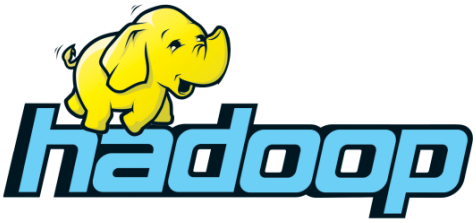
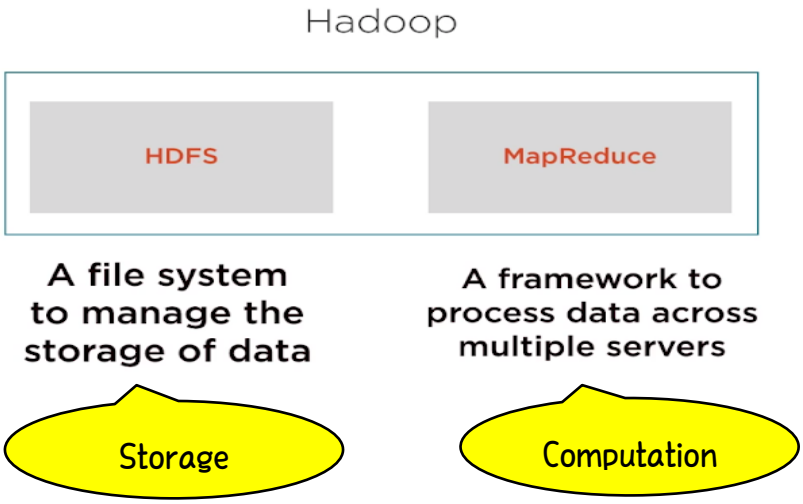
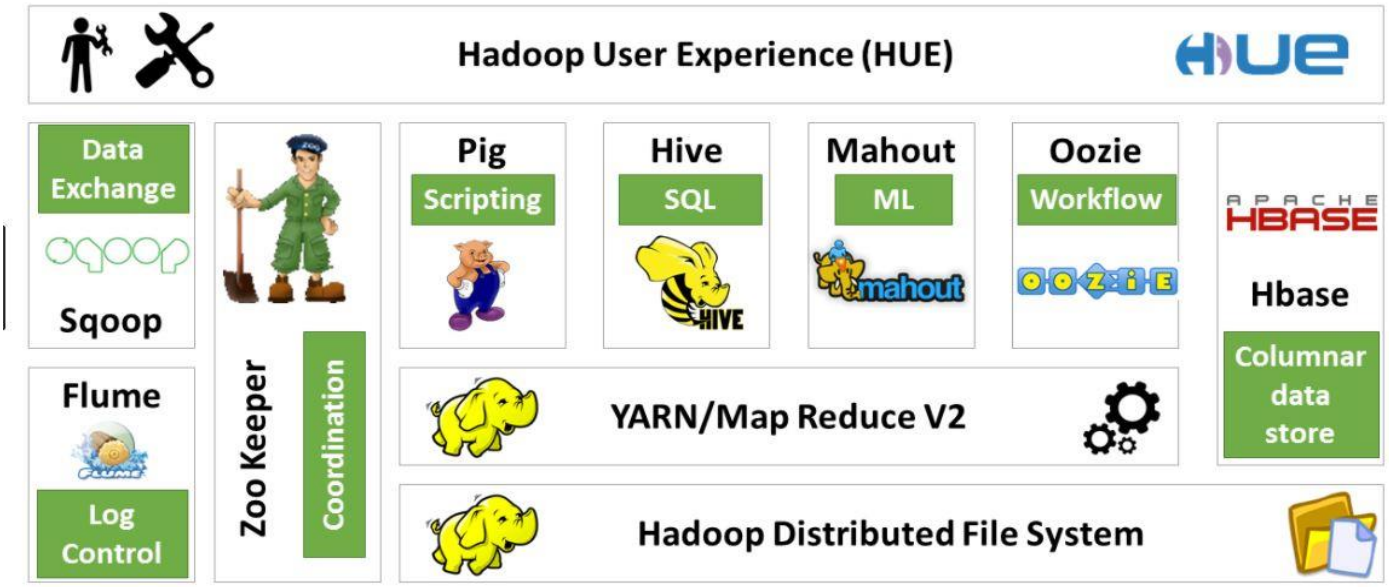


