University of Zurich[UZH]

High Performance Computing Lecture 10

Douglas Potter
Jozef Bucko
Noah Kubli

Today

Main Topic is MapReduce

Hadoop

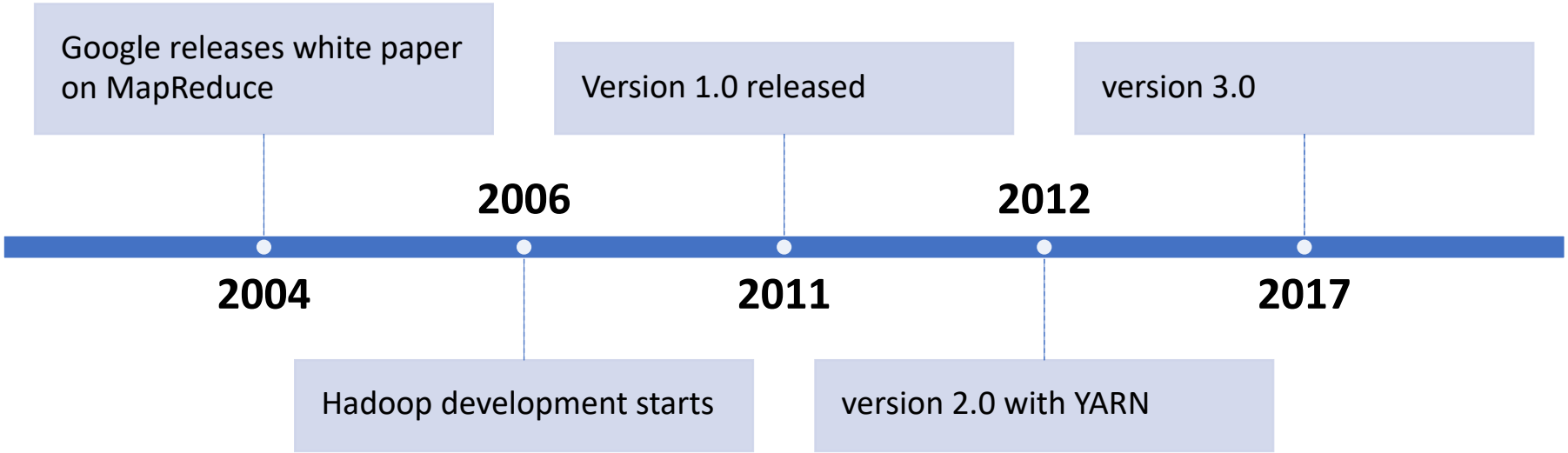
- Distributed File System
- Yarn

SSH Port Forwarding

WordCount Example

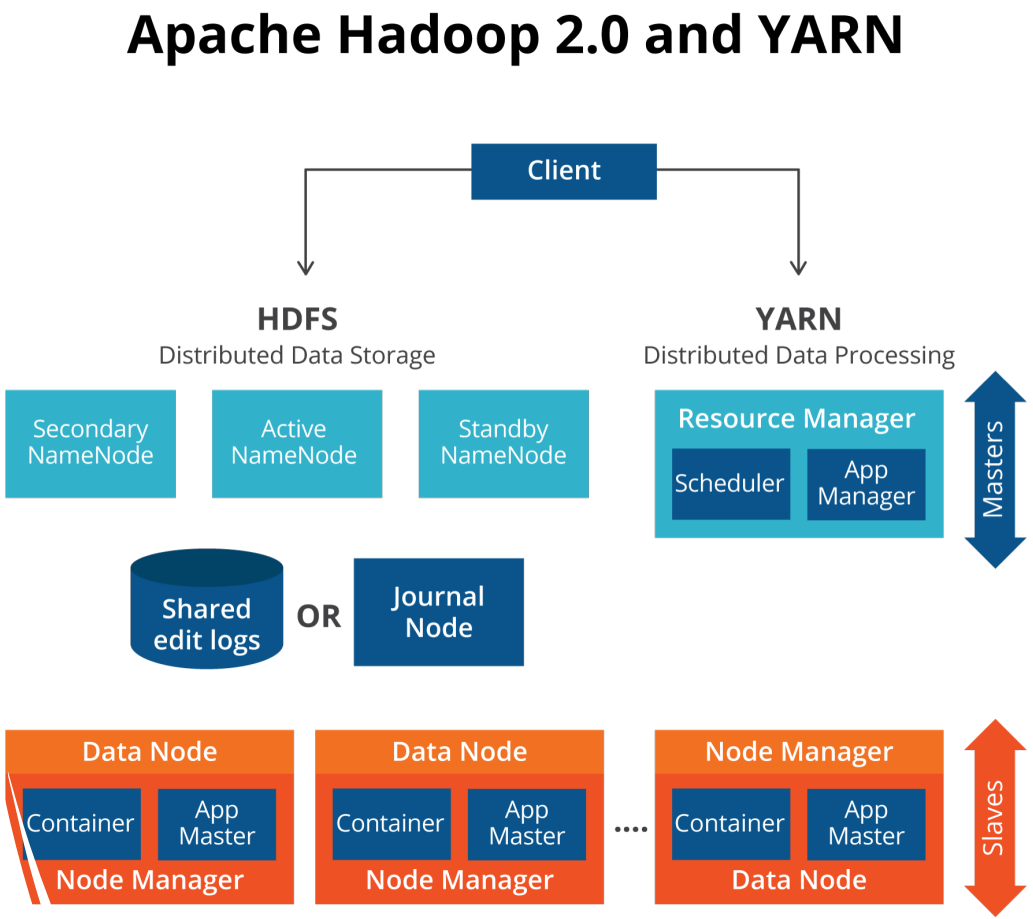
- Hadoop uses Java
- We will use Python

History

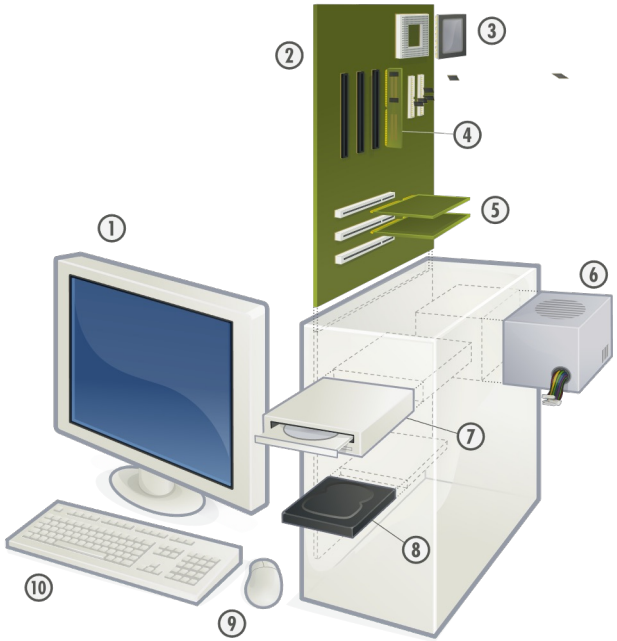

Google releases white paper on MapReduce
Version 1.0 released
version 3.0
2006
2012
2004
2011
2017
Hadoop development starts
version 2.0 with YARN

