Hadoop MapReduce

Felipe Meneses Besson

IME-USP, Brazil

July 7, 2010

Agenda

- What is Hadoop?
- Hadoop Subprojects
- MapReduce
- HDFS
- Development and tools

What is Hadoop?

A framework for large-scale data processing (Tom White, 2009)**:**

- Project of Apache Software Foundation
- Most written in Java
- Inspired in Google MapReduce and GFS (Google File System)

A brief history

- 2004: Google published a paper that introduced MapReduce and GFS as a alternative to handle the volume of data to be processed
- 2005: Doug Cutting integrated MapReduce in the Hadoop
- 2006: Doug Cutting joins Yahoo!
- 2008: Cloudera[1] was founded
- 2009: Hadoop cluster sort 100 terabyte in 173 minutes (on 3400 nodes)[2]

Nowadays, Cloudera company is an active contributor to the Hadoop project and provide Hadoop consulting and commercial products.

[1]Cloudera: http://www.cloudera.com

[2] Sort Benchmark: http://sortbenchmark.org/

Hadoop Characteristics

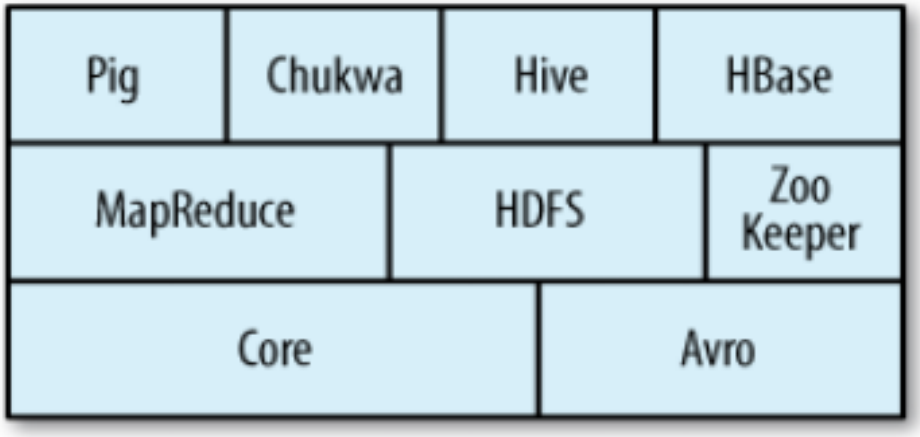
- A scalable and reliable system for shared storage and analyses.
- It automatically handles data replication and node failure
- It does the hard work – developer can focus on processing data logic
- Enable applications to work of petabytes of data in parallel

Who's using Hadoop

Source: Hadoop wiki, September 2009

Hadoop Subprojects

Apache Hadoop is a collection of related subprojects that fall under the umbrella of infrastructure for distributed computing.



All projects are hosted by the Apache Software Foundation.

MapReduce

MapReduce is a programming model and an associated implementation for processing and generating large data sets (Jeffrey Dean and Sanjay Ghemawat, 2004)

- Based on a functional programming model
- A batch data processing system
- A clean abstraction for programmers
- Automatic parallelization & distribution
- Fault-tolerance

MapReduce

Programming model

Users implement the interface of two functions:

map (in_key, in_value) ->

(out_key, intermediate_value) list

reduce (out_key, intermediate_value list) ->

out_value list

MapReduce

Map Function

Input:

- **Records from some data source (e.g., lines of files, rows of a databases, …) are associated in the (key, value) pair**
  - **Example: (filename, content)**

