Optimize the execution of local physics analysis workflows using Hadoop

INFN CCR - GARR Workshop
14 - 17 May Napoli

Hassen Riahi	INFN-PERUGIA
Giacinto Donvito	INFN-Bari
Livio Fano‘	INFN-PERUGIA
Massimiliano Fasi	INFN-PERUGIA
Andrea Valentini	INFN-PERUGIA

Outline

- Introduction
- Apache Hadoop framework
- Motivation
- ROOT files analysis using Hadoop MapReduce
- Results
- Summary and future works

Introduction

- Hadoop is a data processing system that follows the MapReduce paradigm for scalable data analysis.
- Largest install is at Facebook, a huge Hadoop user
  - 30PB of online disk.
- Being used by many other companies such as Yahoo, Last.fm, New York Times ...

Hadoop FS: HDFS

- HDFS is currently used at some CMS Tier-2 sites and it has shown satisfactory performance
- Advantages:
  - Manageability
    - Decommissioning Hardware
    - Recovery from Hardware failure
  - Reliability
    - Replication works well
  - Usability
    - Provide a Posix-like interface though Fuse
  - Scalability

Hadoop Processing: MapReduce

- Is a framework for distributed data processing
- Provides a simple programming model for efficient distributed computing
  - It works like a Unix pipeline:

    Input | Map | Sort | Reduce | Output
- Let's users plug-in own code

Advantages of Hadoop MapReduce

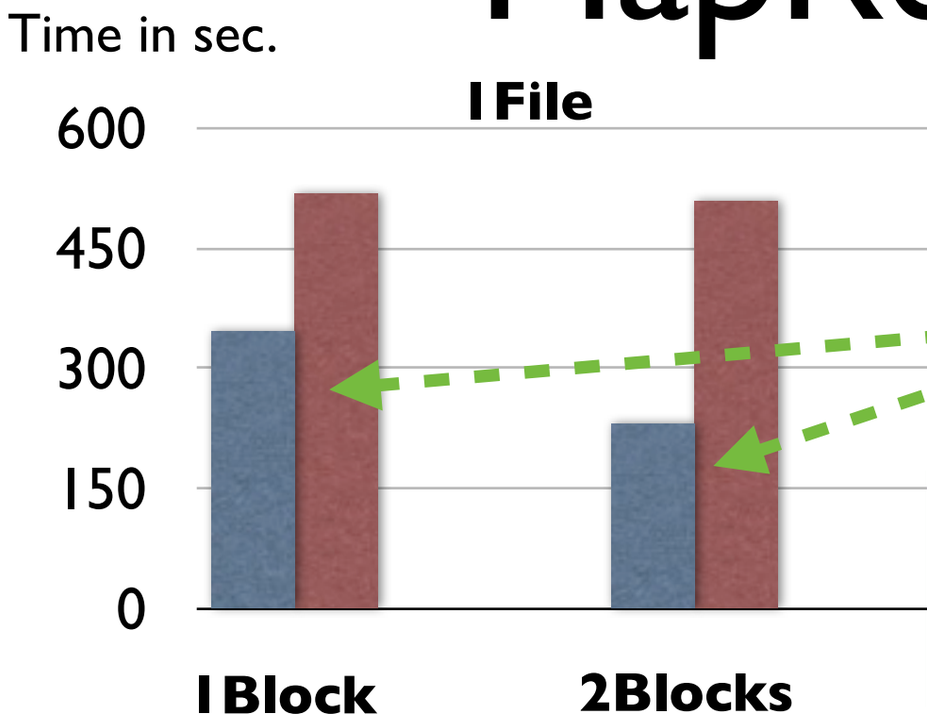
- Runs MapReduce programs written in various language
- Scalability
  - Fine grained Map and Reduce tasks
    - Improved load balancing
    - Faster recovery from failed tasks
  - Locality optimizations
    - Tasks are scheduled close to the inputs when possible
- Reliability
  - Automatic resubmission on failure
    - Framework re-executes failed tasks
- Manageability
  - Powerful Web monitoring interface

