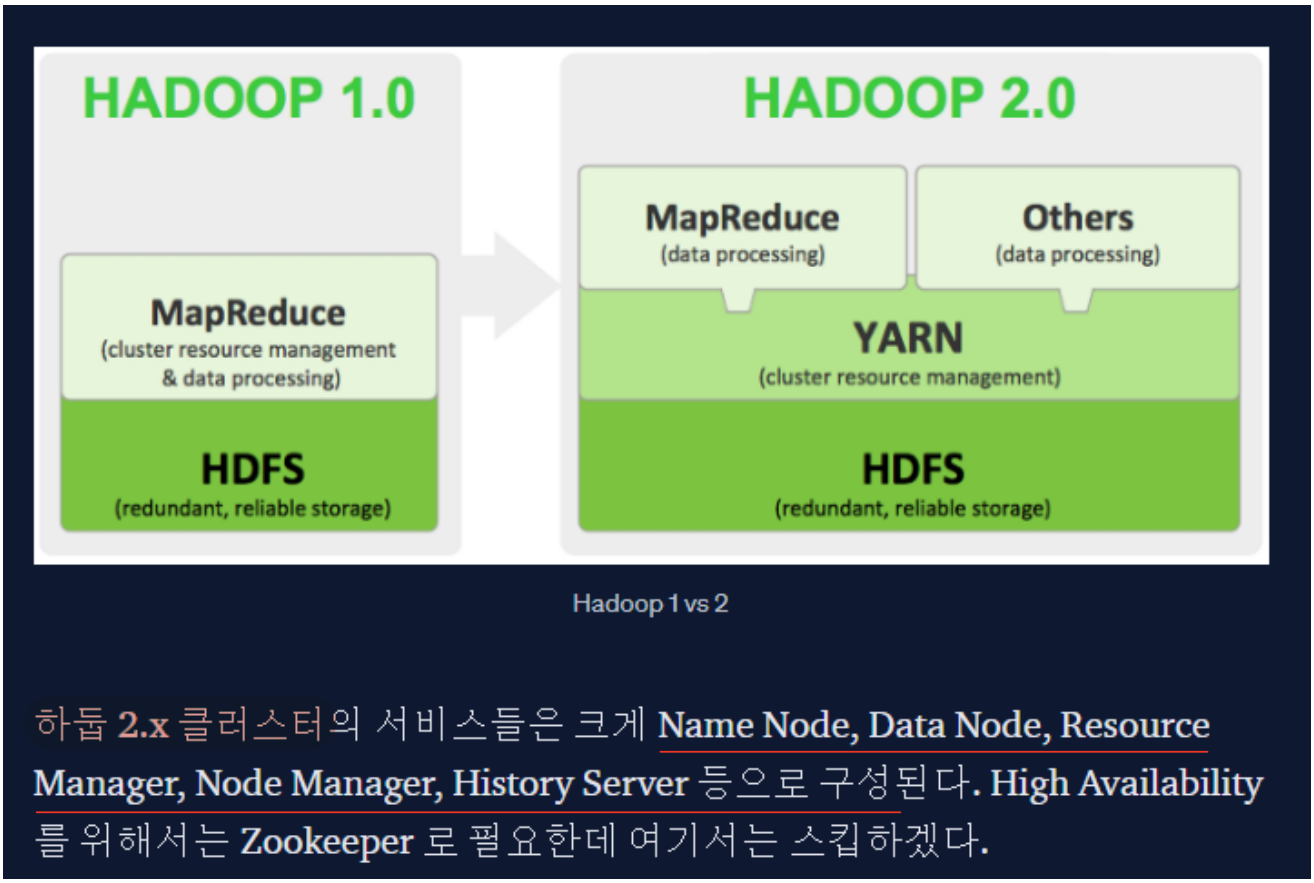
하둡은 세 가지 다른 모드로 사용된다.

- 단일모드(The standalone mode)
  - 데몬 프로세스 없이 모든 프로그램이 하나의 JVM위에서 동작하는 모드.
  - 이 모드는 테스트 용도로만 추천하며 기본 모드이기 때문에 어떠한 다른 설정도 필요 없다.(분산 운영 모드가 아니므로 실제 환경에서는 부적합하다.)
  - HDFS 파일이 아닌 로컬 파일시스템 사용
  - NameNode, DataNode, JobTracker, TaskTracker 같은 모든 데몬은 단일 자바 프로세스로 작동한다.
- 의사 분산 모드(Pseudo-distributed mode)
  - 1대의 컴퓨터에 하둡 데몬 프로세스가 여러개 분리되어 작동하는 모드. (실제로 물리적으로는 분산되진 않았으나 그런 척 하도록 만드는 것)
  - 단일모드와 마찬가지로 싱글노드에서 동작
  - HDFS 사용, JVM 단 위에서 여러개의 자바 데몬 프로세스가 작동 가능
  - 작은 규모의 클러스터를 테스트, 디버깅 하는 등에 사용
- 완전 분산 모드(Fully distributed mode)
  - 하둡 데몬 프로세스가 클러스터로 구성된 여러 컴퓨터에 나누어 동작하는 모드이다.
  - 데이터들은 데이터 노드에, 이들에 대한 메타정보는 네임 노드에서 관리하는 운영모드
  - 실제 환경에서 동작하고 운영하는 환경



