- ...

To Parallelize or Not?

Distribute the computation

- Hardware is (relatively) cheap
- Plenty of parallel algorithms developed

But parallel programming is hard

- Threaded programs are difficult to test. One successful run is not enough
- Threaded programs are difficult to read, because you need to know in which thread each piece of code could execute
- Threaded programs are difficult to debug. Hard to repeat the conditions to find bugs
- More machines means more breakdowns

MapReduce

MapReduce makes parallel programming easy

- Tracks the jobs and restarts if needed
- Takes care of data distribution and synchronization

But there's no free lunch:

- Imposes a structure on the data
- Only allows for certain kinds of parallelism

MapReduce Setting

Data:

- "Which search queries co-occur?"
- "Which friends to recommend?"
- Data stored on disk or in memory

Computation:

- Many commodity machines

MapReduce Basics

Data:

- Represented as <Key, Value> pairs

Example: A Graph is a list of edges

- Key = (u,v)
- Value = edge weight

(u,v)	w_{uv}

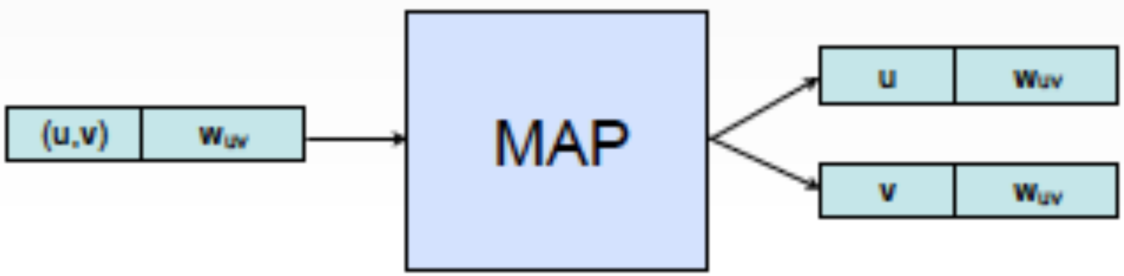
MapReduce Basics

Data:

- Represented as <Key, Value> pairs

Operations:

- Map: <Key, Value> → List(<Key, Value>)
  - Example: Split all of the edges



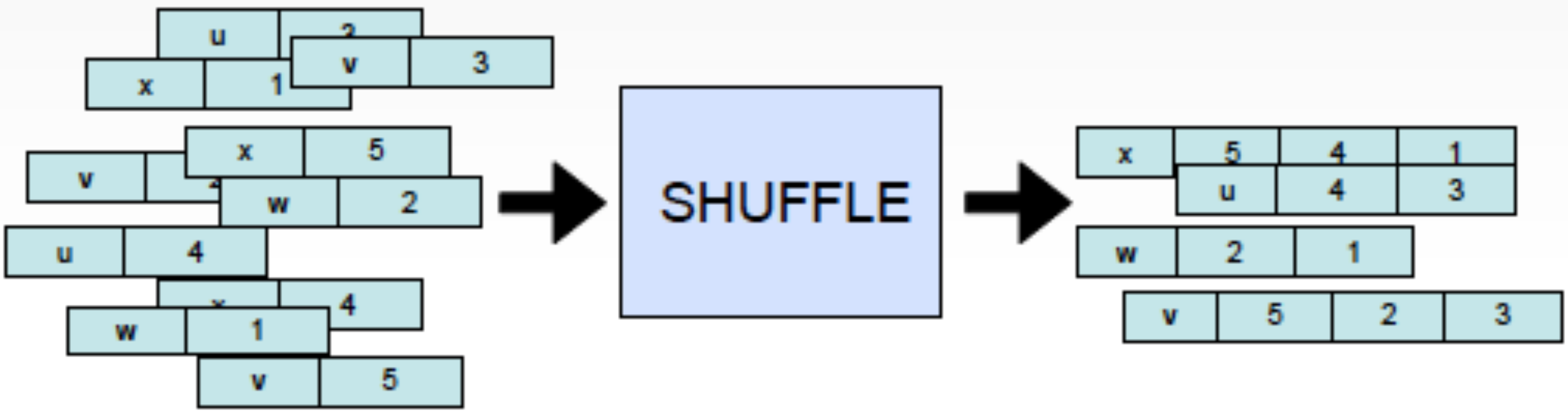
MapReduce Basics

Data:

