

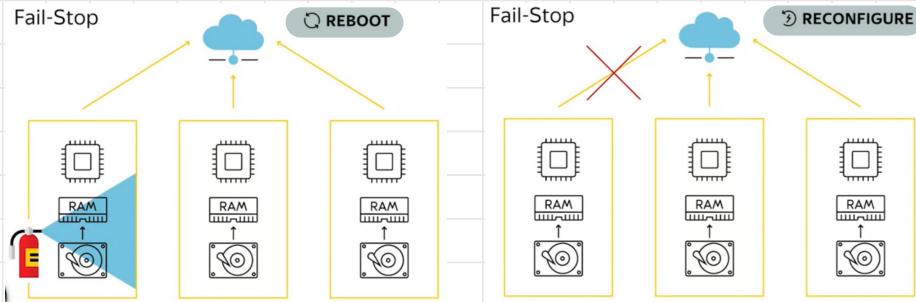
Hadoop MapReduce:

How to build reliable system from unreliable components.

- Distributed systems are often built from **unreliable components**, cluster nodes can break any time because of power supply, disk damages, overheated CPUs, and so on.
- Types of Node Failure.

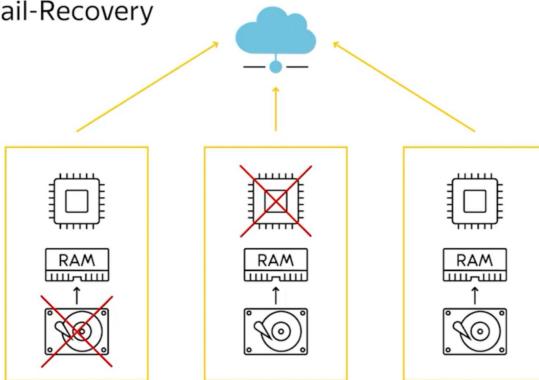
1. Fail-Stop

- means that if machines get out of service **during a computation**, then you have to have an external impact to bring system back to a working state.
- Reboot**: fix the node and reboot the whole system or part of.
- Reconfigure**: retire the broken machine and reconfigure the distributed system, **not robust to node crashes**.



2. Fail Recovery.

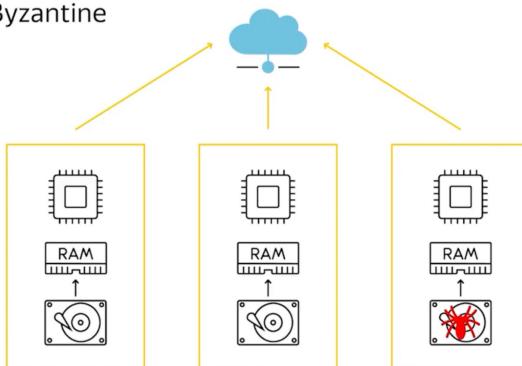
Fail-Recovery



- during computations, nodes can arbitrary crash and return back to servers.
- doesn't influence correctness and success of computation.
no external impact necessary to reconfiguring the system.
- If hard drive was damaged, system administrator can physically change the hard drive. After reconnection, this node will be automatically picked up by a distributed system.

3. Byzantine failure.

Byzantine

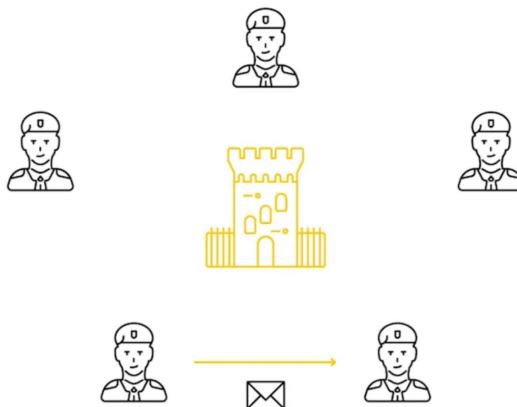


- A distributed system is robust Byzantine failure if it can work

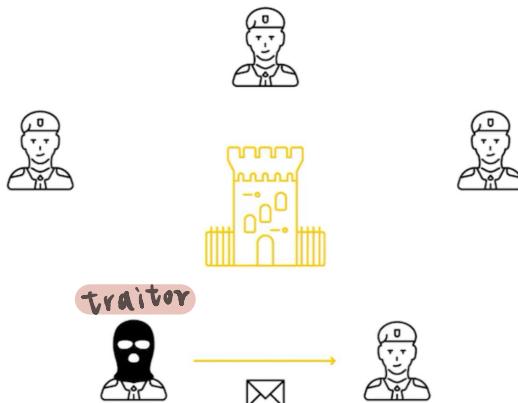
despite some of nodes behaving out of protocol.

- **Byzantine Generals' Problem**

- group of generals of the Byzantine army camped with their troops around an enemy city, communicating only by a messenger, the generals must agree upon a common battle plan.



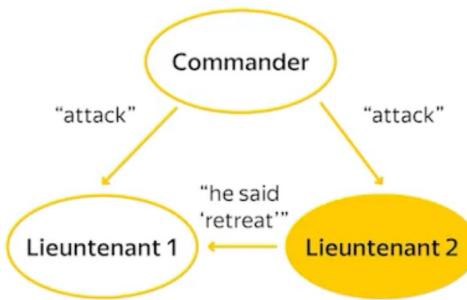
However, one or more of them may be a **traitor** who trying to confuse the others.



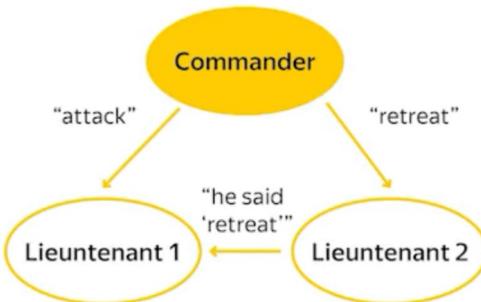
It is shown that using only unsigned messages, you have to have more than $\frac{2}{3}$ of loyal generals.

In the case of one commanding general, two lieutenants:

1. If second lieutenant is traitor.

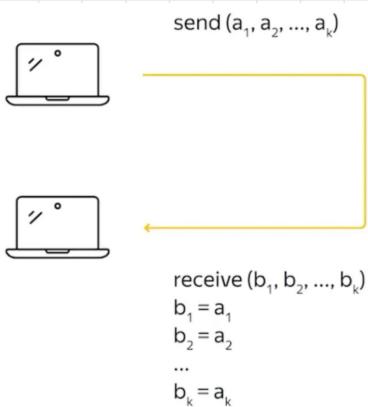


2. If Commander is traitor.



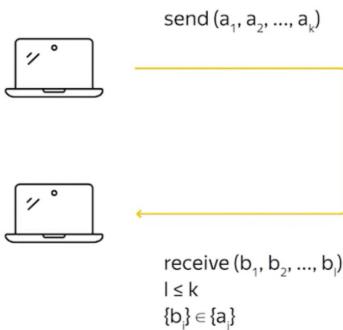
- Types of Link Failures

1. Perfect Link :

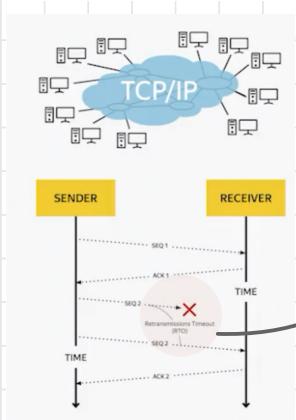


all the sent messages must be delivered and received without any modification in the same order.

