COMP 4901Q: High Performance Computing (HPC)

Lecture 13: Introduction to Hadoop and Spark

Instructor: Shaohuai SHI (shaohuais@cse.ust.hk)
Teaching assistants: Mingkai TANG (mtangag@connect.ust.hk)
Yazhou XING (yxingag@connect.ust.hk)

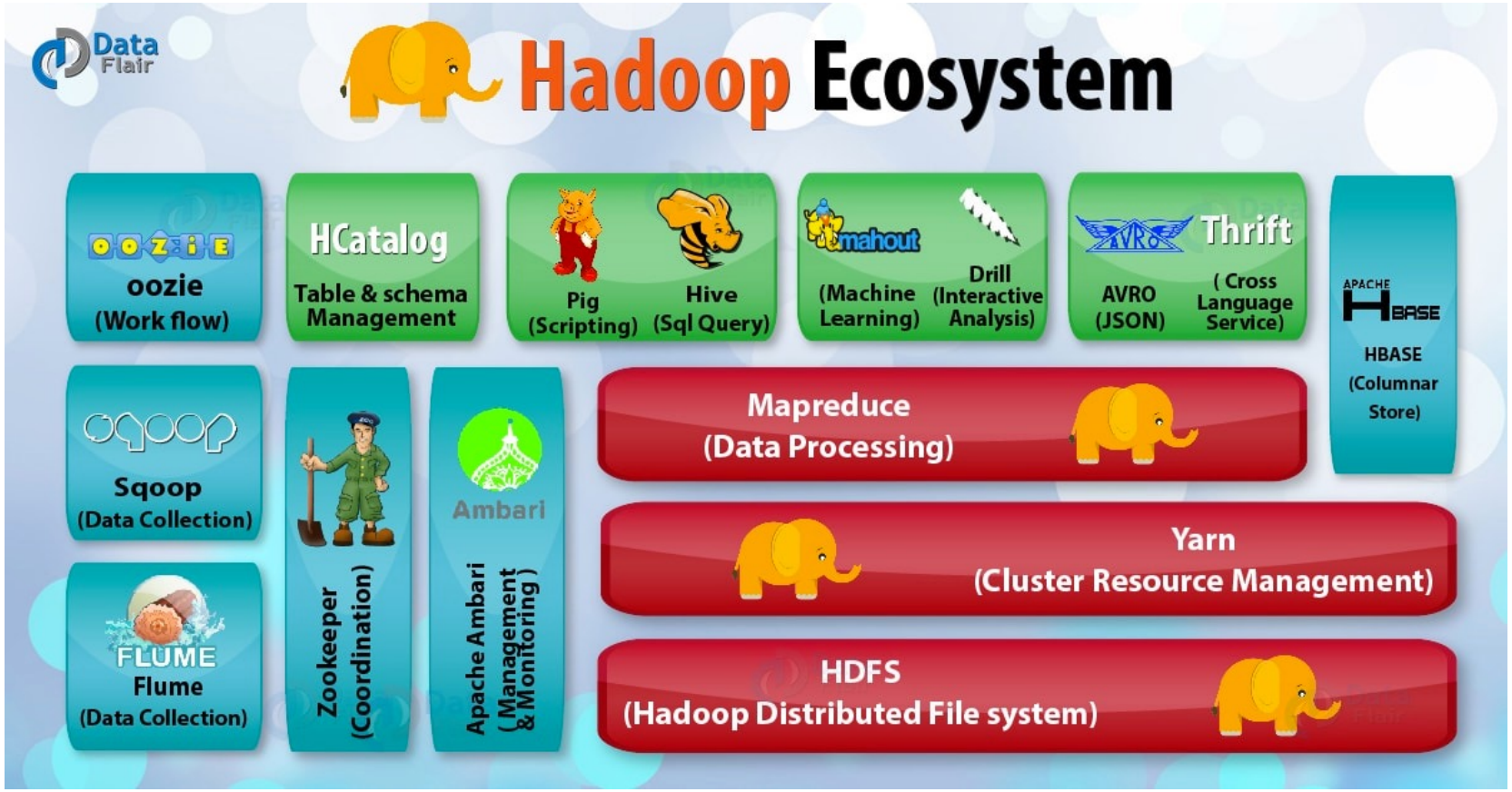
Course website: https://course.cse.ust.hk/comp4901q/

Outline

- Hadoop Ecosystem
- MapReduce and Spark

Hadoop Ecosystem

- The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models.
  - https://hadoop.apache.org/



Source: https://data-flair.training/blogs/hadoop-ecosystem-components/

Apache Hadoop Basic Modules

- Hadoop Common
  - The common utilities that support the other Hadoop modules
- Hadoop Distributed File System (HDFS)
  - A distributed file system that provides high-throughput access to application data
- Hadoop YARN
  - A framework for job scheduling and cluster resource management
- Hadoop MapReduce
  - A YARN-based system for parallel processing of large data sets
- Hadoop Ozone
  - An object store for Hadoop.

