CS 696 Intro to Big Data: Tools and Methods
Fall Semester, 2016
Doc 18 Hadoop Intro
Nov 1, 2016

History

2002 Nutch

- Doug Cutting & Mike Cafarella
- Apache Web search engine project
- 1 billion pages

2003 Google's distributed filesystem paper

2004 Google's MapReduce paper

2006

- Distributed file system & MapReduce moved out of Nutch to create Hadoop
- Doug Cutting joins Yahoo
- Yahoo supported development of Hadoop

2006

- Google's Bigtable paper
- Hadoop sorts 1.8 TB on 188 nodes in 47.9 hours

More History

2007 Hadoop includes HBase, Pig created

2008

- Yahoo! search index uses 10,000-core Hadoop cluster
- Hadoop sorts 1 TB in 209 seconds using 910-node cluster
- Cloudera, Hadoop distributer founded
- Google sorts 1 TB in 68 seconds

2010 Facebook 2,300 cluster/40 Petabytes

- Hive

2011

- Yahoo 42K Hadoop nodes
- Facebook, LinkedIn, eBay and IBM collectively contribute 200,000 lines of code
- Hortonworks spun out of Yahoo

2012 - Hahoop 1.0

2014 San Jose Hadoop Summit 3,200 attendees

- Hadoop 2.6

https://en.wikipedia.org/wiki/Apache_Hadoop

What is Hadoop

Framework for distributed storage & distributed processing of very large data sets

Hadoop Modules

Hadoop Common
- Utilities

Hadoop Distributed File System (HDFS)

Hadoop YARN
- Manage computing resources in clusters & schedule users' applications

Hadoop MapReduce
- Implementation of the MapReduce programming model

What is Hadoop

Java program + native C code + shell scripts

Java Jar file

But there is more

Once people start using Hadoop

- How do you get data in and out?
- People use databases to store data not files
- Machine learning?

Apache Pig

Programming Map-Reduce can be low level

Apache Pig - high-level platform for creating programs for Hadoop

Pig Latin

```
input_lines = LOAD '/tmp/my-copy-of-all-pages-on-internet' AS (line:chararray);
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
filtered_words = FILTER words BY word MATCHES '\\w+';
word_groups = GROUP filtered_words BY word;
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS
                                   count, group AS word;

ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tmp/number-of-words-on-internet';
```

https://en.wikipedia.org/wiki/Pig_(programming_tool)

Apache Hive

SQL is common way to interact with data

Hive provides SQL like query language for HDFS, Amazon S3 data

HiveQL - converted into MapReduce

```
DROP TABLE IF EXISTS docs;
CREATE TABLE docs (line STRING);
LOAD DATA INPATH 'input_file' OVERWRITE INTO TABLE docs;
CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
 (SELECT explode(split(line, '\s')) AS word FROM docs) temp
GROUP BY word
ORDER BY word;
```

Apache HBase

BigTable for Hadoop

Non-relational distributed database

Fault-tolerant way of storing large quantites of sparse data

Apache Sqoop

People have data in non-hadoop databases

Sqoop

- Transferring data between relational databases & Hadoop

Apache Phoenix

But SQL is common

Phoenix

- Massively parallel relational database for Hadoop
- Uses HBase to store data

Apache Spark

Hadoop has latency issues - reads data from disk
MapReduce is not conducive to solving all problems

Spark
- Uses distributed shared memory: Resilient distributed dataset (RDD)
- Iterative algorithms
- Implemented in Scala

Spark Core
Spark SQL
- Dataframes & SQL

Spark Streaming
Spark MLlib
- Machine learning

Apache Mahout

Hadoop does not have machine learning libraries

Mahout

- Environment for quickly creating scalable machine learning applications
- Samsara - R-line syntax & environment

Apache Flink, Apache Storm

Hadoop does batch jobs

Spark streaming has delays

Fling & Storm

- Each calin to have high throughput and low latency streaming

Apache BigTop

There are a lot of programs in the Hadoop Ecosystem so useful to have a way to install them