2. Fair-Loss Link:

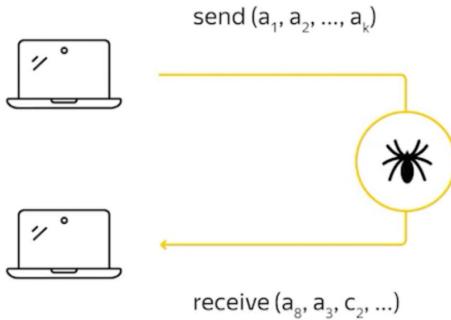


Some part of the messages can be lost, but the probability of message loss doesn't depend on contents of a message.



Packet loss is a very common problem for network connection. For instance, the well-known TCP/IP protocol tries to solve this problem by re-transmitting messages if they were not received

3. Byzantine Link



Some messages can be filtered according to some rule, and some messages could be created out of nowhere. Nothing comes out of nowhere according to the law of conservation of energy.

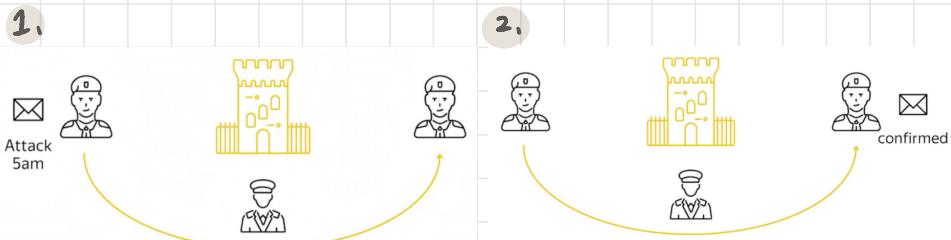
- The Two Generals Paradox.

- started in 1975 and then popularized by Jim Gray in 1998.
- the first computer communication problem to be proved **unsolvable**
- There are two generals on a campaign. They have an objective, if they simultaneously march the objective, they win. They can communicate only via messages.

Unfortunately, the world is occupied by the city's defenders and there is a chance that any given method sent through the wall will be captured. The problem is to find a protocol that allows the generals to march together even. There is a simple proof that **no fixed plans protocol exists**.

1. The first general send a message "attack at 5 a.m." how does he know that the second general will receive?
2. The second general should send an acknowledgement message confirmed. But, how does he know that the first general will receive the acknowledgement message?

⇒ **continue infinitely!!**



- Types of clock synchronization problem.

1. **clock skew**: time can be different on different machines.

2. **clock drift**: different clock rate.

- Any clock synchronization mechanism is subject to some precision,

⇒ **logical clock** were invented.

- help to track happened before events, and therefore, **order events** to build reliable protocols.

- logical clocks were named after his inventor, Leslie Lamport : Lamport logical clocks.

- **Synchronous System**.

- every messages between nodes is delivered within limited time.

- clock drift is limited

- each instruction execution is also limited.

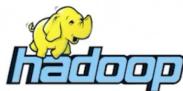
- Different Distributed Systems.

1. Fail-Stop + Perfect Link + Synchronous

- usually referred as a **parallel computation model** and widely adopted by **supercomputers** where many process connected by a local high speed computer bus.

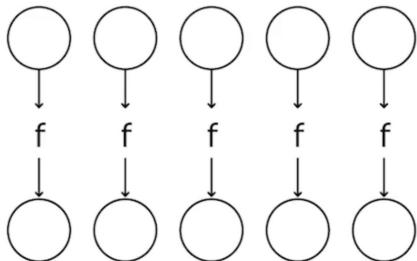


2. Fail Recovery + Fair-Loss Link + Asynchronous.



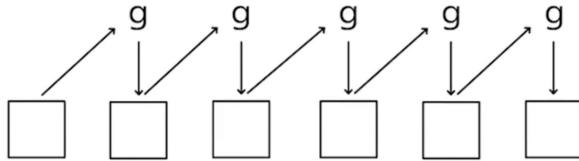
3. Byzantine-Failure + Byzantine Link + Asynchronous

- It is a model adopted for the systems where computational components spread across the globe of unreliable and untrusted network connections.
- The common representative of this model is grid computing.
- MapReduce
 - Invented by Jeffery Dean and Sanjay Ghemawat, presented on Symposium and Operating Systems Design and implementation in 2004.
 - Map: apply the same function to each element of your collection.



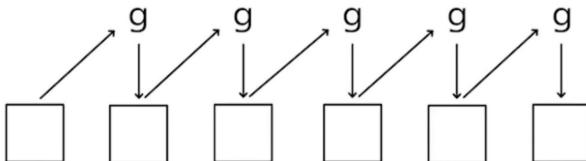
```
>>> map(lambda x: x*x, [1,2,3,4])  
[1, 4, 9, 16]
```

- Reduce (Fold / Aggregate)



```

>>> reduce(operator.sum, [1, 4, 9, 16])
>>> reduce(operator.sum, [5, 9, 16])
>>> reduce(operator.sum, [14, 16])
30
  
```



```

>>> average = lambda x, y: (x + y) / 2.
>>> reduce(average, [1, 2, 3])
2.25
  
```

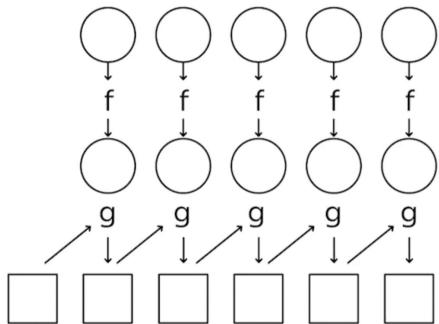
different result.

1.75

- Reduce operator cause a sequence of elements by applying the following procedure iteratively.

- reducing function sum is associative.
- mean function is not associative: changing the order of the atom effects the result.