- Represented as <Key, Value> pairs

Operations:

- Map: <Key, Value> → List(<Key, Value>)
- Shuffle: Aggregate all pairs with the same key



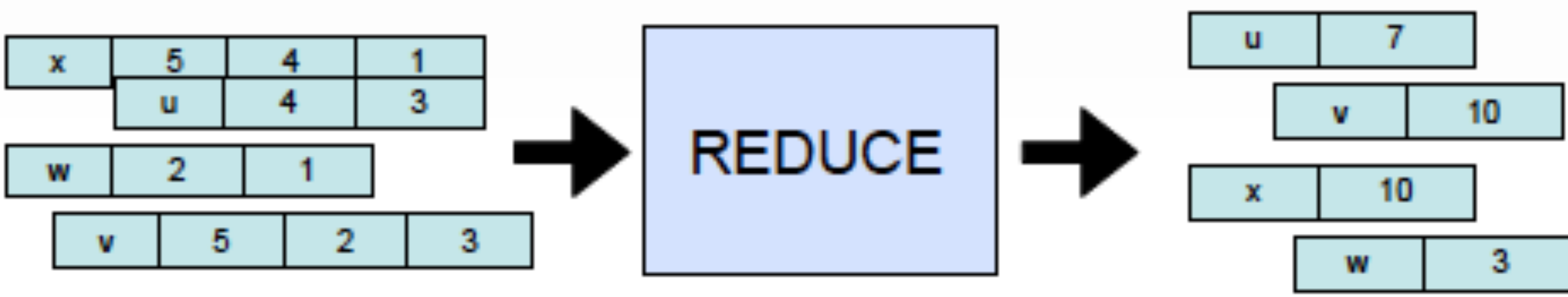
MapReduce Basics

Data:

- Represented as <Key, Value> pairs

Operations:

- Map: <Key, Value> → List(<Key, Value>)
- Shuffle: Aggregate all pairs with the same key
- Reduce: <Key, List(Value)> → <Key, List(Value)>
  - Example: Add values for each key



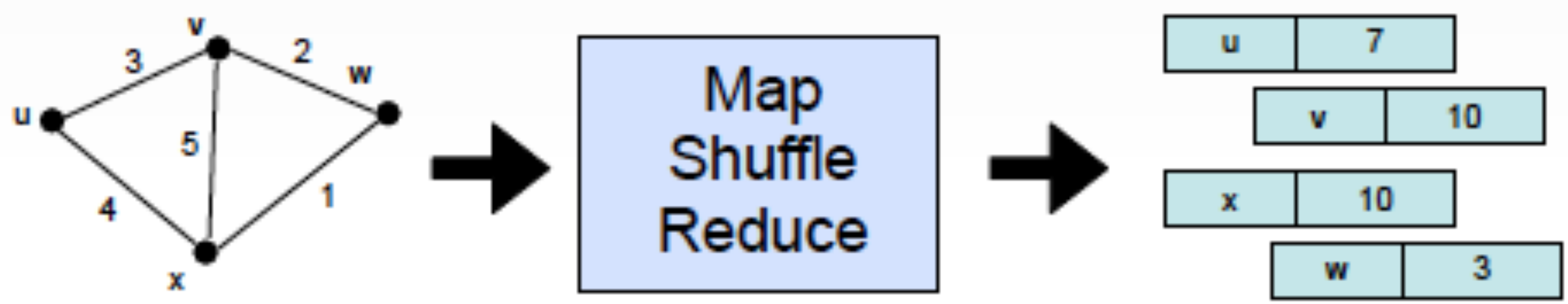
MapReduce Basics

Data:

