MapReduce paradigm and Apache Hadoop

Marek Rychlý

Faculty of Information Technology Brno University of Technology
Božetěchova 1/2. 612 66 Brno - Královo Pole

rychly@fit.vutbr.cz

2 December 2020

Contents

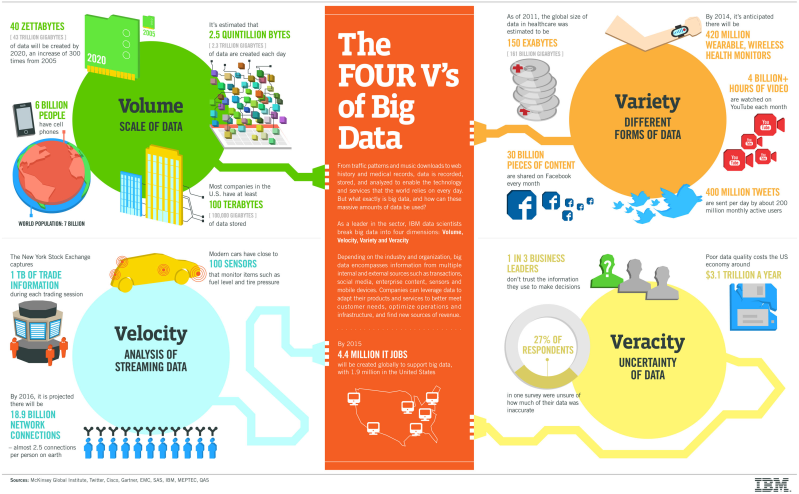
OLTP/OLAP and BigData

- IT was used to working with structured data.
  (e.g. relational and post-relational database with a clear scheme)
- OLTP systems at lower level, OLAP systems at higher level.
  (i.e. "online transaction/analytical processing", common routine vs. complete data analysis)
- Efforts to address the issue of nonstructured data in NoSQL databases.
  (i.e. database without scheme, usually just a storage of "key:value")
- Absence of database scheme is not the only issue.
  (e.g. data streams processed sequentially and real-time, ie. without the option of random access, or stopping the stream)
- Working with such data is outside of the scope of OLTP/OLAP approach $\Rightarrow$ BigData.
  (BigData complete OTLP/OLAP, not replace; OLTP/OLAP still commonly used)

Why BigData?

- Large, nonstructured and quickly growing data collections.
  (can't be processed by the usual means due to their properties)
- They require new approaches for storing, processing and displaying data.
  (capture, pre-processing, storing, searching, sharing/transfer, analysis, visualization)
- It's necessary to use parallel and distributed storage and algorithms/cloud.
  (data can't be stored/processed by a central system due to size, data source location, performance)
- Parallel and distributed processing means more issues.
  (how to ensure appropriate data and computation distribution, how to deal with unreliability/infrastructure crashes, how and where to deliver the results, etc.)
- BigData are necessary for data processing and querying
  - from social networks and news (Facebook, Twitter, . . .),
  - from extensive measuring (data generated constantly by thousands of sensors, various services usage statistics, etc.)
  - from everchanging nonstructured data sets (phone calls, internet communication, video or audio data streams, etc.).

Why BigData?



(diagram taken from "The Four V's of Big Data, IBM")

MapReduce paradigm

- Google's Dean&Ghemawat post from 2004 "MapReduce: Simplified Data Processing on Large Clusters".
- Paradigm based on Map and Reduce functions.
  (inspired by functions from Lisp and other functional languages)
  ;; (map unary-op list1 (list2 list3 . . .))
  (**map** square '(1 2 3 4)) *;; result* = (1 4 9 16)
  ;; (reduce binary-op list1 (list2 list3 . . .))
  (**reduce** + '(1 4 9 16)) *;; result* = (+ 16(+ 9(+ 4 1))) = 30
- Multiple Map and Reduce tasks running simultaneously
  1. input split in parts, each assigned to one comp. node, (Lisp: multiple lists from input data)
  2. each node runs Map for each element of lists simultaneously, (Lisp: parallel execution of Map function for each list)
  3. results are collected from nodes and grouped according to key, (Lisp: lists prepared for Reduce function, one per key value)
  4. groups are split between nodes according to key values, each runs Reduce, (Lisp: parallel execution of Reduce function for each list)
  5. results of all Reduce tasks are collected and stored on output