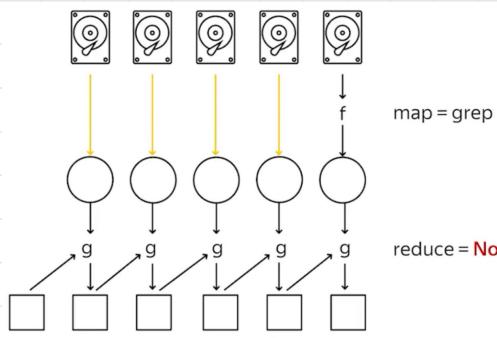
- MapReduce : combine Map and Reduce together.



```

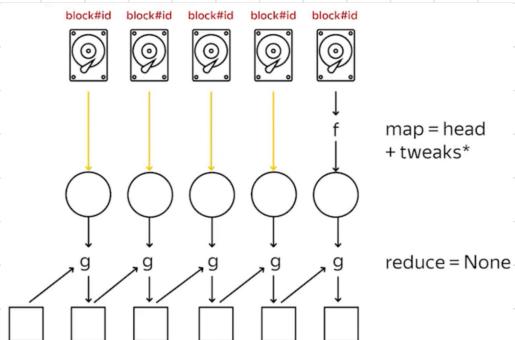
>>> reduce(operator.add, map(lambda x: x*x,
[1, 2, 3, 4]))
30
  
```

• Distributed Shell: grep



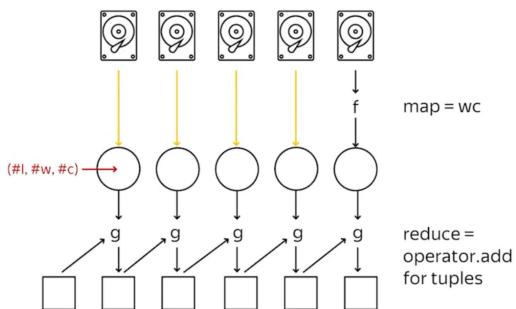
reduce = None (in MapReduce application, you don't always need map or reduce function)

• Distributed Shell: head



reduce = None

• Distributed Shell: wc



reduce = operator.add
for tuples

• Distributed Shell: Word Count

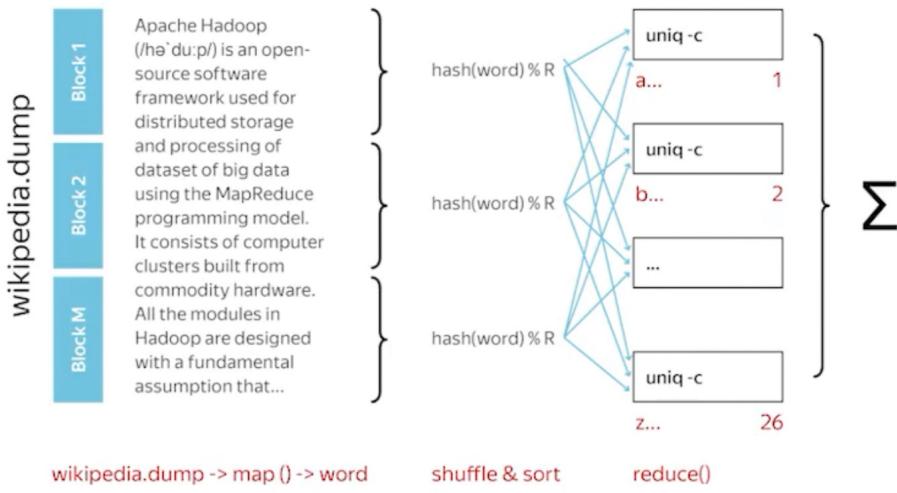
distributed: cat * | tr ' ' '\n' | sort | uniq -c

map=sort

reduce=sort (doesn't fit in Memory / Disk) \Rightarrow introduces shuffle and sort.

Map → Shuffle & Sort → Reduce

- text is split into words.
- words are distributed to a reduce phase in the way that reduce function can be executed independently on different machines.



• MapReduce Formal Model.

- All input and output of map and reduce function should be a **key value pair**.

map: (key, value) → (key, value)

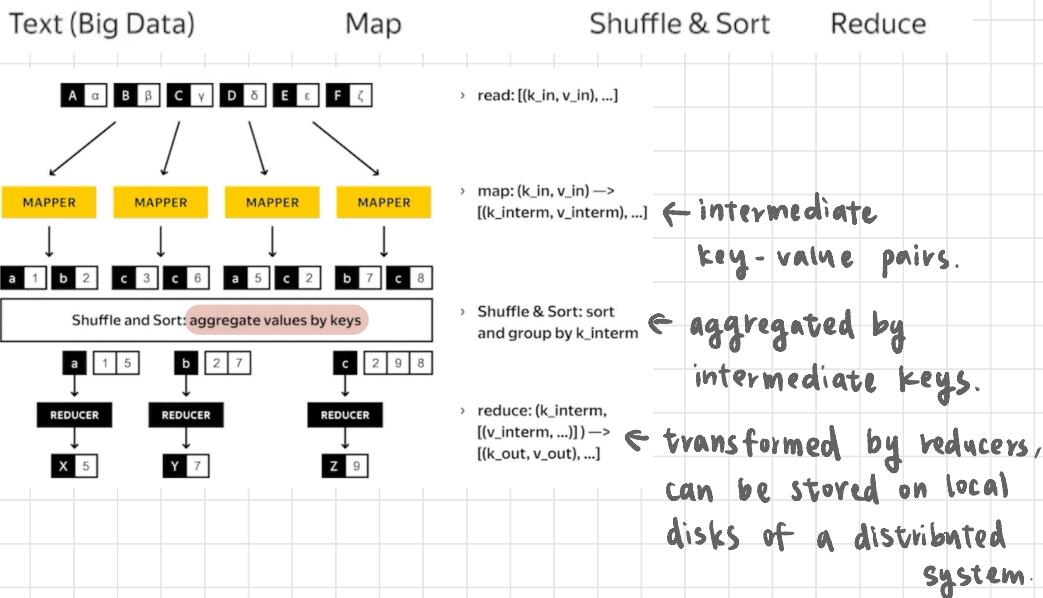
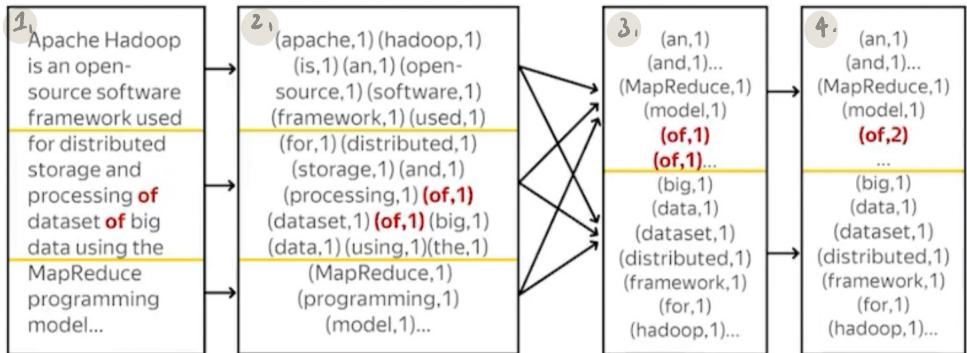
reduce: (key, value) → (key, value)