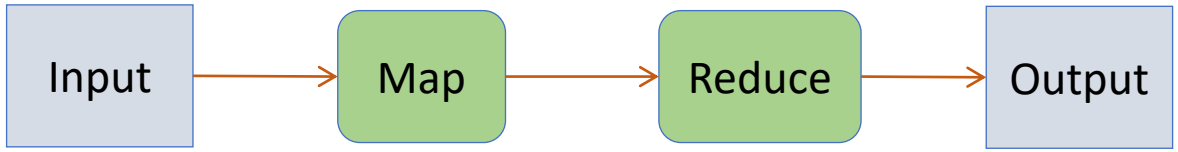
Hadoop MapReduce

- MapReduce is simple programming paradigm for the Hadoop
- Traditional parallel programming requires expertise of different computing/systems concepts
  - Examples: multithreads, synchronization mechanisms (locks, semaphores, and monitors)
  - Incorrect use: can crash your program, get incorrect results, or severely impact performance
  - Usually not fault tolerant to hardware failure
- The MapReduce programming model greatly simplifies running code in parallel

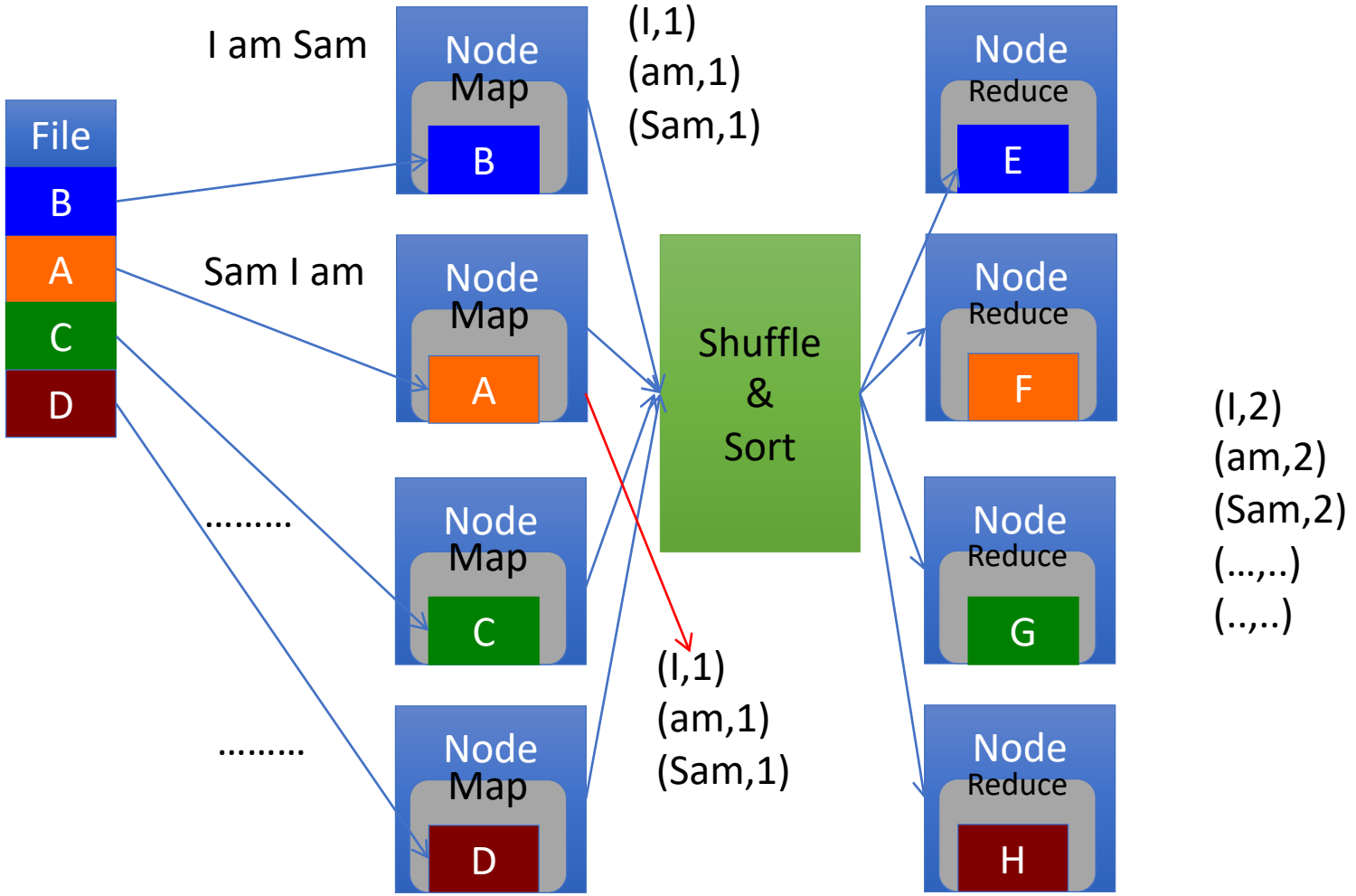
MapReduce Paradigm

- Map and Reduce are based on functional programming

- Breaking the processing into two phases:
  - Map phase: takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs)
