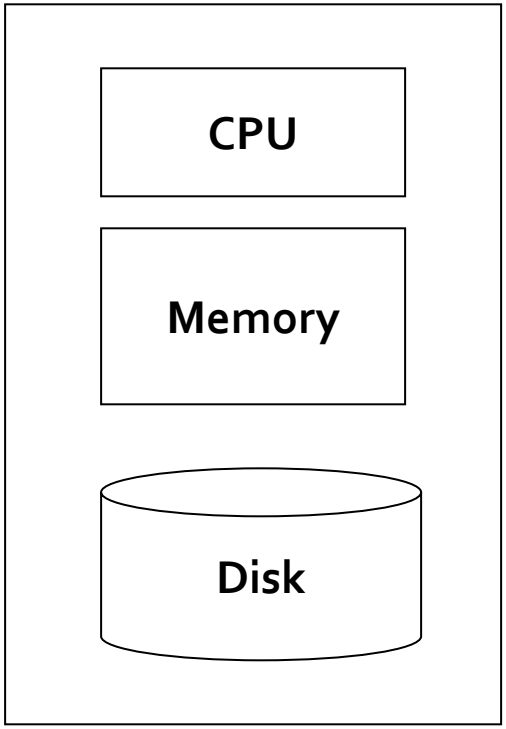
Hadoop mapreduce practice

MapReduce

- Much of the course will be devoted to **large scale computing** for **data mining**
- **Challenges:**
  - How to distribute computation?
  - Distributed/parallel programming is hard
- **Map-reduce** addresses all of the above
  - Google's computational/data manipulation model
  - Elegant way to work with big data

Single Node Architecture



Machine Learning, Statistics

Small data

Data can be completely loaded in main memory

"Classical" Data Mining

Large data

Data can not be completely loaded in main memory

- Load in main memory one chunk of data at a time
- Process it and store some statistics
- Combine statistics to compute the final result

Motivation: Google Example

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
  - ~4 months to read the web
- ~1,000 hard drives to store the web
- Takes even more to **do** something useful with the data!
- **Today, a standard architecture for such problems is emerging:**
  - Cluster of commodity Linux nodes
  - Commodity network (ethernet) to connect them

Storage Infrastructure

- **Problem:**
  - If nodes fail, how to store data persistently?
- **Answer:**
  - **Distributed File System:**
    - Provides global file namespace
    - Google GFS; Hadoop HDFS;
- **Typical usage pattern**
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

Data volume

- The amount of data increases every day
- Some numbers (~**2012**):
  - Data processed by Google every day: 100+ PB
  - Data processed by Facebook every day: 10+ PB
- To analyze them, systems that scale with respect to the data volume are needed

Data volume

- Analyze 10 billion web pages
- Average size of a webpage: 20KB
- Size of the collection: 10 billion x 20KBs = 200TB
- HDD hard disk read bandwidth: 150MB/sec
- Time needed to read all web pages(without analyzing them): 2 million seconds = more than **15 days**
- A single node architecture is not adequate

Data volume

- Analyze 10 billion web pages
- Average size of a webpage: 20KB
- Size of the collection: 10 billion x 20KBs = 200TB
- SSD hard disk read bandwidth: 550MB/sec
- Time needed to read all web pages(without analyzing them): 2 million seconds = more than **4 days**
- A single node architecture is not adequate

Failures

- Failures are part of everyday life, especially in data center
- A single server stays up for 3 years (~1000 days)
  - 10 servers →1 failure every 100 days (~3 months)
  - 100 servers →1 failure every 10 days
  - 1000 servers →1 failure/day
- Sources of failures
  - Hardware/Software
  - Electrical, Cooling, ...
  - Unavailability of a resource due to overload

Network bandwidth

- Network becomes the bottleneck if big amounts of data need to be exchanged between nodes/servers
  - Network bandwidth (in a data center): 10Gbps
  - Moving 10 TB from one server to another takes more than 2 hours
    - **Data** should be **moved across nodes only when** it is **indispensable**
- Usually, codes/programs are small (few MBs)
  - **Move code** (programs) and **computation** to data

Scale up vs. scale out

- Vertical scalability (scale up)
  - Add more power/resources (main memory, CPUs) to a single node (high-performing server)
    - Cost of super-computers is not linear with respect to their resources
- Horizontal scalability (scale out)
  - Add more nodes (commodity servers) to a system
    - The cost scales approximately linearly with respect to the number of added nodes
    - But data center efficiency is a difficult problem to solve
- Horizontal scalability (scale out) is preferred for big data applications
  - But distributed computing is hard
  - New systems hiding the complexity of the distributed part of the problem to developers are needed

Lambda architecture: Definition #1

- ***Hadoop**, for example, can **parallelize large-scale batch computations** on very large amounts of data, but the computations have **high latency**. You don't use Hadoop for anything where you need low-latency results.*
- *"**Lambda architecture** is a **data-processing architecture** designed to **handle massive** quantities of **data** by taking advantage of both **batch** and **stream-processing** methods.*

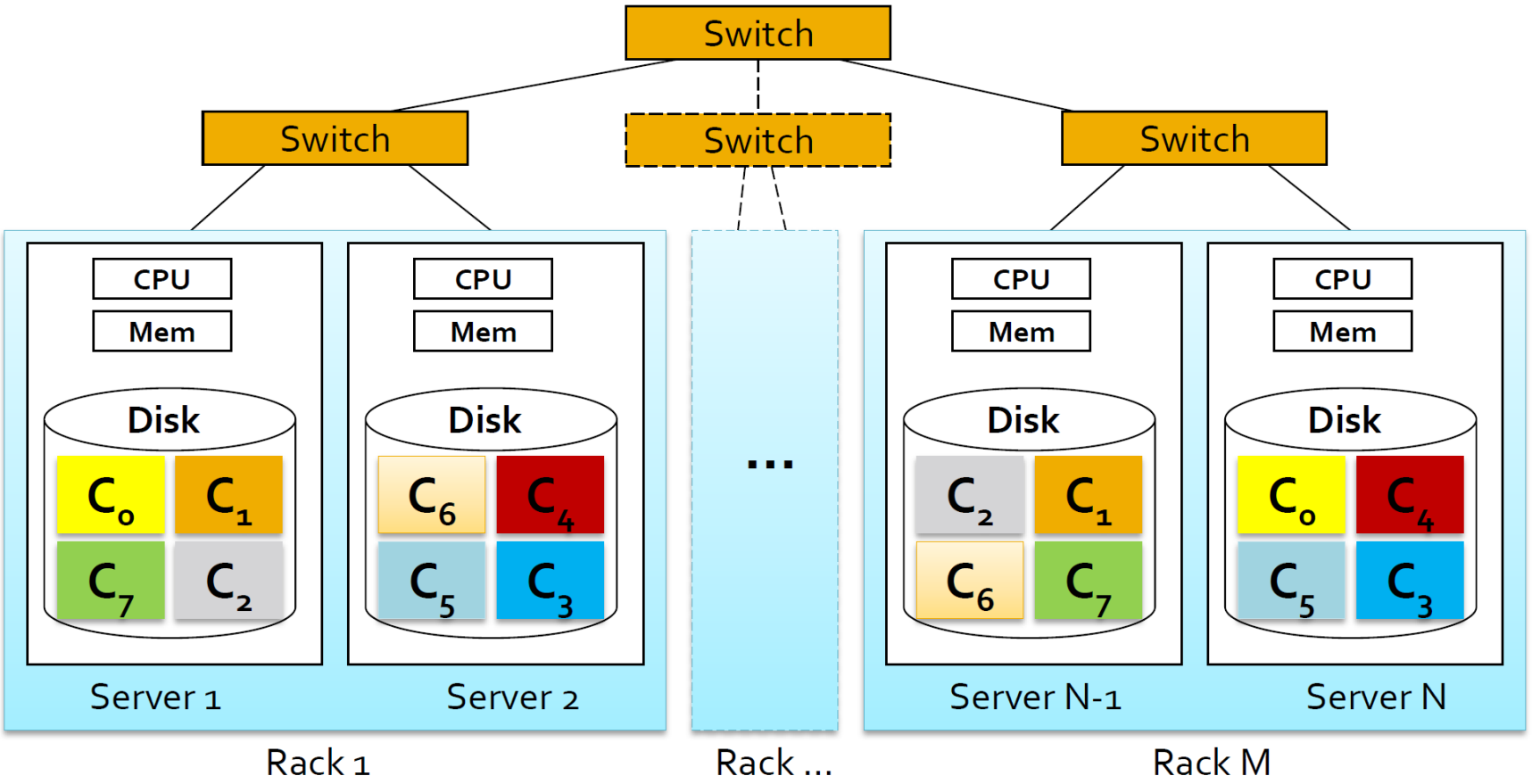
Hadoop
Designed for **Data intensive workloads**

HPC(High-performance computing)
Designed for **CPU intensive tasks**
Usually it is used to process "small" data sets

Data locality

- Traditional distributed systems (e.g., HPC) move data to computing nodes (servers)
  - This approach cannot be used to process TBs of data
  - The network bandwidth is limited
- Hadoop moves code to data
  - Code (few KB) is copied and executed on the servers where the chunks of data are stored
  - This approach is based on "**data locality**"