Hadoop Components

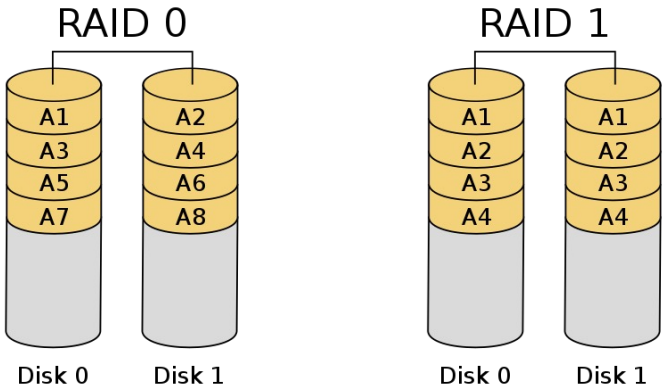
- Client (you)
- HDFS – storage for data
- NameNode – metadata
- YARN – resource scheduling
- Data Node
  - Where HDFS stores data
- Node Manager
  - Runs MapReduce jobs
  - Paired with a Data Node

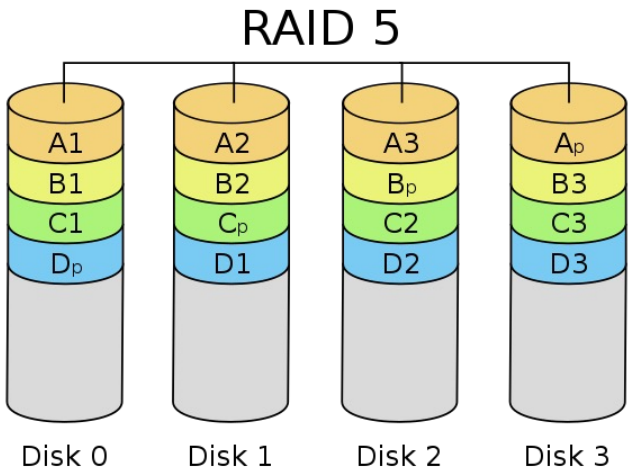


Standard File System (ext4,ntfs)



- What is used on your laptop
- Provides “files” (POSIX)
- “Metadata” is on the drive
- Several drives can be attached
- They can be “combined”
  - Looks like a single drive
  - Can provide data security (RAID)
- Share over the network
  - Network File System (NFS)
  - Samba (SMB/CIFS)
  - Apple Filing Protocol (AFP)





RAID Levels

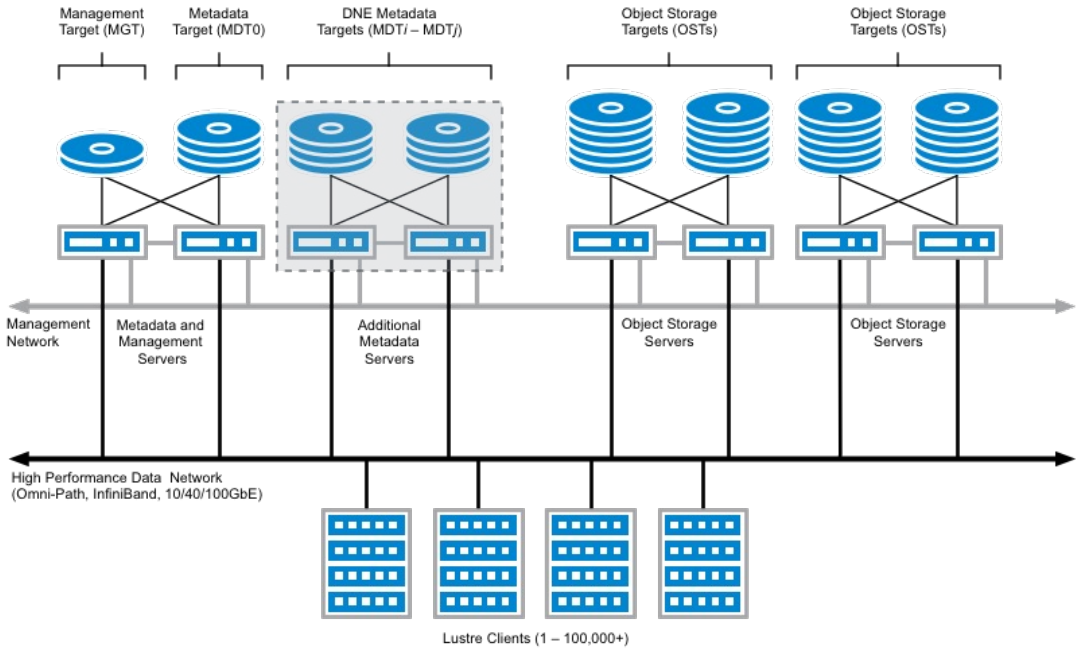
- Level 0 – striping
  - Maximizes the storage
  - High performance – read from both drives
  - "0" is the number of files you get back if there are disk errors
- Level 1 – mirror
  - Provides half of the storage
  - If one disk breaks, files are still present
- Level 5 – distributed partity
  - Mix of performance, reliability and cost
  - Provides 66% (or more) of the storage

File Servers

- Is a server with CPU, memory, …
- Has multiple drives
  - Usually RAID (on node redundancy)
  - Sometimes just JBOD (no redundancy on node)
- Optimized for file I/O
- Network performance is important

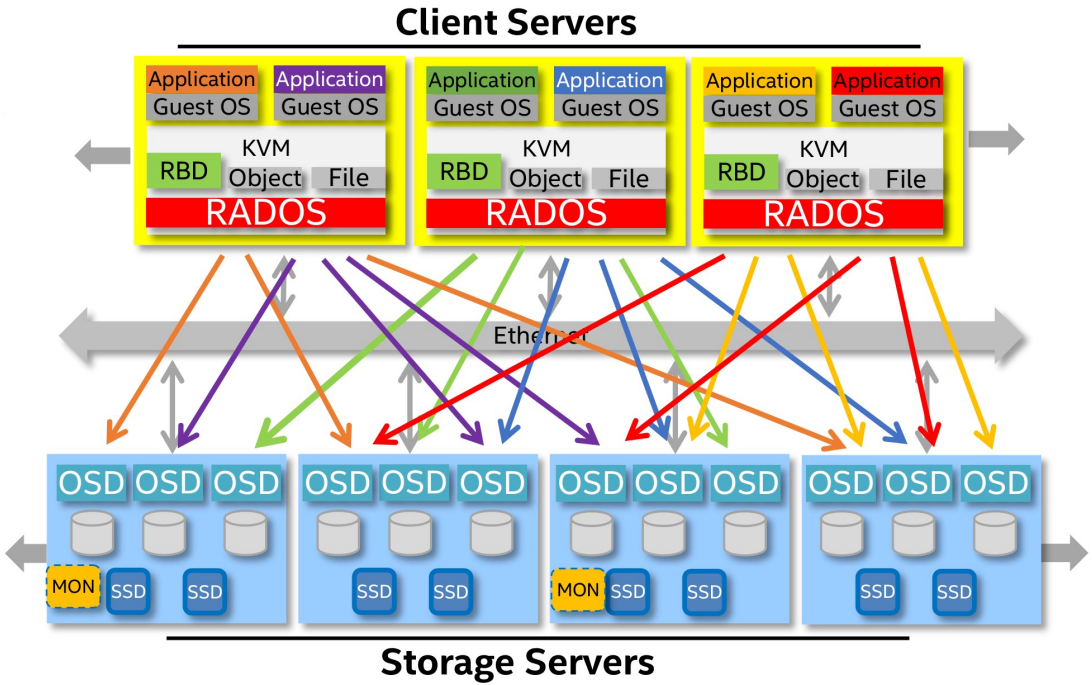
Parallel File System (Lustre)

- Provides “files” (POSIX)
- Many “file” servers
- Files are assigned to server(s)
- Single file access is above average (up to 10 times faster)
- Agregate performance scales as the network and number of file servers.
  - Normal HD: 0.2 GB/s
  - $SCRATCH: 138 GB/s