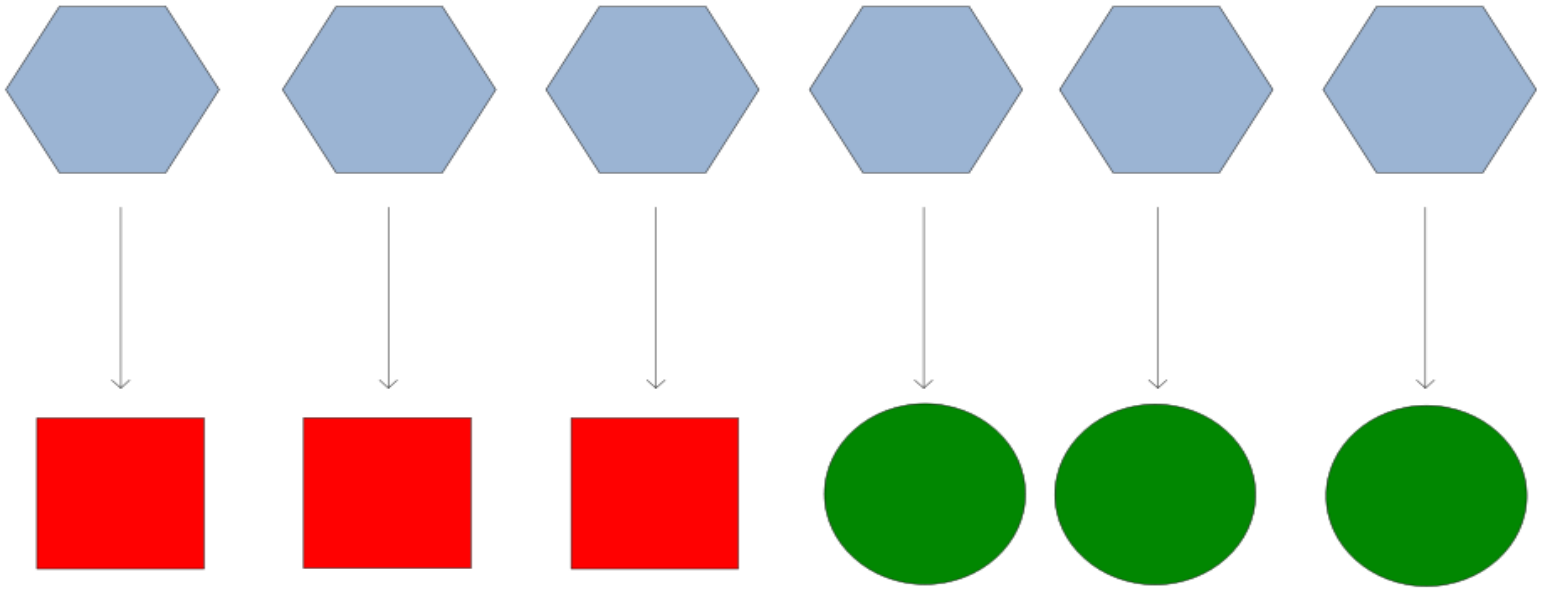
Output:

- **One or more intermediate values in the (key, value) format**
  - **Example: (word, number_of_occurrences)**

MapReduce

Map Function

map (in_key, in_value) → (out_key, intermediate_value) list



Source: (Cloudera, 2010)

MapReduce

Map Function

Example:

map (k, v):

if (isPrime(v)) then emit (k, v)

(“foo”, 7) ➡ (“foo”, 7)

(“test, 10) ➡ (nothing)

MapReduce

Reduce function

After map phase is over, all the intermediate values for a given output key are combined together into a list

Input:

