A Model of Computation for MapReduce

Philip Schill

November 11, 2014
Contents

1 Introduction

2 MapReduce

3 MRC

4 MST

5 MRC algorithms
Why parallelism?

- Problem sizes are getting bigger
- CPUs are getting smaller, not faster
  → Use more CPUs
Why MapReduce?

Good parallelism is hard to implement!

**OpenMP:** *(run multiple loop-iterations at once)*
- Only for shared-memory machines

**MPI:** *(send messages between processes)*
- Who needs to talk to whom?
- Who needs which part of the data?
Why MapReduce?

MapReduce:

Only answer one question:
"How can I split my algorithm into Map and Reduce functions?"
Example

Naive counting algorithm
MapReduce

**Input:** Set of \( <k, v> \) pairs

**Map:** (mapper \( \mu \))
- \( In: \) One of the pairs
- \( Out: \) Some new pairs

**Reduce:** (reducer \( \rho \))
- \( In: \) All map-pairs with the same key \( k_0 \)
- \( Out: \) Some new pairs with the key \( k_0 \)
A MapReduce program is a sequence $< \mu_1, \rho_1, \ldots, \mu_R, \rho_R >$ of mappers and reducers.
Restrictions

Given a problem of size $n$ and $\varepsilon \in (0, 1)$.

**Machines:**
- Number of machines limited to $O(n^{1-\varepsilon})$

**Memory:**
- Space of single machine limited to $O(n^{1-\varepsilon})$
- Problem size bigger than memory of single machine
- Total available space $O(n^{2-2\varepsilon})$

**Time:**
- Map and reduce functions should be $O(n^k)$
  (must be ensured by algorithm designer)
The MapReduce Class

Definition

Let $\varepsilon \in (0, 1)$. The MapReduce Class ($\mathcal{MRC}_\varepsilon$) is the class of MapReduce programs, that fulfill:

- Each mapper takes $O(n^{1-\varepsilon})$ space and time polynomial in $n$.
- Each reducer takes $O(n^{1-\varepsilon})$ space and time polynomial in $n$.
- The total space used by the $<k, v>$ pairs after mapping is $O(n^{2-2\varepsilon})$.
- The output is correct with probability at least $\frac{3}{4}$.
The Deterministic MapReduce Class

Definition

Let $\varepsilon \in (0, 1)$. The **Deterministic MapReduce Class** $(DMRC_\varepsilon)$ consists of the $MRC_\varepsilon$ algorithms, that always produce the correct output.

**Remark:** We write $MRC$ and $DMRC$ instead of $MRC_\varepsilon$ and $DMRC_\varepsilon$. 
Restrictions

- No. of machines: $O(n^{1-\varepsilon})$
- Space per machine: $O(n^{1-\varepsilon})$
- Space per mapper: $O(n^{1-\varepsilon})$
- Space per reducer: $O(n^{1-\varepsilon})$
- Total available space: $O(n^{2-2\varepsilon})$
- Allowed space for mapping: $O(n^{2-2\varepsilon})$
  → Up to $O(n^{2-2\varepsilon})$ reducers
  → Up to $O(n^{1-\varepsilon})$ reducers per machine!

Almost all restrictions are needed for the shuffling step!
The shuffling problem

What is the shuffling problem?

- Up to $O(n^{1-\varepsilon})$ reducers per machine
- Distribute the map-pairs in a way, that the memory does not exceed
- Minimum makespan problem (NP-hard)

The shuffling problem
The shuffling problem

Lemma

Let $K$ be the keys, $V$ be the values and $V_k$ be the values with key $k$ after the mapping step of an MRC algorithm. Then $K$ and $V$ be partitioned across $O(n^{1-\varepsilon})$ machines using $O(n^{1-\varepsilon})$ space per machine such that $k$ and $V_k$ go to the same machine.
The shuffling problem

Proof of the lemma
Minimum Spanning Tree

**Definition**

A **spanning tree** $T$ of a connected, undirected graph $G$ is a subgraph of $G$ that fulfills:

- $T$ is a tree,
- $T$ connects all vertices of $G$.

**Definition**

A **minimum spanning tree** (MST) $M$ of a connected, undirected graph $G$ is a spanning tree of $G$ that fulfills:

- Among all spanning trees of $G$, $M$ has the smallest sum of edge weights.
MST MapReduce algorithm

**Task:**
Compute the MST of a connected, undirected graph $G = (V, E)$ with $|V| = N$ vertices and $|E| = m \geq N^{1+c}$ edges.

**Algorithm:**

- Choose $k \in \mathbb{N}$ and $V_i$ with $V = \bigcup_{i=1}^{k} V_i$ and $|V_i| \approx \frac{N}{k}$.
- For $i \neq j$ let $E_{i,j} := \{(u, v) \in E | u, v \in V_i \cup V_j\}$.
  - For $i \neq j$ let $G_{i,j} := (V_i \cup V_j, E_{i,j})$.
- Compute $M_{i,j} := \text{MSF}(G_{i,j})$ (minimum spanning forest).
- Let $M := \bigcup_{i,j} M_{i,j}$.
- Compute $H := \text{MST}((V, M))$. 
MST MapReduce algorithm

Proof that the algorithm works
MST MapReduce algorithm

**Input:** Pairs $\langle \ldots, v \rangle, \langle \ldots, e \rangle$ of vertices and edges.

**Map 1:**

$\langle \ldots, v \rangle \mapsto \langle (i, j), v \rangle$ for all $(i, j)$ with $v \in G_{i,j}$

$\langle \ldots, e \rangle \mapsto \langle (i, j), e \rangle$ for all $(i, j)$ with $e \in G_{i,j}$

**Reduce 1:**

$\langle (i, j), (v_1, \ldots, v_k, e_1, \ldots, e_l) \rangle \mapsto \langle (i, j), MSF(G_{i,j}) \rangle$

**Map 2:**

$\langle (i, j), M_{i,j} \rangle \mapsto \langle $, $M_{i,j} \rangle$

**Reduce 2:**

$\langle $, $M_{i,j}$ for all $(i, j)$ $\rangle \mapsto \langle $, $MST((V, \bigcup_{i,j} M_{i,j})) \rangle$. 
MST MapReduce algorithm

Lemma

If $\varepsilon < \frac{c}{2+2c}$, then the MST MapReduce algorithm is in $DMRC_\varepsilon$.