BigTop

- Install via RPM, DEBs or Docker
- Includes test

Hadoop Ecosystem

- **Hadoop**
    - **HDFS**
    - **MapReduce**
    - **YARN**
- Tez
- **Pig**
- Hive
- **Hbase**
- Sqoop
- Oozie
- Falcon
- **Spark**
- ZooKeeper
- Mahout
- Phoenix
- BigTop
- + others

Two language Problem

Performance	Interactive
Safety	Expressive
Large scale development	Quick development

Java	Scala	Python
C#	Clojure	Matlab
C++	Julia	Ruby
Swift		R
Go		
Rust		

Hadoop & Java

Hadoop is written in Java

All the parts of Hadoop ecosystem are written in JVM language

- Spark - Scala
- Storm - Clojure

JVM code will be more performant in Hadoop Ecosystem

Java not ideal for exploration

Everyone Else

Hadoop Streaming

- Any code that reads standard in write standard out can be used

Pig Latin, HiveQL

- DSL for using Hadoop

Language specific API to Hadoop Ecosystem

- HDFS.jl
- Ely.jl
- DistributedArrays.jl
- Spark.jl

Goal

Understanding basic concepts

Making all the pieces work

- MapReduce
  - Java
  - Streaming
- HDFS
- YARN

Learning how to use map-reduce to solve problems

Hadoop Components

MapReduce
    Does the computation

Hadoop Distributed File System
    Distributes data

YARN - Yet Another Resource Negotiator
    Cluster resource management

Hadoop MapReduce - Outline

Hadoop sends each record to map function

Map
- Input - Individual record as key-value pair
- Output - Key-value pairs

Hadoop
- Sorts the map output by the keys
- Combines values with same key together
- Sends to reduce function

Reduce
- Input
    - One key
    - All the values with that key
- Output
    - Key-value pairs

Examples

Word Count

Find max temperature for year from hourly weather data

Word Count Example

Given a file(s) of text count the number of times each word occurs

Hello World example of Hadoop ecosystem

Examples don’t worry about what a word is

Weather Example for Text

Given raw data from weather station

Find the high temperature for each year

Data set

- Years 1901-2001
- 10,000’s of weather stations

Weather Example From Book

Each line contains multiple measurements without seperators

0057 **332130** 99999 **19500101** 0300 4 +51317 +028783 FM-12 +0171 99999 V020

```
0057
332130   # USAF weather station identifier
99999    # WBAN weather station identifier
19500101 # observation date
0300     # observation time
4
+51317   # latitude (degrees x 1000)
+028783  # longitude (degrees x 1000)
FM-12
+0171    # elevation (meters)
99999
V020
320      # wind direction (degrees)
1        # quality code
N
0072
```

Weather - Sample Input File -> map input

0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
0043012650999991949032418004...0500001N9+00781+99999999999...

$$\downarrow$$

(0, 006701199099999**1950**051507004...9999999N9**+0000**1+99999999999...)
(106, 004301199099999**1950**051512004...9999999N9**+0022**1+99999999999...)
(212, 004301199099999**1950**051518004...9999999N9**-00111**+99999999999...)
(318, 004301265099999**1949**032412004...0500001N9**+01111**+99999999999...)
(424, 004301265099999**1949**032418004...0500001N9**+00781**+99999999999...)

Map Function

Map function gets one key-value pair

Needs to parse value to find year and temperature

Output key-value pair

- key - year
- value - temperature

Repeated multiple times

(0, 0067011990999991**950**051507004...9999999N9**+0000**1+99999999999...)

$\downarrow$

(1950, 0)

MapReduce framework Shuffle

Combines all map function key-value pairs output with same key into one key-value pair

$$\begin{tikzcd}
\begin{array}{l}(1950, 0)\\(1950, 22)\\(1950, -11)\\(1949, 111)\\(1949, 78)\end{array} \arrow[r] & \begin{array}{l}(1949, [111, 78])\\(1950, [0, 22, -11])\end{array}
\end{tikzcd}$$

