A Model of Computation for MapReduce

Karloff, Suri and Vassilvitskii

\( o(n) \) Big Data Reading Group

presented by:
Babak Bagheri Hariri

Presentation is prepared based on a presentation by Xing Xie.

Sep 19, 2014
Map Reduce

- A new framework for parallel computing originally developed at Google (before 2004)
- Parallelization of data intensive computation
  - interleaves sequential and parallel computation
  - Tera- and petabytes data set (search engines, internet traffic, bioinformatics, etc)
What is MapReduce (cont.)

Three-stage operations:

- **Map-stage**: mapper operates on a single pair \( \langle \text{key}, \text{value} \rangle \), outputs any number of new pairs \( \langle \text{key}', \text{value}' \rangle \);
  - operation is stateless (parallel)

- **Shuffle-stage**: all values that are associated to an individual key are sent to a single machine (done by the system)

- **Reduce-stage**: reducer operates on the all the values and outputs a multiset of \( \langle \text{key}, \text{value} \rangle \).
  - stage can only start when all Map operations are done.
An example: $k^{th}$ frequency moment of a large data set

- Input: a finite string of symbols $s = a_1, a_2, \ldots, a_n$;
- Let $f(x)$ be the frequency of the symbol $x$,
  - note: $\sum_{x \in s} f(x) = n$;
- Want to compute $\sum_{x \in s} f^k(x)$;

example:

$s = 1, 1, 2, 4, 1$

$f^1(x) = 3^1 + 1^1 + 1^1 = 5$;

$f^2(x) = 3^2 + 1^2 + 1^2 = 11$. 
An example (cont.)

- **Input to each mapper:** \( \langle i, x_i \rangle \)
  - \( \mu_1(\langle i, x_i \rangle) = \langle x_i, i \rangle \) (\( i \) is the index).

- **Input to each reducer:** \( \langle x_i, \{i_1, i_2, \ldots, i_m\} \rangle \)
  - \( \rho_1(\langle x_i, \{i_1, i_2, \ldots, i_m\} \rangle) = \langle x_i, m^k \rangle \);

- **Map the values to a single reserved symbol '\$'**
  - \( \mu_2(\langle x_i, v \rangle) = \langle $, v \rangle \);

- **A single reducer for summing up the values:**
  - \( \rho_2(\langle $, \{v_1, \ldots, v_l\} \rangle) = \langle $, \sum v_i \rangle \).
Formal Definition

- The input is a finite sequence of pairs of binary strings \( \langle \text{key}, \text{value} \rangle \);
  - \( U_0 = \langle k_1, v_1 \rangle, \cdots \langle k_m, v_m \rangle \)

- A MapReduce program consists of a finite sequence of mappers and reducers;
  - \( \mu_1, \rho_1, \mu_2, \rho_2, \cdots, \mu_l, \rho_l \);

- Execution: For \( r = 1, 2, \ldots, l \)
  - (Map) feed each \( \langle k, v \rangle \) in \( U_{r-1} \) to mapper \( \mu_r \).
    - Let the output be \( U'_r \);
  - for each \( k \)
    - (Shuffle) \( V_{k,r} \) is the multiset of values \( v \), s.t., \( \langle k, v_i \rangle \in U_{r-1} \);
    - feed \( k \) and \( V_{k,r} \) to a separate instance of \( \rho_r \);
    - (Reduce) Let \( U_r \) be the multiset of \( \langle \text{key}, \text{value} \rangle \) generated by all instances of \( \rho_r \).
  - Output \( U_l \).
The MapReduce Class ($\mathcal{MRC}$)

- On input $I$ with size: $n = \sum_{(k,v) \in I} (|k| + |v|)$
  - Memory: Memory: each mapper/reducer uses $O(n^{1-\epsilon})$ space;
  - Machines: There are $O(n^{1-\epsilon})$ machines available;
  - Time: each machine runs in time polynomial in $n$, (not in the length of the input they receive);
  - Randomized algorithms for map and reduce;
  - The algorithm outputs the correct answer with probability at least $3/4$;
  - Rounds: Shuffle is expensive:
    - $\mathcal{MRC}^i$. number of rounds $= O(\log^i n)$

- $\mathcal{DMRC}$: the deterministic variant.

Lemma

For all rounds of an algorithm in $\mathcal{MRC}$, it is possible to partition the output of the mappers among reducers such that the memory restrictions of $\mathcal{MRC}$ would not be violated.
Recall the Frequency Moments Algorithm

Does this algorithm fit in the restrictions of $MRC$?