Hadoop MapReduce + HDFS solution characteristics

- Commodity HW: add inexpensive servers
  - Use replication across servers to deal with unreliable storage/servers
  - Handle task resubmission on failure
- Simple FS design: Metadata-data separation
- Support for moving computation close to data
  - Servers have 2 purposes: data storage and computation

Use MapReduce + HDFS solution to perform physics analysis in HEP

Performance test MapReduce vs Fuse

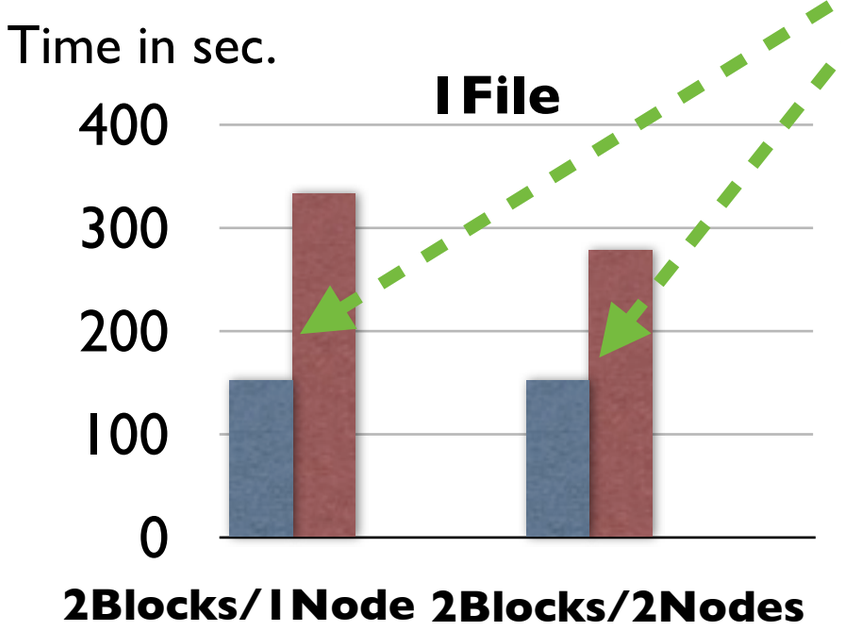


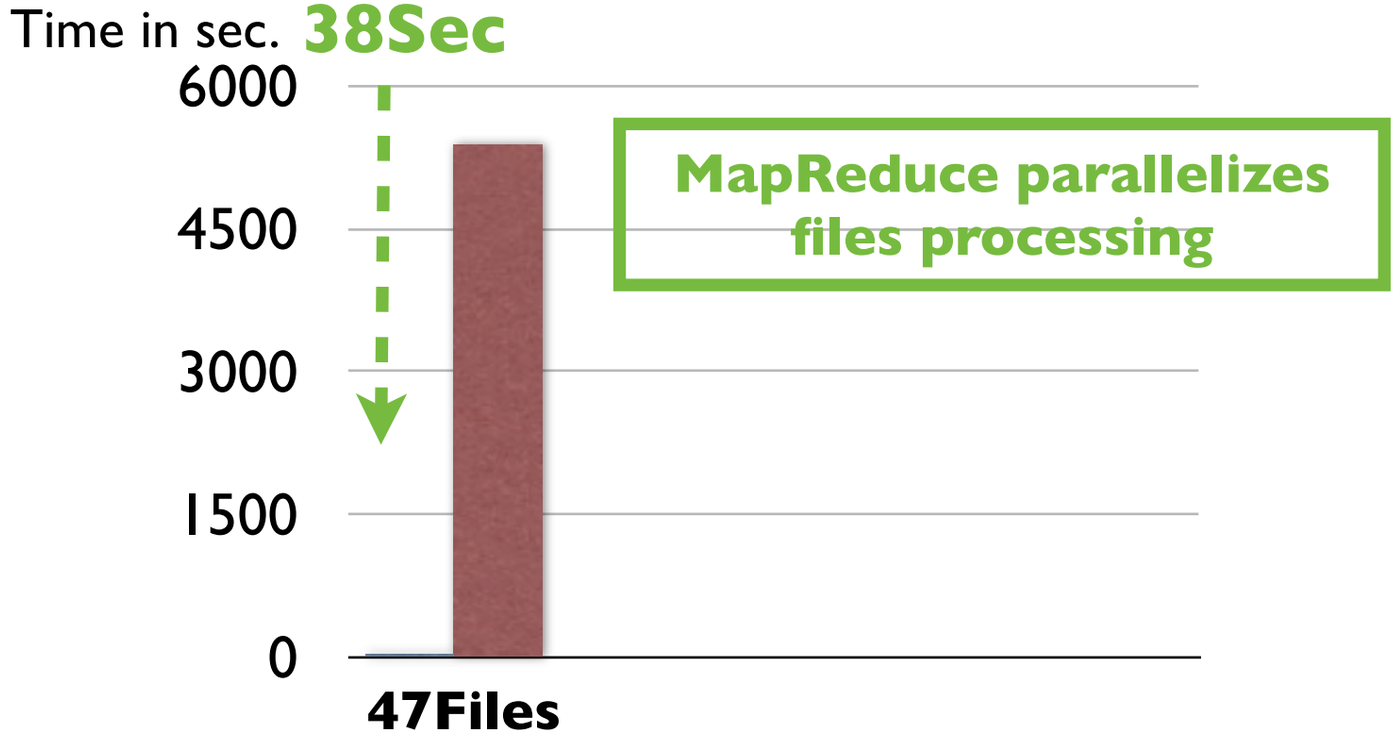
Time in seconds to process 2G files with MapReduce and Fuse