Reduce Function

Reduce function gets one key-value pair

Finds max temperature

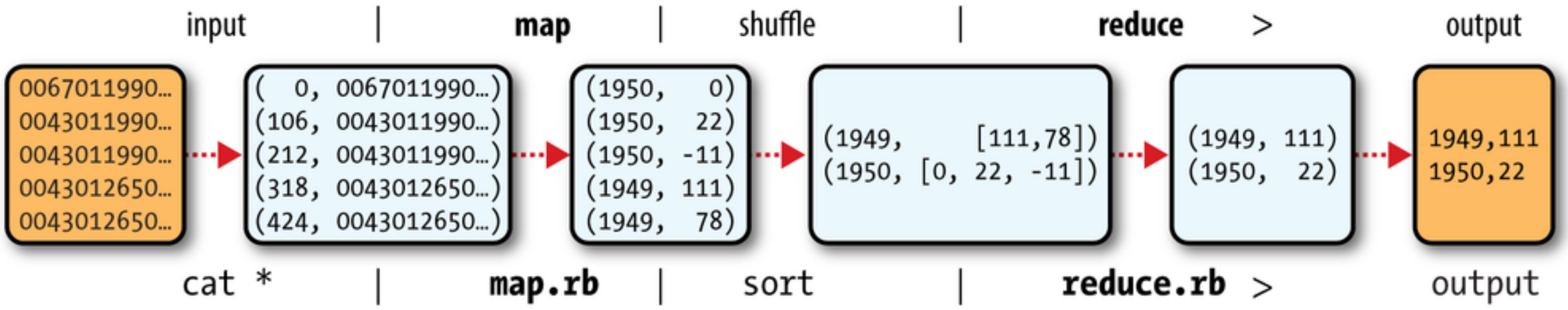
Output key-value pair

- key - year
- value - max temperature

Repeated multiple times

$$\text{(1949, [111, 78])} \longrightarrow \text{(1949, 111)}$$

$$\text{(1950, [0, 22, −11])} \longrightarrow \text{(1950, 22)}$$


input | map | shuffle | reduce > output
0067011990...
0043011990...
0043011990...
0043012650...
0043012650...
(0, 0067011990...)
(106, 0043011990...)
(212, 0043011990...)
(318, 0043012650...)
(424, 0043012650...)
(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
(1949, 78)
(1949, [111,78])
(1950, [0, 22, -11])
(1949, 111)
(1950, 22)
1949,111
1950,22
cat * | map.rb | sort | reduce.rb > output

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

  private static final int MISSING = 9999;

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    String line = value.toString();
    String year = line.substring(15, 19);
    int airTemperature;
    if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs
      airTemperature = Integer.parseInt(line.substring(88, 92));
    } else {
      airTemperature = Integer.parseInt(line.substring(87, 92));
    }
    String quality = line.substring(92, 93);
    if (airTemperature != MISSING && quality.matches("[01459]")) {
      context.write(new Text(year), new IntWritable(airTemperature));
    }
  }
}
```

```
public class MaxTemperatureReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {

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

    int maxValue = Integer.MIN_VALUE;
    for (IntWritable value : values) {
      maxValue = Math.max(maxValue, value.get());
    }
    context.write(key, new IntWritable(maxValue));
  }
}
```

```
public class MaxTemperature {
 public static void main(String[] args) throws Exception {
  if (args.length != 2) {
   System.err.println("Usage: MaxTemperature <input path> <output path>");
   System.exit(-1);
  }

  Job job = new Job();
  job.setJarByClass(MaxTemperature.class);
  job.setJobName("Max temperature");

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

  job.setMapperClass(MaxTemperatureMapper.class);
  job.setReducerClass(MaxTemperatureReducer.class);

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

  System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

Runing the Program in Standalone Mode

```
export HADOOP_CLASSPATH=hadoop-examples.jar
hadoop MaxTemperature pathToInputFile outputDir
```

Terms

MapReduce Job

Tasks
- Map
- Reduce

Input splits
- Divided into records
- Splits handled in parallel
- Ideal is to map split to machine holding data

HDFS Block size - 128 MB
- If split is larger than 128 MB may span machines

Use HDFS to distribute data into 128MB block
Each block resides on a machine

Per machine
- Read block as split
- Divide split into records
- Send each record to map
- Store map results on local file system
- Send map function results to reduce function