Cluster Architecture

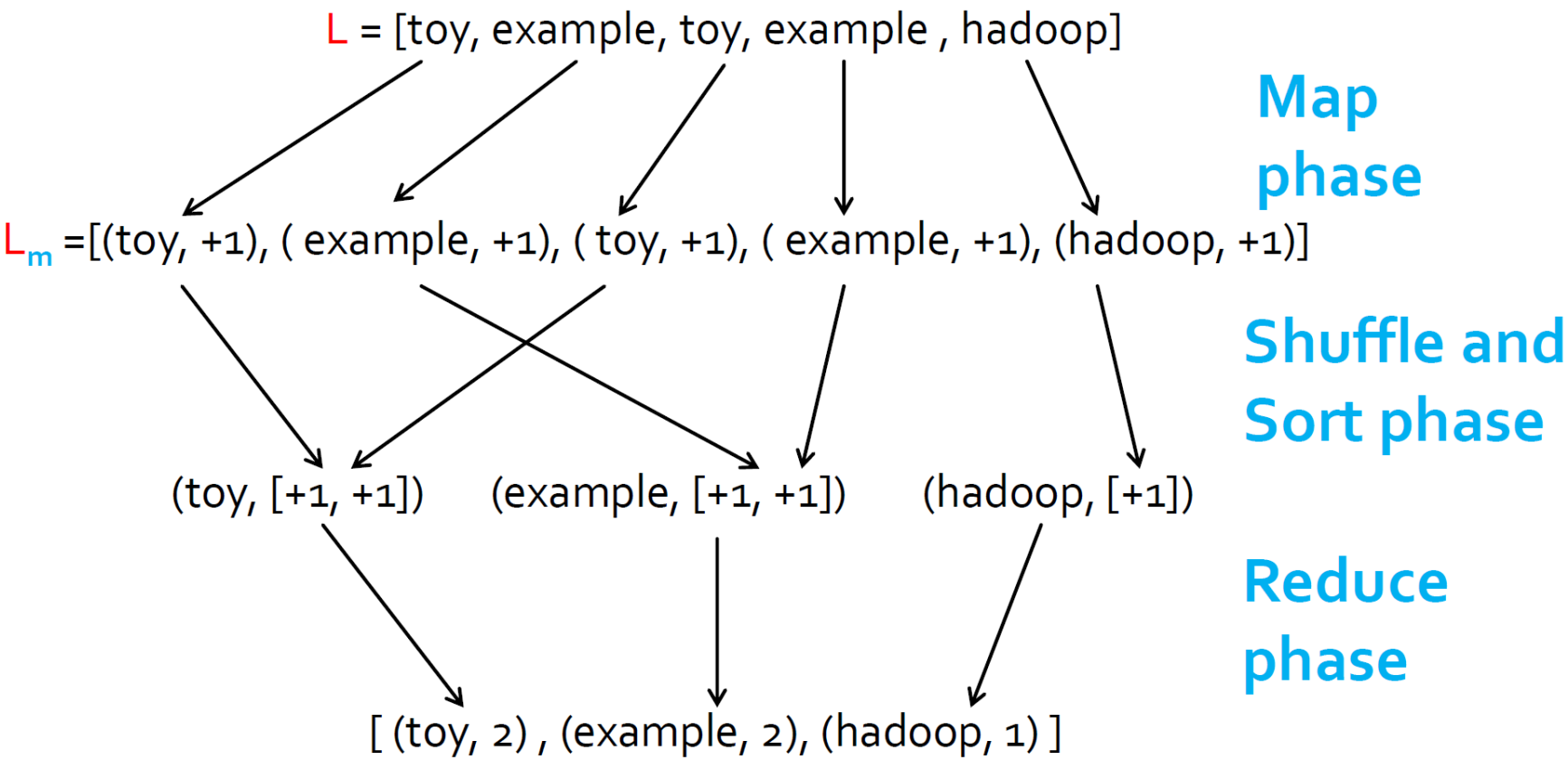


Example with number of replicas per chunk = 2

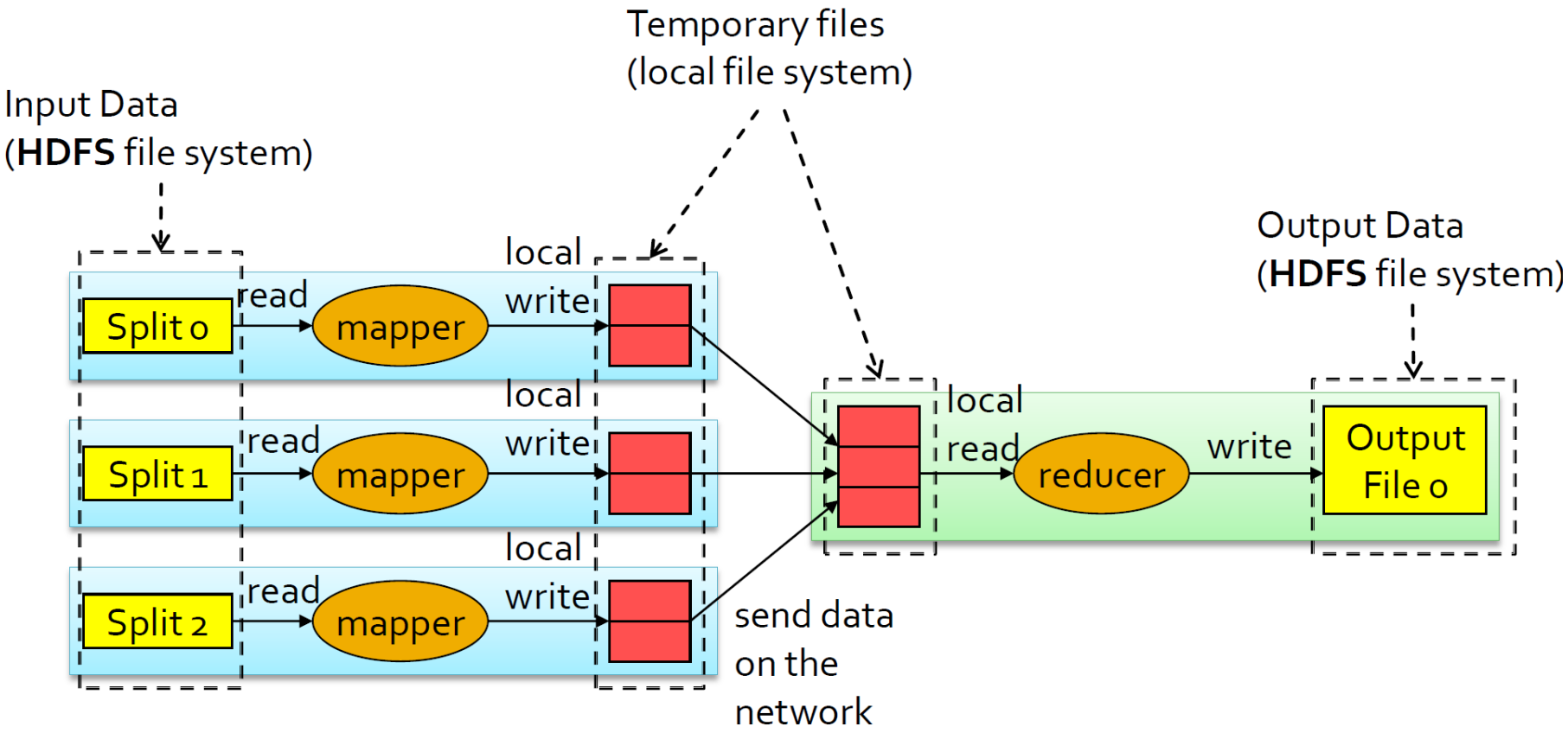
hadoop

- Hadoop/MapReduce is designed for
  - Batch processing involving (mostly) full scans of the input data
- Data-intensive applications
  - Read and process the whole Web (e.g., PageRank computation)
  - Read and process the whole Social Graph (e.g., Link Prediction, a.k.a. “friend suggestion”)
  - Log analysis (e.g., Network traces, Smart-meter data, ..)
- Hadoop/MapReduce is not the panacea for all Big Data problems
- Hadoop/MapReduce does not feet well
  - Iterative problems
  - Recursive problems
  - Stream data processing
  - Real-time processing