- **Intermediate values**
  - **Example: (“A”, [42, 100, 312])**

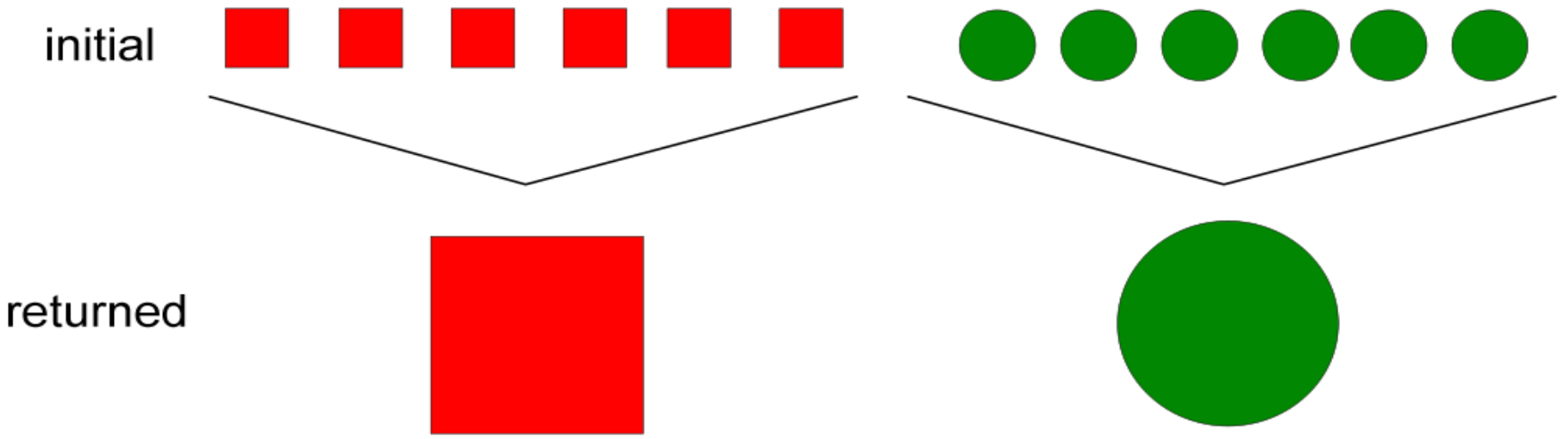
Output:

- **usually only one final value per key**
  - **Example: (“A”, 454)**

MapReduce

Reduce Function

reduce (out_key, intermediate_value list) → out_value list



Source: (Cloudera, 2010)

MapReduce

Reduce Function

Example:

```
reduce (k, vals):
    sum = 0
    foreach int v in vals:
        sum += v
    emit (k, sum)
```

("A", [42, 100, 312]) ➡ ("A", 454)

("B", [12, 6, -2]) ➡ ("B", 16)

MapReduce

Terminology

Job: unit of work that the client wants to be performed

- Input data + MapReduce program + configuration information

Task: part of the job

- map and reduce tasks

Jobtracker: node that coordinates all the jobs in the system by scheduling tasks to run on tasktrackers