MapReduce Fuse

MapReduce process 2 HDFS blocks of the same file in parallel

Same performance when processing 2 blocks in the same node and 2 blocks in 2 different nodes

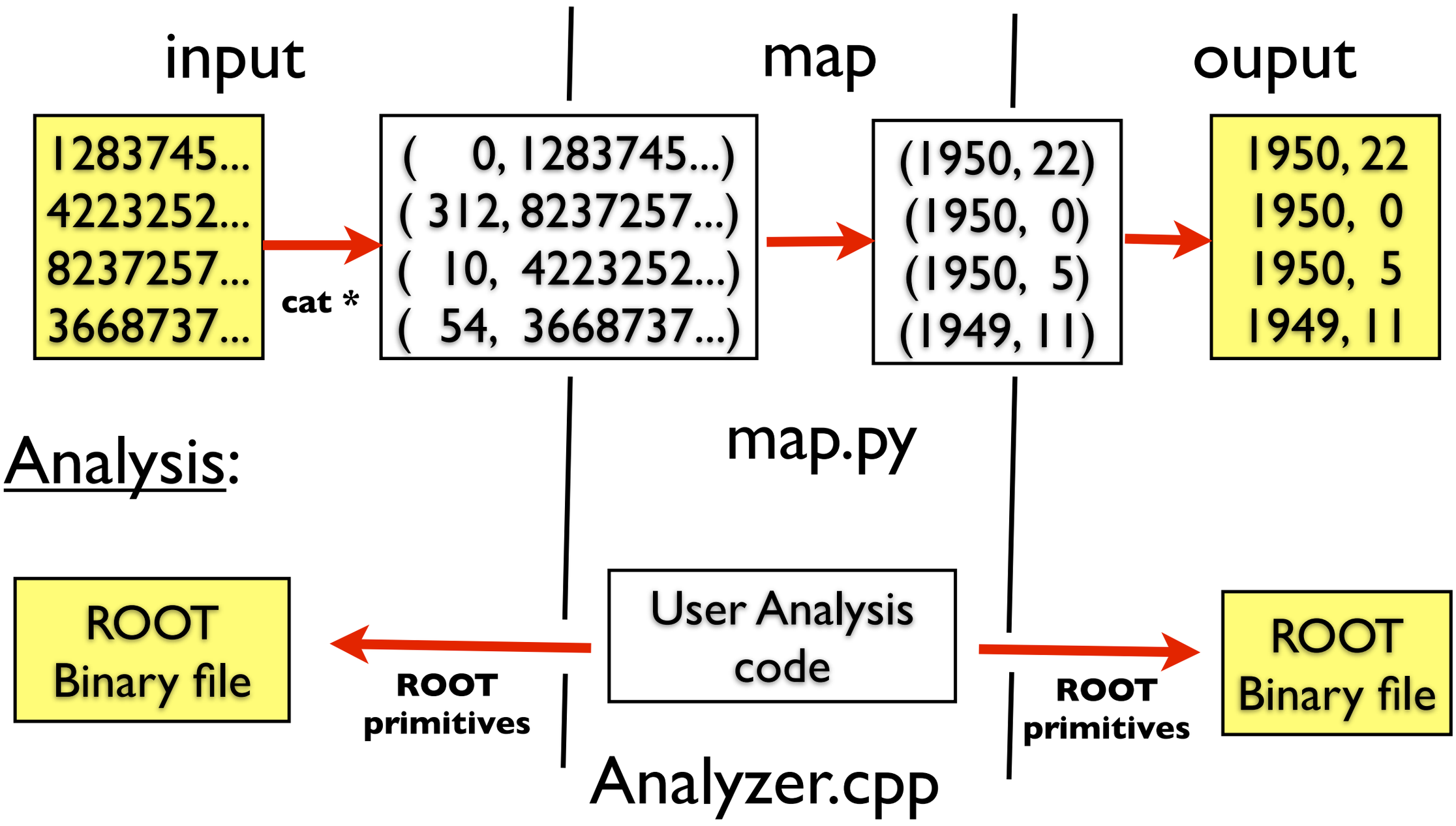




MapReduce parallelizes files processing

Data flows

MapReduce:



Analysis:

ROOT files analysis using Hadoop MapReduce

MapReduce workflow

- Input: default TXT files
  - Split per line
- Map
  - Process TXT input streams
- Output: default TXT files

CMS Analysis workflow

- Input: ROOT files
  - Split per file
- Analyzer
  - Open and process ROOT files
- Output: ROOT files

ROOT file analysis execution

1. Use the dedicated InputFormat, RootFileInputFormat, developed to support the splitting and read of ROOT files

2. Use map.input.file environment to locate the path of the input ROOT file in HDFS

3. Open the ROOT file using the HDFS Plugin of ROOT

- Patch submitted to export correctly the CLASSPATH environment variable in the Plugin

4. Process the ROOT file content

5. Write the output into HDFS

Testbed and setup description

• Hadoop 0.20 is installed on a dedicated Cluster composed of 3 machines (Debian GNU/Linux 6.0, 24 Cores, 24 G RAM and 1 disk 500 G):