Map reduce example

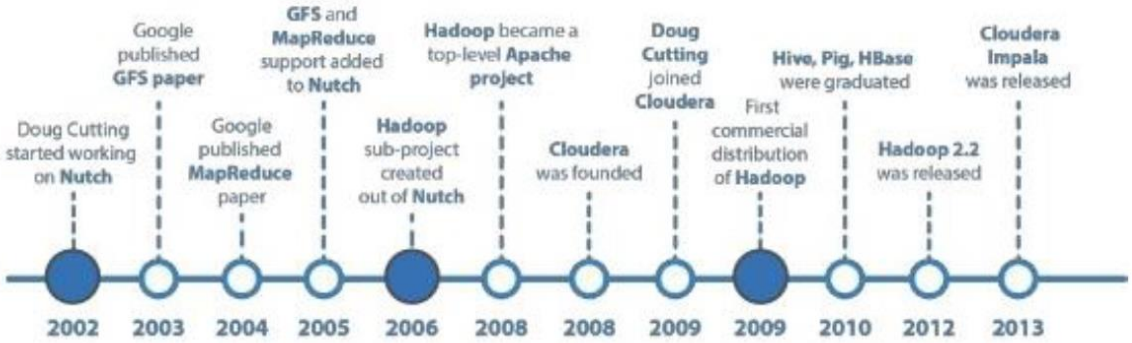


HDFS


Temporary files
(local file system)
Input Data
(HDFS file system)
Output Data
(HDFS file system)
local
write
read
Split 0
mapper
local
write
read
Split 1
mapper
local
write
read
Split 2
mapper
local
read
reducer
write
Output
File 0
send data
on the
network

Hadoop benefits

- 2008年4月，Hadoop打破世界紀錄，成為最快排序1TB資料的系統，它採用一個由910個節點構成的集群進行運算，排序時間只用了209秒
- 在2009年5月，Hadoop更是把1TB資料排序時間縮短到62秒。Hadoop從此名聲大震，迅速發展成為巨量資料時代最具影響力的開源分散式開發平臺，並成為巨量資料處理標準



Hadoop benefits

- Hadoop是一個能夠對大量資料進行分散式處理的軟體框架，它具有以下幾個方面的特性：
  - 高效能
    - 採用分散式儲存與分散式處理兩大核心技術，具有處理PB等級資料的能力。
  - 成本低
    - 可以用在由一般PC所架設的集群環境內。
  - 高可靠性
    - 採用冗餘資料儲存，當一個副本發生故障，可由其它副本資料正常對外服務。
  - 高容錯性
    - 採用冗餘資料儲存，當某個節點發生錯誤，系統能即時自動的取得副本資料以及重新佈署運算資源與任務。
  - 運行在Linux平台
  - 支援多種程式設計語言
  - 高可擴展性(Scale Out擴充能力)

Task: Word Count

Case 1:

- File too large for memory, but all <word, count> pairs fit in memory

Case 2:

- Count occurrences of words:
  - **`words(doc.txt) | sort | uniq -c`**
    - where **`words`** takes a file and outputs the words in it, one per a line
- Case 2 captures the essence of **MapReduce**
  - Great thing is that it is naturally parallelizable

Word count using map reduce pseudocode

Input file: a textual document with one word per line
The map function is invoked over each word of the input file

```
map(key, value):
    // key: offset of the word in the file
    // value: a word of the input document
    emit(value, 1)
```

```
reduce(key, values):
    // key: a word; values: a list of integers
    occurrences = 0
    for each c in values:
            occurrences = occurrences + c

    emit(key, occurrences)
```

Mapper.py

```
import sys
import re

for line in sys.stdin:
    words = re.split("\W+",line)
    for word in words:
        word = word.lower()
        if word != '':
            print("%s\t%d"%(word,1))
```

Reducer.py

```
import sys

cur_word = ""
cur_count = 0
for line in sys.stdin:
    word, count = line.strip().split('\t')
    count = int(count)
    if word == cur_word:
        cur_count += count
    else:
        if cur_word != "":
            print("%s\t%d"%(cur_word,cur_count))
        cur_word = word
        cur_count = count

print("%s\t%d"%(cur_word,cur_count))
```

Test mapreduce with script

- 修改權限
  - chmod +x ./mapper.py
  - chmod +x ./reducer.py
- 執行
  - echo "foo foo quux labs foo bar quux" | python3 mapper.py
  - echo "foo foo quux labs foo bar quux" | python3 mapper.py | sort -k1,1
  - echo "foo foo quux labs foo bar quux" | python3 mapper.py | sort -k1,1 | python3 reducer.py

```
foo     1
foo     1
quux    1
labs    1
foo     1
bar     1
quux    1
```

```
bar     1
foo     3
labs    1
quux    2
```

lamda

Python提供了一個簡易的function define：lambda，用完即丟

```
def func(x, y, z):
return x + y + z
>>> func(1, 2, 3)
>>> 6

func2 = lambda x,y,z : x+y+z
>>> func2(1, 2, 3)
>>> 6
```

filter

以傳入的boolean function作為條件函式，iterate所有的sequence的元素並收集 function(元素) 為True的元素到一個List。

```
>>> def fn(x):
    return x if x > 5 else None

>>> a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> b = filter(fn, a)
>>> b [6, 7, 8, 9]
```

Map

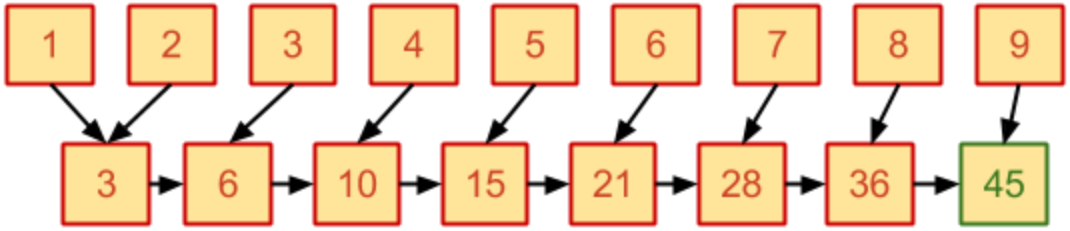
Format: map(function, sequence)
iterate所有的sequence的元素並將傳入的function作用於元素，最後以List作為回傳值。

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
def fn(x):
return x*2
c = map(fn, a)

>>> c [2, 4, 6, 8, 10, 12, 14, 16, 18]
```

reduce

Format: reduce(function, sequence)
必須傳入一個binary function(具有兩個參數的函式)，最後僅會回傳單一值。
reduce會依序先取出兩個元素，套入function作用後的回傳值再與List中的下一個元素一同作為參數，以此類推，直到List所有元素都被取完。

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
def fn(x, y):
return x+y
d = reduce(fn, a)
>>>d
45
```



yield

- **yield** allows you to make generators with ease
- The **yield** statement resembles **return** in many ways
- When **yield** is called, the value is outputted and the function is halted until next value is requested.
- **return** in a generator will raise a **StopIteration** exception

```
>>> def mygenerator():
...     yield 1
...     print("Hello, World!")
...     yield 2
...     return 3
...
>>> for i in mygenerator(): print(i)
...
1
Hello, World!
2
```

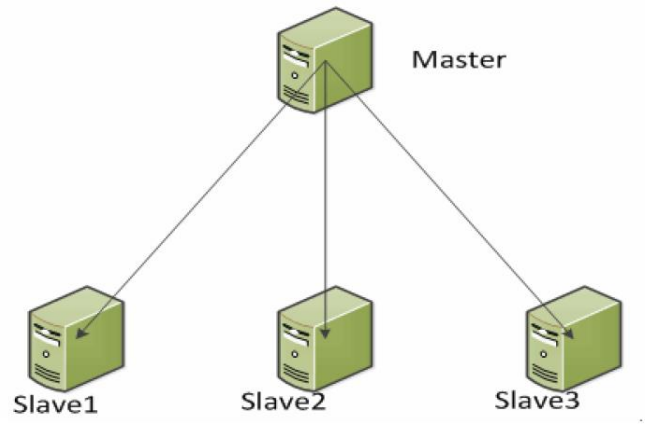
Another mapper and reducer

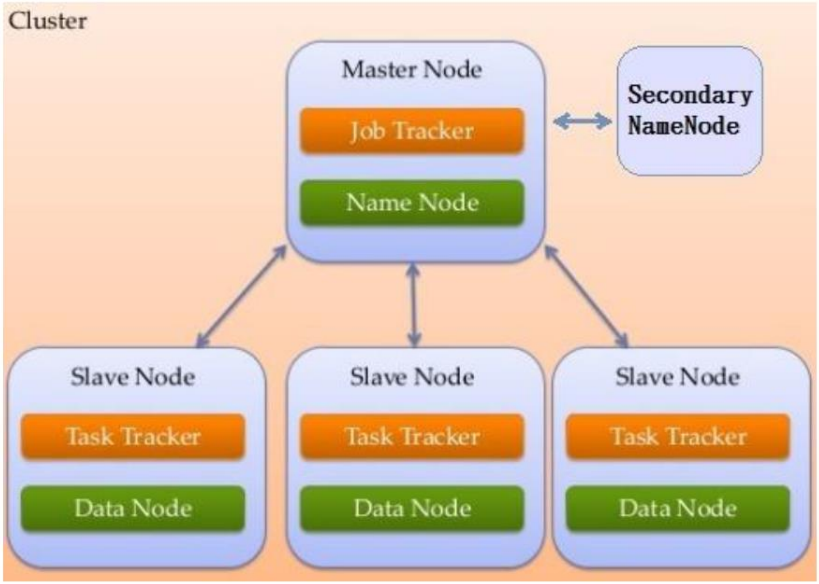
```
#!/usr/bin/python
import sys
word2count = {}
for line in sys.stdin:
    line = line.strip()
    words = filter(lambda word:word,line.split())
    for word in words:
        print("%s\t%s" % (word,1))
```

```
#!/usr/bin/python
from operator import itemgetter
import sys

word2count = {}
for line in sys.stdin:
    line = line.strip()
    word,count = line.split()
    try:
        count = int(count)
        word2count[word] = word2count.get(word,0) + count
    except ValueError as err:
        print(err)
        pass

sorted_word2count = sorted(word2count.items(),key=itemgetter(0))
for word,count in sorted_word2count:
    print("%s\t%s" % (word, count))
```