MapReduce

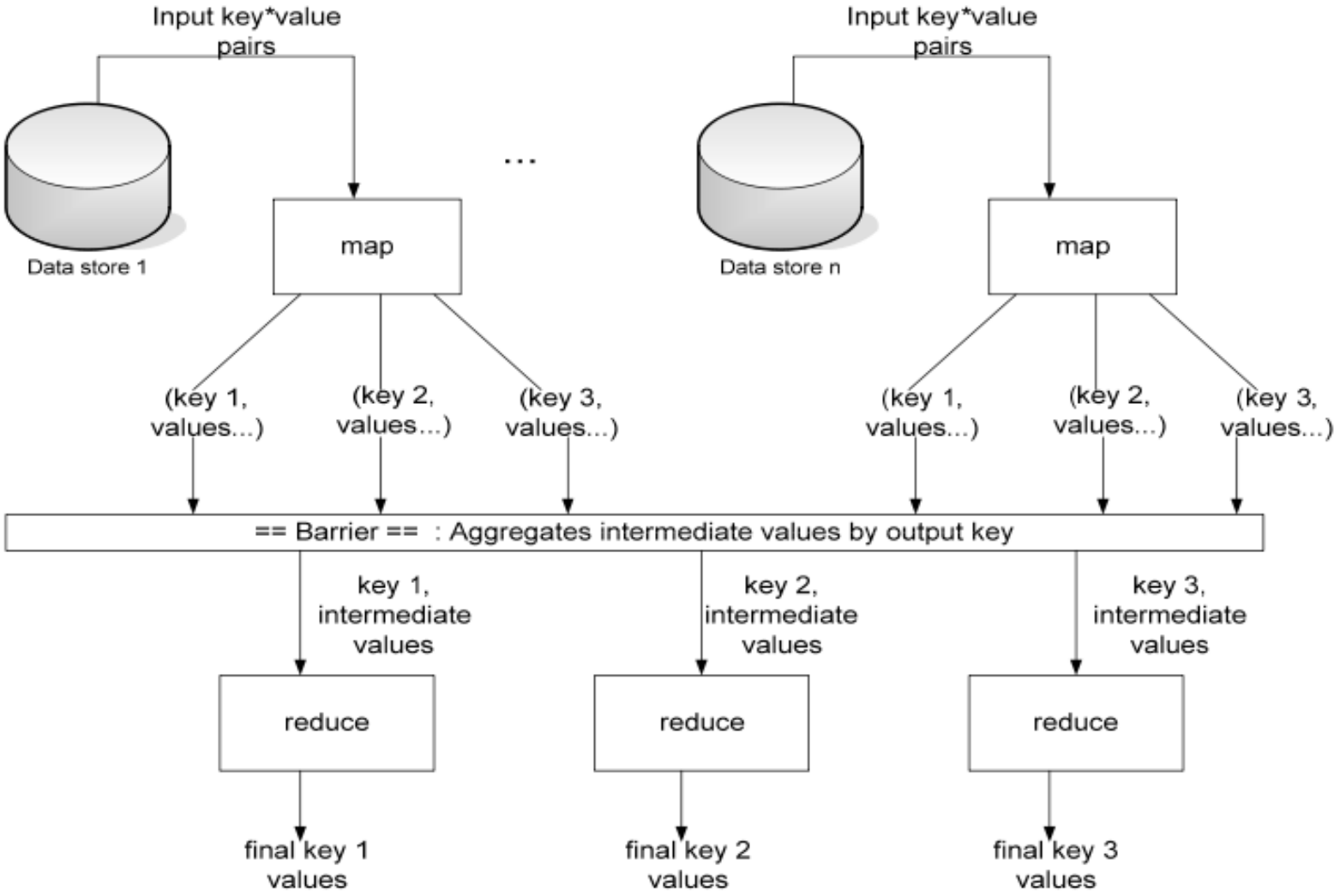
Terminology

Tasktracker: nodes that run tasks and send progress reports to the jobtracker

Split: fixed-size piece of the input data

MapReduce

DataFlow



Source: (Cloudera, 2010)

MapReduce

Real Example

```
map (String key, String value):
        // key: document name
        // value: document contents
        for each word w in value:
            EmitIntermediate(w, "1");
```