Reducer function not likely on same machine

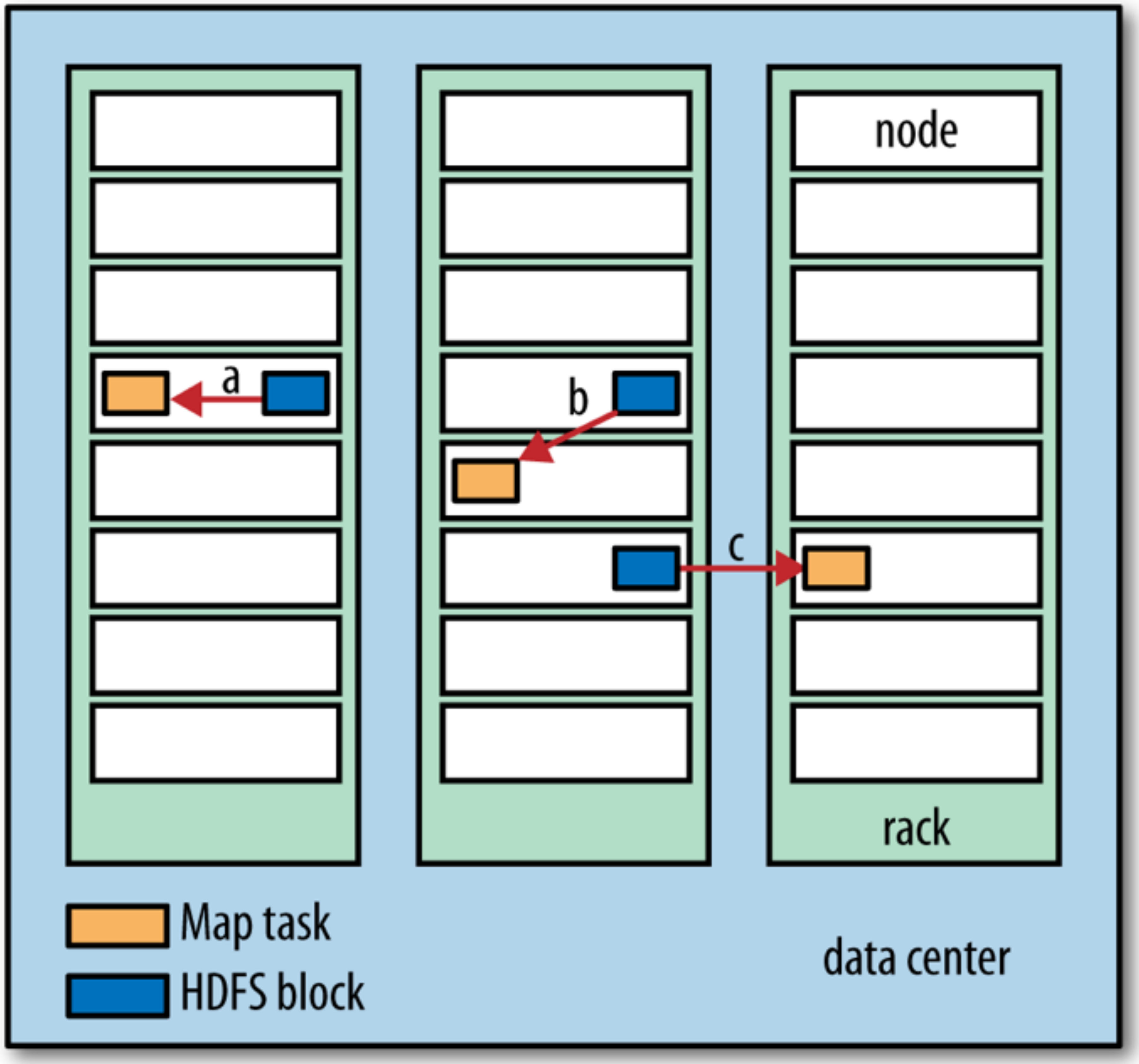
Potential Bottlenecks

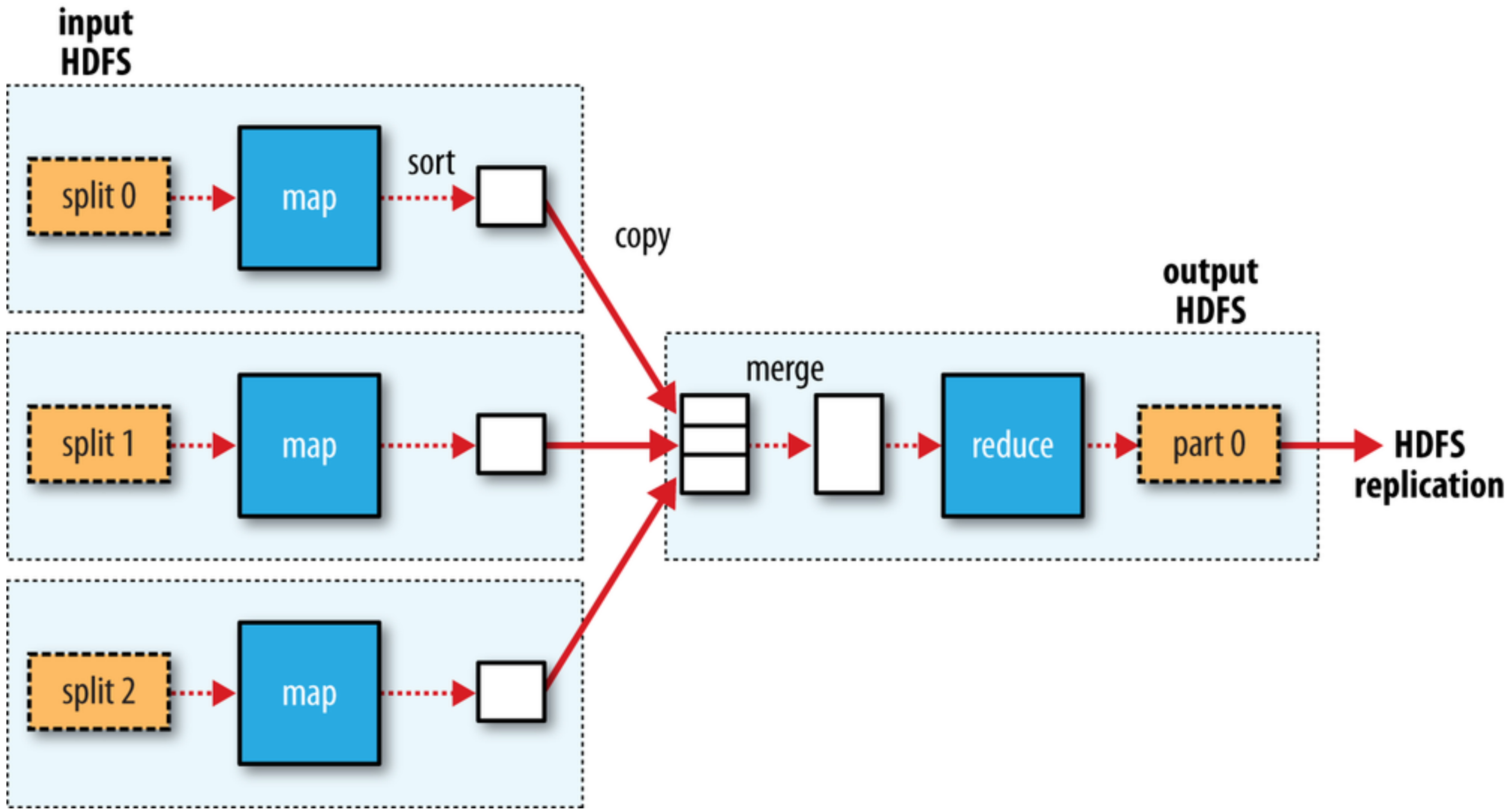
Writing map results to disk