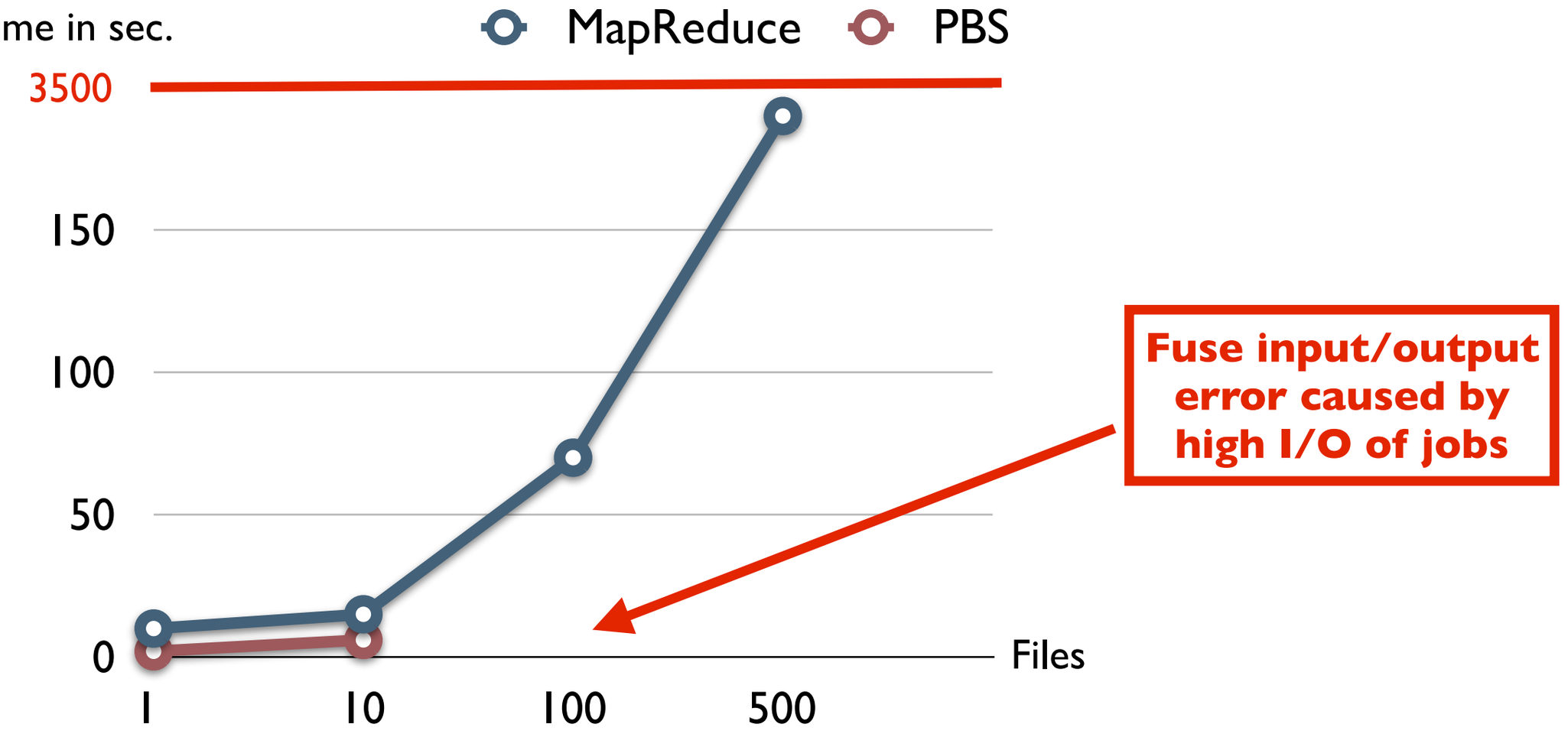
- AdminNode configured as JobTracker, NameNode, and SecondaryNameNode
- Salve nodes configured as TaskTrackers and DataNodes

• AdminNode setup:

1. Add a Hadoop group and Hadoop user for that group
2. Configure and test SSH operation on all nodes
3. Change ownership of all the release contents to permit use by the Hadoop user
4. Add and export for JAVA_HOME and HADOOP_HOME to the .bashrc file
5. Uncommenting the export for JAVA_HOME in hadoop-env.sh
6. Edit the file <yourHadoopHomePath>/conf/core-site.xml to suit your needs
7. Create the hadoop temp datastore directory and change ownership to allow use by the Hadoop user
8. Test Hadoop execution: Start/Stop the single-node cluster: start-all.sh/stop-all.sh

Preliminary test

- Read a Tree from ROOT file and write TH1 histogram from one of its variables in a new output ROOT file
- Size of the ROOT file read is **100MB**
- R/W into HDFS through Fuse