Hadoop 1 vs 2

하둡 2.x 클러스터의 서비스들은 크게 Name Node, Data Node, Resource Manager, Node Manager, History Server 등으로 구성된다. High Availability 를 위해서는 Zookeeper 로 필요한데 여기서는 스킵하겠다.

2

As far as I know HDFS does not care about the physical file system on which it is running. I installed Hadoop on several different file system, for example I also used solaris ZFS.

The blocks of hadoop/hdfs are written as ordinary files on each datanode. The namenode takes to role of inodes or FAT in the OS filesystems. HDFS is a layer above the physical filesystem on each datanode.

You can list the stored blocks in your hadoop/hdfs filesystem just by listing directory content on the data node:

```
/srv/hadoop/hadoop_data/hdfs/datanode/current/BP-1458088587-192.168.1.51-1394008575227/current/finalized$ ls -
alh ./blk_1073741838
-rw-r--r-- 1 hadoop hadoop 1.4M Mar  6 10:55 ./blk_1073741838
```

Name Node - HDFS에서 실제 파일들에 대한 메타정보를 가지고 있는 서비스로 하둡에서 가장 중요한 서비스라고 볼 수 있다. 1.x 에서는 이 부분이 Single-Point-Of-Failure였는데 2.x 에서는 Zookeepr 를 통한 Stand-by Name Node로의 Fail-over 를 통해 High Availability를 고려할 수 있다.

Data Node - HDFS에서 실제 파일 데이터(파일시스템 블럭)를 저장하고 있는 서비스다. 반드시 Name Node를 알고 있어야하며 Scale-out의 대상이 되는 서비스이다. Name Node와 Data Node만 있으면 HDFS 수행이 가능하다.

Resource Manager - YARN의 코어 서비스. 이녀석이 각 Node Manager 들을 모두 관리하면서 클라이언트 Job 수행에 대한 스케쥴링을 제공한다.

Node Manager - Resource Manager로 부터 스케쥴된 요청들을 실제로 수행할 노드에서 실행되는 서비스. 이녀석은 Resource Manager를 반드시 알고 있어야하며 Scale-out 의 대상이다.

History Manager - 이건 YARN을 통해 수행된 모든 Job 들에 대한 이력을 저장하고 관리하는 서비스이다. (옵셔널일듯)

주요 설정

환경변수 HADOOP_CONF_DIR - 하둡에 필요한 config 파일들을 담고 있는 directory 설정. 주요 config 파일로는 core-site.xml, hdfs-site.xml, yarn-site.xml 이 있다.

$HADOOP_CONF_DIR/core-site.xml - Name Node의 Endpoint에 대한 설정히 핵심 (하둡에서 Name Node 가 핵심이다 보니까 core 라고 이름을 지었나 봄)

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://namenode:8020</value>
</property>
```

$HADOOP_CONF_DIR/hdfs-site.xml - HDFS 서비스의 옵션에 대한 설정

```
<property>
  <name>dfs.webhdfs.enabled</name>
  <value>true</value>
</property>
```

$HADOOP_CONF_DIR/yarn-site.xml - Resource Manager 관련 설정