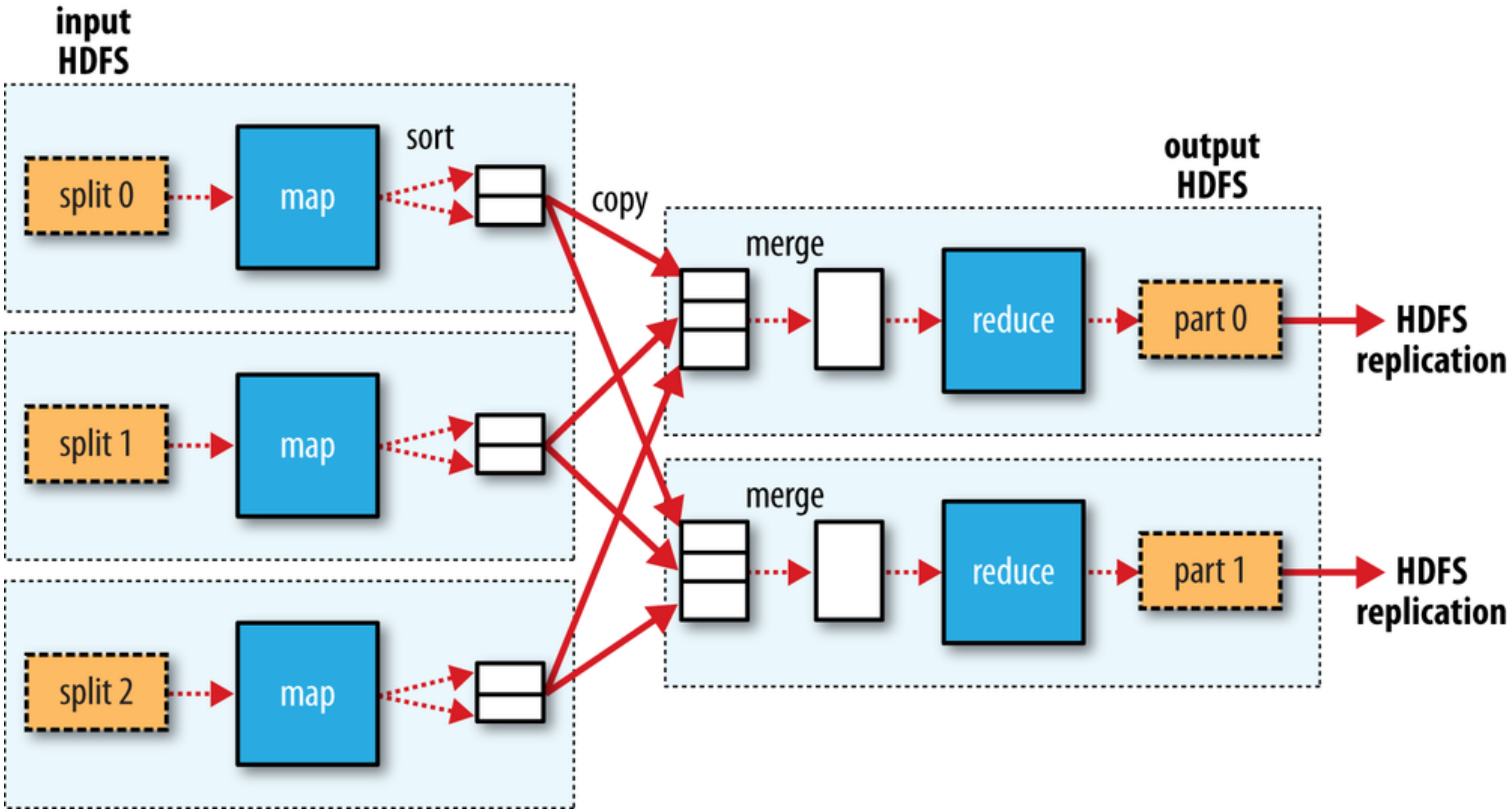
Node busy with other jobs

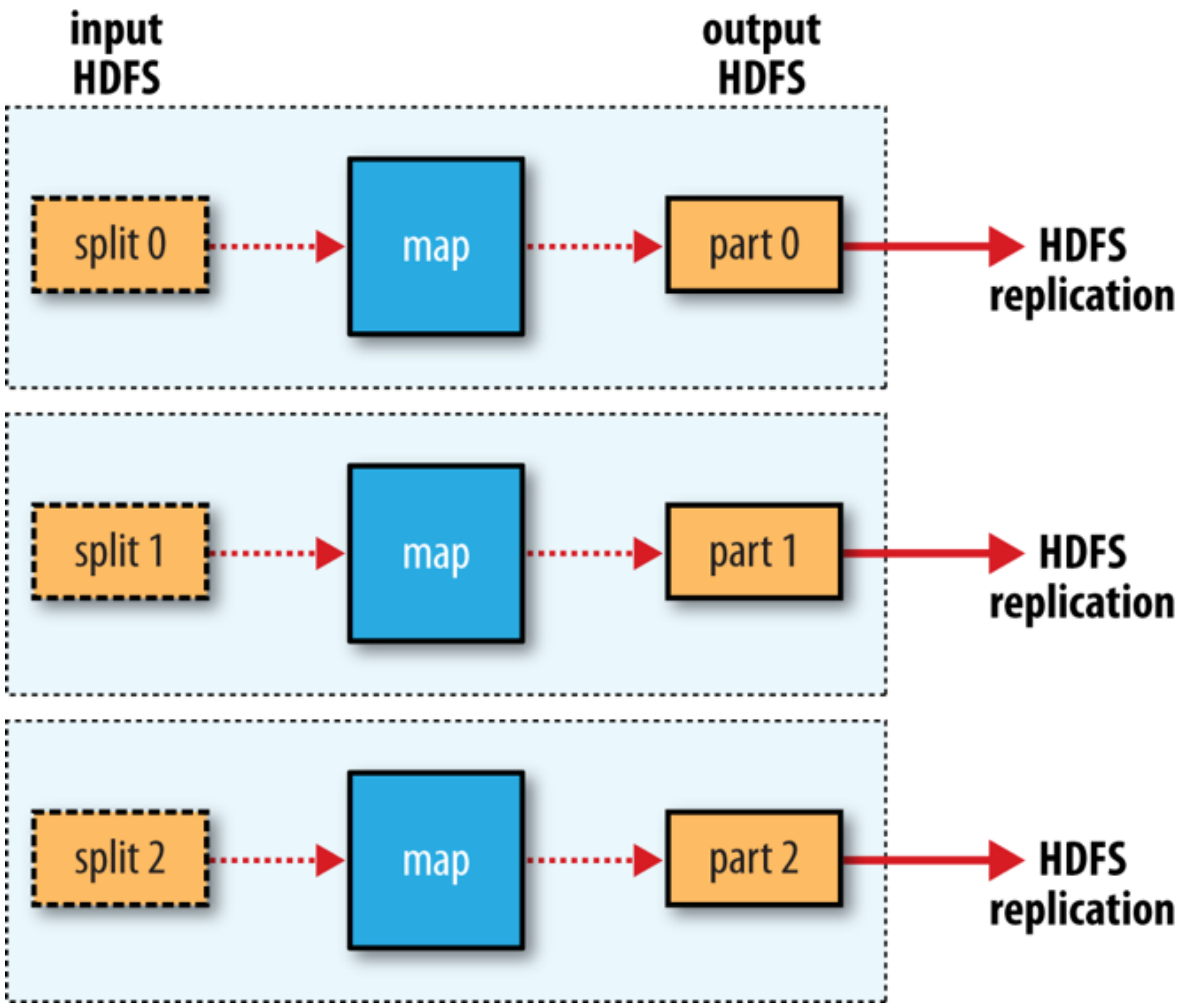
Amount of data sent to reducer function

Number of reducers


node
a
b
c
rack
Map task
HDFS block
data center


input
HDFS
split 0
map
sort
split 1
map
split 2
map
copy
output
HDFS
merge
reduce
part 0
HDFS
replication


input
HDFS
split 0
map
sort
split 1
map
split 2
map
copy
output
HDFS
merge
reduce
part 0
HDFS
replication
merge
reduce
part 1
HDFS
replication


input HDFS
output HDFS
split 0
map
part 0
HDFS replication
split 1
map
part 1
HDFS replication
split 2
map
part 2
HDFS replication