Map and Reduce functions

MapReduce applications are comprised of Map and Reduce functions defined on data represented by couples "key:value"

$Map(k_1, v_1) \rightarrow list(k_2, v_2)$

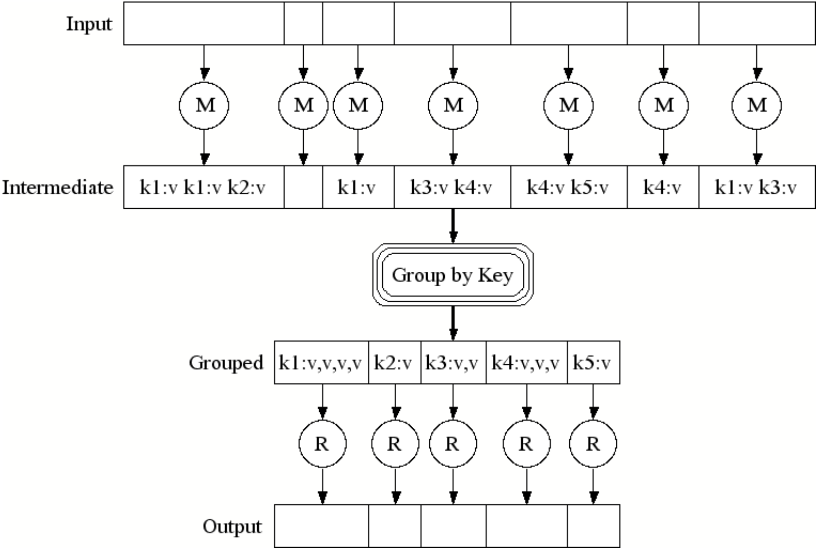
Map is applied to each input data $k_1 : v_1$ and creates a list of outputs $k_2 : v_2$ for each input

$Reduce(k_2, list(v_2)) \rightarrow list(k_3, v_3)$

Outputs of all Map applications are grouped according to a key and then Reduce is applied to each list for the given key and creates a list of output values from them

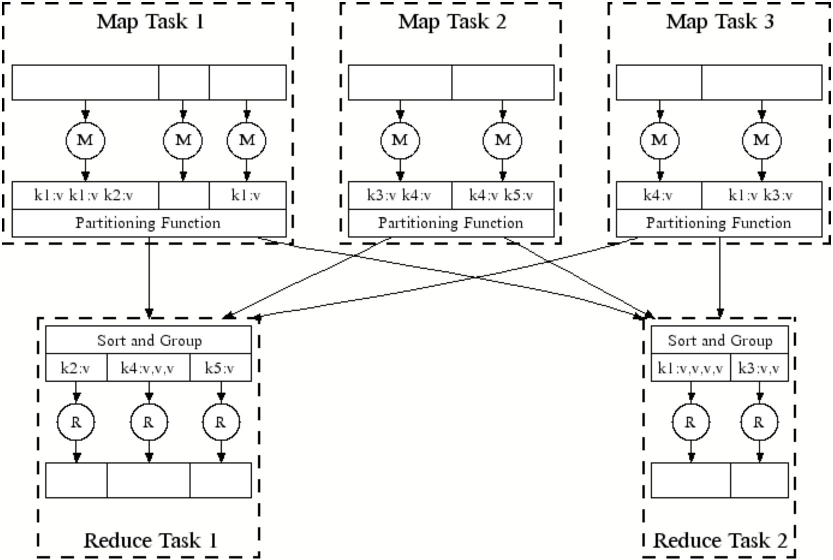
k_1	key value of the input data item, e.g. order, specifies split between nodes
v_1	data of the input data item, i.e. item to be processed
k_2, k_3	key value of the output data, e.g. name of the data processing result
v_2	intermediate result from data processing, obtained from individual inputs
v_3	result of processing by agregation of intermediate results for each key k_2