  - Reduce phase: takes the output from the Map as an input and combines those data tuples based on the key and accordingly modifies the value of the key



MapReduce: Word Count Example



Shortcoming of MapReduce

- Forces your data processing into Map and Reduce

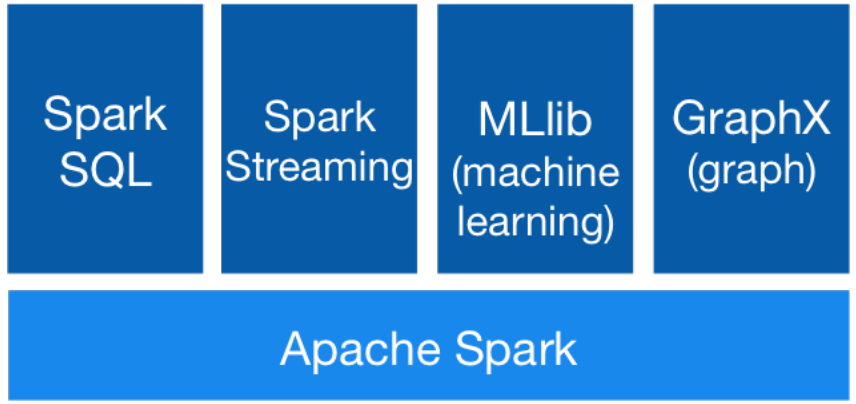
- Based on "Acyclic Data Flow" from Disk to Disk (HDFS)
  - Read and write to Disk before and after Map and Reduce (stateless machine)
  - Not efficient for iterative tasks, i.e. Machine Learning

- Only Java natively supported

- Only for Batch processing
  - Interactivity, streaming data

Apache Spark

- Apache Spark is a unified analytics engine for large-scale data processing.
  - Runs on many frameworks
  - High speed: ~100x faster
  - Ease-of-use: supports Java, Scala, Python, R, and SQL.
  - A stack of libraries: SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming



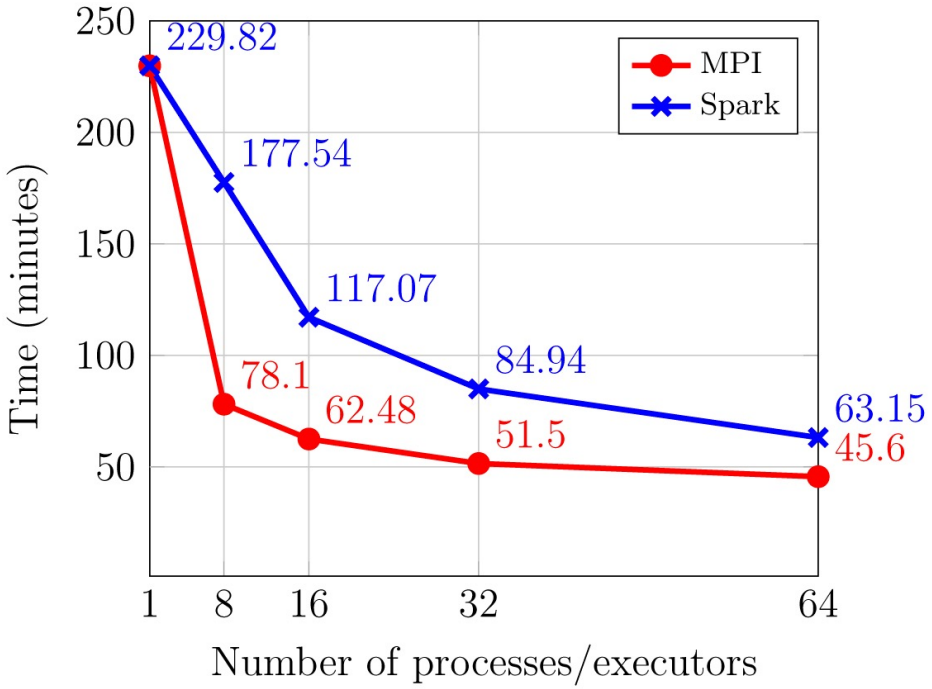
Source: https://spark.apache.org/

Word Count

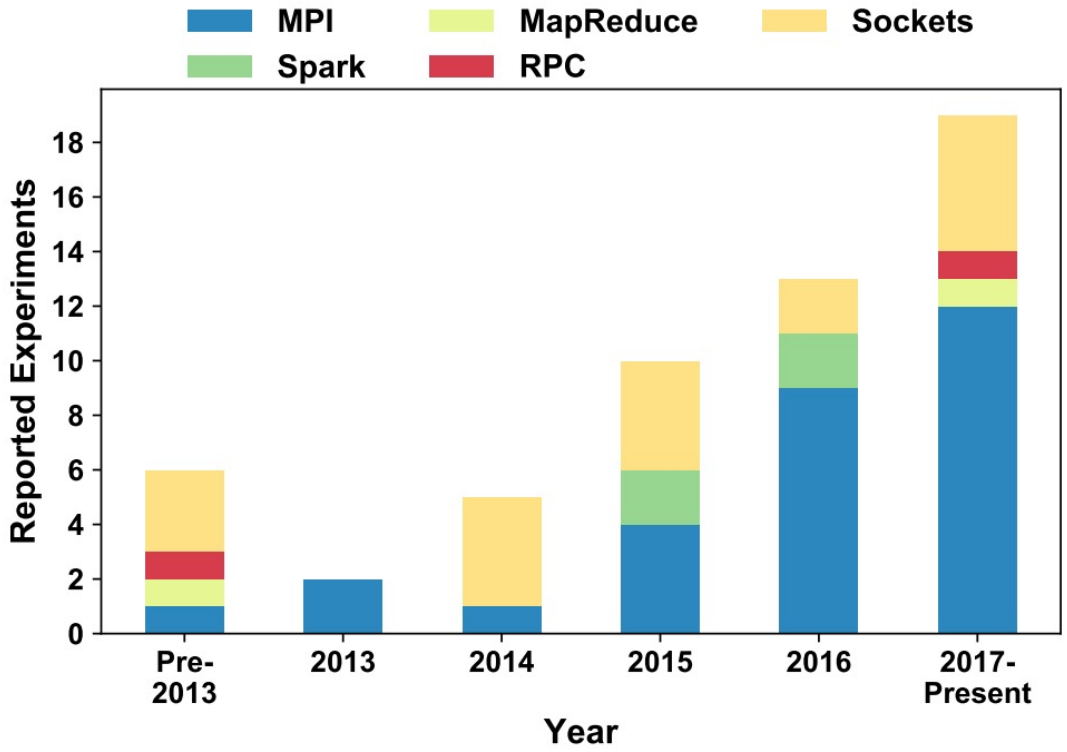
```
text_file = sc.textFile("hdfs://input/book.txt")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://output/wordCount.txt")
```

```
public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }
```

Spark vs. MPI



Performance comparison [1]



Deep Learning research [2]

[1] Abuín, J. M., Lopes, N., Ferreira, L., Pena, T. F., & Schmidt, B. (2020). Big Data in metagenomics: Apache Spark vs MPI. *Plos one*, *15*(10), e0239741.
[2] Ben-Nun, T., & Hoefler, T. (2019). Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. ACM Computing Surveys (CSUR), 52(4), 1-43.