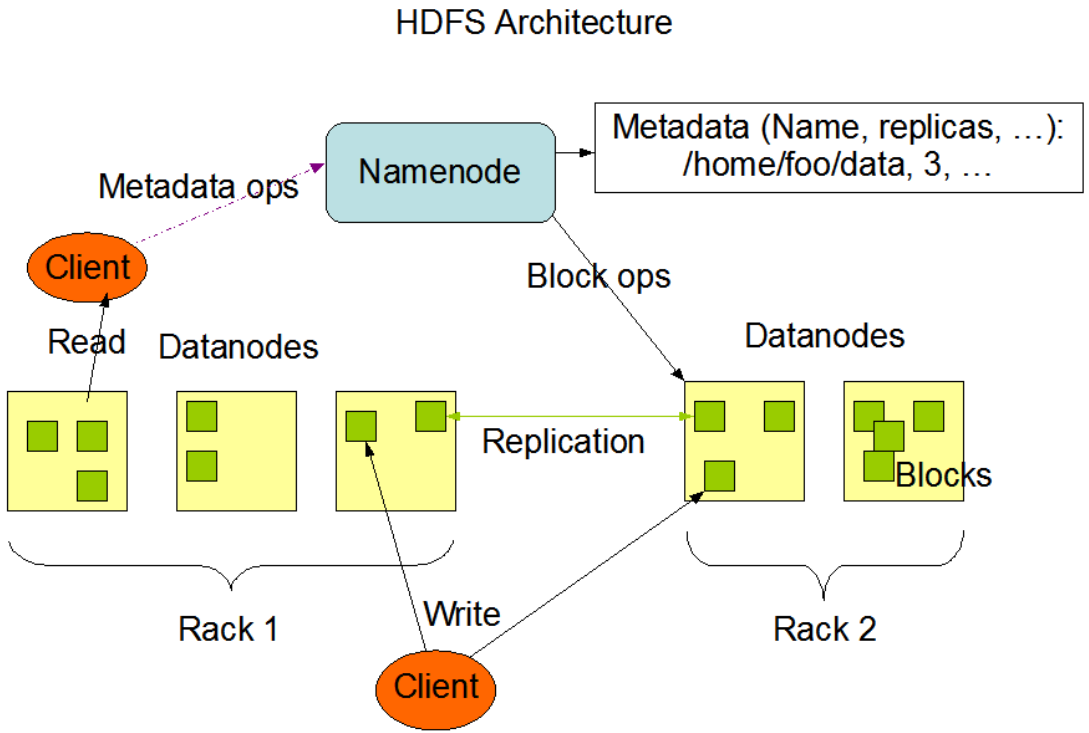
Object Storage (ceph)

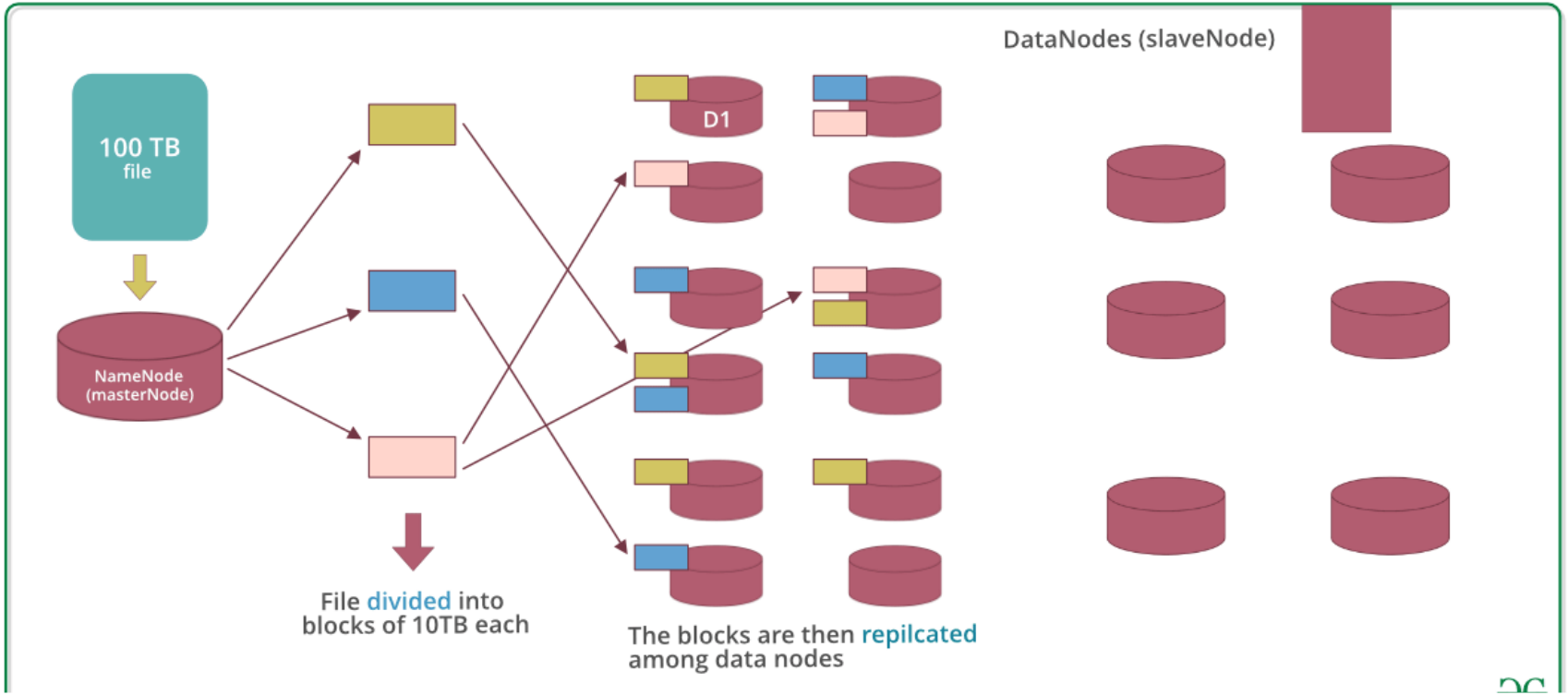
- Object (blob) storage
- Objects are duplicated
  - Exist on multiple servers
  - Typically REPLICA-3
- No redundancy on a server
- Metadata separate (SSD)
- SSD/HDD hierarchy on node



Distributed File System (hdfs)

- Designed for large files
  - Petabyte (1000 TB) range
- Streaming Data Access
  - Write once, read many
- Blocks are distributed
  - Individual files
  - Parts of large files
- Blocks are replicated
- Compute is sent to the data!