• WordCount example

```
$ cat -n wikipedia.dump | tr ' ' '\n' | sort | uniq -c
```

```
> cat -n wikipedia.dump: [(line_no, line), ...] 1
> tr ' ' '\n': (-, line) → [(word, 1), ...] 2
> sort: Shuffle & Sort 3,
> uniq -c: (word, [1, ...]) → (word, count) 4.
```

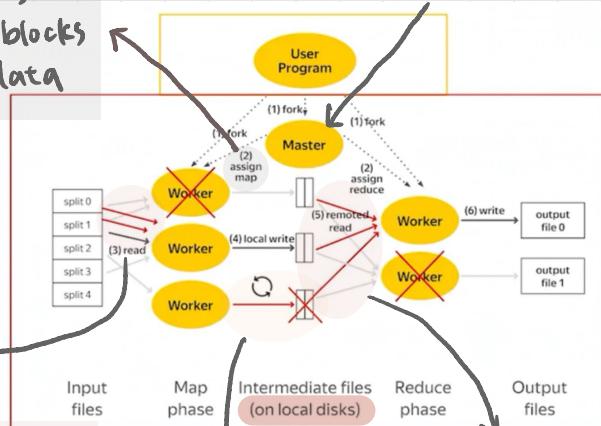
1. read data and get pairs with line number, line content.
2. map phase: ignore line number and split lines into words.
3. shuffle and sort phase: spread the words by the hashes.
4. reduce phase: sum up to get the answer.



• MapReduce framework: Fault Tolerance Model

launch mappers to process input blocks or splits of data

control the execution



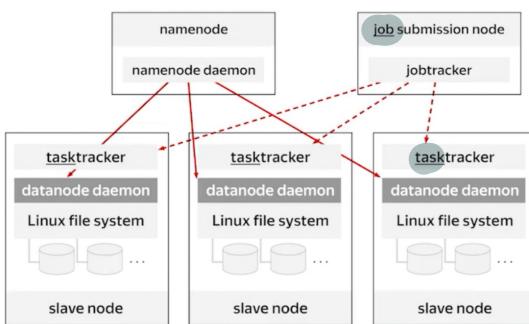
assign another worker to execute mapper against these data.

re-execute mapper on another alive worker if this data is lost

shuffle and sort data for this particular reducer to another worker.

- All this complexity is hidden in MapReduce framework to make your life way more easier.
- You only need to provide deterministic map and reduce functions.

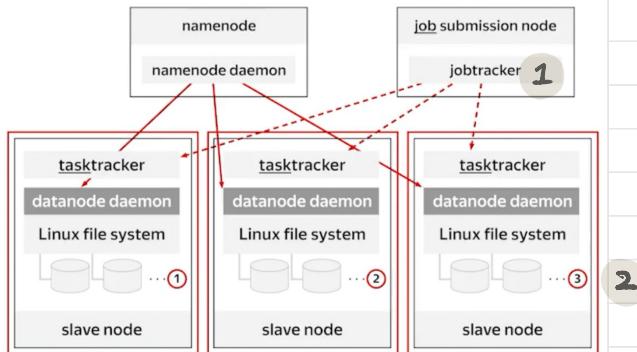
• MapReduce concepts



jobs: One MapReduce application is a job, is a big chunk of work.

task: A map or reduce function applied to some chunk of data.

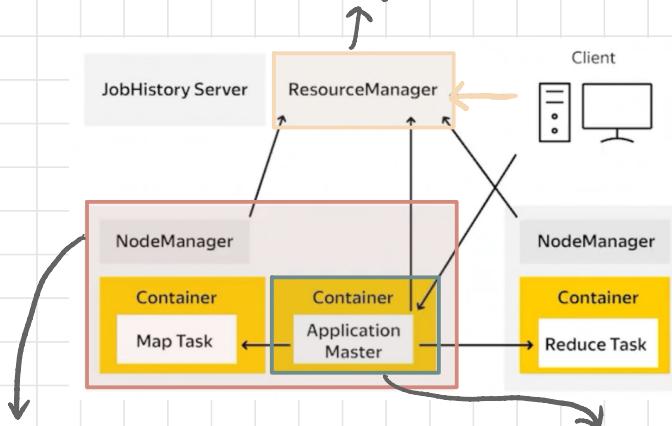
- Hadoop MapReduce v1.



1. There was one global JobTracker to direct execution of MapReduce jobs. It is usually located on one high-cost and high-performance node with HDFS namenode.
2. TaskTrackers are located once per every node where you store data or where datanode daemon is working. TaskTracker spawns workers from mapper or reducer.
3. In this scenario, JobTracker is a single point of failure. That is why to reduce the load on JobTracker, some functionality of future version was delegated to other cluster nodes.

• YARN (Yet Another Resource Negotiation)

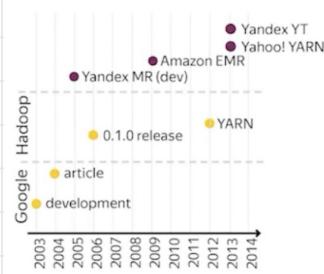
ResourceManager oversees NodeManagers and client request resources for execution.
MapReduce applications can work on top of this resource layer.



TaskTrackers are substituted by NodeManagers who can provide a layer of CPU and RAM containers.

No global JobTracker because Application Master can start on any node.

• MapReduce Frameworks: History Timeline



- > [2003] Google MapReduce (development)
- > [2004] Google MapReduce (article)
- > [2005] Yandex MapReduce (development)
- > [2006] Hadoop 0.1.0 release
- > [2009] Amazon EMR (Hadoop inside)
- > [2012] MapReduce → YARN
- > [2013] Yahoo! YARN deployed in production
- > [2013] Yandex YT
- > ...
- > MapReduce in MongoDB, Riak, ...

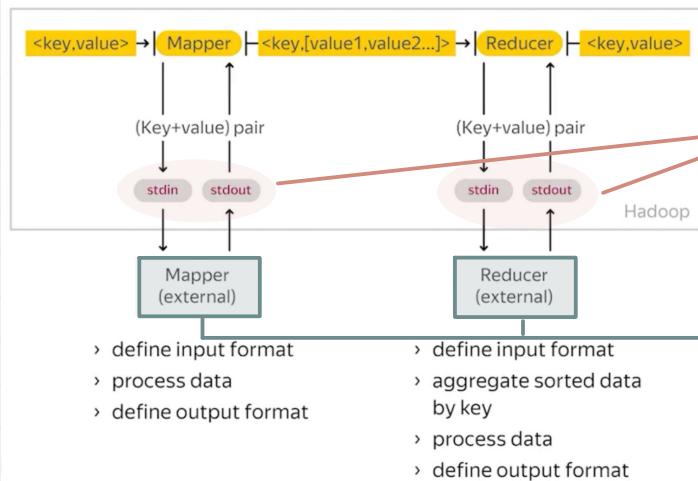
Hadoop MapReduce Streaming Applications

in Python.