Hadoop architecture

- Hadoop的運作概念是基於Master/Slave架構。
- Hadoop的分散式檔案系統HDFS在Master/Slave皆有行程執行相關任務：
  - 運行在master上的：NameNode、Secondary NameNode
  - 運行在slave上的：DataNode
- Hadoop的分散式計算框架MapReduce在Master/Slave皆有行程執行相關任務：
  - 運行在master上的：JobTracker
  - 運行在slave上的：TaskTracker





Hadoop architecture

- HDFS架構的節點類型可分成：
  - NameNode
    - 又稱Master Node
    - 負責記錄與維護HDFS的Metadata
      - Metadata有：檔案名稱，檔案與其區塊(Blocks)的對映關係，各區塊所在的資料節點(DataNode)，其它的檔案屬性(如：建立時間，複本數量…等)
    - 僅有一個NameNode，故NameNode掛掉，會使整個HDFS無法正常存取，致使所有的工作失敗(SPOF – Single Point of Failure)。
  - DataNode
    - 又稱Slave Node
    - 檔案區塊(Blocks)實際儲存的地方，通常有多部DataNode。
    - DataNode會定期傳送現有的Blocks狀態給NameNode。
    - 若NameNode發現某個Block之複本數量少於現有的備份設定時，NameNode會自動增加該Block的複本。
    - 當某個DataNode掛掉時，NameNode會自動將此DataNode上的所有Blocks重新配置到其它DataNode上
  - SecondaryNameNode
    - 會定期與NameNode進行通訊與監控，以作為NameNode的冷備份

Hadoop architecture

- MapReduce架構的節點類型可分成：
  - JobTracker (工作分派者)
    - Master Node
    - 負責接收由Client端傳來的工作
    - 負責將執行該工作所需執行的各項作業(即：Map與Reduce)分配給TaskTracker
    - Single Point of Failure
  - TaskTracker (作業執行者)
    - Worker Nodes
    - 執行Map與Reduce的作業
    - 儲存結果與回報作業狀態
    - 有多個TaskTracker，可分成Mapper與Reducer:
      - Input資料被切割成幾份，就會有幾個Mapper，決定權由Hadoop控制。
      - Reducer數量的決定權由使用者控制。

Hadoop third party

廠商	開放性	易用性(★)	平台功能	性能(★)	總體評價(★)
Apache	完全開放，Hadoop就是託管在Apache社區裡面	安裝：2 使用：2 維護：2	Apache是標準的Hadoop平台，所有廠商都是在Apache平台上進行改善	2	2
Cloudera CDH	與Apache功能同步，部份程式碼開放	安裝：5 使用：5 維護：5	有自主研發的產品，如：Impala、Navigator等	4.5	4.5
Hortonworks HDP	與Apache功能同步，程式碼完全開放	安裝：4.5 使用：5 維護：5	是Apache Hadoop平台最大貢獻者，如：Tez	4.5	4.5
MapR	對Apache Hadoop版本有較多的修改，偏向私有	安裝：4.5 使用：5 維護：5	在Apache平台上最佳化很多、從而形成自己的產品	5	3.5

Hadoop problem

Apache Hadoop問題	第三方版本(CDH, HDP, MapR)優勢
■ 版本複雜，版本管理比較混亂，讓使用者不知所措。 ■ 集群部署、安裝、配置複雜。通常需要編寫大量的配置文件，容易出錯，效率低下。 ■ 對集群的監控與維運，需要安裝第三方的其他軟體，運維難度較大。 ■ 生態環境複雜，組件的選擇、使用，比如Hive，Mahout，Sqoop，Flume，Spark，Oozie等等，需要大量考慮兼容性的問題，版本是否兼容，元件是否有衝突，編譯是否能通過等。	■ 版本管理清晰。 ■ 提供了部署、安裝、配置工具，大大提高了集群部署的效率。 ■ 維運簡單。提供了管理、監控、診斷、配置修改的工具，管理配置方便，定位問題快速、準確，使運維工作簡單，有效。 ■ 比Apache Hadoop在兼容性、安全性、穩定性上有增強。第三方發行版通常都經過了大量的測試驗證與眾多部署實例。 ■ 基於穩定版本Apache Hadoop，並應用了最新Bug修復或Feature的patch

- **建議**
  - 若是為建構實際營運的巨量資料處理環境，以第三方版本為主要選擇對象 (佈署、維運方便，穩定性高)。
  - 若是學習為目的，以Apache Hadoop為主 (易掌握各元件之佈署維運，對Troubleshooting之訓練有一定助益)

Hadoop works

- Hadoop安裝暨運作方式：
  - 單機模式 (Standalone Mode)：為Hadoop的預設模式，不需要進行任何的設定。在此模式下，Hadoop的執行完全在一部機器上，資料儲存是採用本地OS的檔案系統，而不採行分散式檔案系統HDFS。同時，Hadoop上的所有服務如：NameNode、DataNode、TaskTracker…等都變成一個Java程序。
  - 虛擬分散模式 (Pseudo-Distributed Mode)：在此模式下，Hadoop的執行完全在一部機器上，資料儲存是採用分散式檔案系統HDFS。同時，Hadoop上的所有服務如：NameNode、DataNode、TaskTracker、JobTracker…等都執行在同一部機器，以模擬一個具有Hadoop完整功能的小型集群。
  - 完全分散模式 (Fully-Distributed Mode)：在此模式下，Hadoop的執行在多部機器上，資料儲存採用分散式檔案系統HDFS。同時，Hadoop上的所有服務如：NameNode、DataNode、TaskTracker、JobTracker…等都分散執行在不同部機器，形成具有Hadoop完整功能的真正集群。

Hadoop installation

- Hadoop基本安裝配置主要包括以下幾個步驟：
  - 在Ubuntu系統建立Hadoop用戶 (如果需要)
  - SSH登入許可權設置
  - 安裝Java環境
  - 單機安裝配置 / 虛擬分散式安裝配置

SSH login

SSH是什麼？

SSH 為 Secure Shell 的縮寫，是建立在應用層和傳輸層基礎上的安全協議，利用SSH協定可以有效防止遠端系統管理過程中的資訊洩露問題。此外，SSH也是一個壓縮協定，可以節省頻寬、加速傳輸資料速度等。但是，Ubuntu預設是沒有安裝SSH Server，所以需要先行安裝。

配置SSH為無密碼登入的原因：

Hadoop的NameNode需要連線並登入同集群中其它的DataNode。然而，若每次登入其它機器都要輸入密碼也是非常麻煩，若能設定為SSH無密碼登入，則Hadoop的NameNode不需人工輸入密碼即可連線登入至同集群中其它的DataNode會比較方便。無密碼登入不代表登入其它機器不需要輸入密碼，而是產生一個密碼包(金鑰)，並加入到一個授權檔案中，當要登入時系統直接讀取金鑰，而不需要人工輸入密碼。

Hadoop in EC2 (single node)

- **新增 hadoop 使用者**
  - sudo useradd -m hadoop -s /bin/bash　# 增加 hadoop使用者
  - sudo passwd hadoop　# 設定密碼，需要輸入兩次
  - sudo adduser hadoop sudo　# 為 hadoop 使用者增加管理員許可權
  - su hadoop　# 切換 hadoop 使用者，需要輸入密碼
  - sudo apt-get update –y　# 更新 apt 源
  - sudo apt-get upgrade –y　# 升級 apt 源

Install anaconda in EC2

- **Step 1**– Download the version of Anaconda from the below link.
  - **sudo wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh (利用ls -al看權限)**
  - **Step 2- change mod : sudo chmod +x Miniconda3-latest-Linux-x86_64.sh**
- **Step 3 -After download run the downloaded script : sudo ./Miniconda3-latest-Linux-x86_64.sh**
- You will be promoted for the following options:
  - In order to continue the installation process, please review the license agreement.
    Please, press ENTER to continue ———> Press **Enter**
  - Do you accept the license terms? [yes|no] ———> Type **Yes**
  - Anaconda3 will now be installed into this location:
  - [/home/hadoop/anaconda3] >>> ———> Press **Enter**
  - Do you wish the installer to initialize Anaconda3 by running conda init? [yes|no]
    [no] >>> ————> Type **Yes**
- **Step 4** – After installation, use the following command to activate the installation
  - **$ source ~/.bashrc**

Hadoop in EC2 (single node)