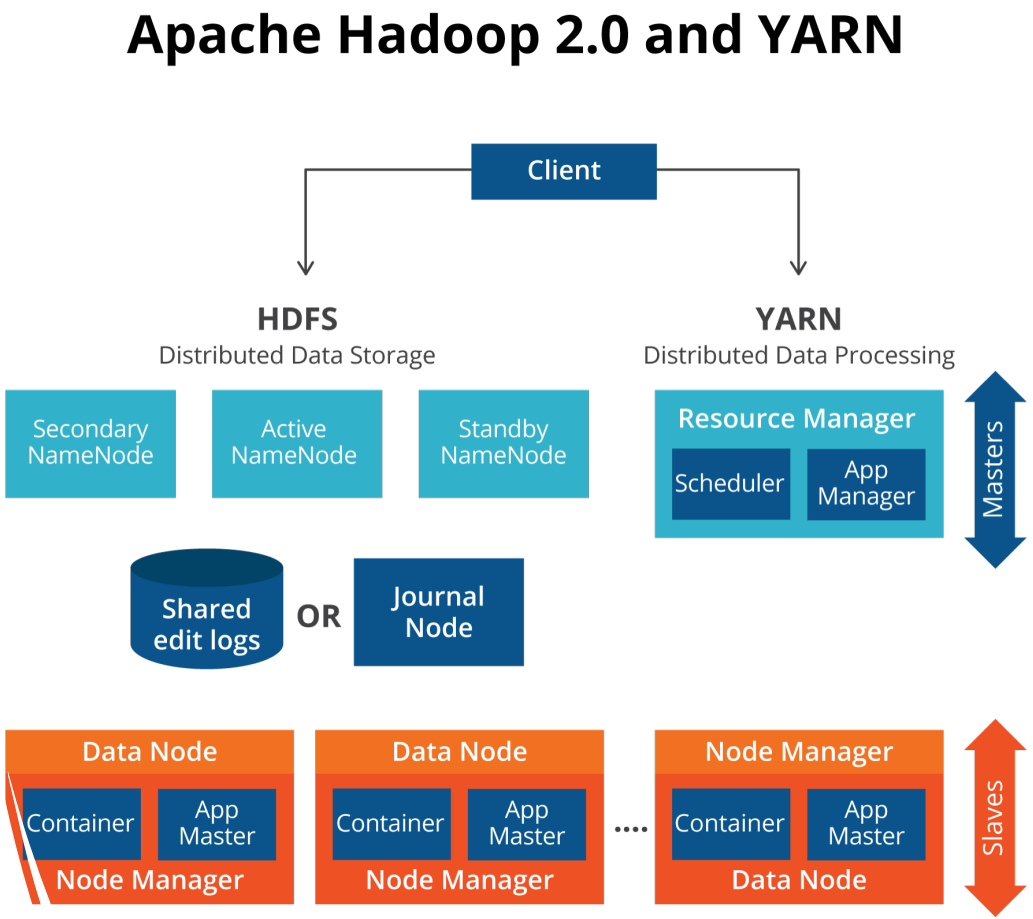



DataNodes (slaveNode)
100 TB
file
NameNode
(masterNode)
D1
File divided into
blocks of 10TB each
The blocks are then repilcated
among data nodes

Injection

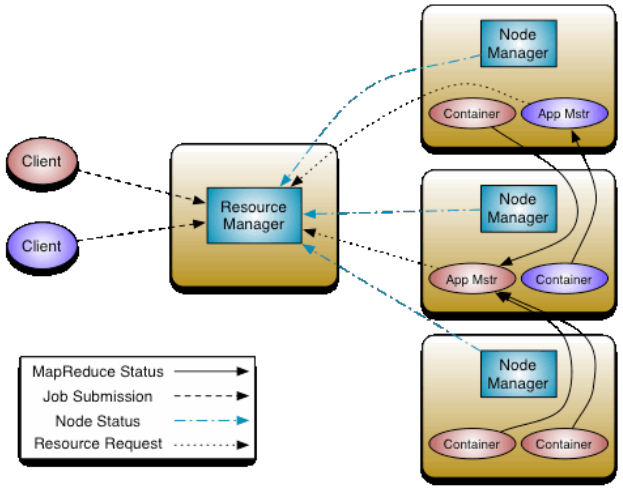
Hadoop Components

- Client (you)
- HDFS – storage for data
- NameNode – metadata
- YARN – resource scheduling
- Data Node
  - Where HDFS stores data
- Node Manager
  - Runs MapReduce jobs
  - Paired with a Data Node

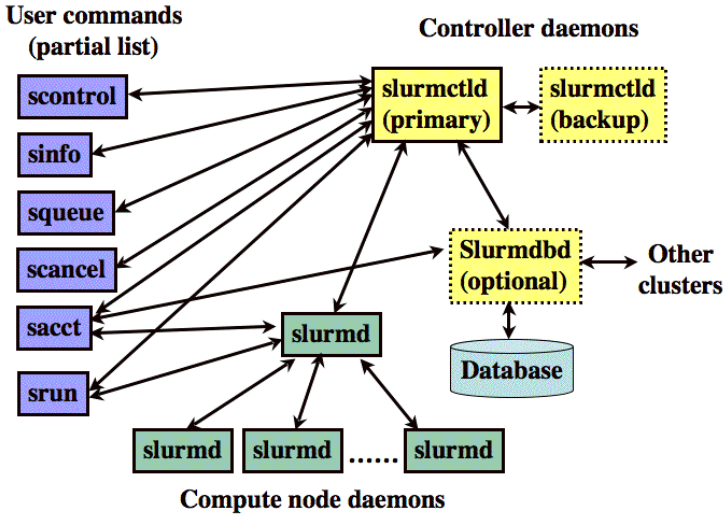


Yarn (Yet Another Resource Negotiator)