MapReduce

Real Example

```
reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

MapReduce

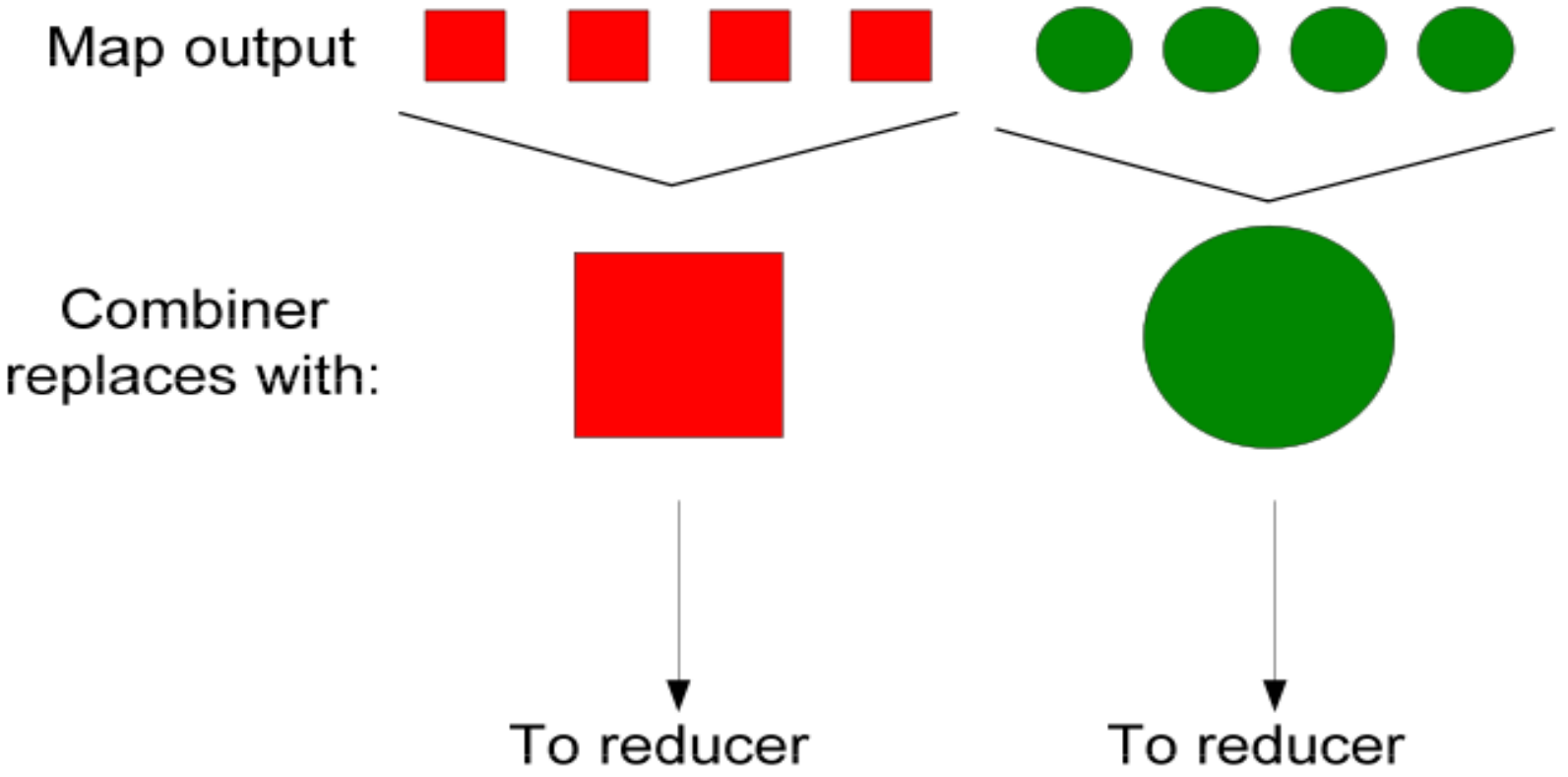
Combiner function

- Compress the intermediate values
- Run locally on mapper nodes after map phase
- It is like a “mini-reduce”
- Used to save bandwidth before sending data to the reducer

MapReduce

Combiner Function

Applied in a mapper machine



Source: (Cloudera, 2010)

HDFS

Hadoop Distributed Filesystem

- **Inspired on GFS**
- **Designed to work with very large files**
- **Run on commodity hardware**
- **Streaming data access**
- **Replication and locality**

HDFS

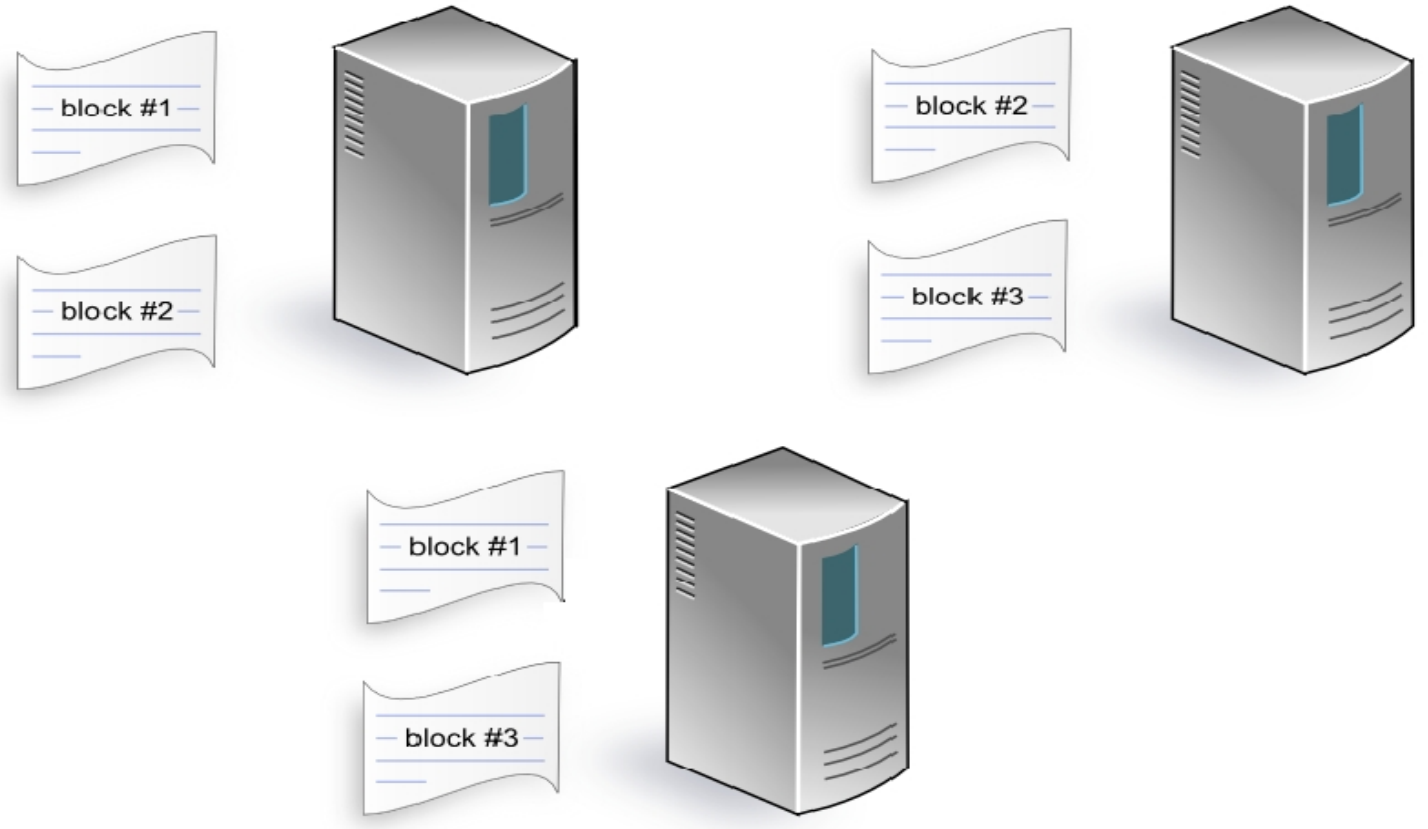
Nodes

- **A Namenode (the master)**
  - Manages the filesystem namespace
  - Knows all the blocks location
- **Datanodes (workers)**
  - Keep blocks of data
  - Report back to namenode its lists of blocks periodically

HDFS

Duplication

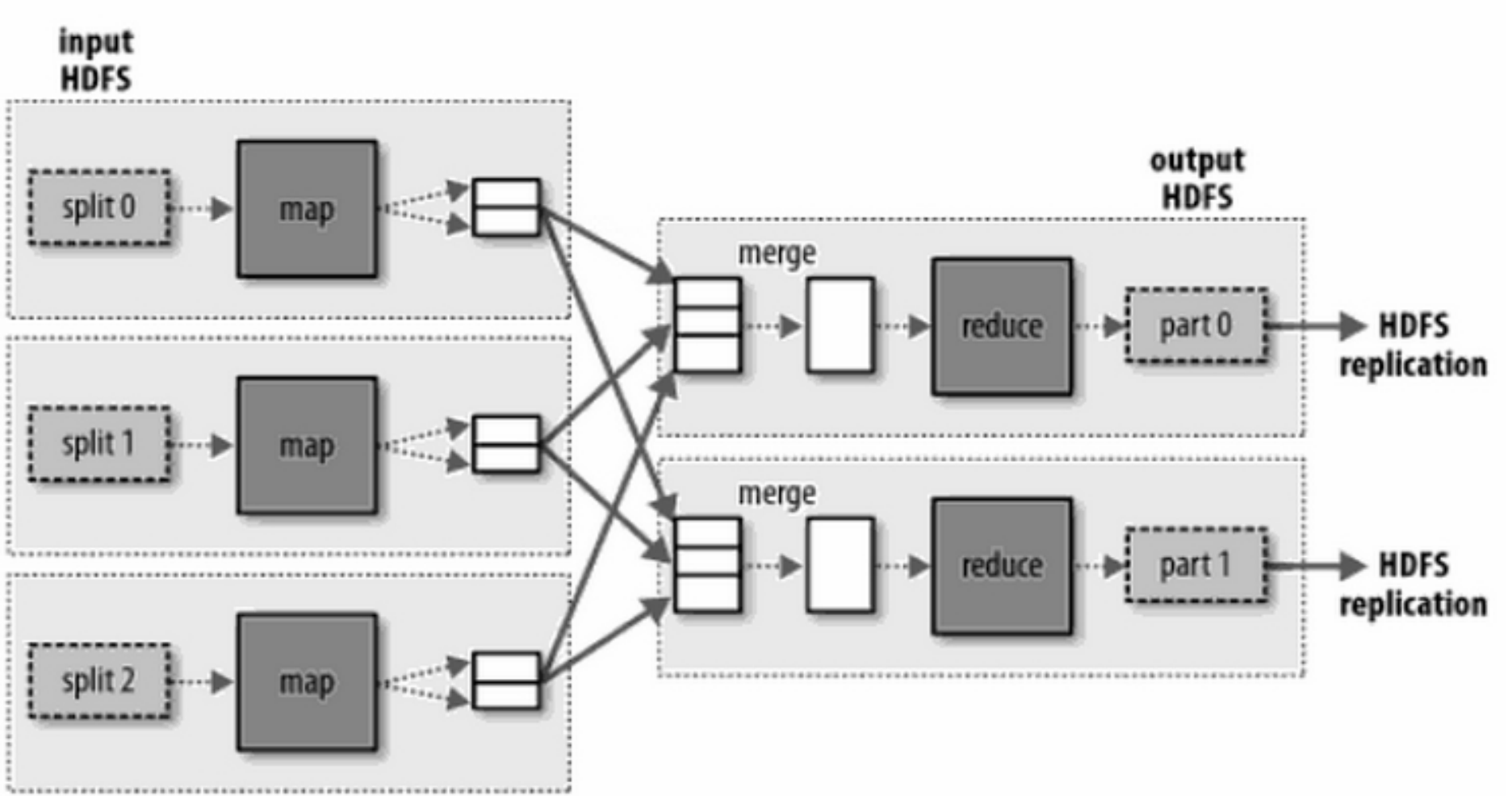
Input data is copied into HDFS is split into blocks