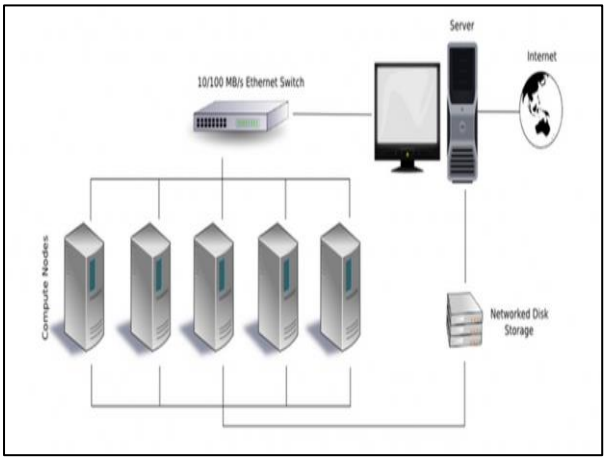


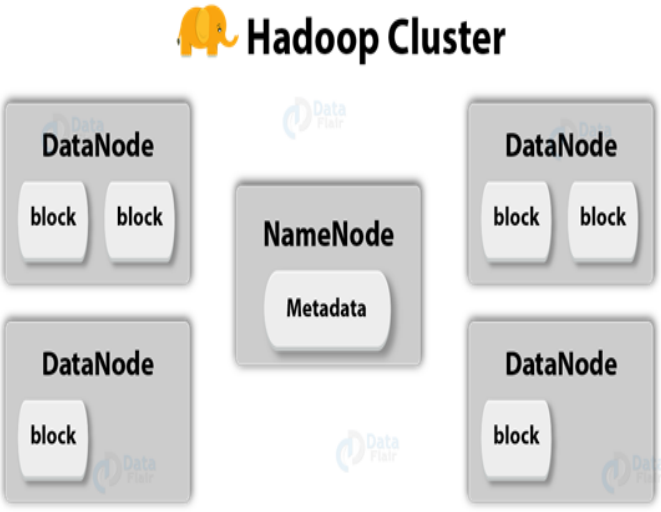
[Hadoop & Eco System]



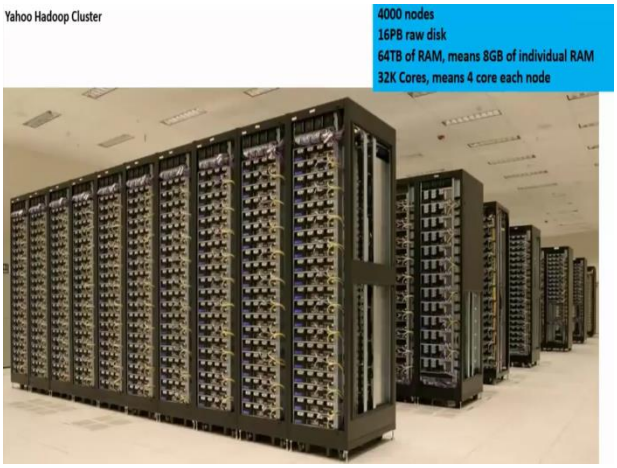
[클러스터]