- Streaming



Hadoop provides a possibility to write mappers, and reducers in any language, which is called **streaming**.



communicate via
standard input and
output channels.

→ You have to implement
your own mappers
and reducers.

- example: Line Count.

```
/opt/cloudera/parcels/CDH-5.9.0-1.cdh5.9.0.p0.23/lib/hadoop-mapreduce/hadoop-streaming.jar
```

```
HADOOP_STREAMING_JAR="/path/to/hadoop-streaming.jar"  
yarn jar $HADOOP_STREAMING_JAR \  
  -mapper 'wc -l' \  
  -numReduceTasks 0 \  
  -input /data/wiki/en_articles \  
  -output wc_mr
```

execute yarn application.

- define the path to the streaming jar.
- a bash command mapper.
- need to be specified if no reducer.
- HDFS folder or file going to process.
- HDFS folder for output.

output:

```
$ hdfs dfs -ls wc_mr
Found 3 items
-rw-r--r-- 3 adral adral          0 2017-03-21 14:48 wc_mr/_SUCCESS
-rw-r--r-- 3 adral adral          6 2017-03-21 14:48 wc_mr/part-00000
-rw-r--r-- 3 adral adral          6 2017-03-21 14:48 wc_mr/part-00001
```

only two mappers were executed.

```
$ hdfs dfs -text wc_mr/*
1986
2114
```

$$1986 + 2114 = 4100 \text{ Wikipedia articles.}$$

provide reducer which aggregates the number of articles from all the mappers:

```
HADOOP_STREAMING_JAR="/path/to/hadoop-streaming.jar"
yarn jar $HADOOP_STREAMING_JAR \
-mapper 'wc -l' \
-reducer "awk '{line_count += \$1} END { print line_count
}'" \
-numReduceTasks 1 \
-input /data/wiki/en_articles \
-output wc_mr
```

escape special characters in the Shell map and reduce commands

```
$ hdfs dfs -ls wc_mr_with_reducer
Found 2 items
-rw-r--r-- 3 adral adral      wc_mr_with_reducer/_SUCCESS
-rw-r--r-- 3 adral adral      wc_mr_with_reducer/part-00000

$ hdfs dfs -text wc_mr_with_reducer/*
4100
```

provide reducer from shell script:

reducer.sh

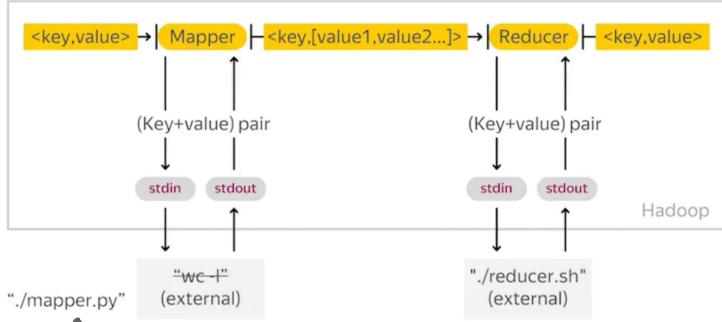
```
#!/usr/bin/env bash
awk '{line_count += $1} END { print line_count }'
```

don't need escaping.

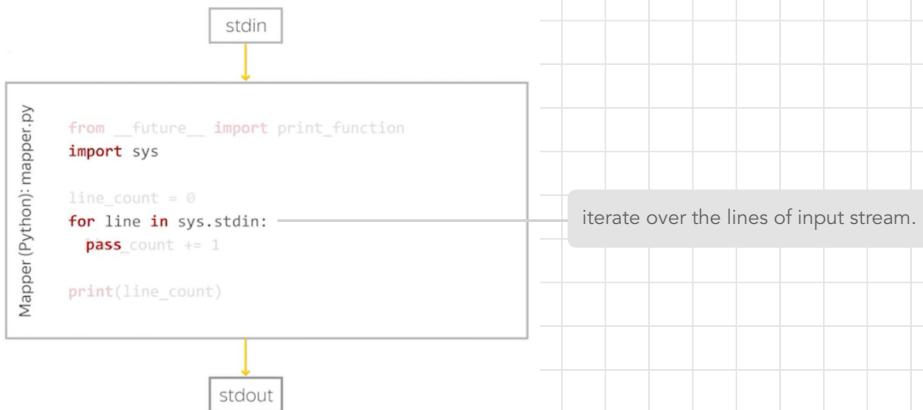
```
HADOOP_STREAMING_JAR="/path/to/hadoop-streaming.jar"
yarn jar $HADOOP_STREAMING_JAR \
-mapper 'wc -l' \
-reducer './reducer.sh' \
-file reducer.sh \
-numReduceTasks 1 \
-input /data/wiki/en_articles \
-output wc_mr_with_reducer
```

copy reducer.sh from the local storage to be available for MapReduce workers.

• Streaming in Python.



replace shell command to Python script that reads data from standard input, and prints it out to the standard output.



```
HADOOP_STREAMING_JAR="/path/to/hadoop-streaming.jar"
yarn jar $HADOOP_STREAMING_JAR \
  -files mapper.py,reducer.sh \
  -mapper 'python mapper.py' \
  -reducer './reducer.sh' \
  -numReduceTasks 1 \
  -input /data/wiki/en_articles \
  -output wc_mr_with_reducer
```

specify multiple files to distribute over the workers as a comma separated list.

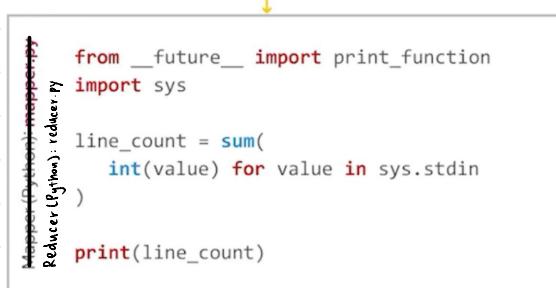
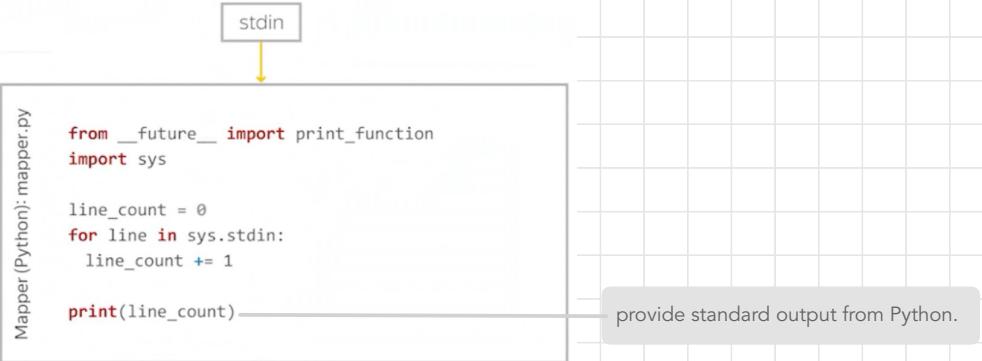
use python mapper.

```
$ hdfs dfs -ls wc_mr_with_reducer
Found 2 items
-rw-r--r-- 3 adral adral 0 <date> wc_mr_with_reducer/_SUCCESS
-rw-r--r-- 3 adral adral 0 <date> wc_mr_with_reducer/part-00000
```