Each data blocks is replicated to multiple machines

HDFS

MapReduce Data flow



Source: (Tom White, 2009)

Hadoop filesystems

Filesystem	URI scheme	Java implementation (all under org.apache.hadoop)	Description
Local	`file`	`fs.LocalFileSystem`	A filesystem for a locally connected disk with client-side checksums. Use `RawLocalFileSystem` for a local filesystem with no checksums.
HDFS	`hdfs`	`hdfs.DistributedFileSystem`	Hadoop's distributed filesystem. HDFS is designed to work efficiently in conjunction with MapReduce.
HFTP	`hftp`	`hdfs.HftpFileSystem`	A filesystem providing read-only access to HDFS over HTTP. (Despite its name, HFTP has no connection with FTP.) Often used with **distcp** to copy data between HDFS clusters running different versions.
HSFTP	`hsftp`	`hdfs.HsftpFileSystem`	A filesystem providing read-only access to HDFS over HTTPS. (Again, this has no connection with FTP.)
HAR	`har`	`fs.HarFileSystem`	A filesystem layered on another filesystem for archiving files. Hadoop Archives are typically used for archiving files in HDFS to reduce the namenode's memory usage.
KFS (CloudStore)	`kfs`	`fs.kfs.KosmosFileSystem`	CloudStore (formerly Kosmos filesystem) is a distributed filesystem like HDFS or Google's GFS, written in C++. Find more information about it at http://kosmosfs.sourceforge.net/.
FTP	`ftp`	`fs.ftp.FTPFileSystem`	A filesystem backed by an FTP server.
S3 (native)	`s3n`	`fs.s3native.NativeS3FileSystem`	A filesystem backed by Amazon S3. See http://wiki.apache.org/hadoop/AmazonS3.
S3 (block-based)	`s3`	`fs.s3.S3FileSystem`	A filesystem backed by Amazon S3, which stores files in blocks (much like HDFS) to overcome S3's 5 GB file size limit.

Source: (Tom White, 2009)

Development and Tools

Hadoop operation modes

Hadoop supports three modes of operation:

- **Standalone**
- **Pseudo-distributed**
- **Fully-distributed**

More details:

http://oreilly.com/other-programming/excerpts/hadoop-tdg/installing-apache-hadoop.html

Development and Tools

Java example