Hadoop

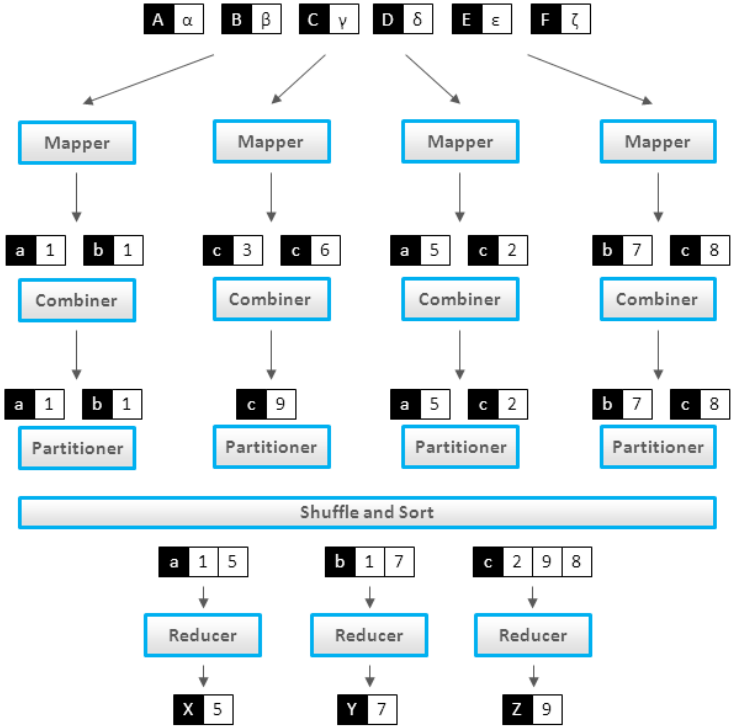


SLURM



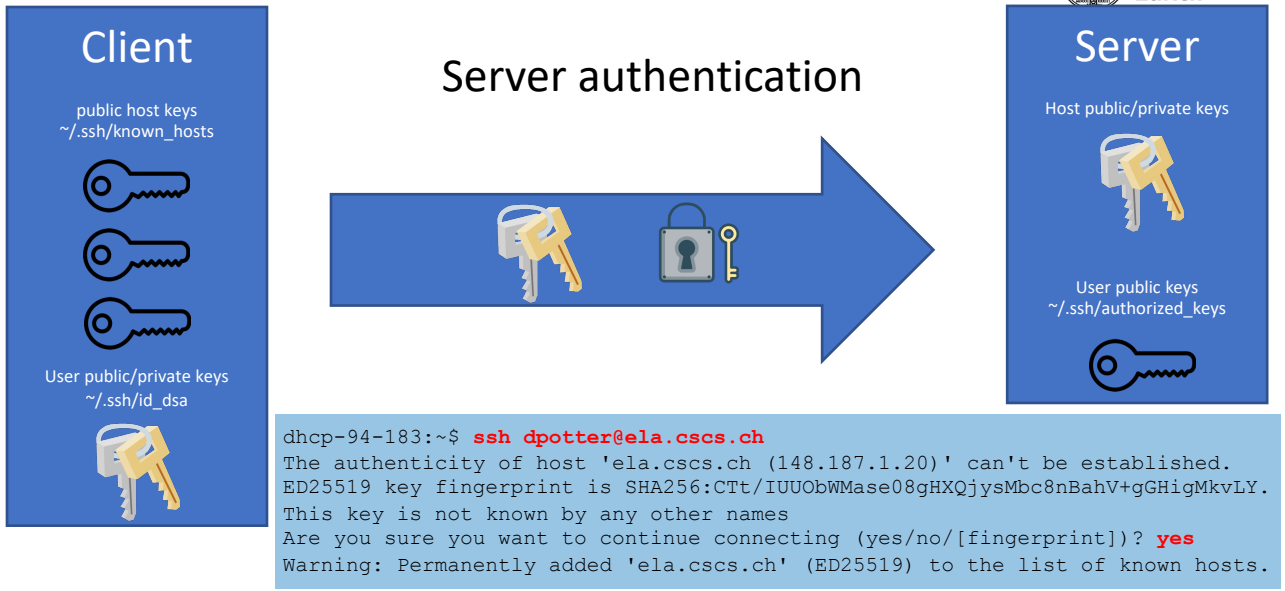
MapReduce

- Key and Value “pairs”
- Our focus is on
  - Mapper
  - Reducer
- Combiner may not run
- Partitioner separates keys into different partitions
- Shuffle moves partitions to the Reducers
- Reducers get 1+ keys

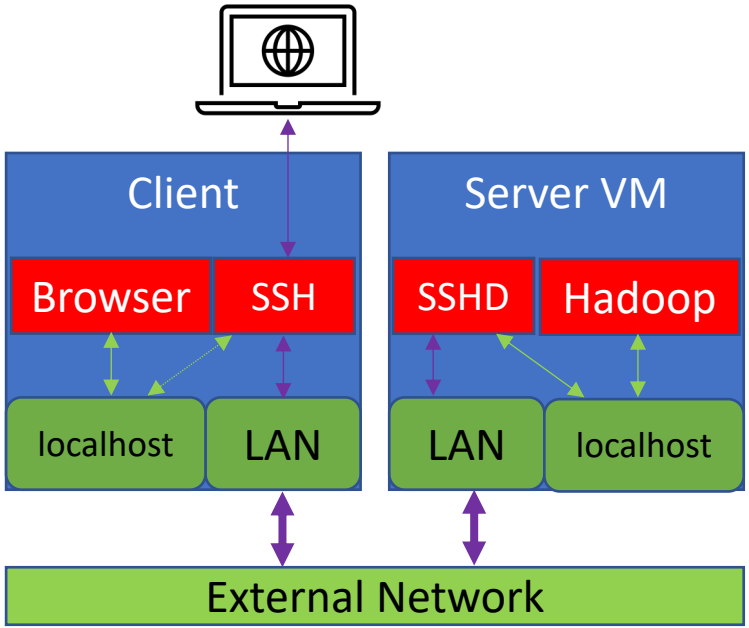


SSH Components

- Servers have public keys too!
- This way we can avoid man-in-the-middle attacks.
- Public host key is saved first time we connect to a host.
  - Thereafter it is verified.
- Session keys are exchanged for faster encryption & decryption.
- Traffic is “tunneled” through our connection.
  - Port forwarding for example.



SSH Port Forwarding



Client Port

Server address

Server Port

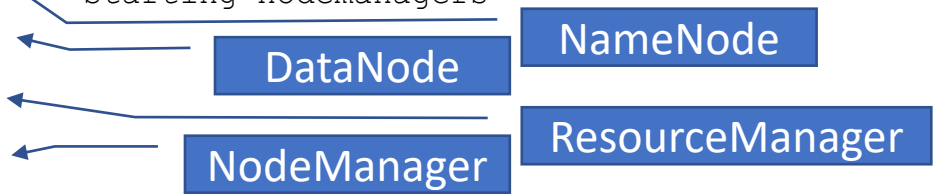
```
ssh\
  -L 9870:localhost:9870\
  -L 8088:localhost:8088\
  ubuntu@172.23.8.109
```

```
$ ping localhost
PING localhost (127.0.0.1)
56(84) bytes of data.
64 bytes from localhost
(127.0.0.1): icmp_seq=1
ttl=64 time=0.030 ms
```

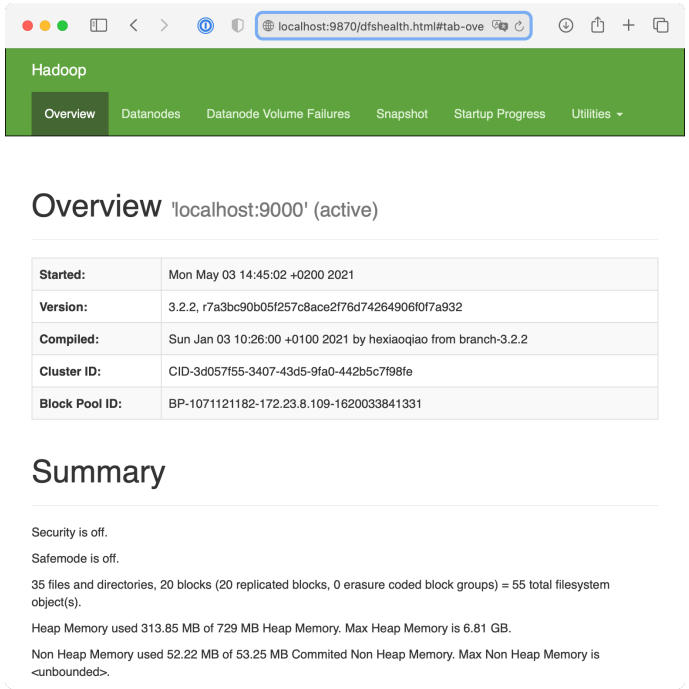
~/.ssh/config

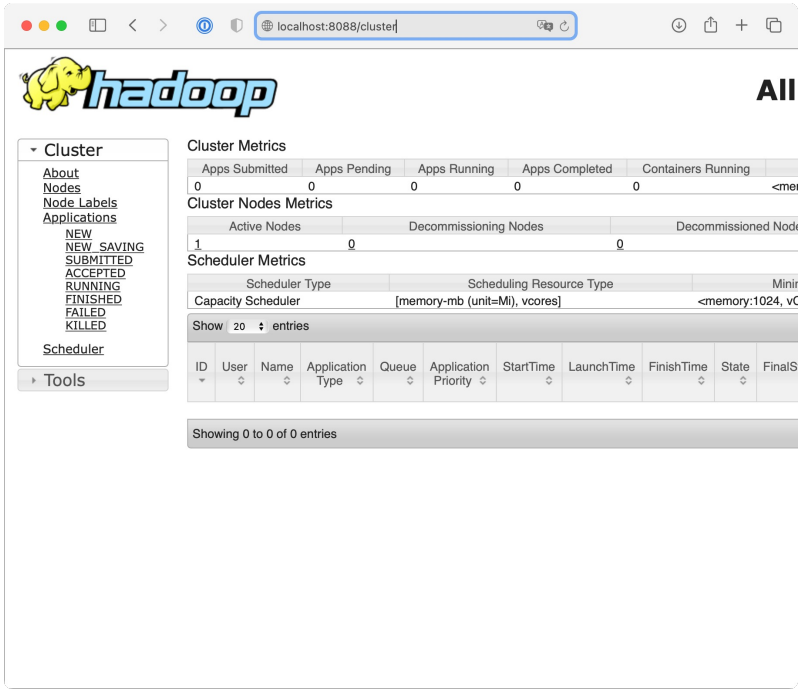
```
Host hadoop
  Hostname 172.23.8.109
  User ubuntu
  ForwardX11 yes
  ForwardX11Trusted yes
  ForwardAgent yes
  LocalForward 9870 localhost:9870
  LocalForward 9864 localhost:9864
  LocalForward 8088 localhost:8088
  LocalForward 8042 localhost:8042
```

```
laptop:~$ ssh hadoop
hadoop:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [hadoop]
hadoop:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
```



Overview in your browser





Test Files (sample)

Interim
Lola Ridge, 1873

The earth is motionless
And poised in space ...
A great bird resting in its flight
Between the alleys of the stars.
It is the wind's hour off ...
The wind has nestled down among the corn ...
The two speak privately together,
Awaiting the whirr of wings.