- **Java套件 (Java環境可選擇Oracle的JDK，或是OpenJDK)**
  - sudo apt-get install openjdk-8-jdk

  or
  - sudo add-apt-repository ppa:webupd8team/java –y
  - sudo apt-get update
  - sudo apt-get install oracle-java8-installer
  - sudo apt-get install oracle-java8-set-default
- **hadoop套件**
  - sudo wget https://archive.apache.org/dist/hadoop/core/hadoop-2.7.3/hadoop-2.7.3.tar.gz
  - sudo tar zxvf hadoop-* -C /usr/local
  - sudo mv /usr/local/hadoop-* /usr/local/hadoop

Hadoop in EC2 (single node)

- **新增啟動路徑**
  - sudo vi ~/.bashrc　　# 開啟文字編輯器
  - # Insert the following lines in bashrc file
  - export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64 (如果使用openjdk)
  - export JAVA_HOME=/usr/lib/jvm/java-8-oracle (如果使用java-8-oracle)
  - export PATH=$PATH:$JAVA_HOME/bin
  - export HADOOP_HOME=/usr/local/hadoop
  - export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
  - export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
  - export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
  - export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
- **執行啟動路徑**
  - source ~/.bashrc
- 測試環境
  - java –version
  - hadoop version

SSH setting

- Setting hadoop environment for password less ssh access.
  - ssh-keygen -t rsa -P ''
  - cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
- Modify the config file
  - sudo vim /etc/ssh/sshd_config
- Find the following line and edit accordingly.
  - PasswordAuthentication yes
- Save the config file. Then restart the ssh service for the update to take action.
  - sudo service ssh restart
- check password less ssh access to localhost
  - ssh localhost
  - exit

Hadoop configuration

- Hadoop可以在單一節點上以虛擬分散式的方式運行，Hadoop行程以分離的Java行程來運行，節點既作為NameNode也作為DataNode，同時，讀取的是HDFS中的文件
- Hadoop的設定檔位於/usr/local/hadoop/etc/hadoop/中
- Copy and paste the below configurations in **hadoop-env.sh**
  - **sudo vi** /usr/local/hadoop/etc/hadoop/**hadoop-env.sh**
  - export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

Hadoop in EC2

- 設定環境
  - **sudo vi /usr/local/hadoop/etc/hadoop/core-site.xml**

```
<configuration>
<property>
<name>hadoop.tmp.dir</name>
<value>/usr/local/hadoop/tmp</value>
<description>Abase for other temporary directories.</description>
</property>
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost:9000</value>
</property>
</configuration>
```

hadoop.tmp.dir表示存放臨時資料的目錄。
fs.defaultFS表示hdfs的登入位址及其port號之**邏輯名稱**

Hadoop in EC2

- 設定環境
  - **sudo vi /usr/local/hadoop/etc/hadoop/hdfs-site.xml**

```
<configuration>
<property>
<name>dfs.name.dir</name>
<value>/home/hadoop/hadoopdata/hdfs/namenode
</value>
</property>
<property>
<name>dfs.data.dir</name>
<value>/home/hadoop/hadoopdata/hdfs/datanode
</value>
</property>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
</configuration>
```

dfs.replication表示**副本的數量**，虛擬分散式要設置為**1**

dfs.namenode.name.dir表示NameNode的本地硬碟目錄，是儲存Metadata的地方

dfs.datanode.data.dir表示DataNode的本地硬碟目錄，HDFS資料存放block的地方

Hadoop in EC2

- 設定環境
  - **sudo vi /usr/local/hadoop/etc/hadoop/mapred-site.xml**

```
<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
</configuration>
```

Hadoop in EC2 (optimal)

- 設定環境
  - **sudo vi /usr/local/hadoop/etc/hadoop/yarn-site.xml**

```
<configuration>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
</configuration>
```

Hadoop starting

- Formatting the HDFS file system via NameNode (after installing hadoop, for the first time we have to format the HDFS file system to make it work)
  - hdfs namenode –format
- Issue the following commands to start Hadoop
  - start-dfs.sh
  - start-yarn.sh

  or
  - start-all.sh
- Check for hadoop processes /daemons running on hadoop with Java Virtual Machine Process Status Tool.
  - jps
- Issue the following commands to stop Hadoop
  - stop-dfs.sh
  - stop-yarn.sh

  or
  - stop-all.sh

HDFS

- The content of a HDFS file can be accessed by means of
  - Command line commands
  - A basic web interface provided by Apache Hadoop
    - The HDFS content can only be browsed and its files downloaded from HDFS to the local file system
- The hdfs command can be executed in a Linux shell to read/write/modify/delete the content of the distributed file system
- The parameters/arguments of hdfs command are used to specify the operation to execute

HDFS command

- Example
  - hdfs dfs -ls .
- shows the content of the home of the current user
  - i.e., the content of /user/*current_username*
  - **.** = user home

HDFS command

- Show the content of a file of the HDFS file system
  - hdfs dfs -cat *file*
- Example
  - hdfs dfs -cat /user/garza/document.txt
- Shows the content of the /user/garza/document.txt file stored in HDFS

HDFS command

- Copy a file from the local file system to the HDFS file system
  - hdfs dfs -put *local_file HDFS_path*
  - hdfs dfs –copyFromLocal *local_file HDFS_path*
- Example
  - hdfs dfs -put /data/document.txt /user/garza/
- Copy the local file /data/document.txt in the folder /user/garza of HDFS

HDFS command

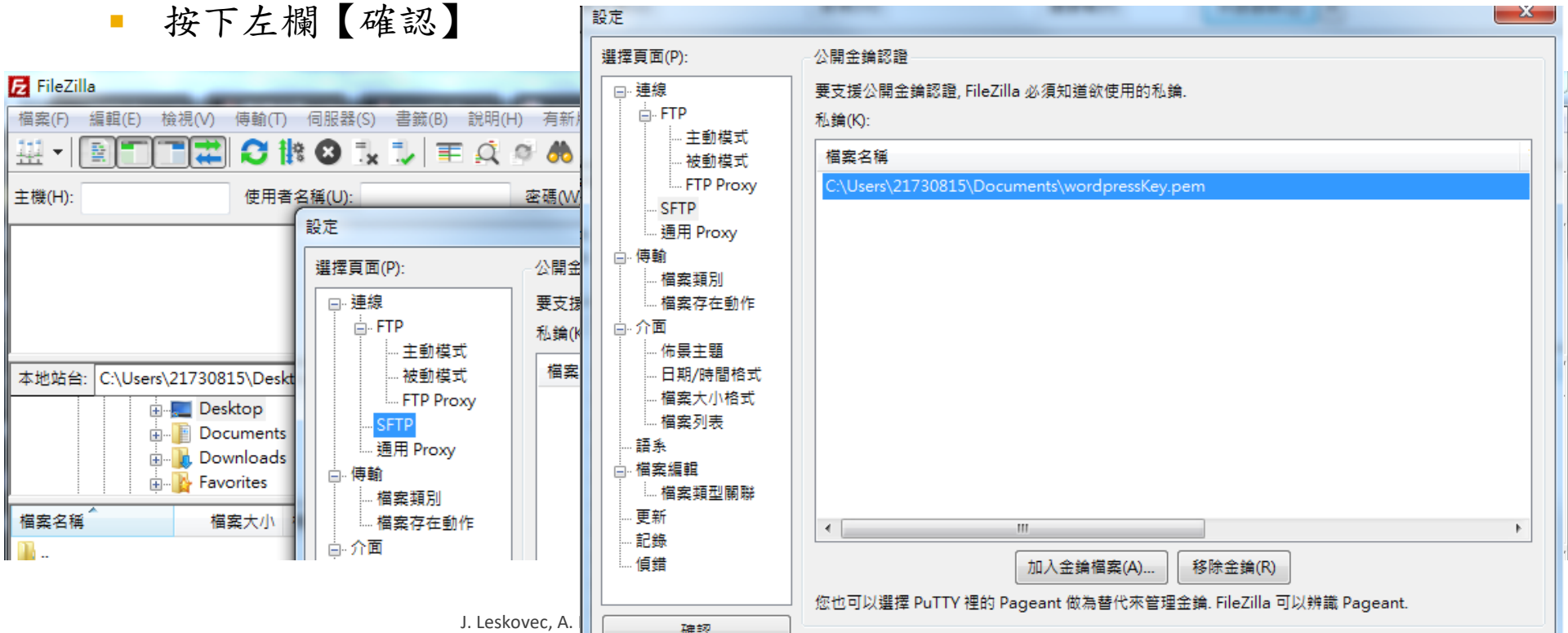
- Copy a file from the HDFS file system to the local file system
  - hdfs dfs -get *HDFS_path local_file*
- Example
  - hdfs dfs -get /user/garza/document.txt /data/
- Copy the HDFS file /user/garza/document.txt in the local file system folder /data/

HDFS command

- Delete a file from the HDFS file system
  - hdfs dfs -rm *HDFS_path*
- Example
  - hdfs dfs -rm /user/garza/document.txt
- Delete from HDFS the file /user/garza/document.txt

