MR - A MapReduce Programming Language

W4115 Programming Language & Translator
Fall 2011

Siyang Dai
Zhi Zhang
Zeyang Yu
Jinxiong Tan
Shuai Yuan
class HelloWorld {
    public static void main ( String argv[] ) {
        System.out.println("Hello World!");
    }
}

All we need actually is one line:
    System.out.println("Hello World!");
Motivation

Word Count

Hello world for Mapreduce
Motivation

Why not like this:

```python
#JobName = "WordCount"

def wordcount_map(offset, line) <(Int, Text) -> (Text, Int)> : Mapper

  List<Text> words = split line by " ";
  foreach word in words
    emit(word, 1);

def wordcount_reduce (word, counts) <(Text, Int) -> (Text, Int)> : Reducer

  Int total = 0;
  foreach count in counts
    total = total + count;
  emit(word, total);
```
#Tutorial: wordcount.mr

```python

#JobName = "WordCount"

//map function definition
def wordcount_map <(Int, Text) -> (Text, Int)> (offset, line): Mapper
{
    List<Text> words;
    words = split line by " ";
    foreach Text word in words {
        emit(word, 1);
    }
}

//reduce function definition
def wordcount_reduce <(Text, Int) -> (Text, Int)> (word, counts): Reducer
{
    Int total;
    total = 0;
    foreach Int count in counts {
        total = total + count;
    }
    emit(word, total);
}
```
Map function

//map function definition
def wordcount_map <(Int, Text) -> (Text, Int)> (offset, line): Mapper
{
    List<Text> words;
    words = split line by " ";
    foreach Text word in words {
        emit(word, 1);
    }
}
Reduce function

// reduce function definition

def wordcount_reduce  
  \((\text{Text} , \text{Int}) \rightarrow (\text{Text}, \text{Int})\)  
  \((\text{word}, \text{counts})\): 
  Reducer 
  
  
  \{ 
  \text{Int} \text{ total}; 
  \text{total} = 0; 
  \textbf{foreach} \text{Int} \text{ count in} \ \text{counts} \ 
  \{} 
  \text{total} = \text{total} + \text{count}; 
  \} 
  \textbf{emit}(\text{word}, \text{total}); 
  \}


Compiler Structure
Translator

Mapper / Reducer (Parameters in Hadoop types) {
  mapper / reducer logic
  emit the values in Hadoop types
}

Problem:
  Hadoop types do not support basic operations.
Types Puzzle

- Hadoop Types
- Ocaml Types
- Java Types
- MR Types
Translator

Mapper / Reducer (Parameters in Hadoop types) {
    Convert from Hadoop types to Java types
    mapper / reducer logic
    Convert from Java types to Hadoop types
    emit the values in Hadoop types
}

Solution:
    Another layer of indirection
Type Checker

Prevent Ridiculous Expression/Statement

E.g.

```java
Int a = "oops"; // illegal assignment

if("oops") { // it should be boolean type
    ...
}
```
<table>
<thead>
<tr>
<th>Type Checker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Definition</td>
</tr>
<tr>
<td>Assignment Operation</td>
</tr>
<tr>
<td>Binary Operation (Arithmetic, Relational, Logical)</td>
</tr>
<tr>
<td>Split Expression</td>
</tr>
<tr>
<td>If-Expression</td>
</tr>
<tr>
<td>Foreach Statement</td>
</tr>
<tr>
<td>Emit Statement</td>
</tr>
</tbody>
</table>
Lessons Learned

1. Parametric Polymorphism
2. Think Recursively
3. Universal Solution - Indirection
4. Teamwork
Demo