The Destroyer
Lola Ridge 1873 - 1941

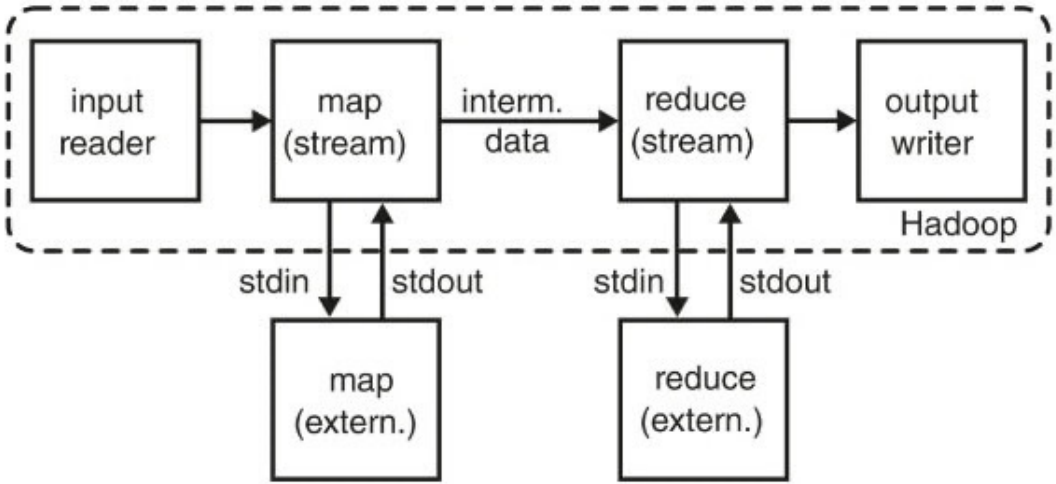
I am of the wind ...
A wisp of the battering wind ...

I trail my fingers along the Alps
And an avalanche falls in my wake ...
I feel in my quivering length
When it buries the hamlet beneath ...

I hurriedly sweep aside
The cities that clutter our path ...
As we whirl about the circle of the globe ...
As we tear at the pillars of the world ...
Open to the wind,
The Destroyer!
The wind that is battering at your gates.

Hadoop Streaming

- Hadoop is Java
- Mapper and Reducer can be any language by “streaming”.
- Streaming:
  - Read the data from stdin
  - Write results to stdout
- We will use Python



Java example of “WordCount”

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

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);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(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(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

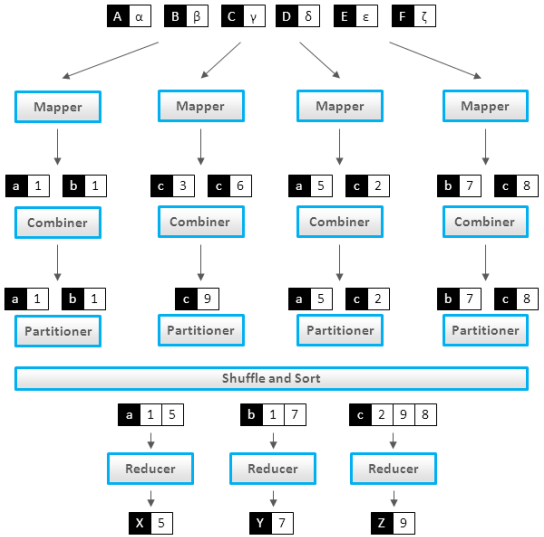
The Mapper

```
#!/usr/bin/env python3
"""mapper.py"""

import sys
import string

trans=str.maketrans('','',string.punctuation)

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace and punctuation
    line = line.strip().translate(trans).lower()

    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print(f'{word}\t1')
```

- Read each line from input
- Remove leading/trailing spaces
- Remove punctuation
- Convert to lowercase
- Split into individual words
- Output words with count of 1

The Mapper in action

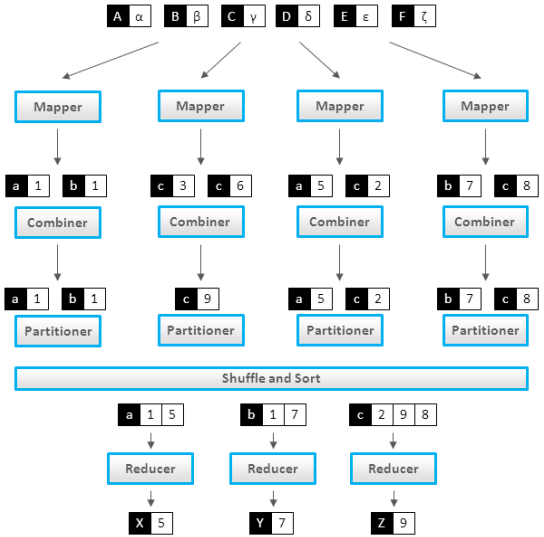
```
$ cat File0.txt
Interim
Lola Ridge, 1873

The earth is motionless
And poised in space ...
A great bird resting in its flight
Between the alleys of the stars.
It is the wind's hour off ...
The wind has nestled down among the corn ...
The two speak privately together,
Awaiting the whirr of wings.

$ cat File0.txt | ./mapper.py
interim         1
lola            1
ridge           1
1873            1
the             1
earth           1
is              1
motionless      1
and             1
poised          1
in              1
space           1
a               1
great           1
bird            1
resting         1
in              1
its             1
flight          1
between         1
the             1
alleys          1
of              1
the             1
stars           1
it              1
is              1
the             1
winds           1
hour            1
off             1
the             1
wind            1
has             1
nestled         1
down            1
among           1
the             1
corn            1
the             1
two             1
speak           1
privately       1
together        1
awaiting        1
the             1
whirr           1
of              1
wings           1
```



The Reducer

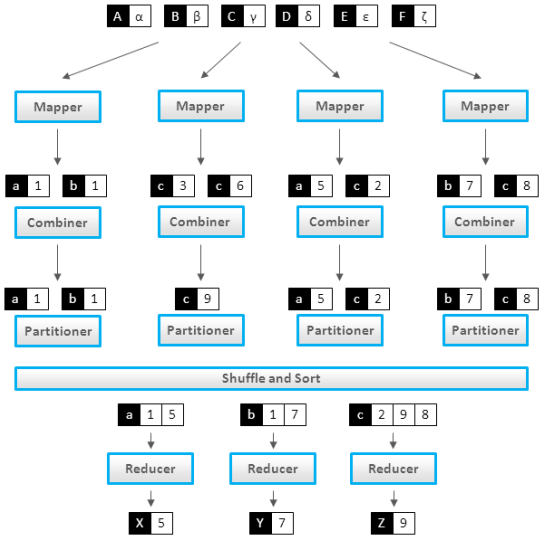
```
#!/usr/bin/env python3
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

for line in sys.stdin: # input comes from STDIN
    line = line.strip() # remove leading and trailing whitespace
    word, count = line.split('\t', 1) # parse the input we got from mapper.py

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        continue # count was not a number, silently ignore/discard this line

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word: # write result to STDOUT
            print(f'{current_word}\t{current_count}')
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print(f'{current_word}\t{current_count}')
```

- Input has been sorted by key
- Read each word and count
- For repeated words sum counts