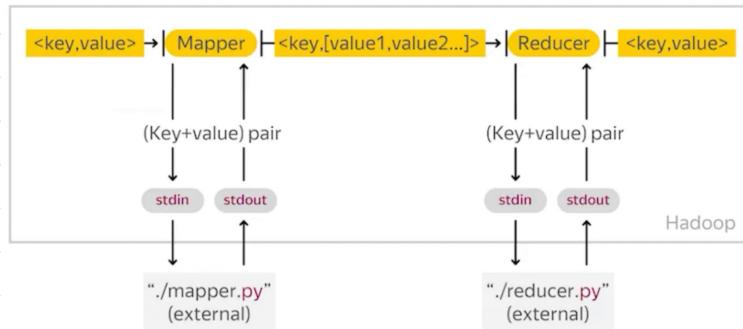
of replicas filesize

```
$ hdfs dfs -text wc_mr_with_reducer/*
--
```

empty output by running this streaming MapReduce job.



```
HADOOP_STREAMING_JAR="/path/to/hadoop-streaming.jar"
yarn jar $HADOOP_STREAMING_JAR \
    -files mapper.py,reducer.py \
    -mapper 'python mapper.py' \
    -reducer 'python reducer.py' \
    -numReduceTasks 1 \
    -input /data/wiki/en_articles \
    -output wc_mr_with_reducer
```



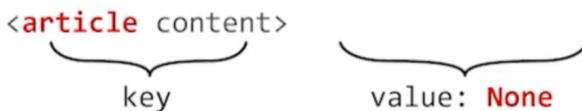
• Word Count in Python.

- key-value pairs for the input and output streams.



key: before tab , value: after tab

if no tab character in the line :



mapper.py

```
from __future__ import print_function
import sys

for line in sys.stdin:
    article_id, content = line.split("\t", 1)
    words = content.split()
    for word in words:
        print(word, 1, sep="\t")
```

```
yarn jar $HADOOP_STREAMING_JAR \
    -files mapper.py \
    -mapper 'python mapper.py' \
    -numReduceTasks 0 \
    -input /data/wiki/en_articles \
    -output word_count
```

```
$ hdfs dfs -text /data/wiki/en_articles/* | head -c 80
12 <tab> Anarchism      Anarchism is often defined as a
political philosophy which ...
```

```
$ hdfs dfs -ls -h word_count
Found 3 items
-rw-r--r-- 3 adral adral 0 2017-03-22 11:40 word_count/_SUCCESS
-rw-r--r-- 3 adral adral 47.8 M 2017-03-22 11:40 word_count/part-00000
-rw-r--r-- 3 adral adral 47.9 M 2017-03-22 11:40 word_count/part-00001
```

result :

part-00000	part-00001
Basel 1	Anarchism 1
Basel 1	Anarchism 1
(1	is 1
) 1	often 1
or 1	defined 1
...	...

```
yarn jar $HADOOP_STREAMING_JAR \
    -files mapper.py \
    -mapper 'python mapper.py' \
    -numReduceTasks 1 \
    -input /data/wiki/en_articles \
    -output word_count
```

one reducer with default implementation which does nothing, the shuffle and sort phase will be executed.

```
$ hdfs dfs -text word_count/part-00000 | head
! 1
! 1
! 1
! 1
...
sorted output!
```

modify mapper.py

```
from __future__ import print_function
import re
import sys

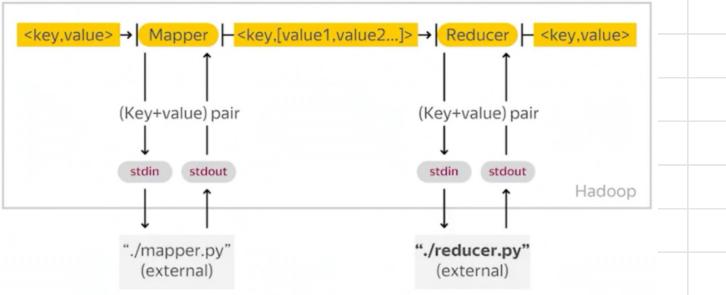
for line in sys.stdin:
    article_id, content = line.split("\t", 1)
    words = re.split("\W+", content) ——————
    for word in words:
        if word:
            print(word, 1, sep="\t")
```

split all non-word characters.

```
yarn jar $HADOOP_STREAMING_JAR \
    -files mapper.py \
    -mapper 'python mapper.py' \
    -numReduceTasks 1 \
    -input /data/wiki/en_articles \
    -output word_count
```

```
$ hdfs dfs -text word_count/part-00000 | head -4
0 1
0 1
0 1
0 1
```

```
$ hdfs dfs -tail word_count/part-00000 | tail -4
zyu1 1
zyu1 1
zz 1
zz 1
```



- > define input format
- > **aggregate sorted data by key**
- > process data
- > define output format

Reducer has responsibility to aggregate value by keys.

reducer.py

```
from __future__ import print_function
import sys

current_word = None
word_count = 0

for line in sys.stdin:
    word, counts = line.split("\t", 1)
    counts = int(counts)
    if word == current_word:
        word_count += counts
    else:
        if current_word:
            print(current_word, word_count, sep="\t")
        current_word = word
        word_count = counts

if current_word:
    print(current_word, word_count, sep="\t")
```

initialize current_word and word_count.

parse input key-value pair.

if see the same word, increase the counter.

otherwise,
1. output aggregate stats for previous word.
2. update the counter for a new key.

output accumulated stat for the last key.

```
yarn jar $HADOOP_STREAMING_JAR \
  -files mapper.py,reducer.py \
  -mapper 'python mapper.py' \
  -reducer 'python reducer.py' \
  -numReduceTasks 1 \
  -input /data/wiki/en_articles \
  -output word_count
```

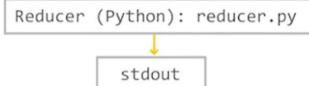
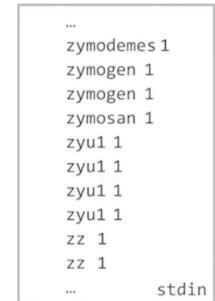
one reducer

```
$ hdfs dfs -ls -h word_count
```

