# Hadoop

- Hadoop은 방대한 데이터를 변환하거나 분석하는 데이 있어서 강력한 툴이다.
  - 그러나 수백가지의 서로 다른 기술로 구성되어 있어서 이해하는 데에 어려움이 있을 수 있다.

- 가상 머신(virtual machine)이 설치되어 있어야한다. (https://www.virtualbox.org/ (https://www.virtualbox.org/))
- 하둡 이미지가 필요하다. (https://hortonworks.com/products/sandbox/ (https://hortonworks.com/products/sandbox/))

  - Hadoop의 분산 저장을 위한 주요 오픈소스 소프트웨어 플랫폼이다.
  - 컴퓨터 클러스터의 매우 큰 데이터 세트를 처리한다.

## 1. HDFS (Hadoop Distributed File System)

- 전체 클러스터에서 배포 및 분리 할 수있는 매우 큰 파일을 처리하는 데 최적화되어 있다.
- 128 MB의 블록을 분해한다.
  - 여러 대의 컴퓨터에서 여러 부분으로 나누어 병렬적으로 처리할 수 있다.

- SQL query를 사용한다.

- Hadoop은 특정 데이터 센터의 전체 PC 클러스터에서 실행이된다.
  - 여러 대의 컴퓨터가 빅데이터를 처리할 수 있다.
  - 분산형 저장소에 의해 클러스터에 컴퓨터를 추가할 수 있다.
  - 모든 하드드라이브에 분산된 데이터를 볼 수 있다.
  - 모든 데이터의 백업 복사본을 유지하기 때문에 매우 안정적이다.
  - 전체 클러스터에서 모든 CPU 코어를 설정하여 문제를 병렬로 처리 할 수 있다.
    - 데이터를 매우 빠르게 처리 할 수 있다.

### (1) HDFS 조작 - Web

- virtual machine 실행 후 http://127.0.0.1:8888/ (http://127.0.0.1:8888/) url을 통해 접근

  - ID : maria_dev
  - PW : maria_dev

- Ambari로 이동되어 Hive를 통해 데이터를 import 할 수 있다.
  - 클러스터의 뷰를 제공하고 클러스터에서 실행중인 것을 시각화
  - 데이터셋의 이름과 각 컬럼명 설정 가능
  - Hadoop과 상호작용하여 관계형 데이터 베이스인 것처럼 보인다. (실제로는 그렇지 않다.)

- 직관적인 인터페이스로 파일 추가 및 설정이 쉽다.

### (2) HDFS 조작 - Prompt

- PuTTY를 통해 접근 (https://putty.org/ (https://putty.org/))
  - HOST_NAME : maria_dev@127.0.0.1
  - Port : 2222
  - Connection type : SSH
  - PW : maria_dev
- 기본 command는 linux command와 동일하나 모든 command 앞에 'hadoop fs -'이 붙는다.

```
$ hadoop fs -ls 디렉토리 : HDFS 디렉토리 내의 파일과 폴더 확인
$ hadoop fs -mkdir 폴더명 : HDFS 내 폴더 생성 (make directory)
$ hadoop fs -copyFromLocal 파일명 디렉토리 : 로컬 파일 시스템에서 HDFS의 디렉토리로 전송
                                                (ex: hadoop fs -copyFromLoca u.data ml-100k/u.data)
$ hadoop fs -rm 디렉토리/파일명 : HDFS 디렉토리 내의 파일 삭제 (remove)
$ hadoop fs -rmdir 디렉토리 : HDFS 디렉토리 삭제 (remove directory)

$ hadoop fs : Hadoop에서 사용 가능한 command 목록을 보여준다. (로컬 파일 시스템 command와 같다.)
```

- 가상 머신으로 업로드 하기 위해서는 로컬 파일 시스템에서 서버에 업로드 해야한다. 아래는 로컬 파일 시스템에서 사용하는 명령어이다.

```
$ ls : 현재 디렉토리의 파일 및 폴더 확인
$ pwd : 작업 디렉토리 확인
4 wget url : url의 데이터를 다운로드
```

## 2. Yarn(yet another resource negotiator)

- 자원 (CPU, 메모리) 관리
- ResourceManager(RM)와 ApplicationMaster(AM)로 분리

# 3. MAPREDUCE

- Hadoop의 핵심 부분 중 하나이다.
- Hadoop이 클러스터에서 데이터 처리를 분산 할 수있는 수단을 제공한다.
  - 모든 데이터를 클러스터에서 병렬로 처리 할 수 있는 파티션으로 나누고 파티션의 처리 방법과 오류 처리 방법 등을 관리한다.