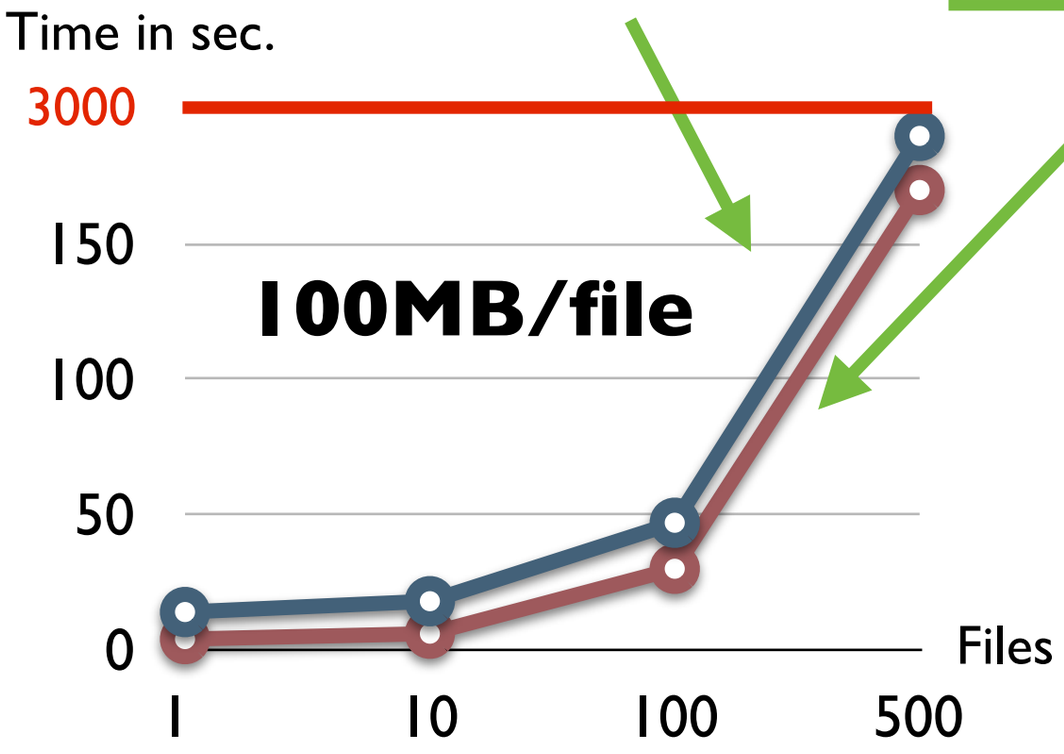


Results

- Read a Tree from ROOT file and write TH1 histogram from one of its variables in a new output ROOT file
- Read from HDFS using ROOT's HDFS Plugin
- Write into HDFS using libhdfs