Reminder: $|V| = N$, $|E| = m \geq N^{1+c}$. 
Proof that the algorithm is in $\text{DMRC}$
**Definition**

Let $S$ be a set. A function $f$ on $S$ is **MRC-parallelizable**, if there are functions $g$, $h$ such that:

- For any partition $S = \bigcup_{i=1}^{k} T_i$, it is
  
  $$f(S) = h(g(T_1), \ldots, g(T_k)).$$

- $g$ and $h$ can be computed in time polynomial in $|S|$. 
**MRC algorithms**

<table>
<thead>
<tr>
<th>Lemma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given a universe $U$ of size $n$ and $k \leq n^{2-3\varepsilon}$. Let $f_1, \ldots, f_k$ be MRC-parallelizable functions on $S_1, \ldots, S_k$ and $\sum_i</td>
</tr>
</tbody>
</table>

**Algorithm designer can focus on**

- dividing the problem into small parts,
- implementing the functions for those parts.

**Question:** How do we deal with $f_1, \ldots, f_k$?
The functions \( f_i \) are \( MRC \)-parallelizable, let \( g_i, h_i \) be the corresponding functions.

**Input:** Pairs \(< i, g_i >, < i, h_i >, < i, u >\) with \( u \in S_i \).

**Idea:** Send \( g_i, h_i \) and elements of \( S_i \) to the same reducer.

**Problem:** What if \(|S_i| > n^{1-\varepsilon}\)?

**Initialize:** Use \( M := n^{1-\varepsilon} \) reducers. Group them into \( t := n^{1-2\varepsilon} \) blocks \( B_1, \ldots, B_t \) of size \( B := n^\varepsilon \). Construct hash functions

\[
hash_1 : \{1, \ldots, k\} \rightarrow \{1, \ldots, t\},
\]
\[
hash_2 : \{1, \ldots, k\} \rightarrow \{1, \ldots, M\}.
\]
**MRC algorithms**

Map 1:

\[
<i, u> \mapsto <r, (u, i)> \text{ for a random reducer } r \text{ in } B_{hash_1}(i),
\]

\[
<i, g_i> \mapsto <b, (g_i, i)> \text{ for all reducers } b \text{ in } B_{hash_1}(i),
\]

\[
<i, h_i> \mapsto <b, (h_i, i)> \text{ for all reducers } b \text{ in } B_{hash_1}(i).
\]

Reduce 1:

For one or more values of \(i\) we get:

**In:** \(<r, ((u_1, i), \ldots, (u_l, i), (g_i, i), (h_i, i))>\)

Let \(T_j := \{u_1, \ldots, u_l\} \subset S_i\), compute \(g_i(T_j)\).

**Out:** \(<r, (g_i(T_j), i, h_i)>\).
**MRC algorithms**

**Map 2:**
\[
< r, (g_i(T_j), i, h_i) > \mapsto < \text{hash}_2(i), (g_i(T_j), i, h_i) >
\]

**Reduce 2:**
For one or more values of \( i \) we get:

**In:** \(< \text{hash}_2(i), (g_i(T_j), i, h_i) >\)

Now we have
\[
\bigcup_j T_j = S_i
\]
\[
\Rightarrow f_i(S_i) = h_i(g_i(T_1), \ldots, g_i(T_B))
\]

**Out:** \(< \text{hash}_2(i), f_i(S_i) >\).

**Alternative out:** \(< \text{hash}_2(i), (i, f_i(S_i)) >\).
**MRC algorithms**

Why is the data split evenly?

**Map 1:** Choose random reducer in current block.

Say a block with 3 reducers gets 5 parts of the input data.
**MRC** algorithms

**Example:** Word counting.

Let $\mathcal{L}$ be the string alphabet.

**Input:** $<i, a_i>$ where $a_i \in \mathcal{L}$ is the symbol at index $i$.

**Strategy:**

- For $a \in \mathcal{L}$ define $S_a := \{<i, a_i> \mid a_i = a\}$.
- We want to compute $f_a(S_a) = |S_a|^k$.
  
  $\rightarrow g(\{t_1, \ldots, t_m\}) = m$ and $h(i_1, \ldots, i_n) = (i_1 + \ldots + i_n)^k$.  

Conclusion

Many problems are suited for MapReduce:

Given

- $MRC$-parallelizable functions $f_1, \ldots, f_k$,
- input $S_1, \ldots, S_k$ of size $\sum_i |S_i| \leq n^{2-2\varepsilon}$,

we can compute $f_1(S_1), \ldots, f_k(S_k)$ in parallel only by thinking about

"How can we split $f_i$ in functions $g_i, h_i$, such that

$$f_i(S_i) = h_i(g_i(T_1), \ldots, g_i(T_k))$$

for any partition $S_i = \bigcup_{j=1}^k T_j$?"
Conclusion

Thank you for listening!