- Represented as <Key, Value> pairs

Operations:

- Map: <Key, Value> → List(<Key, Value>)
- Shuffle: Aggregate all pairs with the same key
- Reduce: <Key, List(Value)> → <Key, List(Value)>



Matrix Transpose

Given a sparse matrix in row major order

Output same matrix in column major order

Given:

row 1	(col 1, a)	(col 2, b)
row 2	(col 2, c)	(col 3, d)
row 3	(col 2, e)	

a	b	
	c	d
	e	

Matrix Transpose

Map:

- Input: <row i, (col_{i1}, val_{i1}), (col_$_{i2}$, val_{i2}), ... >
- Output: <col_{i1}, (row i, val_{i1})>
- <col_{i2}, (row i, val_{i2})>
- ....

a	b	
	c	d
	e	

row 1	(col 1, a)	(col 2, b)	$\longrightarrow$	col 1	(row 1, a)	col 2	(row 1, b)
row 2	(col 2, c)	(col 3, d)	$\longrightarrow$	col 2	(row 2, c)	col 3	(row 2, d)
row 3	(col 2, e)		$\longrightarrow$	col 2	(row 3, e)		

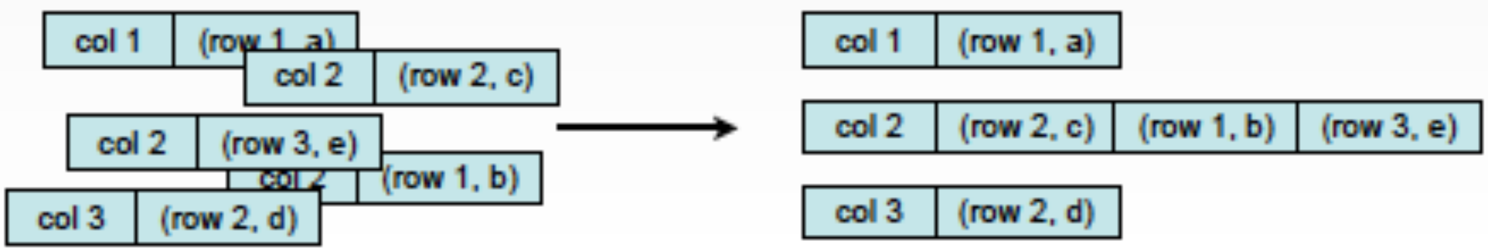
Matrix Transpose

a	b	
	c	d
	e	

Map:

- Input: <row i, (col_{i1}, val_{i1}), (col_{i2}, val_{i2}), ... >
- Output: <col_{i1}, (row i, val_{i1})>
- <col_{i2}, (row i, val_{i2})>
- ....

Shuffle:



Matrix Transpose

Map:

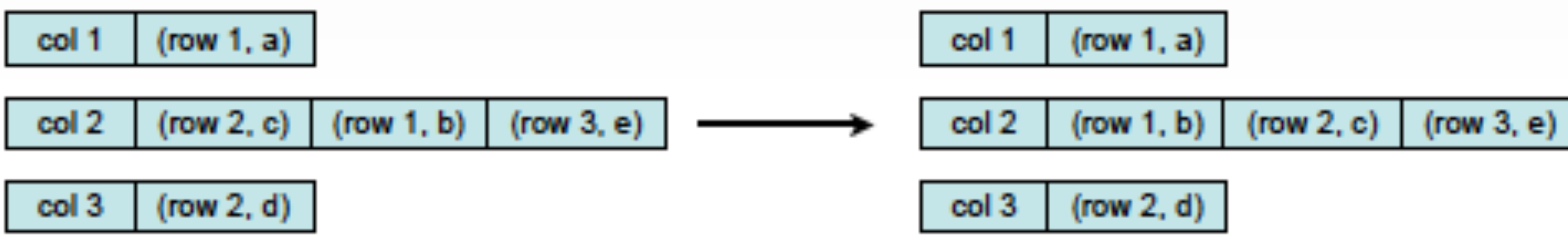
- Input: <row i, (col_{i1}, val_{i1}), (col_$_{i2}$, val_{i2}), ... >
- Output: <col_{i1}, (row i, val_{i1})>
- <col_{i2}, (row i, val_{i2})>
- ....