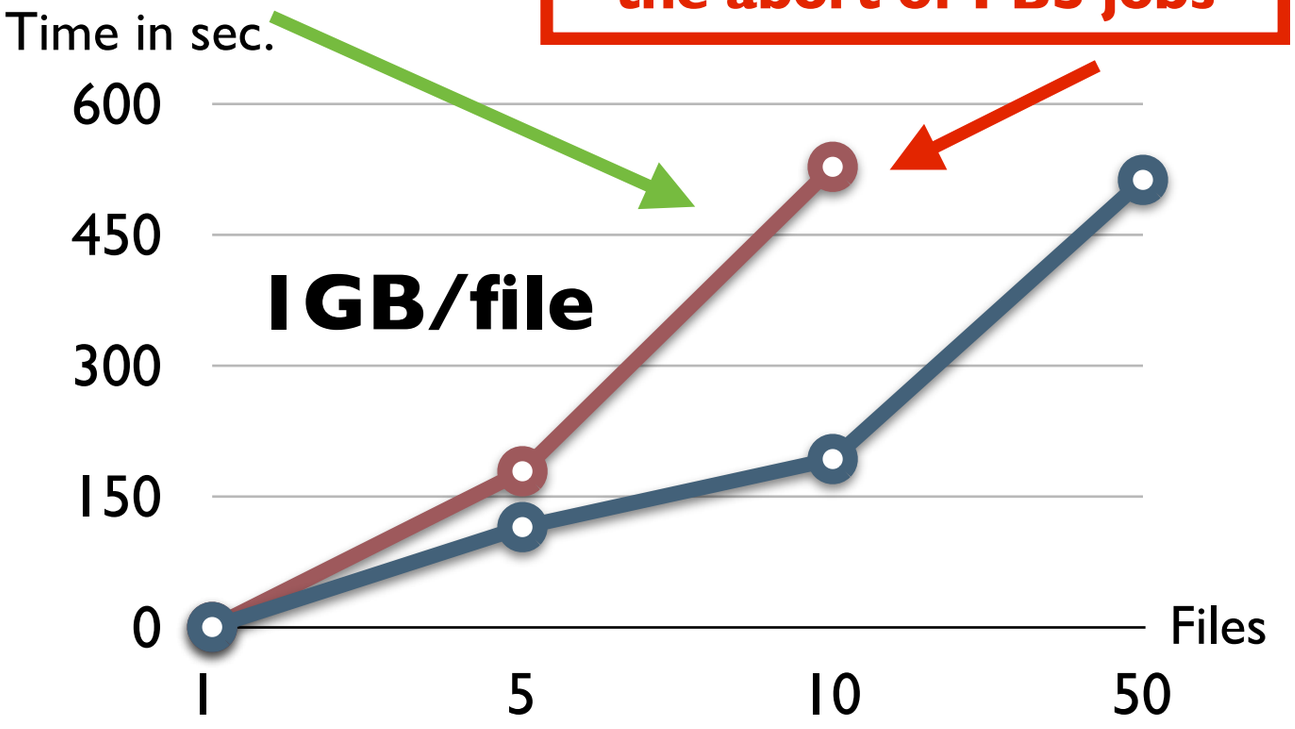
MapReduce
PBS

Only little overhead introduced by MapReduce



MapReduce scales better with large files

The high load on worker machines when processing 50 files cause the abort of PBS jobs



~ 95 % MapReduce jobs benefit from the data locality

Summary and future works

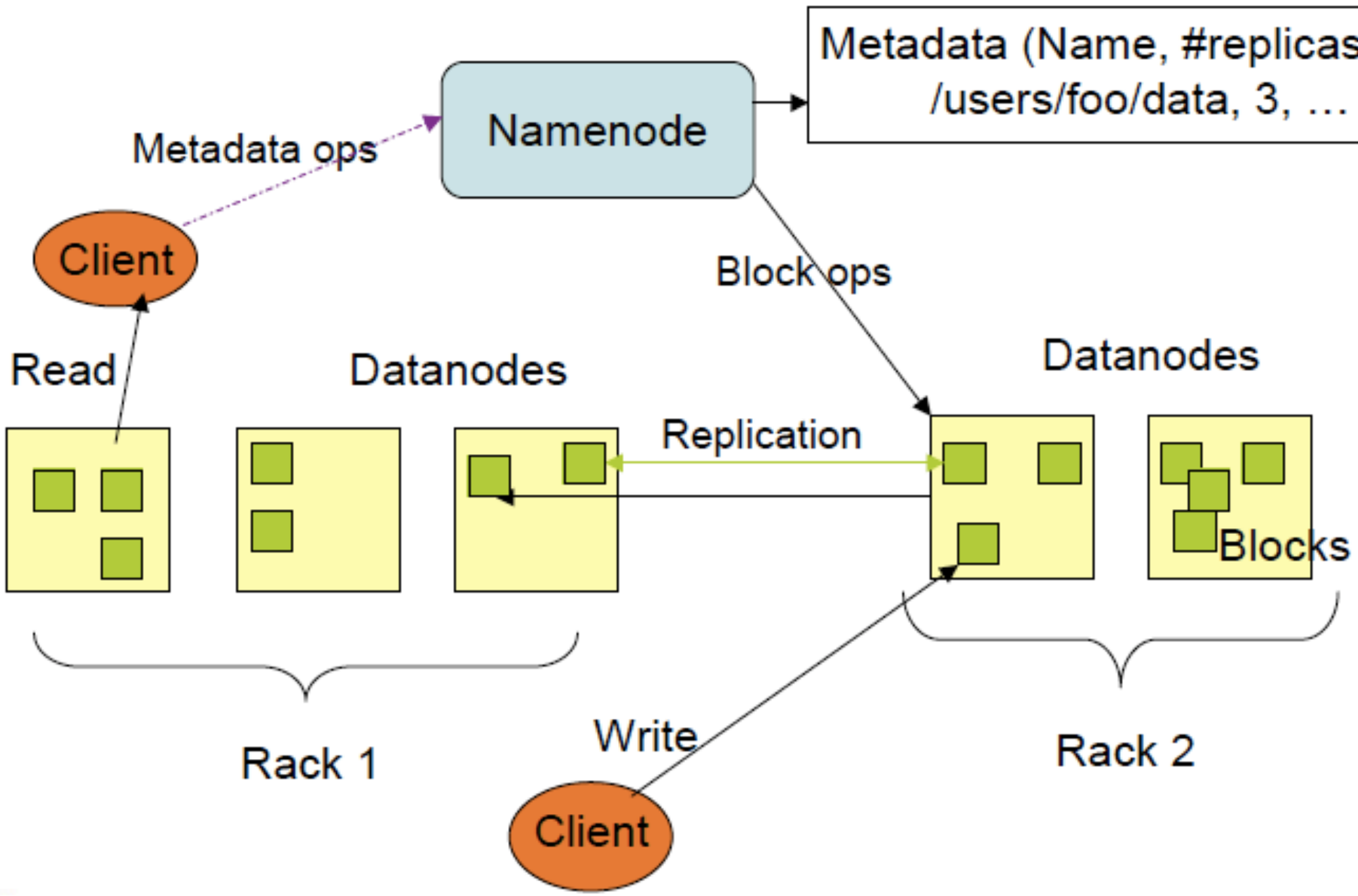
- ROOT files analysis workflow has been implemented with success using Hadoop
- The execution of ROOT files analysis using Hadoop has shown promising performance
- Extend the workflow to include a "reduce" in the analysis workflow (such as merging the ROOT output files)
- Perform scale tests of the analysis workflow in large Cluster
- Implement Plugins to allow the deployment of a Hadoop-based Grid Computing Element