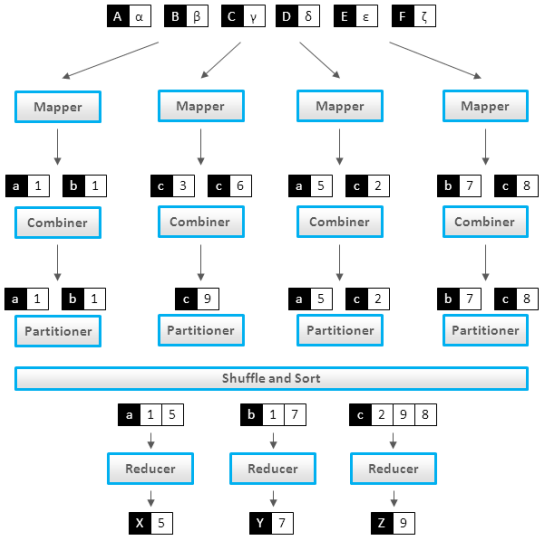
The Reducer expects sorted keys

```
$ cat File0.txt | ./mapper.py | sort
1873        1
a           1
alleys      1
among       1
and         1
awaiting    1
between     1
bird        1
corn        1
down        1
earth       1
flight      1
great       1
has         1
hour        1
in          1
in          1
interim     1
is          1
is          1
it          1
its         1
lola        1
motionless  1
nestled     1
of          1
of          1
off         1
poised      1
privately   1
resting     1
ridge       1
space       1
speak       1
stars       1
the         1
the         1
the         1
the         1
the         1
the         1
the         1
the         1
the         1
together    1
two         1
whirr       1
wind        1
winds       1
wings       1
```



The Reducer in action

```
$ cat File0.txt | ./mapper.py | sort | ./reducer.py
1873        1
a           1
alleys      1
among       1
and         1
awaiting    1
between     1
bird        1
corn        1
down        1
earth       1
flight      1
great       1
has         1
hour        1
in          2
interim     1
is          2
it          1
its         1
lola        1
motionless  1
nestled     1
of          2
off         1
poised      1
privately   1
resting     1
ridge       1
space       1
speak       1
stars       1
the         8
together    1
two         1
whirr       1
wind        1
winds       1
wings       1
```



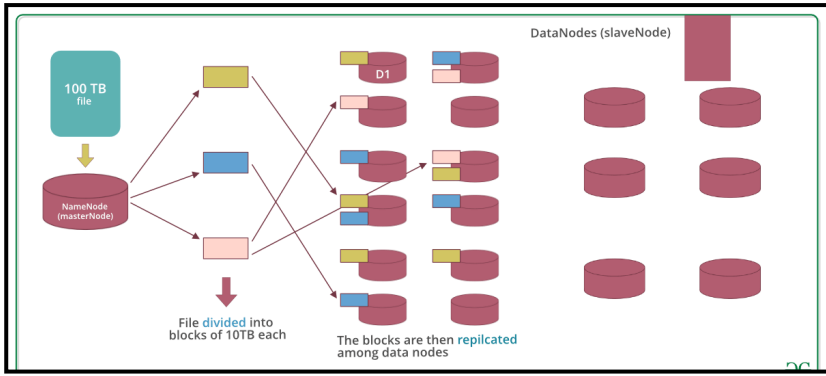
What we expect for results (all files)

```
$ ( for F in File?.txt ; do cat $F | ./mapper.py ; done ) | sort | ./reducer.py
1873	4
1941	3
a	7
about	1
again	1
alleys	1
along	1
alps	1
altitude	1
am	1
among	1
an	1
and	12
anger	1
are	1
as	5
ascends	1
aside	1
at	2
avalanche	1
awaiting	1
battering	2
be	1
beneath	1
between	1
bind	1
bird	1
buries	1
burn	3
but	2
catch	1
circle	1
cities	1
cleanly	1
clutter	1
corn	1
darkness	1
destroyer	2
down	1
droning	1
dust	1
earth	1
ever	2
falls	1
feel	1
fingers	1
fire	3
flames	1
flight	1
for	1
gates	1
globe	1
great	2
green	2
hamlet	1
hand	1
hands	2
has	2
have	1
here	1
him	3
hour	1
how	1
hunger	1
hurriedly	1
i	5
if	1
in	6
interim	1
into	1
is	4
it	3
its	2
length	1
like	1
little	2
lola	4
low	1
made	1
men	5
mens	2
motionless	1
my	3
nestled	1
newwashed	1
oak	1
of	12
off	2
old	3
one	2
open	1
our	1
over	1
pain	1
path	1
pillared	1
pillars	1
poised	1
poke	1
privately	1
quivering	1
remote	1
resting	1
ridge	4
rises	1
round	2
rub	1
run	1
shaken	1
sheer	1
silence	2
sinking	1
sit	1
sky	1
slow	1
slow…	1
smoke	2
space	1
speak	1
spreads	1
stars	1
sweep	1
tear	1
that	3
the	41
their	2
thereof	1
they	2
things	1
throw	1
to	3
together	1
torment	2
touches	1
trail	1
trembling	1
trunk	1
two	1
up	1
valleys	1
wake	1
warm	1
way	1
we	2
when	1
where	1
whirl	1
whirr	1
wind	6
winds	1
wings	1
wisp	1
with	2
withes	3
wonder	1
world	3
would	1
wrath	1
you	1
young	4
your	1
```

Login and remember to start the services

```
laptop:~$ ssh hadoop
hadoop:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [hadoop]
hadoop:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
```

Upload the test files to Hadoop FS




```
$ wc File?.txt
  12   52  281 File0.txt
  21  125  620 File1.txt
  18   93  460 File2.txt
  14   56  316 File3.txt
  65  326 1677 total
$ hadoop fs -ls
$ hadoop fs -mkdir LolaRidge
$ hadoop fs -ls
Found 1 items
drwxr-xr-x   - ubuntu supergroup          0 2021-05-04 07:48 LolaRidge
ubuntu@hadoop:~$ hadoop fs -put File?.txt LolaRidge/
ubuntu@hadoop:~$ hadoop fs -ls LolaRidge/
Found 4 items
-rw-r--r--   1 ubuntu supergroup        281 2021-05-04 07:50 LolaRidge/File0.txt
-rw-r--r--   1 ubuntu supergroup        620 2021-05-04 07:50 LolaRidge/File1.txt
-rw-r--r--   1 ubuntu supergroup        460 2021-05-04 07:50 LolaRidge/File2.txt
-rw-r--r--   1 ubuntu supergroup        316 2021-05-04 07:50 LolaRidge/File3.txt
```