- Mapper
  - 구조화하고 추출하기 원하는 데이터를 key:value의 형태로 변환한다.
  - 자동으로, 같은 key를 가진 value를 list의 형태로 합쳐준다. (Shuffle & Sort)

- Reducer
  - 전체 클러스터에서 모든 값을 모은다.
  - 원하는 output을 출력하기 위해 무엇을 할 것인지 결정한다.

# 4. MRJob

- python을 사용항여 mapreduce를 구현하는 코드(추후 HDP root에 저장한다.)

In [ ]:

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class RatingsBreakdown(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_ratings,
                   reducer=self.reducer_count_ratings)
        ]

    def mapper_get_ratings(self, _, line):
        (userID, movieID, ratings, timestamp) = line.split('₩t')
        yield ratings, 1

    def reducer_count_ratings(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    RatingsBreakdown.run()

```

- **Install at HDP 2.6.5 (HDP 2.5의 경우, 시스템 리소스가 적게 요구되기 때문에 HDP 2.5가 권장된다.)**

```
$ yum install python-pip : 파이썬 pip 설치
$ pip install mrjob==0.5.11 : mrjob 다운
$ yum install nano : nano editor 다운
$ wget file_url : 파일 및 파이썬 스크립트 등 다운
  (ex: http://media.sundog-soft.com/hadoop/ml-100k/u.data / http://media.sundog_soft.com/hadoop/RatingsBreakdown.py)
```

- **Install at HDP 2.5**

```
$ cd /etc/yum.repos.d : 디렉토리 이동
$ cp sandbox.repo /tmp : 충돌을 일으키는 파일을 삭제하기 전에 다른 경로에 복사
$ rm sandbox.repo  : 충돌을 일으키는 파일 삭제
$ cd ~ : 홈 디렉토리로 이동
$ yum install python-pip

$ pip install google-api-python-clinet==1.6.4
$ pip install mrjob==0.5.11

$ yum install nano

$ wget file_url
```

- **mrjob 실행**

1. virtual box의 Hortonworks 를 실행한다..
2. PuTTY를 통해 HDP에 접속한다. (PW: maria_dev)
3. root로 이동한다. (su root)
4. locall 실행 (Hadoop 실행보다 빠르다.)
    - python python_script file_name (ex: python RatingsBreakdown.py u.data)
5. Hadoop 실행
    - python python_cripts -r hadoop --hadoop-streaming-jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar file_name (큰 데이터 용량 등으로 locall에 파일이 존재하지 않을 경우 HDFS의 경로로 대체한다 - hdfs:// ~)

# Exercise

- **Basic Exercise**
  - 각 영화를 인기도 순서로 정렬하여 출력한다.

In [ ]:

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class RatingsBreakdown(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_ratings,
                   reducer=self.reducer_count_ratings),
              MRStep(educer=self.reducer_sorted_output)
        ]

    def mapper_get_ratings(self, _, line):
        (userID, movieID, ratings, timestamp) = line.split('₩t')
        yield movieID, 1

    def reducer_count_ratings(self, key, values):
        yield str(sum(values)).zfill(5), key

    def reducer_sorted_output(self, count, movies):
        for movie in movies:
            yield key, sum(values)

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

## NOTE

- 여러 개의 mapreduce를 사용하는 경우, 아래의 코드를 사용하여 mapreduce를 묶을 수 있다.

In [1]:

```
def steps(self):
    return [
        MRStep(mapper=self.mapper_get_ratings,
            reducer=self.reducer_count_ratings),
        MRStep(reducer=self.reducer_count_ratings)
        ]
```

- streaming은 input과 output을 string만을 다룬다.(alphabetically)
- 때문에 숫자 그대로 사용할 경우 제대로 정렬이 되지 않을 수 있다.(ex: 100 보다 4가 크게 나온다.)
  - 아래 코드와 같이 zfill()을 사용하여 숫자를 zero-pad 할 경우, 적절하게 정렬할 수 있다.(해결할 수 있는 여러가지 방법들이 존재한다.)

In [2]:

```
def reducer_count_ratings(self, key, values):
    yield str(sum(values)).zfill(5), key
```