Combiner Functions

128 MB of weather data will produce a lot of map output

Since we only want the max temperature per year

Find max temperature per year before send to reduce

Can add combiner function to combine the map output

Combiner Functions

May be called multiple times on same data or not at all

Combiner functions need to be
- associative
  - $a * (b * c) = (a * b) * c$
- commutative
  - $a * b = b * a$

Not all functions are are associative & commutative

```
public class MaxTemperatureWithCombiner {

  public static void main(String[] args) throws Exception {
    Job job = new Job();
    job.setJarByClass(MaxTemperatureWithCombiner.class);
    job.setJobName("Max temperature");

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

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setCombinerClass(MaxTemperatureReducer.class);
    job.setReducerClass(MaxTemperatureReducer.class);

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

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Hadoop Streaming

- Standard Unix Streams
  - Raw bytes & typed bytes for more efficient transfer

  - map
    - Reads data from standard in
    - Can read multiple records at a time
    - Write key-value pairs to standard out
    - tab separates key from value

  - Reduce
    - Reads key-tab-value pairs from standard in
    - Can read multiple records at a time
    - Write key-value pairs to standard out
    - tab separates key from value

- Hadoop Pipes - for C++

Typed Bytes & Overhead

Early version of Hadoop

300 GB of web logs to count how many time IP address appears

Python Streaming using Dumbo

	Time in Minutes
Java	8
Streaming with Type Bytes	10
Hive	13
Streaming without Type Bytes using Special Java IO classes	16

https://dumbotics.com/2009/02/24/hadoop-1722-and-typed-bytes/

Sample Map function

```
#!/usr/bin/env ruby

STDIN.each_line do |line|
  val = line
  year, temp, q = val[15,4], val[87,5], val[92,1]
  puts "#{year}\t#{temp}" if (temp != "+9999" && q =~ /[01459]/)
end


#!/usr/bin/julia

for line in eachline(STDIN)
  year, temperature, q = (line[16:19], line[88:92], line[93:93])
  if (temperature != "+9999" && ismatch(r"[01459]", q))
    print("$year\t$temperature")
  end
end
```

Running Streaming Program

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
  -files max_temperature_map.rb,\
                   max_temperature_reduce.rb \
  -input input/ncdc/all \
  -output output \
  -mapper max_temperature_map.rb \
  -combiner max_temperature_reduce.rb \
  -reducer max_temperature_reduce.rb
```

Installing Hadoop - Linux & Mac

http://hadoop.apache.org/docs/r2.7.3/hadoop-project-dist/hadoop-common/SingleCluster.html
https://goo.gl/rtls7t

Download tar file and unpack

Startwith Standalone Operation

Run the Standalone example on the install guide

Apache Hadoop Tutorial & Sample Programs

http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

https://goo.gl/dt2Vwy

Sample Programs

aggregatewordcount:, aggregatewordhist:,
grep:
join:
multifilewc:
pentomino:
pi:
randomtextwriter:
randomwriter:
secondarysort:
sort:
sudoku:
terasort:
wordcount:, wordmean:, wordmedian:, wordstandarddeviation:

Running Hadoop

Standalone Operation

Pseudo-Distributed Operations

Cluster Operation

Installing on Windows

http://wiki.apache.org/hadoop/Hadoop2OnWindows

https://goo.gl/WlEhJc

Requirements

Windows System
JDK 1.6+
Maven 3.0 or later
Findbugs 1.3.9 (if running findbugs)
ProtocolBuffer 2.5.0
CMake 2.6 or newer
Windows SDK or Visual Studio 2010 Professional
Unix command-line tools from GnuWin32 or Cygwin: sh, mkdir, rm, cp, tar, gzip
zlib headers (if building native code bindings for zlib)
Internet connection for first build (to fetch all Maven and Hadoop dependencies)

Installing on Windows - Option 2

Install Linux on
- Separate partition
- USB drive
- In VM
    - VMWare
    - Docker
    - VirtualBox

Follow regular instruction on installing Hadoop

Hortonworks Hadoop Sandbox

All of Hadoop Ecosystem in VM

- VirtualBox
- VMware
- Docker

11 GB download

http://hortonworks.com/products/sandbox/