Found 2 items

-rw-r--r--	3	adral adral	0 2017-03-22 13:05	word_count/_SUCCESS
-rw-r--r--	3	adral adral	3.2 M 2017-03-22 13:05	word_count/part-00000

one file in output



```
$ hdfs dfs -text word_count/part-00000
0 14905
00 844
0008186
...
zymodemes 1
zymogen 2
zymosan 1
zyu1 4
zz 2
```



```
yarn jar $HADOOP_STREAMING_JAR \
    -files mapper.py,reducer.py \
    -mapper 'python mapper.py' \
    -reducer 'python reducer.py' \
    -input /data/wiki/en_articles \
    -output word_count
```

remove -numReduceTasks argument,
MapReduce job will have an arbitrary
number of reducer.

```
$ hdfs dfs -ls -h word_count
Found 11 items
-rw-r--r-- 3 adral adral 0 2017-03-22 13:19 word_count/_SUCCESS
-rw-r--r-- 3 adral adral 331.0 K 2017-03-22 13:18 word_count/part-00000
-rw-r--r-- 3 adral adral 332.1 K 2017-03-22 13:18 word_count/part-00001
-rw-r--r-- 3 adral adral 331.7 K 2017-03-22 13:18 word_count/part-00002
-rw-r--r-- 3 adral adral 329.8 K 2017-03-22 13:18 word_count/part-00003
-rw-r--r-- 3 adral adral 326.1 K 2017-03-22 13:18 word_count/part-00004
-rw-r--r-- 3 adral adral 332.2 K 2017-03-22 13:18 word_count/part-00005
-rw-r--r-- 3 adral adral 332.3 K 2017-03-22 13:18 word_count/part-00006
-rw-r--r-- 3 adral adral 331.4 K 2017-03-22 13:18 word_count/part-00007
-rw-r--r-- 3 adral adral 330.5 K 2017-03-22 13:19 word_count/part-00008
-rw-r--r-- 3 adral adral 330.7 K 2017-03-22 13:19 word_count/part-00009
```

will see several files in the output.

```
$ hdfs dfs -tail word_count/part-... | tail -5
```

part-00000	part-00005
...	...
zsu 1	
zuang 1	
zucchini 5	
zuchetto 1	
zuerst 1	
zure 1	
...	...

In each file, data is sorted by keys, but not globally sorted as I shuffled between reducers.

TotalOrderPartitioner

can sort keys between
the reducers.

• Distributed Cache

if I only want to count some specific word in word count example ?

mapper.py

```
from __future__ import print_function
import re
import sys

def read_vocabulary(file_path):
    return set(word.strip() for word in open(file_path))

vocabulary = read_vocabulary("vocabulary.txt")

for line in sys.stdin:
    article_id, content = line.split("\t", 1)
    words = re.split("\W+", content)
    for word in words:
        if word in vocabulary:
            print(word, 1, sep="\t")
```

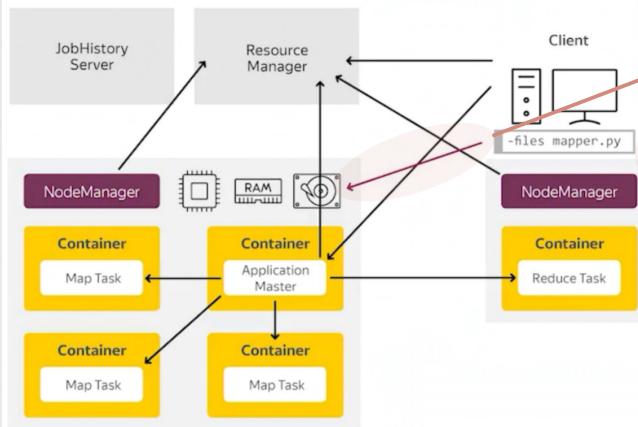
read vocabulary in memory.

```
yarn jar $HADOOP_STREAMING_JAR \
  -files mapper.py,reducer.py,vocabulary.txt \
  -mapper 'python mapper.py' \
  -reducer 'python reducer.py' \
  -input /data/wiki/en_articles \
  -output word_count
```

filter words by these vocabularies.

add this vocabulary.txt into distribute files during execution.

• Distributed Cache.



If you provide -files, each of this file will be copied once by each node before any task execution.

- ways to distribute files.

1. -files

2. -archives : better utilize network profile transmission.

all archives will be unpacked on worker nodes.

3. -libjars : JAR stands for Java Archive.

- example for -archives : count the most popular male and female name.

(1) create tar file



(2) execute MapReduce application with -archives

```
yarn jar $HADOOP_STREAMING_JAR \
    -files mapper.py,reducer.py,vocabulary.txt \
    -archives names.tar \
    -mapper 'python mapper.py' \
    -reducer 'python reducer.py' \
    -input /data/wiki/en_articles \
    -output word_count
```

(3) mapper.py

```
from __future__ import print_function
import re
import sys

def read_vocabulary(file_path):
    return set(word.strip() for word in open(file_path))

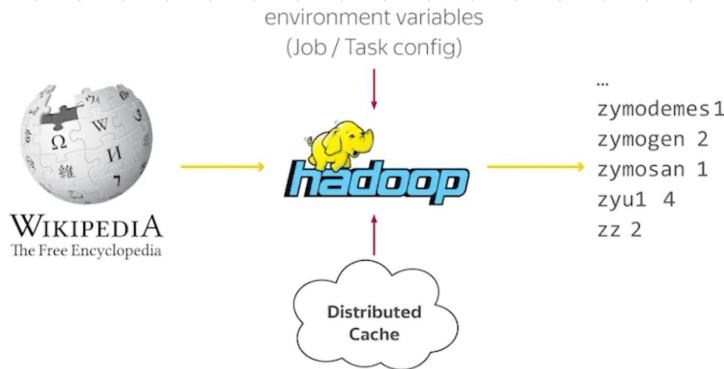
male_names = read_vocabulary("names.tar/male.txt")
female_names = read_vocabulary("names.tar/female.txt")

for line in sys.stdin:
    article_id, content = line.split("\t", 1)
    words = re.split("\W+", content)
    for word in words:
        if word in male_names or word in female_names:
            print(word, 1, sep="\t")
```

tar file will be unpacked to the folder with the same name

• Environment, Counters.

Distributed cache is not the only way to pass information to script, you can also get job configuration options through environment variables.



When you launch MapReduce application, framework will assign of data to available workers, you can access these data from your script.

`os.environ["mapreduce_task_id"]`

Get an absolute task id, is usually available on job tracker UI.