여러 대의 컴퓨터들이 연결되어 하나의 시스템처럼 동작하는 컴퓨터들의 집합

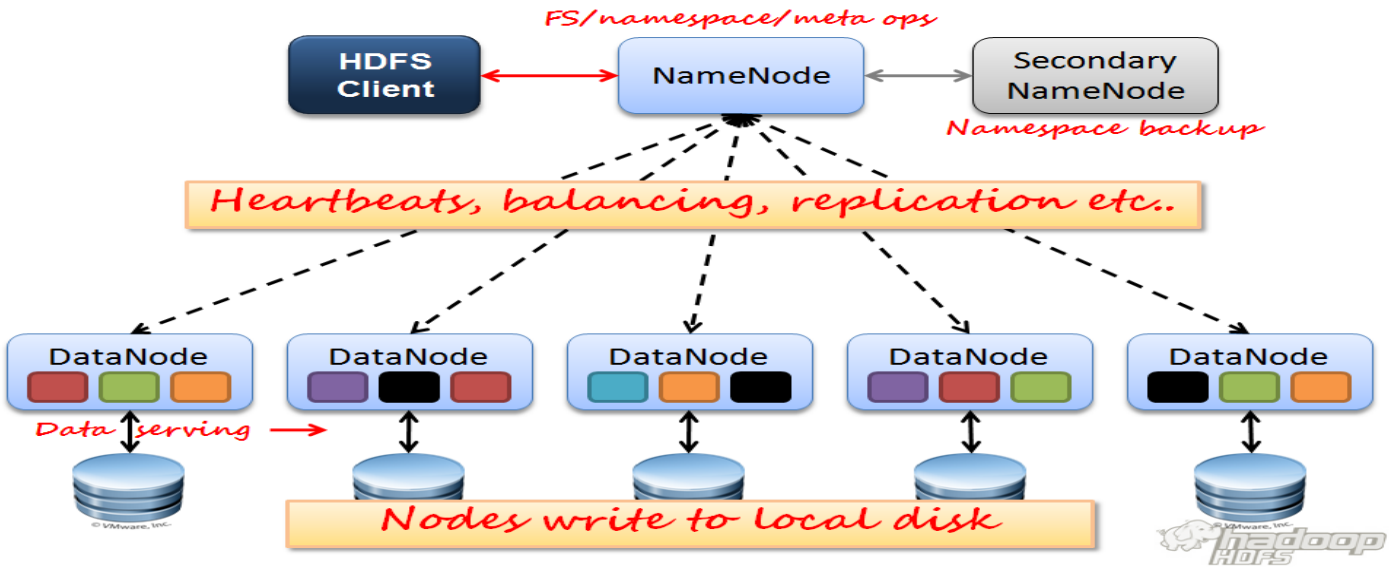




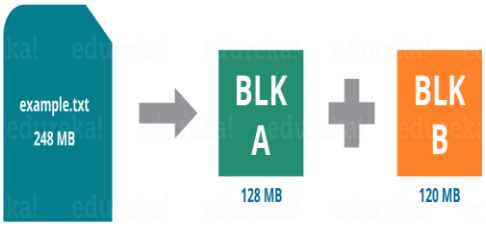
[분산 컴퓨팅 시스템]

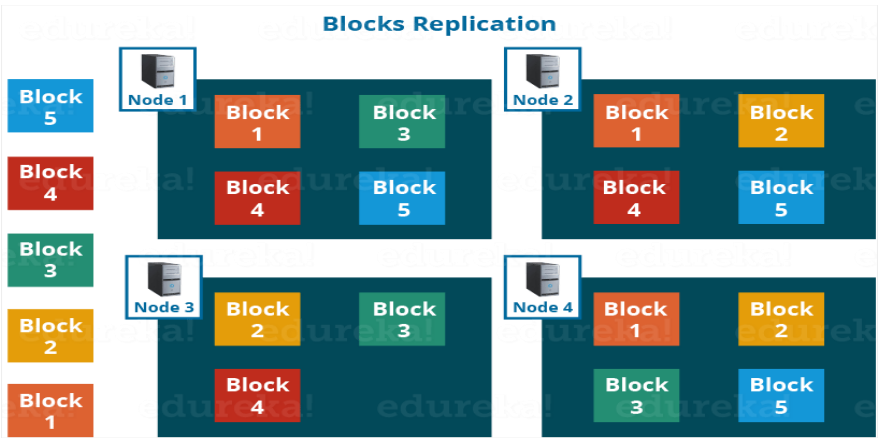


[Hadoop Cluster의 구조]



