Improving Software *Robustness* and *Efficiency* Through *Programming Language* and *Compiler* Innovations

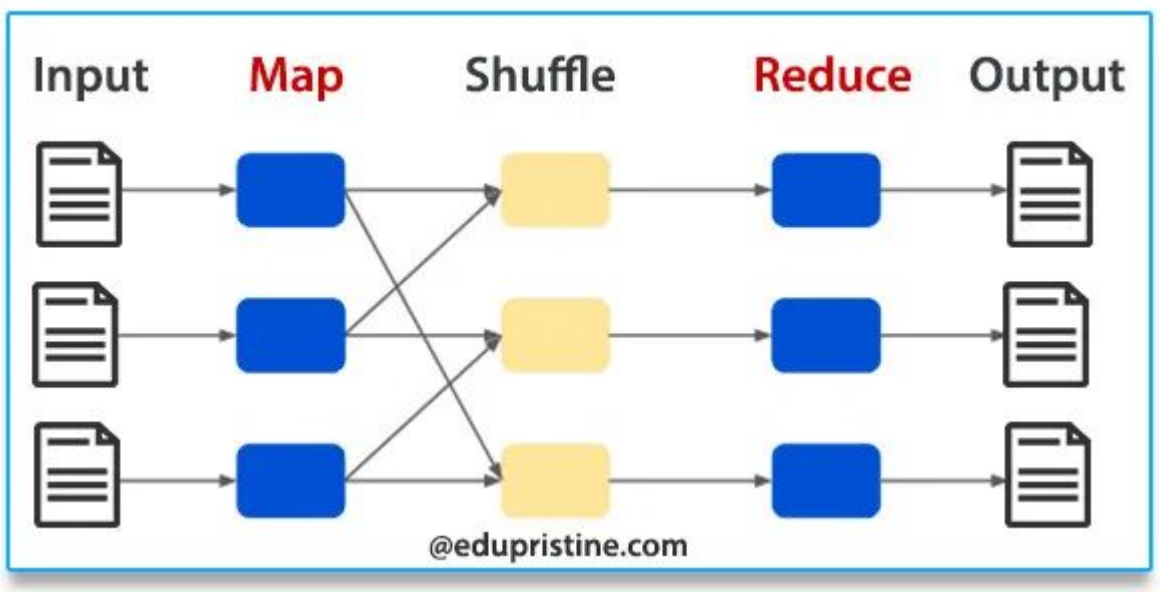
Lionel Parreaux, HKUST CSE *(joining in 2 weeks!)*
Research and Technology Forum 2021

Main points of this short presentation

- **High-level** programming languages are **desirable** in general
- Their **robustness** is improved by advanced **type systems**
- There are **ways** of making them execute **very efficiently**

  (as fast as low-level code)

MapReduce (eg Hadoop)



MapReduce (eg Hadoop)

Low-level

boilerplate

```
public class WordCount {
  ...extends MapReduceBase implements
  ...gWritable, Text, Text, IntWritable> {
  ...ntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable>
                  output, Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
      word.set(tokenizer.nextToken());
      output.collect(word, one);
}}}

public static class Reduce extends MapReduceBase implements
          Reducer<Text, IntWritable, Text, IntWritable> {
  public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
                     IntWritable> output, Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) { sum += values.next().get(); }
    output.collect(key, new IntWritable(sum));
}}

public static void main(String[] args) throws Exception {
  JobConf conf = new JobConf(WordCount.class);
  conf.setJobName("wordcount");
  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(IntWritable.class);
  conf.setMapperClass(Map.class);
  conf.setCombinerClass(Reduce.class);
  conf.setReducerClass(Reduce.class);
  conf.setInputFormat(TextInputFormat.class);
  conf.setOutputFormat(TextOutputFormat.class);
  FileInputFormat.setInputPaths(conf, new Path(args[0]));
  FileOutputFormat.setOutputPath(conf, new Path(args[1]));

  JobClient.runJob(conf);
}}
```

Map function

Reduce function

Run this program as a MapReduce job

Spark Framework

Compose powerful combinators

```
val textFile = sc.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

Software Robustness and Type Systems

- **High-level languages** can accommodate **advanced type systems**
- Types prevent **whole classes of bugs**
  - **prove properties** about program, need **fewer tests**
- **Dependent types** and **formal methods**
  - still in infancy
  - need lots of research to make practical

$$\frac{x : \sigma \in \Gamma \quad \sigma \sqsubseteq \tau}{\Gamma \vdash_S x : \tau}$$

$$\frac{\Gamma \vdash_S e_0 : \tau \to \tau' \qquad \Gamma \vdash_S e_1 : \tau}{\Gamma \vdash_S e_0\ e_1 : \tau'}$$

$$\frac{\Gamma,\ x : \tau \vdash_S e : \tau'}{\Gamma \vdash_S \lambda x\ .\ e : \tau \to \tau'}$$

Efficient Execution of High-Level Programs

- **Naive execution** of HL languages is **more expensive**
  (overhead adds up quickly)
- **Significant problem** – Amdahl's law & end of Moore's law
- Several avenues to **allow for efficient execution**
  - Define **domain-specific** sub-**languages** & **compilation backend**
  - Use **metaprogramming** to **remove abstractions** automatically
  - Design **new optimization techniques** for HL languages

Domain-specific languages & compilation backend

use special domain-aware compilers

Futhark

purely functional data-parallel array **programming language** for the GPU

julia

numerical analysis and computational science

just-in-time compiler

Halide

image and array processing

DSL/compiler

PyTorch

TorchScript **just-in-time compiler** for machine learning

tvm

deep learning

compiler stack

TensorFlow

XLA: **optimizing compiler** for machine learning