- $\mu_1(\langle i, x_i \rangle) = \langle x_i, i \rangle$;
- $\rho_1(\langle x_i, \{i_1, i_2, \ldots, i_m\} \rangle) = \langle x_i, m^k \rangle$;
- $\mu_2(\langle x_i, v \rangle) = \langle $, $ v \rangle$;
- $\rho_2(\langle $, $ \{v_1, \ldots, v_l\} \rangle) = \langle $, $ \sum v_i \rangle$.

Consider the input $I = \langle 1, a \rangle, \langle 2, a \rangle, \ldots, \langle n, a \rangle$. 
Comparing $\mathcal{MRC}$ with other Complexity Classes

Easy relation: $\mathcal{MRC} \subseteq \mathcal{P}$;

Lemma

If $\mathcal{NC} \neq \mathcal{P}$, then $\mathcal{DMRC} \not\subseteq \mathcal{NC}$;

Proof idea: There exists a $\mathcal{P}$-complete problem solvable in $\mathcal{DMRC}$:

- Padded Circuite Value Problem (PCV) is a $\mathcal{P}$-complete problem;
- For a given PCV problem with input size $n$, append the input with $n^2 - n$ special character $\#$;
- The problem is in $\mathcal{DMRC}$;
- But it cannot be in $\mathcal{NC}$; otherwise, we would have $\mathcal{NC} = \mathcal{P}$!

Open question: $\mathcal{P} \subseteq \mathcal{DMRC}$?
Example: Finding an MST

Problem:
Find the Minimum Spanning Tree (MST) of a dense graph.

The algorithm:
- Randomly partition the vertices of $G$ into $k$ parts;
- For each pair of vertex sets, find the MST of the subgraph induce by these two sets;
- Take the union $H$ of all the edges in the MST of each pair;
- Compute an MST of $H$

Theorem
the MST tree of $H$ is an MST of $G$

Proof idea: we did not discard any relevant edge when sparsifying the input graph $G$
Finding an MST (cont.)

Why the algorithm is in MRC?

• Let $N = |V|$ and $m = |E| = N^{1+c}$, for $0 < c \leq 1$;
• So input size $n$ satisfies $n = N^{1+c}$;
• Pick $k = N^{c/2}$;

Lemma

*With high probability, each subgraph has size $N^{1+c/2}$.*

• so the input to any reducer is $n^{1-\epsilon}$;
• the size of $H$ is also in $n^{1-\epsilon}$.
Functions Lemma

A very useful building block for designing MapReduce algorithms:

\[ \text{\textbf{MRC}-parallelizable function} \]

Let \( S \) be a finite set. We say a function \( f \) on \( S \) is \( \text{MRC} \)-parallelizable if there are functions \( g \) and \( h \) so that the followings hold:

- For all partition of \( S \), \( S = T_1 \cup T_2 \cup \cdots \cup T_k \), \( f \) can be written as:
  \[
  f(S) = h(g(T_1), g(T_2), \ldots, g(T_k));
  \]
- \( g \) and \( h \) each can be represented in \( O(\log n) \) bits;
- \( g \) and \( h \) can be computed in time polynomial in \( |S| \);
- all possible outputs of \( g \) can be expressed in \( O(\log n) \) bits.
Application of Functions Lemma (1): the Frequency Moments Algorithm

Input $\mathcal{I} = \{\langle 1, l_1 \rangle, \ldots, \langle m, l_m \rangle \}$;

- define $f_{k,l}(\mathcal{I}) = |\text{occurrences of the element } l \text{ in the input } \mathcal{I}|^k$;
- $k^{\text{th}}$-frequency moment of $\mathcal{I}$ is $\sum_{l} f_{k,l}(l)$;
- $f_{k,l}(l)$ is $\mathcal{MRC}$-parallelizable:
  - $g(t_1, \ldots, t_n) = n$;
  - $h(i_1, \ldots, i_r) = (i_1 + \ldots + i_r)^k$. 
Application of the Functions Lemma (2): $s - t$ connectivity

$s - t$ Connectivity Problem:

Given a graph $G$ and two nodes, are they connected in $G$?

- for dense graphs: easy, powering adjacency matrix;
- Sparse graphs?
A $\log n$-round MapReduce algorithm for $s-t$ connectivity

- Initially every node is active;
- For $i = 1, 2, \ldots, O(\log n)$ do
  - Each active node becomes a leader with probability $1/2$;
  - For each non-leader active node $u$, find a node $v$ in the neighbor of $u$’s current connected component
    - If the connected component of $v$ is non-empty, then $u$ become passive and re-label each node in $u$’s connected component with $v$’s label.
- Output true if $s$ and $t$ have the same label, false otherwise.
Thanks!