MapReduce process



(diagram taken from "MapReduce: Simplified Data Processing on Large Clusters")

Parallel MapReduce process



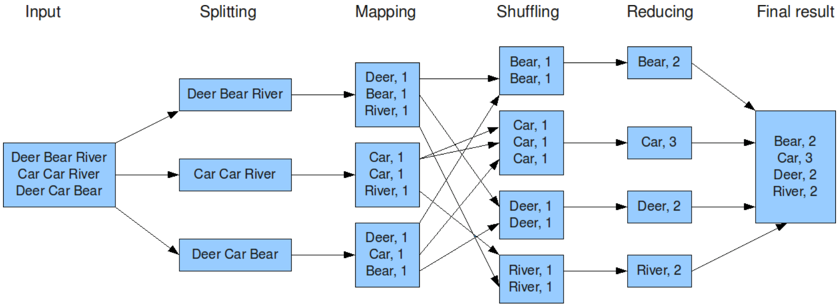
(diagram taken from "MapReduce: Simplified Data Processing on Large Clusters")

Example Map and Reduce functions (from Google2004 article)

```
map(String input_key, String input_value):
  // input_(key, value): (document_name, document_contents)
  for each word w in input_value:
    EmitIntermediate(w, "1");
reduce(String output_key, Iterator intermediate_values):
  // output_(key, values): (a word, a list of counts)
  int result = 0;
  for each val in intermediate_values:
    result += ParseInt(val);
  Emit(AsString(result));
```

- specified application counts occurences of words in input data
- Map is launched for each document (row) in the input and creates output for each word in the document (in row)
  (input key is document name (row number), value is the document contents(row); output key is a word, value represents occurences "1")
- Reduce takes an input word and a list of it's occurences, and returns total occurence count as output
  (input key is a word, value is a list containing element "1"for each occurence of the word; e.g. for three occurences, there is a list (1, 1, 1); output value is a sum of all elements of the input list)

Outputs of phases of MapReduce process

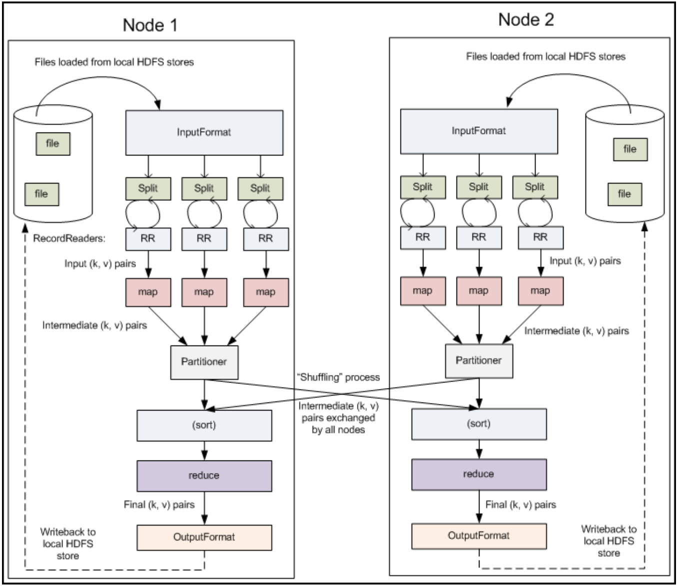


(diagram taken from "Data Mining 2.0: Mine your data like Facebook, Twitter and Yahoo")

Data processing and MapReduce programming

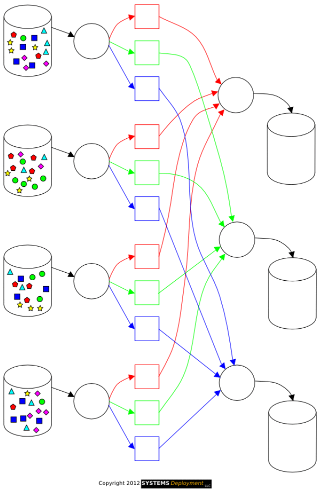
1. input&splitting
   (load data from source and split between Map nodes)
2. Map function
   (Map function execution for individual input parts)
3. shuffling (partitioning&comparing)
   (sort Map outputs, split between Reduce nodes and data transfer)
4. Reduce function
   (Reduce fuction execution for individual intermediate values)
5. output
   (collect Reduce results and write to output)