a	b	
	c	d
	e	

Shuffle

Reduce:

- Sort by row number



Matrix Transpose

Given a sparse matrix in row major order

Output same matrix in column major order

Given:

row 1	(col 1, a)	(col 2, b)
row 2	(col 2, c)	(col 3, d)
row 3	(col 2, e)	

a	b	
	c	d
	e	

Output:

col 1	(row 1, a)		
col 2	(row 1, b)	(row 2, c)	(row 3, e)
col 3	(row 2, d)		

MapReduce Implications

Operations:

- Map: <Key, Value> → List(<Key, Value>)
  - Can be executed in parallel for each pair.
- Shuffle: Aggregate all pairs with the same Key
  - Synchronization step
- Reduce: <Key, List(Value)> → <Key, List(Value)>
  - Can be executed in parallel for each Key

MapReduce Implications

Operations:

- Map: <Key, Value> → List(<Key, Value>)
  - Can be executed in parallel for each pair
  - Provided by the programmer
- Shuffle: Aggregate all pairs with the same Key
  - Synchronization step
  - Handled by the system
- Reduce: <Key, List(Value)> → <Key, List(Value)>
  - Can be executed in parallel for each Key
  - Provided by the programmer

The system also:

- Makes sure the data is local to the machine
- Monitors and restarts the jobs as necessary

Trying MapReduce

Hadoop:

- Open source version of MapReduce
- Can run locally

Amazon Web Services

- Upload datasets, run jobs
- Run jobs ... (Careful: pricing round to nearest hour, so debug first!)

The World of MapReduce

Practice:

- Used very widely for big data analysis
- Google, Yahoo!, Amazon, Facebook, LinkedIn, ...

Beyond Simple MR:

- Many similar implementations and abstractions on top of MR: Hadoop, Pig, Hive, Flume, Pregel, ...
- Same computational model underneath

MapReduce: Overview

Multiple Processors:

- 10s to 10,000s processors

Sublinear Memory

- A few Gb of memory/machine, even for Tb+ datasets
- Unlike PRAMs: memory is not shared

Batch Processing

- Analysis of existing data
- Extensions used for incremental updates, online algorithms

Modeling

For an input of size n :

Memory

- Cannot store the data in memory
- Insist on sublinear memory per machine: $O(n^{1-\epsilon})$ for some $\epsilon > 0$

Machines

- Machines in a cluster do not share memory
- Insist on sublinear number of machines: $O(n^{1-\epsilon})$ for some $\epsilon > 0$

Synchronization

- Computation proceeds in rounds
- Count the number of rounds
- Aim for $O(1)$ rounds

Example

Distributed Sum:

- Given a set of n numbers: $a_1, a_2, \ldots, a_n \in \mathbb{R}$, find $S = \sum_i a_i$

MapReduce:

- Compute $M_j = a_{jk} + a_{jk+1} + \ldots + a_{j(k+1)-1}$ for $k = \sqrt{n}$ in Round 1
- Round 2: add the $\sqrt{n}$ partial sums.

Example

- Given a graph $G = (V, E)$ on $|V| = N$ vertices and $|E| = M \geq N^{1+c}$ edges for some constant $c > 0$, compute Minimum Spanning Tree of the graph.

- Idea: Distribute edges randomly to machines. Compute MST on the local edges. Combine and Repeat!