[Data Block] – 블록 사이즈 디폴트 : 128MB





[Hadoop HDFS 명령어]

https://hadoop.apache.org/docs/r2.7.7/hadoop-project-dist/hadoop-common/FileSystemShell.html

hdfs dfs -ls /[디렉토리명 또는 파일명]

지정된 디렉토리의 파일리스트 또는 지정된 파일의 정보를 보여준다.

hdfs dfs -ls -R /[디렉토리명]

지정된 디렉토리의 파일리스트 및 서브디렉토리들의 파일 리스트도 보여준다.

hdfs dfs -mkdir /디렉토리명

지정된 디렉토리를 생성한다.

hdfs dfs -cat /[디렉토리/]파일

지정된 파일의 내용을 화면에 출력한다.

hdfs dfs -put 사용자계정로컬파일 HDFS디렉터리[/파일]

지정된 사용자계정 로컬 파일시스템의 파일을 HDFS 상 디렉터리의 파일로 복사한다.

hdfs dfs -get HDFS디렉터리의파일 사용자계정로컬 디렉터리[/파일]

지정된 HDFS상의 파일을 사용자계정 로컬 파일시스템의 디렉터리나 파일로 복사한다.

hdfs dfs -rm /[디렉토리]/파일

지정된 파일을 삭제한다.

hdfs dfs -rm -r /디렉토리

지정된 디렉터리를 삭제. 비어 있지않은 디렉터리도 삭제하며 서브 디렉토리도 삭제한다.

hdfs dfs -tail /[디렉토리]/파일

지정된 파일의 마지막 1kb 내용을 화면에 출력한다.

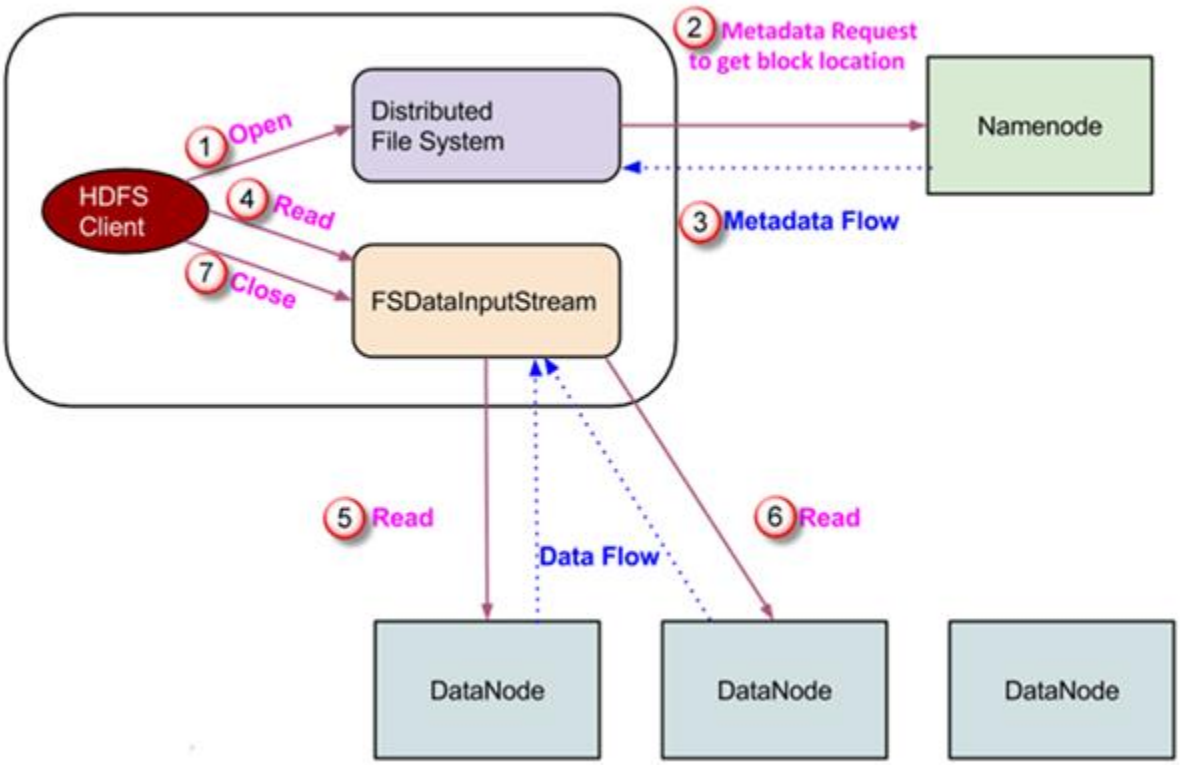
hdfs dfs -chmod 사용자허가모드 /[디렉토리명 또는 파일명]