```
<property>
  <name>yarn.resourcemanager.store.class</name>

<value>org.apache.hadoop.yarn.server.resourcemanager.recovery.FileSys
temRMStateStore</value>
</property>
<property>
  <name>yarn.resourcemanager.resource-tracker.address</name>
  <value>hadoop-resourcemanager:8031</value>
</property>
<property>
  <name>yarn.resourcemanager.hostname</name>
  <value>hadoop-resourcemanager</value>
</property>
<property>
  <name>yarn.timeline-service.hostname</name>
  <value>hadoop-historyserver</value>
</property>
<property>
  <name>yarn.log.server.url</name>
  <value>http://hadoop-
historyserver:8188/applicationhistory/logs/</value>
</property>
<property>
  <name>yarn.resourcemanager.scheduler.address</name>
  <value>hadoop-resourcemanager:8030</value>
</property>
<property>
  <name>yarn.resourcemanager.address</name>
  <value>hadoop-resourcemanager:8032</value>
</property>
<property>
  <name>yarn.resourcemanager.bind-host</name>
  <value>0.0.0.0</value>
</property>
<property>
  <name>yarn.nodemanager.bind-host</name>
  <value>0.0.0.0</value>
</property>
<property>
  <name>yarn.nodemanager.bind-host</name>
  <value>0.0.0.0</value>
</property>
<property>
  <name>yarn.timeline-service.bind-host</name>
  <value>0.0.0.0</value>
</property>
```

클러스터란?

특정한 기능수행을 위해 여러 대의 컴퓨터가 네트워크로 연결된 것을 의미하며,

이때 클러스터를 구성하는 개별 컴퓨터를 node라고 칭한다.

In HDFS architecture there is a concept of blocks. A typical block size used by HDFS is 64 MB. When we place a large file into HDFS it chopped up into 64 MB chunks(based on default configuration of blocks), Suppose you have a file of 1GB and you want to place that file in HDFS, then there will be 1GB/64MB = 16 split/blocks and these block will be distribute across the DataNodes. These blocks/chunk will reside on a different different DataNode based on your cluster configuration.

↳ block들이 여러 데이터 노드들에 분배된다.

Data splitting happens based on file offsets. The goal of splitting of file and store it into different blocks, is parallel processing and fail over of data.

Difference between block size and split size.

↳ input split.

Split is logical split of the data, basically used during data processing using Map/Reduce program or other dataprocessing techniques on Hadoop Ecosystem. Split size is user defined value and you can choose your own split size based on your volume of data(How much data you are processing).

↳ input split 개수 만큼, Mapper의 개수가 결정된다.

Split is basically used to control number of Mapper in Map/Reduce program. If you have not defined any input split size in Map/Reduce program then default HDFS block split will be considered as input split.

Example:

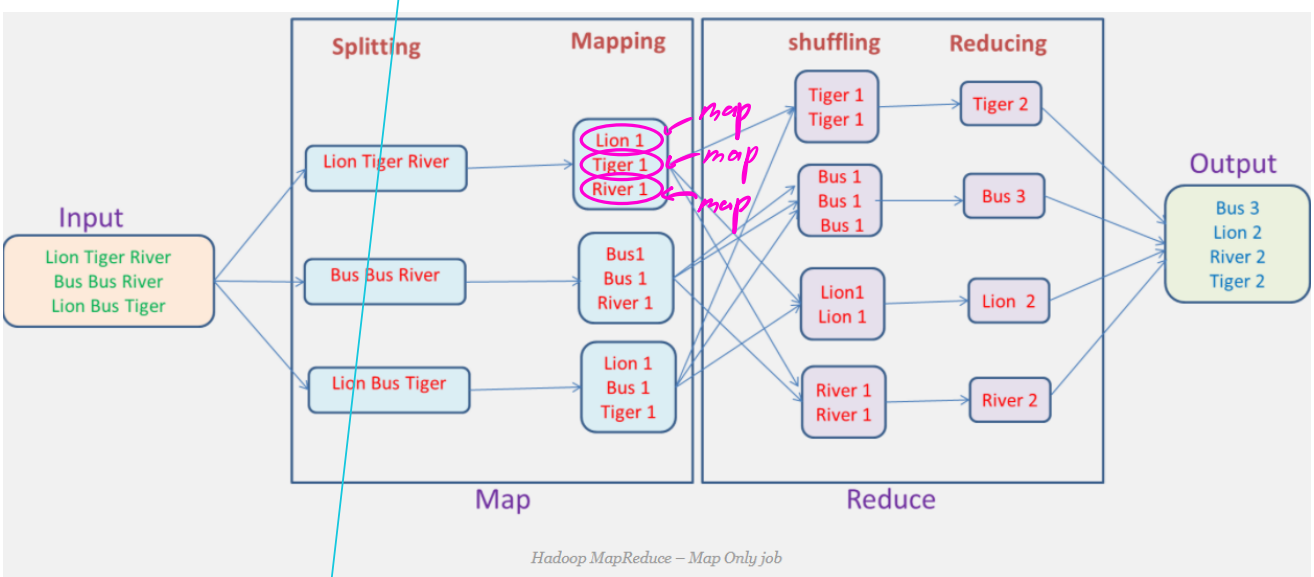
Suppose you have a file of 100MB and HDFS default block configuration is 64MB, then it will chopped in 2 split and occupy 2 blocks. Now you have a Map/Reduce program to process this data but you have not specified any input split then based on the number of blocks(2 block) input split will be considered for the Map/Reduce processing and 2 mapper will get assigned for this job.

But suppose, you have specified the split size(say 100MB) in your Map/Reduce program then both blocks(2 block) will be considered as a single split for the Map/Reduce processing and 1 Mapper will get assigned for this job.

Suppose, you have specified the split size(say 25MB) in your Map/Reduce program then there will be 4 input split for the Map/Reduce program and 4 Mapper will get assigned for the job.

Conclusion:

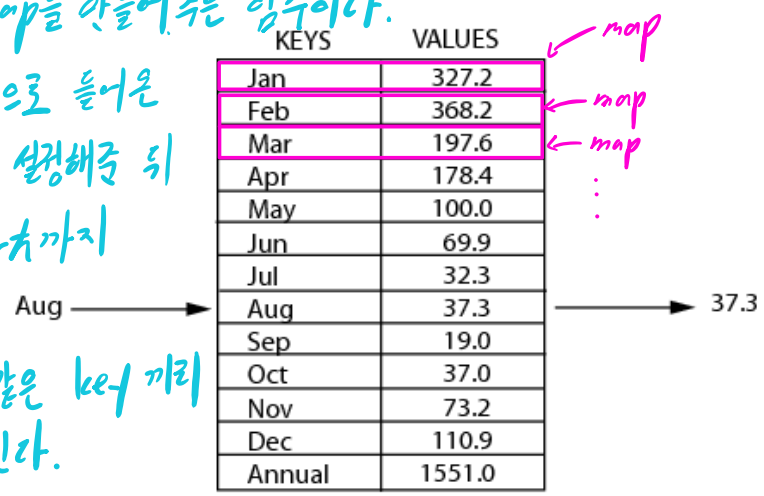
1. Split is a logical division of the input data while block is a physical division of data.
2. HDFS default block size is default split size if input split is not specified.
3. Split is user defined and user can control split size in his Map/Reduce program.
4. One split can be mapping to multiple blocks and there can be multiple split of one block.
5. The number of map tasks (Mapper) are equal to the number of splits.



Hadoop MapReduce – Map Only job

일단 **맵(Map)** 이라는 것은 지도? 아니구요, :) 데이터를 담아두는 자료 구조 중의 하나입니다. 맵은 키와 밸류라는 두개의 값을 쌍으로 가지고 있는 형태입니다. 수학시간에 좌표를 표시할때 순서쌍이라고 하죠, (x,y) 이렇게 하던 바로 그 개념입니다. 여기서 x가 키이고, y가 밸류 즉 값인거죠. 그리고 함수 f(x) => y 도 생각나시죠? x를 알면 y를 알 수 있는 구조로 관리 됩니다.

※ Mapper는 map을 만들어주는 함수이다.
Mapper는 입력으로 들어온 요소들에 key를 설정해준 뒤 key에 따라 sort까지 실시한다.
이후, 자동으로 같은 key끼리 'GROUP BY'된다.



리듀스(Reduce)는 이 맵을 정리해 나가는(줄여나가는) 방법이라고 할 수 있습니다. 키를 기준으로 (같은 키 값을 가진 맵들의) 개수를 센다든지, 같은 키를 기준으로 밸류를 모두 더하거나, 평균을 내거나 하는 것들이 있습니다. 글 마지막 부분에 더 많은 예제가 있으니까 끝까지 꼭 읽어보세요. :)

쇼핑몰을 방문한 사용자를 집계하고 싶을때는

로그를 한 줄씩 읽으면서 (사용자아이디, 1) 형태로 맵을 만듭니다.

리듀스 할때 밸류를 서로 더해 주셔도 되고, key의 개수를 세어주는 함수가 제공된다면 그걸 그냥 호출하면 됩니다.

대략 적으로 코드 형태를 보여드리자면 이런 식입니다. 일종의 pseudo code랄 수 있겠네요. :)