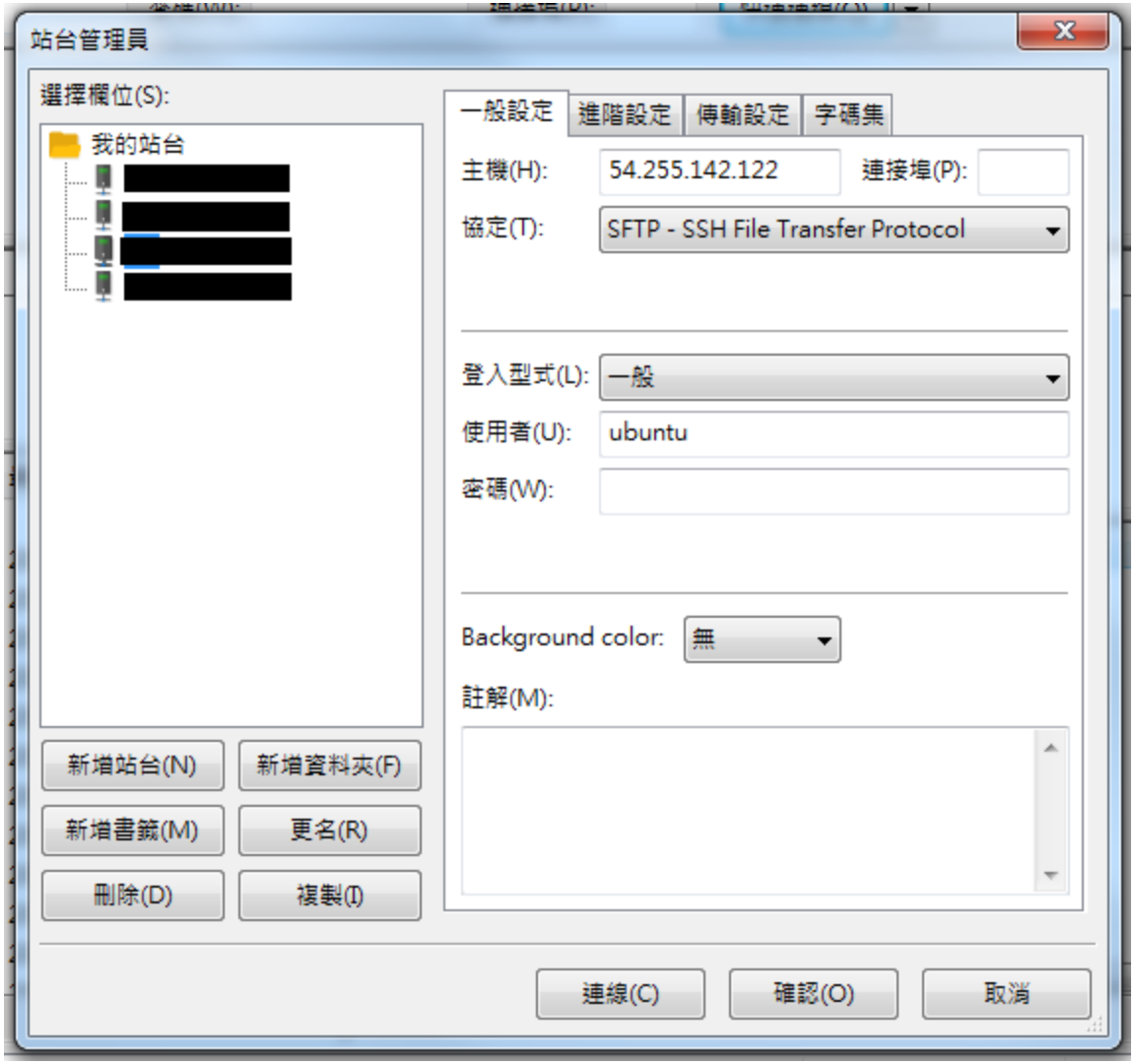
Install FileZilla in your NB or PC

- 透過 FileZilla 使用 key-pairs 登入 AWS EC2 存取檔案
  - 軟體(官方下載連結): FileZilla Client
- 開啟FileZilla，點選選單【編輯】->【設定】
- 在彈出視窗左欄選取【SFTP】
- 點選【加入金鑰檔案】，選取您的.pem金鑰檔案。
  - 注意! .pem路徑必須**不含中文字**，不然會產生連線錯誤/無法連線的情況。
- 按下左欄【確認】



Install FileZilla in your NB or PC

- 新增站台
- 站台設定值如下:
  - 主機 – 填入EC2 public IP
  - 協定 – 選擇SFTP
  - 登入型式 – 選擇一般
  - 使用者 – 若你申請的是Linux機器的話，那麼這邊請填【ubuntu】
  - 密碼 – 維持空白即可，因為是使用.pem金鑰認證登入。



Hadoop mapreduce example

- hdfs dfs -mkdir /input
- hdfs dfs -copyFromLocal /home/ubuntu/shakespeare.txt /input
- hdfs dfs –ls /input
- hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -files mapper.py,reducer.py -input /input/shakespeare.txt -output /output -mapper "python3 mapper.py" -reducer "python3 reducer.py“
- hdfs dfs –cat /output/part-00000
- hdfs dfs -copyToLocal /output/part-00000 .

mrjob

- mrjob is a Python MapReduce library, created by Yelp, that wraps Hadoop streaming, allowing MapReduce applications to be written in a more Pythonic manner
- MapReduce jobs written with mrjob can be tested locally, run on a Hadoop cluster, or run in the cloud using Amazon Elastic MapReduce (EMR).
  - pip install mrjob

Mrjob example

python3 WordFrequency.py ./shakespeare.txt

```
from mrjob.job import MRJob

class MRWordCount(MRJob):

    def mapper(self, _, line):
        for word in line.split():
            yield(word, 1)

    def reducer(self, word, counts):
        yield(word, sum(counts))

if __name__ == '__main__':
    MRWordCount.run()
```

Configuration in multi-node

- 確認每台主機（master 跟其它 slave）名稱，是否分別為「master、slave01、slave02」：
  - sudo vi /etc/hostname
- 修改每台主機的 hosts 檔案：
  - sudo vi /etc/hosts
- 修改完成後，全部重新啟動，才能在終端機看到修正後的變化。
- 試著 ping 每一台，確認相互連通
  - Ping –c 4 slave01

```
127.0.0.1       localhost
192.168.56.100 master
192.168.56.101 slave01
192.168.56.102 slave02
```

Private IP

/etc/hosts

```
hadoop@master:~/.ssh$ ping -c 4 slave01
PING slave01 (192.168.56.101) 56(84) bytes of data.
64 bytes from slave01 (192.168.56.101): icmp_seq=1 ttl=64 time=0.515 ms
64 bytes from slave01 (192.168.56.101): icmp_seq=2 ttl=64 time=0.674 ms
64 bytes from slave01 (192.168.56.101): icmp_seq=3 ttl=64 time=0.637 ms
64 bytes from slave01 (192.168.56.101): icmp_seq=4 ttl=64 time=0.675 ms

--- slave01 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
rtt min/avg/max/mdev = 0.515/0.625/0.675/0.067 ms
hadoop@master:~/.ssh$
```

SSH configuration in multi-node

- Add the following to the .ssh/config file of namenode instance
- Copy the public key (id_rsa.pub) to the ~/.ssh/authorized_keys file of datanode instances and then try to access the datanodes from namenode instance using ssh.
  - scp ~/.ssh/id_rsa.pub hadoop@slave01:/home/hadoop/
  - scp ~/.ssh/id_rsa.pub hadoop@slave02:/home/hadoop/

~/.ssh/authorized_keys

```
Host namenode
 HostName <public DNS of namenode instance>
 User ubuntu
Host datanode1
 HostName <public DNS of datanode1 instance>
 User ubuntu
Host datanode2
 HostName <public DNS of datanode2 instance>
 User ubuntu
```

Configuration in multi-node

- 我們需要修改 /usr/local/hadoop/etc/hadoop 的五個配置文件：slaves、core-site.xml、hdfs-site.xml、mapred-site.xml、yarn-site.xml
- 進入配置文件資料夾
  - cd /usr/local/hadoop/etc/Hadoop
- 在slave文件中把原先裡面的「localhost」刪除，改成
  slave01
  slave02
  slave03

Hadoop in EC2 (multi-nodes)

- 設定環境
  - **sudo vi /usr/local/hadoop/etc/hadoop/core-site.xml**

```
<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://master:9000</value>
</property>
<property>
<name>hadoop.tmp.dir</name>
<value>file:/usr/local/hadoop/tmp</value>
<description>Abase for other temporary directories.</description>
</property>
</configuration>
```

hadoop.tmp.dir表示存放臨時資料的目錄。
fs.defaultFS表示hdfs的登入位址及其port號之**邏輯名稱**

Hadoop in EC2 (multi-nodes)

- 設定環境
  - **sudo vi /usr/local/hadoop/etc/hadoop/hdfs-site.xml**

```
<configuration>
<property>
<name>dfs.namenode.secondary.http-address</name>
<value>master:50090</value>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>file:/usr/local/hadoop/tmp/dfs/name</value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>file:/usr/local/hadoop/tmp/dfs/data</value>
</property>
<property>
<name>dfs.replication</name>
<value>3</value>
</property>
</configuration>
```

dfs.replication表示**副本的數量**，虛擬分散式要設置為**1**

dfs.namenode.name.dir表示NameNode的本地硬碟目錄，是儲存Metadata的地方

dfs.datanode.data.dir表示DataNode的本地硬碟目錄，HDFS資料存放block的地方

Hadoop in EC2 (multi-nodes)