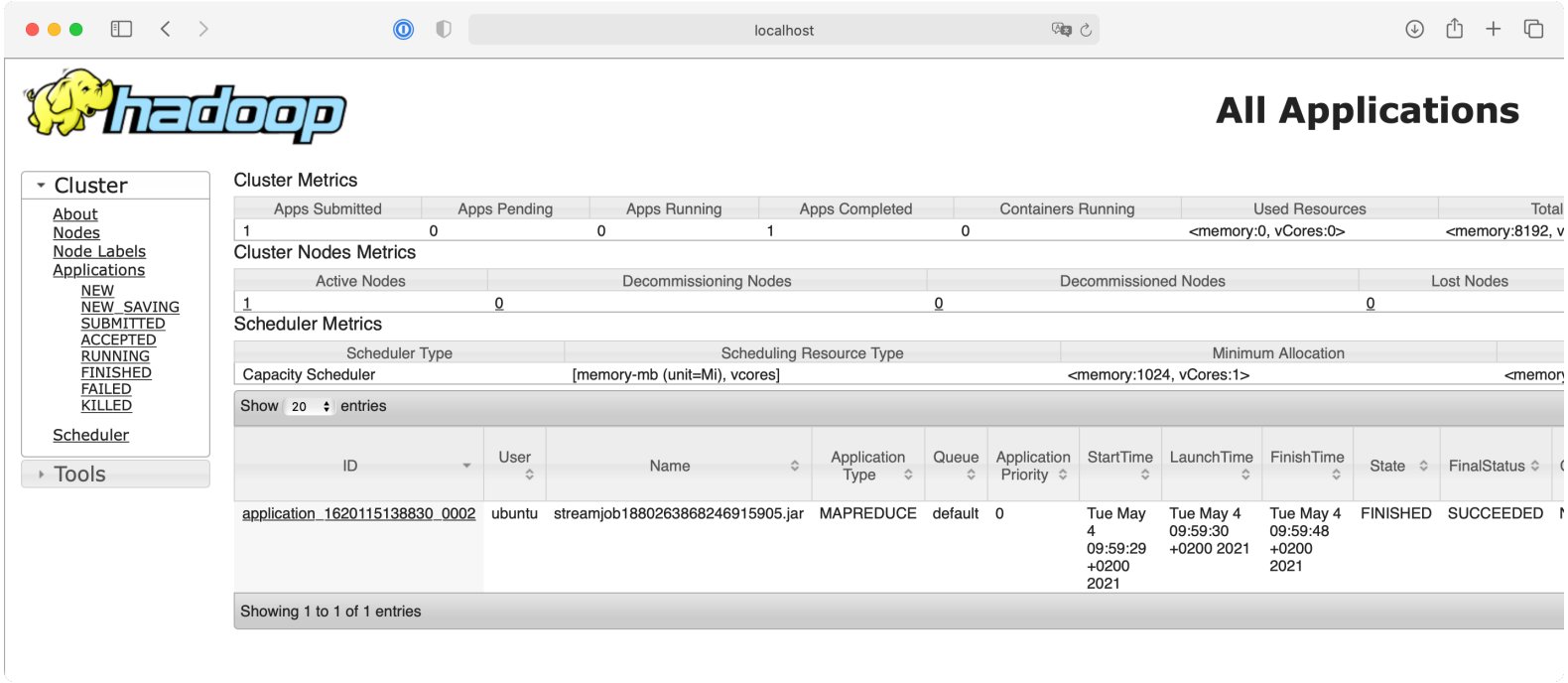
Hadoop Streaming

```
hadoop:~$ cat pymapred.sh
#!/bin/bash
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.2.2.jar\
 -files mapper.py,reducer.py\
 -mapper mapper.py -reducer reducer.py\
 -input LolaRidge/* -output output
```

Running the job

```
ubuntu@hadoop:~$ ./pymapred.sh
packageJarDir: [/tmp/hadoop-unjar5183485775643681398/] [] /tmp/streamjob1880263868246915905.jar tmpDir=null
2021-05-04 07:59:26,563 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
2021-05-04 07:59:26,825 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
2021-05-04 07:59:27,163 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/ubuntu/.staging/job_1620115138830_0002
2021-05-04 07:59:28,242 INFO mapred.FileInputFormat: Total input files to process : 4
2021-05-04 07:59:28,753 INFO mapreduce.JobSubmitter: number of splits:4
2021-05-04 07:59:28,924 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1620115138830_0002
2021-05-04 07:59:28,926 INFO mapreduce.JobSubmitter: Executing with tokens: []
2021-05-04 07:59:29,143 INFO conf.Configuration: resource-types.xml not found
2021-05-04 07:59:29,143 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2021-05-04 07:59:29,505 INFO impl.YarnClientImpl: Submitted application application_1620115138830_0002
2021-05-04 07:59:29,583 INFO mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1620115138830_0002/
2021-05-04 07:59:29,585 INFO mapreduce.Job: Running job: job_1620115138830_0002
2021-05-04 07:59:38,784 INFO mapreduce.Job: Job job_1620115138830_0002 running in uber mode : false
2021-05-04 07:59:38,786 INFO mapreduce.Job:  map 0% reduce 0%
2021-05-04 07:59:44,887 INFO mapreduce.Job:  map 100% reduce 0%
2021-05-04 07:59:48,928 INFO mapreduce.Job:  map 100% reduce 100%
2021-05-04 07:59:49,946 INFO mapreduce.Job: Job job_1620115138830_0002 completed successfully
2021-05-04 07:59:50,063 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=2810
                FILE: Number of bytes written=1198503
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=2097
                HDFS: Number of bytes written=1264
                HDFS: Number of read operations=17
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=4
                Launched reduce tasks=1
                Data-local map tasks=4
                Total time spent by all maps in occupied slots (ms)=14613
                Total time spent by all reduces in occupied slots (ms)=2394
                Total time spent by all map tasks (ms)=14613
                Total time spent by all reduce tasks (ms)=2394
                Total vcore-milliseconds taken by all map tasks=14613
                Total vcore-milliseconds taken by all reduce tasks=2394
                Total megabyte-milliseconds taken by all map tasks=14963712
                Total megabyte-milliseconds taken by all reduce tasks=2451456
        Map-Reduce Framework
                Map input records=65
                Map output records=310
                Map output bytes=2184
                Map output materialized bytes=2828
                Input split bytes=420
                Combine input records=0
                Combine output records=0
                Reduce input groups=161
                Reduce shuffle bytes=2828
                Reduce input records=310
                Reduce output records=161
                Spilled Records=620
                Shuffled Maps =4
                Failed Shuffles=0
                Merged Map outputs=4
                GC time elapsed (ms)=770
                CPU time spent (ms)=3360
                Physical memory (bytes) snapshot=1602109440
                Virtual memory (bytes) snapshot=12895678464
                Total committed heap usage (bytes)=2181562368
                Peak Map Physical memory (bytes)=339447808
                Peak Map Virtual memory (bytes)=2579976192
                Peak Reduce Physical memory (bytes)=260395008
                Peak Reduce Virtual memory (bytes)=2583035904
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=1677
        File Output Format Counters
                Bytes Written=1264
2021-05-04 07:59:50,063 INFO streaming.StreamJob: Output directory: output
```

Results at http://localhost:8088



Output file

```
ubuntu@hadoop:~$ hadoop fs -ls
Found 2 items
drwxr-xr-x   - ubuntu supergroup          0 2021-05-04 07:50 LolaRidge
drwxr-xr-x   - ubuntu supergroup          0 2021-05-04 07:59 output
ubuntu@hadoop:~$ hadoop fs -ls output
Found 2 items
-rw-r--r--   1 ubuntu supergroup          0 2021-05-04 07:59 output/_SUCCESS
-rw-r--r--   1 ubuntu supergroup       1264 2021-05-04 07:59 output/part-00000
ubuntu@hadoop:~$ hadoop fs -cat output/part-00000
1873        4
1941        3
a           7
about       1
again       1
alleys      1
along       1
alps        1
altitude    1
am          1
among       1
an          1
and         12
anger       1
are         1
as          5
ascends     1
aside       1
at          2
avalanche   1
awaiting    1
battering   2
be          1
beneath     1
between     1
bind        1
bird        1
buries      1
…
```

Comparison to running directly

```
hadoop:~$ ( for F in File?.txt ; do cat $F | ./mapper.py ; done ) | sort | ./reducer.py >DIRECT.txt
hadoop:~$ hadoop fs -get output/part-00000
hadoop:~$ wc DIRECT.txt part-00000
 161  322 1264 DIRECT.txt
 161  322 1264 part-00000
 322  644 2528 total
hadoop:~$ md5sum DIRECT.txt part-00000
4f3e933ce5a9fba42b3c7ae380225b16  DIRECT.txt
4f3e933ce5a9fba42b3c7ae380225b16  part-00000
```