```
map(사용자아이디, 1)
reduceByKey((a,b) => a+b)
```

맵을 만들때는 로그의 한 줄을 읽어서 적당한 부분을 잘라내는 파싱 작업이 먼저 이루어져야 하죠. 그래야 map에 저렇게 사용자 아이디를 넣어줄 수 있으니까요.
그리고 reduceByKey에 나오는 a,b는 두 개의 맵을 들고 연산을 수행할때, 각각의 밸류값 입니다. 키는 reduceByKey라는 이름이 알려주듯 같은 키 값을 가진 맵들끼리 계속해서 연산을 해서 맵의 숫자를 줄이는 거죠. 최종적으로는 그 키를 가지는 맵이 한 개 남을때 까지요. 그러니까 키는 정해줄 필요가 없고 첫번째 맵의 밸류 a와 두번째 맵의 밸류 b를 가지고 어떤 연산을 수행할 것인지 (위 예제에서는 더하기(+) 네요) 만 정해 주면 됩니다.

map의 총 개수를 구한다.

- 특정행동이 일어난 건수를 행동 유형별로 알고 싶을때는

  로그를 한 줄씩 읽으면서 (행동유형, 1) 형태로 맵을 만들구요,

  리듀스처리에서는 키가 같은 두개의 맵이 만날때마다 밸류 값을 서로 더해주는 함수를 선언하면 됩니다.

```
map(행동유형, 1)
reduceByKey((a,b) => a+b)
```