- Programmer usually only deals with input&output and Map&Reduce.
- Splitting&partitioning executed automatically by framework implementation
  (usually split between nodes according to key hash, preferably uniform)
- Comparing. i.e. sorting intermediate values, is executed automatically according to the keys.



(diagram taken from "Apache Hadoop, Modeule 4: MapReduce")

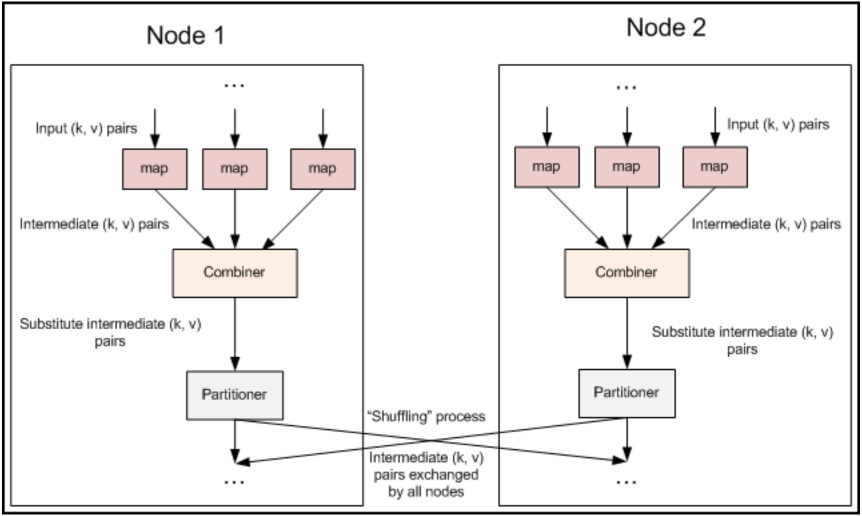
MapReduce process example



(taken from "MapReduce Animation, SYSTEMS Deployment, 20120")

- Map assigns a category to each input record.
  (in the example, Map assigns input objects a category according to their color)
- Shuffle groups records according to their assigned category.
  (in the example, group objects according to their colors)
- Reduce counts/stores records of individuals categories. (in the example, objects belonging to a category according to their color)

"Combiner" extension for MapReduce



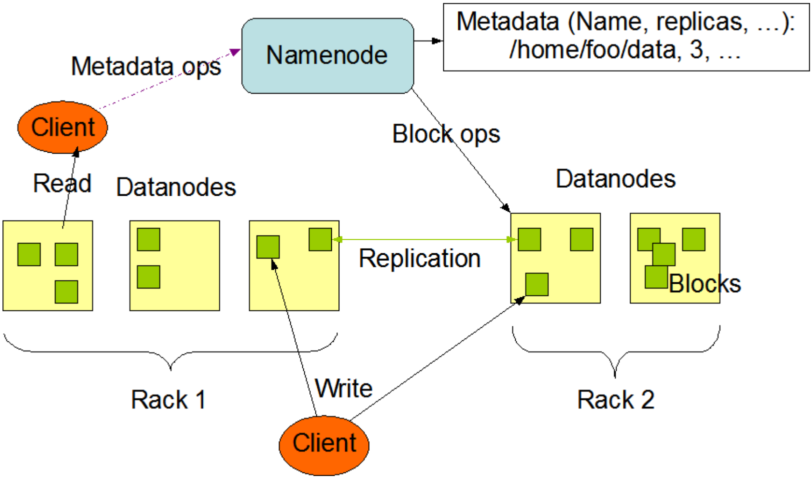
(diagram taken from "Apache Hadoop, Modeule 4: MapReduce")