- 設定環境
  - **sudo vi /usr/local/hadoop/etc/hadoop/mapred-site.xml**

```
<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
</configuration>
```

Hadoop in EC2 (multi-nodes)

- 設定環境
  - **sudo vi /usr/local/hadoop/etc/hadoop/yarn-site.xml**

```
<configuration>
<property>
<name>yarn.resourcemanager.hostname</name>
<value>master</value>
</property>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
</configuration>
```

Hadoop in EC2 (multi-nodes)

- 我們要將 master 上面的 hadoop 資料夾打包，傳輸給所有 slave 主機：
  - cd /usr/local
  - rm -r ./hadoop/tmp
  - sudo tar -zcf ./hadoop.tar.gz ./hadoop
  - scp ./hadoop.tar.gz slave01:/home/Hadoop
  - scp ./hadoop.tar.gz slave02:/home/Hadoop
- 我們分別到 slave01、slave02主機裡，進行操作：
  - sudo tar -zxf ~/hadoop.tar.gz -C /usr/local
  - sudo chown -R hadoop:hadoop /usr/local/hadoop

Practice (mapper for count and sum)

- Data format of each line is: date\ttime\tstore location\titem description\tcost\tmethod of payment

example.csv

```
2012-07-16	15:43	Las Vegas	Men's Clothing	208.97	Visa
2012-06-11	16:17	Miami	Crafts	84.11	Amex
2012-10-17	15:30	Tucson	Crafts	489.93	Cash
2012-10-25	15:01	San Francisco	Men's Clothing	388.3	Visa
2012-07-13	09:01	Dallas	Consumer Electronics	145.63	Cash
2012-11-06	13:02	Tampa	Garden	353.23	MasterCard
2012-09-07	12:58	Washington	Women's Clothing	481.31	MasterCard
2012-08-05	16:34	San Jose	DVDs	492.8	Discover
2012-04-22	13:12	Newark	Consumer Electronics	410.37	Visa
2012-10-19	11:35	Memphis	Garden	354.44	Discover
2012-10-10	13:17	Jersey City	Books	369.07	Amex
2012-04-27	11:54	Plano	Women's Clothing	4.65	Cash
2012-08-28	14:56	Buffalo	Video Games	337.35	Discover
2012-09-17	13:09	Louisville	Music	213.64	Discover
2012-02-24	12:05	Miami	Women's Clothing	154.64	Cash
2012-01-02	10:04	Los Angeles	Pet Supplies	164.5	Discover
2012-11-15	15:46	Birmingham	Men's Clothing	1.64	Cash
2012-03-16	11:18	Mesa	Toys	13.79	Visa
2012-06-25	10:05	Wichita	Consumer Electronics	158.25	Amex
2012-04-05	17:03	Indianapolis	Pet Supplies	152.77	Amex
2012-11-08	15:19	San Bernardino	Video Games	332.43	Discover
2012-08-08	10:09	Indianapolis	Health and Beauty	464.36	Amex
2012-03-02	09:25	Stockton	Men's Clothing	180.61	Discover
2012-02-27	16:12	Austin	Health and Beauty	48.09	Visa
2012-12-29	16:56	Buffalo	Garden	386.56	Amex
2012-03-20	09:02	Santa Ana	Books	2.75	Amex
2012-10-30	11:52	Gilbert	DVDs	11.31	Amex
2012-02-03	11:02	New York	DVDs	221.35	Visa
2012-07-26	16:16	Corpus Christi	Health and Beauty	157.91	Amex
2012-07-20	11:46	Riverside	Video Games	349.41	Visa
2012-10-04	12:25	Chicago	Children's Clothing	364.53	MasterCard
2012-02-04	11:53	Fremont	Video Games	404.17	Cash
2012-05-31	14:43	Rochester	Video Games	460.39	Amex
2012-05-25	16:11	Raleigh	Computers	61.22	MasterCard
```

```
import string
import fileinput

for line in fileinput.input():
    data = line.strip().split("\t")
    if len(data) == 6:
        date, time, location, item, cost, payment = data
        print("{0}\t{1}".format(location, cost))
```

Practice (reducer for count and sum)

- Format of each line is: location\tcost

```
import fileinput
transactions_count = 0
sales_total = 0

for line in fileinput.input():
    data = line.strip().split("\t")
    if len(data) != 2:
        # Something has gone wrong. Skip this line.
        continue
    current_key, current_value = data
    transactions_count += 1
    sales_total += float(current_value)
print(transactions_count, "\t", sales_total)
```

Practice (reducer for max)

- Format of each line is: location\tcost

```
import fileinput
max_value = 0
old_key = None
for line in fileinput.input():
    data = line.strip().split("\t")
    if len(data) != 2:
        # Something has gone wrong. Skip this line.
        continue
     current_key, current_value = data
    # Refresh for new keys (i.e. locations in the example context)
    if old_key and old_key != current_key:
        print(old_key, "\t", max_value)
        old_key = current_key
        max_value = 0
    old_key = current_key
    if float(current_value) > float(max_value):
        max_value = float(current_value)
if old_key != None:
  print(old_key, "\t", max_value)
```

HW1 deadline:4/30

- 請利用python程式語言撰寫map reduce的程式來計算每個地點(location)的cost平均數，處理資料請用之前的範例example.txt，請在AWS EC2環境來實作

- 請利用python程式語言撰寫map reduce的程式來計算計算每個地點(location)的cost最小值，處理資料請用之前的範例example.txt，請在AWS EC2環境來實作

- 作業繳交方式:
  - 請將程式碼內容貼在word上
  - 請將程式跑的過程擷取螢幕畫面一個貼在word上

Practice (mrjob for average)

Fakefriends.csv

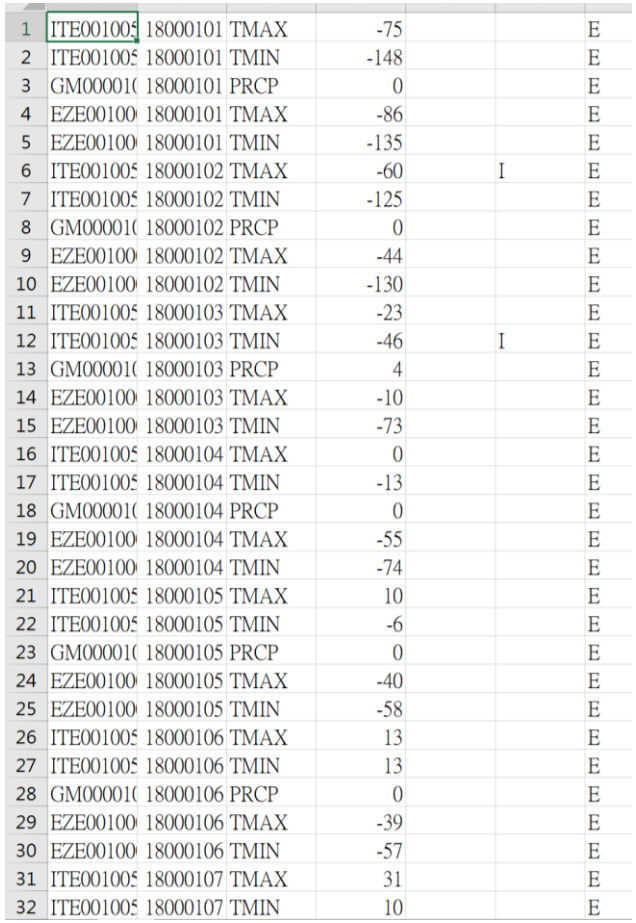
1	0	Will	33	385
2	1	Jean-Luc	26	2
3	2	Hugh	55	221
4	3	Deanna	40	465
5	4	Quark	68	21
6	5	Weyoun	59	318
7	6	Gowron	37	220
8	7	Will	54	307
9	8	Jadzia	38	380
10	9	Hugh	27	181
11	10	Odo	53	191
12	11	Ben	57	372
13	12	Keiko	54	253
14	13	Jean-Luc	56	444
15	14	Hugh	43	49
16	15	Rom	36	49
17	16	Weyoun	22	323
18	17	Odo	35	13
19	18	Jean-Luc	45	455
20	19	Geordi	60	246
21	20	Odo	67	220
22	21	Miles	19	268
23	22	Quark	30	72
24	23	Keiko	51	271
25	24	Julian	25	1
26	25	Ben	21	445
27	26	Julian	22	100
28	27	Leeta	42	363
29	28	Martok	49	476
30	29	Nog	48	364
31	30	Keiko	50	175
32	31	Miles	39	161

fakefriends

```
from mrjob.job import MRJob

class FriendsByAge(MRJob):

        def mapper(self, _, line):
            (ID, name, age, numFriends) = line.split(',')
            yield(age, float(numFriends) )

        def reducer(self, age, numFriends):
            total = 0.0
            n = 0
            for f in numFriends:
                total += f
                n += 1
            avg = total/n
            yield(age, avg)

if __name__ == '__main__':
        FriendsByAge().run()
```