The screenshot shows the Hadoop Job Tracker interface. On the left, there's a sidebar with "Application" selected, showing "Overview", "Configuration", "Map Reduce tasks", and "Tools". The main area is titled "Map Tasks for job_1488734338480_1443". A table lists two tasks:

Name	State	Task	Start Time	Finish Time	Elapsed Time	Start Time	Finish Time	Elapsed Time
task_1488734338480_1443_m_000001	SUCCEEDED	task	Sat Mar 25 19:57:00 2017	Sat Mar 25 19:57:00 2017	4sec	Sat Mar 25 19:57:00 2017	Sat Mar 25 19:57:00 2017	4sec
task_1488734338480_1443_r_000008	SUCCEEDED	task	Sat Mar 25 19:57:01 +0000 2017	Sat Mar 25 19:57:05 +0000 2017	4sec	Sat Mar 25 19:57:01 +0000 2017	Sat Mar 25 19:57:05 +0000 2017	4sec

A yellow arrow points from the "task_1488734338480_1443_m_000001" row to the text "os.environ['mapreduce_task_id']".

Mapper (Python): mapper.py

```

from __future__ import print_function
import re
import sys

if os.environ["mapred_task_is_map"] == "true":
    print("input_file:{}, start:{}, size:{}").format(
        os.environ["mapreduce_map_input_file"],
        os.environ["mapreduce_map_input_start"],
        os.environ["mapreduce_map_input_length"])
)
  
```

environment variables
(Job / Task config)

for line in sys.stdin:

pass

`os.environ["mapreduce_task_partition"]`

Get the relative order of the task within map or reduce phase.

task_1488734338480_1443_m_000001 → 1
task_1488734338480_1443_r_000008 → 8

```
yarn jar $HADOOP_STREAMING_JAR -D word_pattern="\w+\d+" \
    -files mapper.py \
    -mapper 'python mapper.py' \
    -reducer 'python reducer.py' \
    -input /data/wiki/en_articles \
    -output word_count
```

use -D to provide arbitrary environment variables.

Mapper (Python): wc_mapper.py

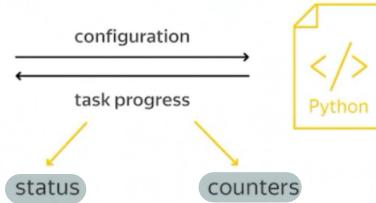
```
from __future__ import print_function
import os
import re
import sys

pattern = re.compile(os.environ["word_pattern"])

for line in sys.stdin:
    article_id, content = line.split("\t", 1)
    words = re.findall(pattern, content)
    for word in words:
        print(word, 1, sep="\t")
```

use os.environ() to get the environment variables which specified in executing yarn app.

There is also a backward communication channel between framework and your script. For instance, you can provide information about task progress: **status** and **counters**.



- **status**

Mapper (Python): reporter_mapper.py

```
from __future__ import print_function
import re
import sys

for line in sys.stdin:
    article_id, content = line.split("\t", 1)
    words = re.findall("\w+", content)
    for index, word in enumerate(words):
        print(word, 1, sep="\t")
        print("reporter:status:processed {} words"
              .format(index + 1), file=sys.stderr)
```

you can provide an arbitrary message in status for each task execution.

counter

Mapper (Python): reporter_mapper.py

```
from __future__ import print_function
import re
import sys

for line in sys.stdin:
    article_id, content = line.split("\t", 1)
    words = re.findall("\w+", content)
    for index, word in enumerate(words):
        print(word, 1, sep="\t")
        print("reporter:status:processed {} words".format(index + 1), file=sys.stderr)
        print("reporter:counter:Personal Counters[word found,1]", file=sys.stderr)
        file=sys.stderr)
    )

```

reporter:counter:<group>, <counter>, <amount>



	Name	Map	Reduce	Total
Map-Reduce Framework	Combine input records	0	0	0
	Combine output records	0	0	0
	CPU time spent (ms)	2066620	63080	269700
	Failed Shuffles	0	0	0
	GC time elapsed (ms)	832	1533	2365
	Input split bytes	264	0	264
	Map input records	4100	0	4100
	Map output bytes	98534099	0	98534099
	Map output materialized bytes	9634808	0	9634808
	Map output records	12396473	0	12396473
	Merged Map outputs	0	20	20
	Physical memory (bytes) snapshot	3370938368	2752958464	6123896832
	Reduce input groups	0	306456	306456
	Reduce input records	0	12396473	12396473
	Reduce output records	0	306456	306456
	Reduce shuffle bytes	0	9634808	9634808
	Shuffled Maps	0	20	20
	Spilled Records	12396473	12396473	24792946
	Total committed heap usage (bytes)	364222736	6243221504	9885450240
	Virtual memory (bytes) snapshot	7705399296	83205345280	90910744576
Personal Counters	word found	12396473	0	12396473



this information is available on JobTracker UI,

• Testing.



• Unit Testing

- test function **edge cases**.
- python library: **pytest**

```
def get_words(input_line):  
    return input_line.split()  
  
def test_get_words_parse_simple_string():  
    assert get_words("a b cd efg ") == ["a", "b", "cd", "efg"]  
  
def test_get_words_parse_empty_string():  
    assert get_words("") == []  
  
def test_get_words_raise_exception_if_no_input():  
    with pytest.raises(AttributeError):  
        get_words(None)
```

```
(venv) > coursesa pytest -v test_function.py  
platform darwin -- Python 2.7.12, pytest-3.0.7, py-1.4.33, pluggy-0.4.6 -- /Users/aadral/workspace/personal/mipt/coursesa/venv/bin/python2.7  
cachedir: .cache  
rootdir: /Users/aadral/workspace/personal/mipt/coursesa, inifile:  
collected 3 items  
  
test_function.py::test_get_words_parse_simple_string PASSED  
test_function.py::test_get_words_parse_empty_string PASSED  
test_function.py::test_get_words_raise_exception_if_no_input PASSED  
  
3 passed in 0.01 seconds
```

prefix test function with test

• Integration Testing.

- validate how mappers and reducers scripts are integrated with Hadoop MapReduce streaming API.
- will be working out of the box if your scripts rely on MapReduce job configuration options.
⇒ Hadoop MapReduce framework provides an **empty config**. for System Testing.

Pytest will scan and execute all the functions with this prefix.

• System Testing.

- execute whole pipeline end to end.

```
hdfs --config $HADOOP_EMPTY_CONFIG dfs -rm -r word_count  
yarn --config $HADOOP_EMPTY_CONFIG jar $HADOOP_STREAMING_JAR
```

```
$ locate "hadoop/conf.empty"
```

If you provide a Hadoop empty config, you execute the MapReduce application in a standalone mode.

find a path to an empty config.

- standalone mode.

HDFS client point out to a local file system, and a node manager is working on the same node. Streaming scripts will be able to communicate with MapReduce framework via environment variables. You will be able to read configuration of variables and validate correctness.

• Acceptance Testing.

- final stage before shipping code into production system.
- validation against a sample dataset without wasting your time and CPU cycles.
- validation against big datasets and measuring performance or efficiency.