Map Reduce Algorithms

David Wemhoener

Acknowledgement: Some slides are taken from
Sergei Vassilivski, Barna Saha

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

Example

Distributed Sum:

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

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.

Word Counting

```
map(String key, String value):
  // key: document name,
  // value: document contents
    for each word w in value:
      EmitIntermediate(w, "1");
reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
    int word_count = 0;
    for each v in values:
      word_count += ParseInt(v);
    Emit(key, AsString(word_count));
```

MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004

Word Counting

- Have a very large document

What else can we do?

- Reverse Web-Link Graph

MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004

What else can we do?

- Reverse Web-Link Graph
  - Map Function: source → <target, source> pairs

MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004

What else can we do?

- Reverse Web-Link Graph
  - Map Function: source → <target, source> pairs
  - Reduce Function: <target, source> pairs → <target, list(source)>

MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004

What else can we do?

- Reverse Web-Link Graph
  - Map Function: source → <target, source> pairs
  - Reduce Function: <target, source> pairs → <target, list(source)>
- Inverted Index

MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004

What else can we do?

- Reverse Web-Link Graph
  - Map Function: source → <target, source> pairs
  - Reduce Function: <target, source> pairs → <target, list(source)>
- Inverted Index
  - Map Function: document → <word, doc ID> pairs

MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004

What else can we do?

- Reverse Web-Link Graph
  - Map Function: source → <target, source> pairs
  - Reduce Function: <target, source> pairs → <target, list(source)>
- Inverted Index
  - Map Function: document → <word, doc ID> pairs
  - Reduce Function: <word, doc ID> pairs → <word, list(doc ID)>

MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004

Example: Host size

- **Suppose we have a large web corpus**
- Look at the metadata file
  - Lines of the form: (URL, size, date, …)
- **For each host, find the total number of bytes**
  - That is, the sum of the page sizes for all URLs from that particular host

Example: Join By Map-Reduce

- **Compute the natural join *R(A,B)* ⋈ *S(B,C)***
- *R* and *S* are each stored in files
- Tuples are pairs *(a,b)* or *(b,c)*

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

R

⋈

B	C
b_2	c_1
b_2	c_2
b_3	c_3

S

=

A	C
a_3	c_1
a_3	c_2
a_4	c_3

Map-Reduce Join

- **A Map process turns:**
  - Each input tuple *R(a,b)* into key-value pair *(b,(a,R))*
  - Each input tuple *S(b,c)* into *(b,(c,S))*

Map-Reduce Join

- **A Map process turns:**
  - Each input tuple $R(a,b)$ into key-value pair $(b,(a,R))$
  - Each input tuple $S(b,c)$ into $(b,(c,S))$
- **Map processes** send each key-value pair with key b to Reduce process $h(b)$

Map-Reduce Join

- **A Map process turns:**
  - Each input tuple *R(a,b)* into key-value pair *(b,(a,R))*
  - Each input tuple *S(b,c)* into *(b,(c,S))*

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.

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!