```
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);
    }
  }
}
```

Development and Tools

Java example

```
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);
  }
}
```

Development and Tools

Java example

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
  if (otherArgs.length != 2) {
    System.err.println("Usage: WordCount <in> <out>");
    System.exit(2);
  }
  Job job = new Job(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
  FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Development and Tools

Guidelines to get started

The basic steps for running a Hadoop job are:

- Compile your job into a JAR file
- Copy input data into HDFS
- Execute hadoop passing the jar and relevant args
- Monitor tasks via Web interface (optional)
- Examine output when job is complete

Development and Tools

Api, tools and training

Do you want to use a scripting language?

- http://wiki.apache.org/hadoop/HadoopStreaming
- http://hadoop.apache.org/core/docs/current/streaming.html

Eclipse plugin for MapReduce development

- http://wiki.apache.org/hadoop/EclipsePlugIn

Hadoop training (videos, exercises, …)

- http://www.cloudera.com/developers/learn-hadoop/training/

Bibliography

Hadoop – The definitive guide

Tom White (2009). Hadoop – The Definitive Guide. O'Reilly, San Francisco, 1st Edition

Google Article

Jeffrey Dean and Sanjay Ghemawat (2004). MapReduce: Simplified Data Processing on Large Clusters. Available on: http://labs.google.com/papers/mapreduce-osdi04.pdf

Hadoop In 45 Minutes or Less

Tom Wheeler. Large-Scale Data Processing for Everyone. Available on: http://www.tomwheeler.com/publications/2009/lambda_lounge_hadoop_200910/twheeler-hadoop-20091001-handouts.pdf

Cloudera Videos and Training

http://www.cloudera.com/resources/?type=Training