- Alongside computational model, we also need distributed data storing.
  (MapReduce is a computational model, GFS/HDFS distributed file system)
- Google designed MapReduce on Google File System (GFS)
- GFS was an inspiration for HDFS during implementation of MapReduce.
  (HDFS = Hadoop Distribute File System; hereinafter HDFS)
- Distributed file system distributes data (and metadata).
  (distribution through IT infrastructure, dislocated nodes of a global storage)
- Solves optimal data storing, performance and failure resistance. (various placement strategies, e.g. close to data origin or consumption ; necessary redundance, not all nodes must be available or have the latest data)

- Virtual distributed file system.
  (built on common file systems of individual nodes, solves the problem of locating a storage and access to darta, data not stored physically on node)
- Designed for sequention file access, not random.
  (MapReduce is a batch processing, reads and writes input/output data sequentially)
- Designed for large files (BigData).
  (most of the resources dedicated to locating storage, read/write are fast)
- File data stored in HDFS in blocks with fixed size. (typically 64 or 128 MB, fast; implemented as a group of blocks of local file systems from various nodes, i.e. you can store more data than the capacity of a single node allows; partially full HDFS blocks only take up necessary number of blocks of local file systems, space is not wasted)
- Individual HDFS blocks are distributed and serve as replication units. (i.e. HDFS blocks level of redundance, single block stored on multiple nodes)

HDFS architecture

- There are two types of nodes in HDFS
  NameNode manages file system and file metadata,
  (directory, filepaths, their atributes and locations)
  DataNode hosts data, individual HDFS blocks of files.
  (NameNode knows, where blocks of files are located)
- Usually only one NameNode, great performance and reliability.
  (so called "single point of failure", backup on "secondary NameNode", etc.)
- Multiple instances of DataNode, don't need great performance and reliability due to redundancy.
  (same HDFS block is stored multiple times on various DataNodes)
- Storage nodes are physically arranged in "racks". (rack is in one location, it's nodes are better connected)
- NameNode places instances of a block to various racks (redundance).
  (number of instances depends on replication factor, usually 3 instances on total 2 racks)



(diagram taken from "HDFS Architecture Guide, Apache.")

Apache Hadoop framework

- Framework for distributed, scalable and batch MapReduce computation.
  Hadoop MapReduce MapReduce paradigm implementation
  Hadoop YARN distributed task scheduling and source management
  Hadoop DFS distributed file system
  Hadoop Common common libraries

Apache Hadoop framework

- Additional tools alongside those mentioned earlier.
  Apache Pig(Latin) high-level MapReduce programming (Yahoo)
  Apache Hive datamining platform on Hadoop (Facebook)
  Apache HBase distributed database on Hadoop (Google)
  Apache/IBM Jaql querying language for JSON data
  Apache Flume Hadoop data flow control service
  and other. . .
  (Mahout, Cassandra, HCatalog, Zookeeper, Oozie, Sqoop,. . .)
- Apache Hadoop and most tools are written in Java.
  (runs on JVM, multiple instances within a single node)