지정된 디렉토리 또는 파일의 사용자 허가 모드를 변경한다.

hdfs dfs -mv /[디렉토리]/old파일 /[디렉토리]/new파일

지정된 디렉토리의 파일을 다른 이름으로 변경하거나 다른 폴더로 이동한다.

[HDFS Read Process]



(1) 클라이언트는 FileSystem(DistributedFileSystem) 객체의 'open()' 메소드를 호출하여 읽기 요청을 시작한다.

(2) FileSystem(DistributedFileSystem) 객체는 RPC를 사용하여 NameNode에 연결하고 파일 블록의 위치 정보를 담고 있는 메타 데이터 정보를 가져온다.

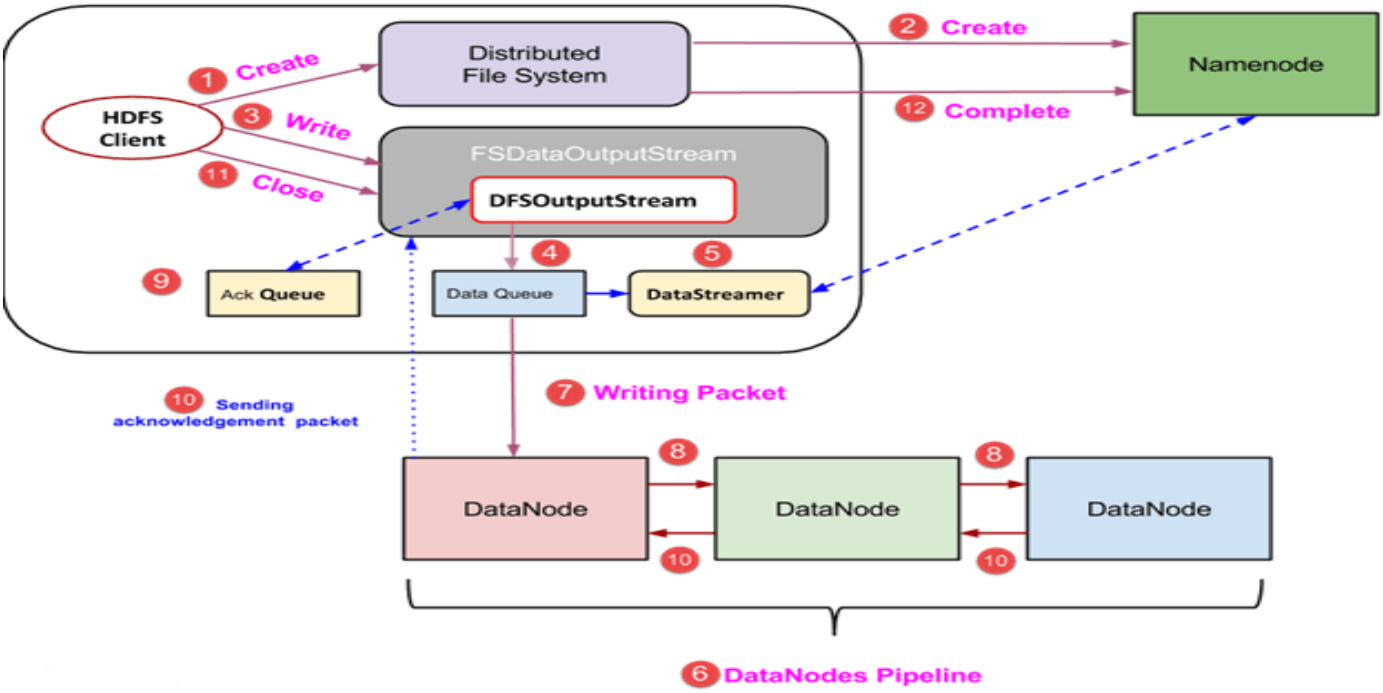
(3) 이 메타 데이터에는 읽고자 하는 파일의 블록에 대한 사본이 저장되어 있는 DataNode 의 주소가 담겨 있다.