Practice (mrjob for min)

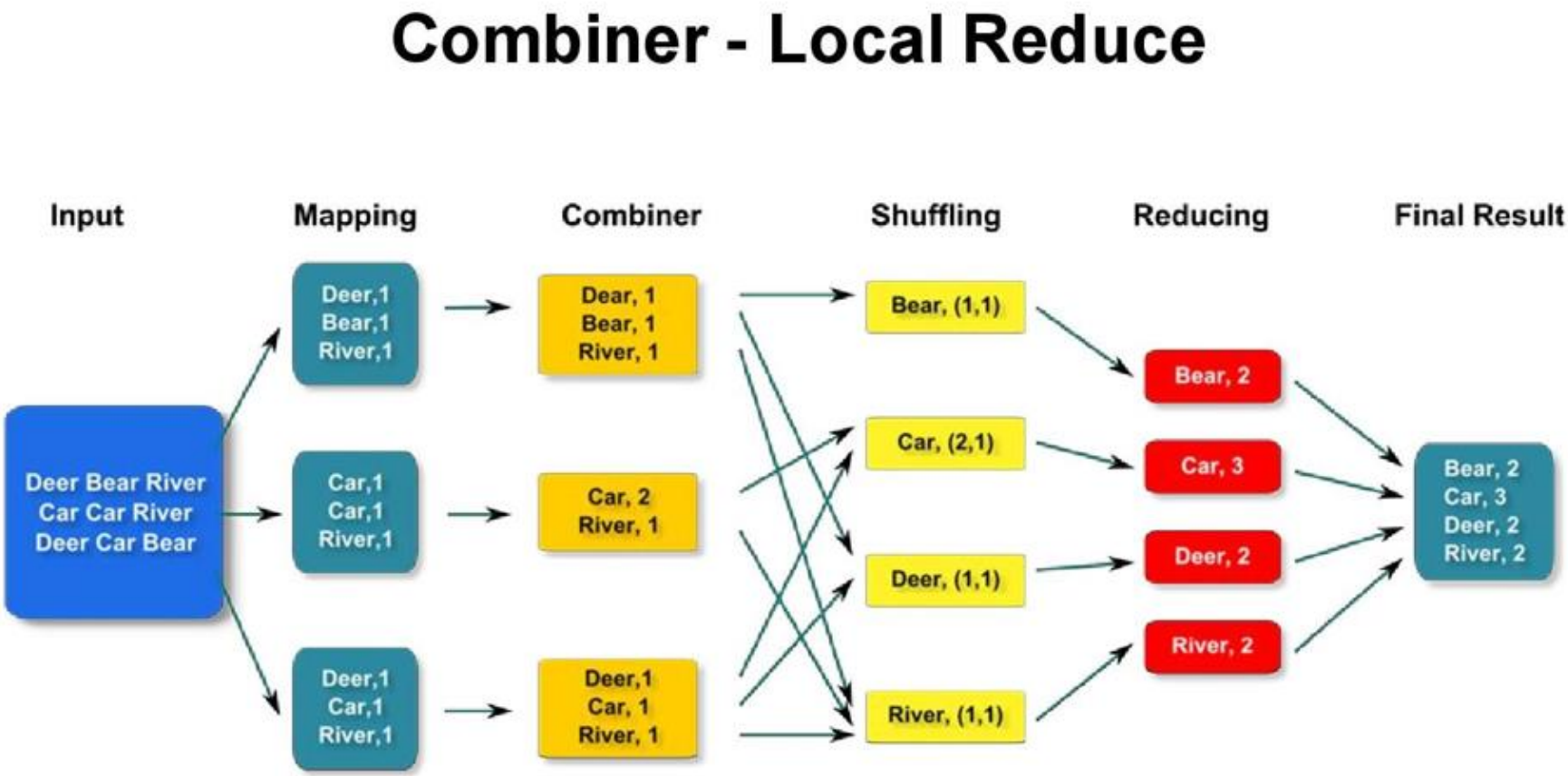
1800.csv



1	ITE001005	18000101	TMAX	-75			E
2	ITE001005	18000101	TMIN	-148			E
3	GM000010	18000101	PRCP	0			E
4	EZE00100	18000101	TMAX	-86			E
5	EZE00100	18000101	TMIN	-135			E
6	ITE001005	18000102	TMAX	-60		I	E
7	ITE001005	18000102	TMIN	-125			E
8	GM000010	18000102	PRCP	0			E
9	EZE00100	18000102	TMAX	-44			E
10	EZE00100	18000102	TMIN	-130			E
11	ITE001005	18000103	TMAX	-23			E
12	ITE001005	18000103	TMIN	-46		I	E
13	GM000010	18000103	PRCP	4			E
14	EZE00100	18000103	TMAX	-10			E
15	EZE00100	18000103	TMIN	-73			E
16	ITE001005	18000104	TMAX	0			E
17	ITE001005	18000104	TMIN	-13			E
18	GM000010	18000104	PRCP	0			E
19	EZE00100	18000104	TMAX	-55			E
20	EZE00100	18000104	TMIN	-74			E
21	ITE001005	18000105	TMAX	10			E
22	ITE001005	18000105	TMIN	-6			E
23	GM000010	18000105	PRCP	0			E
24	EZE00100	18000105	TMAX	-40			E
25	EZE00100	18000105	TMIN	-58			E
26	ITE001005	18000106	TMAX	13			E
27	ITE001005	18000106	TMIN	13			E
28	GM000010	18000106	PRCP	0			E
29	EZE00100	18000106	TMAX	-39			E
30	EZE00100	18000106	TMIN	-57			E
31	ITE001005	18000107	TMAX	31			E
32	ITE001005	18000107	TMIN	10			E

```
from mrjob.job import MRJob

class MrMinTemperature(MRJob):

        def mapper(self, key, value):
            (loc, data, type, data, x,y,z,w)= value.split(',')
             if type == 'TMIN:
                    yield(loc, data)

        def reducer(self, loc, temps):
              yield(loc, min(temps))

if __name__ == '__main__':
        MrMinTemperature().run()
```

Combiner


Combiner - Local Reduce
Input
Mapping
Combiner
Shuffling
Reducing
Final Result
Deer Bear River
Car Car River
Deer Car Bear
Deer,1
Bear,1
River,1
Car,1
Car,1
River,1
Deer,1
Car,1
River,1
Dear, 1
Bear, 1
River, 1
Car, 2
River, 1
Deer,1
Car, 1
River, 1
Bear, (1,1)
Car, (2,1)
Deer, (1,1)
River, (1,1)
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

Practice (mrjob for combiner)

```
from mrjob.job import MRJob
import re

WORD_REGEXP = re.compile(r"[\w']+")

class WordFrequencyCbn(MRJob):
        def mapper(self, key, line):
              words = WORD_REGEXP.findall(line)
              for w in words:
                    yield(w.lower(), 1)

        def combiner(self, word, occurences):
                    yield(word, sum(occurences) )

        def reducer(self, word, occurences):
                    yield(word, sum(occurences))

if __name__ == '__main__':
       WordFrequencyCbn.run()
```

Mrjob pipeline

```
from mrjob.job import MRJob
from mrjob.step import MRStep
import re WORD_RE = re.compile(r"[\w']+")

class MRMostUsedWord(MRJob):
        def steps(self):
                return[ MRStep(mapper = self.mapper_get_words,
                                        combiner = self.combiner_count_words,
                                        reducer = self.reducer_count_words),
                                MRStep(reducer = self.reducer_find_max_word) ]

        def mapper_get_words(self,_,line):
                for word in WORD_RE.findall(line):
                        yield (word.lower(),1)

        def combiner_count_words(self, word, counts):
                yield (word, sum(counts))

        def reducer_count_words(self, word, counts):
                yield None, (sum(counts), word)

        def reducer_find_max_word(self, _, word_count_pairs):
                yield max(word_count_pairs)

if __name__ == '__main__':
        MRMostUsedWord.run()
```

Mrjob run

- Mrjob run locally
  - python3 WordFrequency.py ./shakespeare.txt
- Mrjob run on Hadoop
  - hdfs dfs -mkdir /input
  - hdfs dfs -copyFromLocal ./shakespeare.txt input
  - hdfs dfs –ls input
  - python3 WordFrequency.py -r Hadoop hdfs:///user/hadoop/input -o hdfs:///output/hadoop

HW2 deadline:5/7

- 請利用python的mrjob函示庫撰寫map reduce的程式來計算每個電影被觀看的次數，處理資料請用課程網頁的範例檔案u1.base，請在AWS EC2環境來實作

- 請利用python的mrjob函示庫撰寫map reduce的程式來計算電影被觀看前二十名次數的電影，處理資料請用課程網頁的範例u1.base來，請在AWS EC2 環境來實作
  - Hint: https://github.com/tttonyl/ddd/blob/master/L03/wordcount_top10.py

- 作業繳交方式:
  - 請將程式碼內容貼在word上
  - 請將程式跑的過程擷取螢幕畫面一個貼在word上