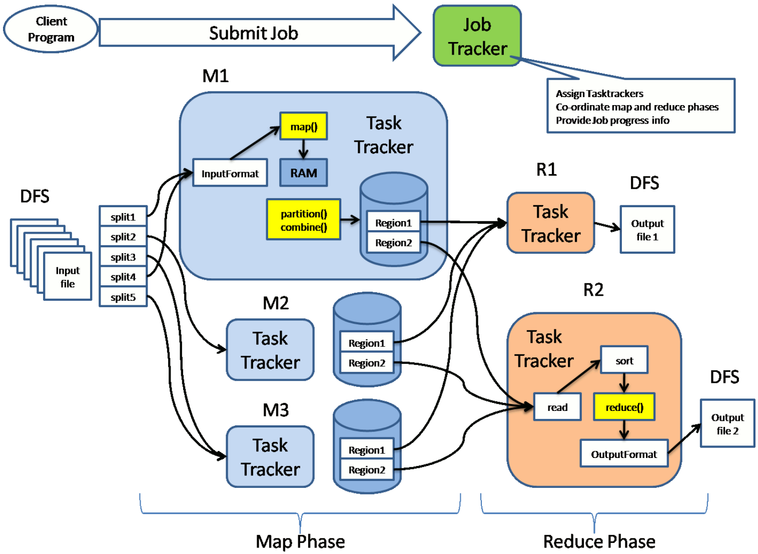
Hadoop Distributions

- Apache Hadoop
- IBM InfoSphere BigInsights
- MapR M3/M5/M7
- Hortonworks Data Platform
- Intel HPC Distribution for Apache Hadoop
- Cloudera CDH
- EMC Pivotal HD
- DataStax Enterprise
- Microsoft Windows Azure HDInsight

Hadoop architecture

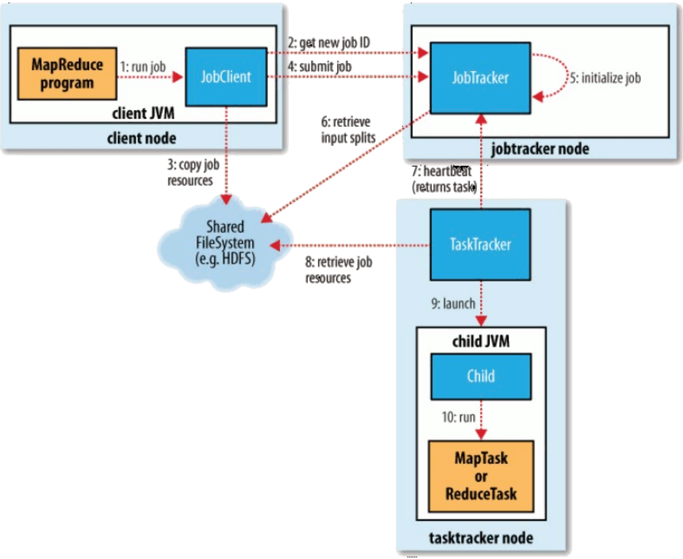
- Hadoop has two additional types of nodes for MapReduce
  JobTracker receives and controls MapReduce applications,
  (only one per cluster, controls TaskTracker)
  TaskTracker runs individual MapReduce operations.
  (at least one per node, runs tasks in individual JVM)
- JobTracker runs client specified MapReduce applications.
  (splits Map and Reduce between TaskTrackers, tracks their completion)
- TaskTrackers receive tasks working with local data.
  (preferably with data located in DataNode on the same node or rack, much like TT)
- TaskTracker does not have to be reliable, JobTracker has to be.
  (if TaskTracker stops sending "heartbeat", JobTracker repeats it's tasks)
- TaskTracker runs each task in an individual JVM.
  (allows absolute control over each task and independence)

Hadoop architecture and job submitting



(diagram taken from "How does hadoop MapReduce works, Big Data Foundation.")

Running MapReduce applications on Hadoop



(diagram taken from "How MapReduce Works with Hadoop")

Abstract from MapReduce paradigm:

- $Map(k_1, v_1) \rightarrow list(k_2, v_2)$
- $Reduce(k_2, list(v_2)) \rightarrow list(k_3, v_3)$

Specific from Java pkg "org.apache.hadoop.mapreduce":

- Interface Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT>

```
protected void map(KEYIN key, VALUEIN val,
org.apache.hadoop.mapreduce.Mapper.Context context) throws
IOException, InterruptedException
```

- Interface Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT>

```
protected void reduce(KEYIN key, Iterable<VALUEIN> values,
org.apache.hadoop.mapreduce.Reducer.Context context)
throws IOException, InterruptedException
```

- Output pair (key, value) is a context method call

```
Context.write(KEYOUT key, VALUEOUT value)
```

Example Hadoop API use - WordCount I

```
package org.myorg;
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {
```

