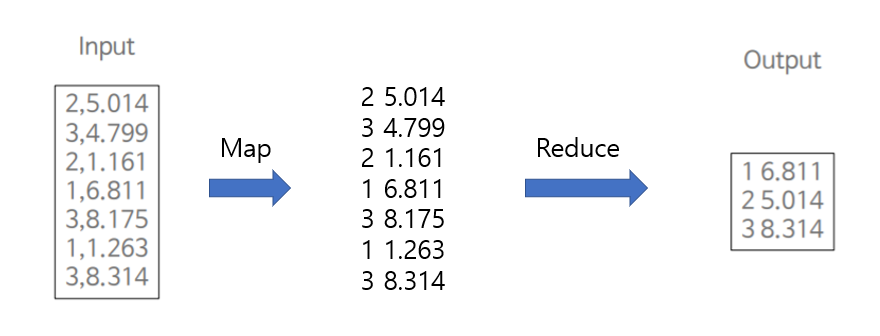
# 프로그램 개요

본 프로젝트는 Azure환경에서 Hadoop의 분산 처리 기법인 Map&Reduce를 구현하는 데 목적이 있다. 이는 Azure의 HDInsight를 통해 실험할 수 있으며, 거대한 사이즈의 데이터에 대해 Group by Max알고리즘을 구현해본다. Group by Max는 각 그룹 별 최대값을 찾는 문제로, 그룹 번호, 값이 key:value형태로 주어졌을 때 각 그룹 마다의 최대값을 구하게 된다. Group by Max를 Mapping, Reducing을 이용하여 Hadoop 환경에서 구현하는 것이 본 프로젝트의 목표이다.

# 프로그램 설명



Hadoop의 MapReduce는 Mapper와 Reducer로 분산 처리 시스템을 구현할 수 있다. MapReduce는 Job이라는 작업단위로 관리되며, Job Tracker는 cluster에 등록된 전체 Job의 스케줄링을 관리하고 모니터링한다. 실제 Mapper와 Reducer을 실행하는 곳은 Task Tracker이다. HDFS에 input.data가 저장되고, Master는 Mapper은 Map을 담당하는 worker에게, Reducer은 Reduce를 담당하는 worker에게 할당한다. Map이 모두 실행된 후, Reduce가 실행되는데 각 Map task는 노드의 데이터를 읽어 key, value쌍을 생성한다. Map이 완료되면 Reducer가 실행되는데 Job Tracker에게 map의 결과를 물어보며 정상적으로 조회되면 key, value값으로 key를 sort한다. 그 다음에는 Reducer가 collective data를 combine/aggregate하여 Output file을 생성한다.

위와 같은 프로그램의 기반 위에서 Group by max를 구현하는 것이 본 프로젝트의 핵심이다. 이를 구현하기 위해 Mapper을 통해 데이터가 key와 value값으로 구분이 되고, 위와 같이 Map과 Reduce가 이루어지도록 코드를 구현하였다. Input과 Mapper.out에는 크게 차이가 없는데, 여기에서는input으로부터 들어온 raw data를 key과 value값으로 보다 분명하게 분리하기 위함이다. 이렇게 다시 만들어진 mapper.out은 reducer.py의 인풋으로 들어가게 되는데 모든 line을 읽으면서 key별로 최대값을 찾아 출력한다. Sort는 따로 코드를 작성해주지 않았는데, reducer.py에서 사전형을 사용하기 때문에 오름차순의 순서가 의미가 없다고 판단했으며, sort없이도 reducer가 가능하기 때문이다.

# 모듈 정의(mapper.py와 reducer.py의 동작)

* Mapper.py

먼저 sys.stdin으로부터 input.data를 읽어온다. Read\_input함수에서 파일을 라인단위로 읽으며, rstrip()하여 오른쪽 공백을 제거한 후 쉽표로 split한다. 이는 input.data의 형식에 따른 것이다. 쉽표로 divide함으로써 그룹번호와 값이 key:value쌍으로 구분된다. Yield를 통해 data에 iterative한 데이터가 저장된다. 메인에서는 이 데이터를 “data”변수에 받아오고, data의 각 변수는 split한 결과인 리스트가 저장되어 있는데 각 리스트는 input.data의 한 줄 단위의 그룹명, 값을 담고 있으며 i[0], i[1]로 참조할 수 있다. intermediate파일로 mapper.ot을 출력하게 되는데 값의 손실을 방지하기 위하여 python의 %f로 포맷팅하기 보다 print()로 그대로 출력하였다. 이러한 mapper.out는 reducer.py의 인풋으로 들어가 reducer을 실행하게 된다.

* Reducer.py

input파일인 mapper.out을 마찬가지로 파일읽기한다. Read\_input함수에서 공백이 제거되고 , split을 통해 key과 value값으로 분리된다. 해당 쌍을 통해 그룹별 최대값을 구할 수 있는데, 먼저

사전형태의 data 변수를 만든다. Data의 사전 키는 그룹번호이고, value는 그룹의 최대값이다. 따라서 파일읽기한 데이터가 담인 inputdata를 하나씩 참조하면서, 해당 쌍의 key가 data사전에 있고 기존의 최대값보다 더 크다면 data[key]의 값을 새로운 값으로 바꾸어준다. 아니라면 그대로 둔다. 만약 key가 data에 없다면 새로 추가하고 새로운 값을 입력해준다. intermediate파일을 모두 읽어 그룹별 최대값이 모두 저장된 사전 data가 완성된 이후에, data의 각 item을 참조하면서 key와 value값을 형식에 맞게 출력해준다.

따라서 정리해보면, input.data로 들어온 데이터를 mapper.py에서 key와 value로 정제해준 후, sort가 이루어지고 reducer에서는 intermediate파일의 key와 value쌍을 이용하여 최대값을 찾는 알고리즘을 수행하게 되는 것이다. 본 프로그램에서 mapper의 역할은 단지 input.data가 key와 value로 잘 들어왔는지 확인하여 그대로 출력하는 역할을 수행할 뿐이지만, 여타 다른 mapReduce에서 mapper은 상당한 역할을 수행할 수도 있다.

# 코드

* Mapper.py

#!/usr/bin/env python

import sys

def read\_input(file):

for line in file:

yield line.strip().split(',')

def main():

data = read\_input(sys.stdin)

for i in data:

print(i[0],i[1])

if \_\_name\_\_ == "\_\_main\_\_":

main()

* Reducer.py

#!/usr/bin/env python

import sys

def read\_mapper\_output(file):

for line in file:

yield line.rstrip().split()

def main():

data = {}

inputdata = read\_mapper\_output(sys.stdin)

for L in inputdata:

group, number = L

number = float(number)

if group in data.keys():

data[group] = max(data[group],number)

else:

data[group]=number

for group, number in data.items():

print("{},{}".format(group, number))

if \_\_name\_\_ == "\_\_main\_\_":

main()