BACKUP

HDFS deployment in HEP

- HDFS is currently used at some Tier-2 sites
- Configured as a Grid Storage Element
  - Interfaced with Globus GridFTP and BeStMan SRM
  - Accessed locally through Fuse to read files using ROOT
- It has shown satisfactory performance

Hadoop Storage: HDFS



INPUT

➡ Write a custom InputFormat to support the splitting and read of ROOT files

RootFileRecordReader.java ← "use" - - - - - **RootFileInputFormat.java**

Old MR API: BUG MAPREDUCE-1122

```
import org.apache.hadoop.conf.Configured;
//import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
//import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapred.InputSplit;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.TaskAttemptContext;
import org.apache.hadoop.mapred.FileSplit;
import java.io.IOException;
import org.apache.hadoop.io.IOUtils;

class RootFileRecordReader implements RecordReader<NullWritable, BytesWritable> {

private FileSplit fileSplit;
private Configuration conf;
private boolean processed = false;

public RootFileRecordReader(FileSplit fileSplit, Configuration conf) throws IOException {
...
public boolean next(NullWritable key, BytesWritable value) throws IOException {
        if (!processed) {
                // byte[] contents = new byte[(int) fileSplit.getLength()];
                byte[] contents = new byte[0];
                Path file = fileSplit.getPath();
                // FileSystem fs = file.getFileSystem(conf);
                // FSDataInputStream in = null;
                // try {
                        // in = fs.open(file);
                        // in.readFully(0, contents, 0, contents.length);
                        value.set(contents, 0, contents.length);
                        conf.set("map.input.file", conf.get("mapred.input.dir") + "/" + file.getName());
                // } finally {
                //      in.close();
                //}
                processed = true;
                return true;
                }
        return false;
}
...
```

Do not read anything as input

```
package org.apache.hadoop.streaming;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.mapred.InputFormat;
import org.apache.hadoop.mapred.InputSplit;
import org.apache.hadoop.mapred.JobContext;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.TaskAttemptContext;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileSplit;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.JobConf;
import java.io.IOException;
import org.apache.hadoop.fs.FileSystem;

public class RootFileInputFormat
        extends FileInputFormat<NullWritable, BytesWritable> {
        protected boolean isSplitable(FileSystem fs, Path filename) {
                return false;
        }
```

Whole file per split

OUTPUT

- HDFS does not support Random write
  - designed for write-once-read-many access model

➡ Write the output in /tmp of WN and then copy it to HDFS using libhdfs

➡ Write into HDFS only successfully produced ROOT files avoiding useless load on the FS

➡ Write 2 times the output file