Example Hadoop API use - WordCount II

```
public static class Map extends Mapper<LongWritable, Text,
Text, IntWritable>{

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

  public void map(LongWritable key, Text value, Context
context) throws IOException, InterruptedException {
    String line = value.toString();

    StringTokenizer tokenizer = new StringTokenizer(line);

    while(tokenizer.hasMoreTokens()) {
      word.set(tokenizer.nextToken());
      context.write(word, one);
    }

  }

}
```

Example Hadoop API use - WordCount III

```
public static class Reduce extends Reducer<Text, IntWritable,
Text, IntWritable>{

  public void reduce (Text key, Iterable<IntWritable>values,
Context context) throws IOException, InterruptedException {

    int sum = 0;

    for (IntWritable val : values) {
      sum += val.get();
    }

    context.write(key, new IntWritable(sum));

  }

}
```

Example Hadoop API use - WordCount IV

```
public static void main (String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = new Job(conf, "wordcount");

  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);

  job.setMapperClass(Map.class);
  job.setReducerClass(Reduce.class);

  job.setInputFileFormatClass(TextInputFormat.class);
  job.setOutputFileFormatClass(TextOutputFormat.class);

  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));

  job.waitForCompeltion(true);

 }

}
```

Running Hadoop from command line

- script hadoop with the following syntax
  `hadoop [-config dir] [COMMAND] [GENERIC_OPTIONS] [COMMAND_OPTIONS]`
- Most commonly used commands:
  - "fs"or "dfs"- working with files on HDFS
    `(hadoop fs [GENERIC_OPTIONS] [COMMAND_OPTIONS])`
  - "jar"- running MapReduce applications distributed in .jar archives
    `(hadoop jar <jar>[mainClass] args...)`
  - "job"- working with running applications on JobTracker
    `(hadoop job [GENERIC_OPTIONS] [-status <job-id>] | ...)`
  - "dfsadmin"- HDFS file system administration
    `(hadoop dfsadmin [GENERIC_OPTIONS] [-report] | ...)`

  See the manual for other commands.

Manipulating files on HDFS

The most commonly used arguments for `hadoop fs <args...>`

- "-ls" to print the contents of HDFS directories
  `(hadoop fs -ls hdfs://myhost/mypath)`
- "-cat" to print the contents of HDFS files
  `(hadoop fs -cat hdfs://myhost/mypath/myfile)`
- "-mdkir" to create HDFS directories
  `(hadoop fs -mkdir hdfs://myhost/mypath/mydir)`
- "-rm" or "-rmr" to delete HDFS files/directories
  `(hadoop fs -rmr hdfs://myhost/mypath/mydir)`
- "-put" to copy local system files to HDFS
  `(hadoop fs -put mylocalpath/myfile hdfs://myhost/myfile)`
- "-get" to copy HDFS files to local system
  `(hadoop fs -get hdfs://myhost/myfile mylocalpath/myfile)`
- "-getmerge" to merge HDFS files into a single local file
  `(hadoop fs -getmerge hdfs://myhost/myf1 hdfs://myhost/myf2 myfile)`

See the manual for other commands.

- The simplest way is to create a single JAR archive and run it.
  1. `javac -cp <hadoop-*-core.jar> -d <class-files-dir><java-files>`
  2. `jar -cvf <myapp.jar> - C <class-files-dir>.`
  3. `hadoop jar <myapp.jar [mainClass] args...`
- It is necessary to copy files to HDFS before running the app.

Example:

```
hadoop fs -put file01 hdfs:/localhost/usr/joe/input/
hadoop fs -put file02 hdfs:/localhost/usr/joe/input/
hadoop jar wordcount.jar org.myorg.WordCount /usr/joe/input
/usr/joe/output
hadoop fs -cat /usr/joe/output/part-00000
```

Summary and conclusion

- MapReduce paradigm for parallel, distributed and batch computation.
  (functions Map and Reduce; OLTP/OLAP add-on, not a replacement)
- Hadoop is a framework for running MapReduce applications.
  (NameNode and DataNode for HDFS, JobTracker and TaskTracker for MapReduce)
- Programmer only needs to implement the Map and Reduce functions.
  (potentially org.apache.hadoop.mapred.Partitioner)

Thank you for your attention!