(4, 5) DataNode의 주소를 받으면 FSDataInputStream 유형의 객체가 클라이언트에 리턴된다. FSDataInputStream 객체는 DFSInputStream 을 포함하고 있으며 읽기 관련 메서드가 호출되면 가장 우선순위가 높은 데이터 노드에서 해당 블록을 읽게 된다.

(6) 블록의 끝 부분에 도달하면 DFSInputStream은 연결을 닫고 다음 블록에 대한 다음 DataNode를 찾는다.

(7) 클라이언트가 읽기를 완료 하면 close() 메소드를 호출한다 .

[HDFS Write Process]



(1) 클라이언트는 새로운 파일을 생성하는 DistributedFileSystem 객체의 'create ()'메소드를 호출하여 쓰기 작업을 시작한다.

(2) DistributedFileSystem 객체는 RPC 호출을 사용하여 NameNode에 연결하고 새 파일 만들기를 시작한다. NameNode의 책임은 생성되는 파일이 이미 존재하지 않고 클라이언트가 새 파일을 생성 할 수 있는 올바른 권한을 가지고 있는지 확인하는 것이다. 파일이 이미 있거나 클라이언트에 새 파일을 작성할 수 있는 충분한 권한이 없는 경우 IOException 을 클라이언트에서 발생시키지만 문제가 없는 경우 NameNode에 의해 파일에 대한 새 레코드가 만들어진다.

(3) NameNode에 의해 새 레코드가 만들어지면 FSDataOutputStream 유형의 객체가 클라이언트에 반환된다. 클라이언트는 이 객체를 사용하여 HDFS에 데이터를 쓰기 위하여 write() 메소드를 호출한다..

(4) FSDataOutputStream은 DataNodes 및 NameNode와의 통신을 모니터링하는 역할의 DFSOutputStream 객체를 포함한다. 클라이언트가 계속해서 데이터를 쓰는 동안 DFSOutputStream 은 이 데이터를 가지고 패킷을 계속 작성하고 DataQueue 라고 하는 대기열에 저장한다. .

(5) DataStreamer는 NameNode에 새로운 블록 할당을 요청하여 복제에 사용할 바람직한 DataNode를 선택한다.

(6) DataNode를 사용하여 파이프 라인을 생성해서 복제 프로세스를 시작한다. 여기서는 복제 수준을 3으로 선택했으므로 파이프 라인에 3 개의 DataNode가 존재하게 된다.

(7) DataStreamer는 패킷을 파이프 라인의 첫 번째 DataNode에 집어 넣는다.

(8) 파이프 라인의 모든 DataNode는 받은 패킷을 저장하고 파이프 라인의 두 번째 DataNode로 전달한다.

(9) 또 다른 큐인 'Ack Queue'는 DFSOutputStream에 의해 유지되어 DataNode로부터 확인을 기다리는 패킷을 저장한다.

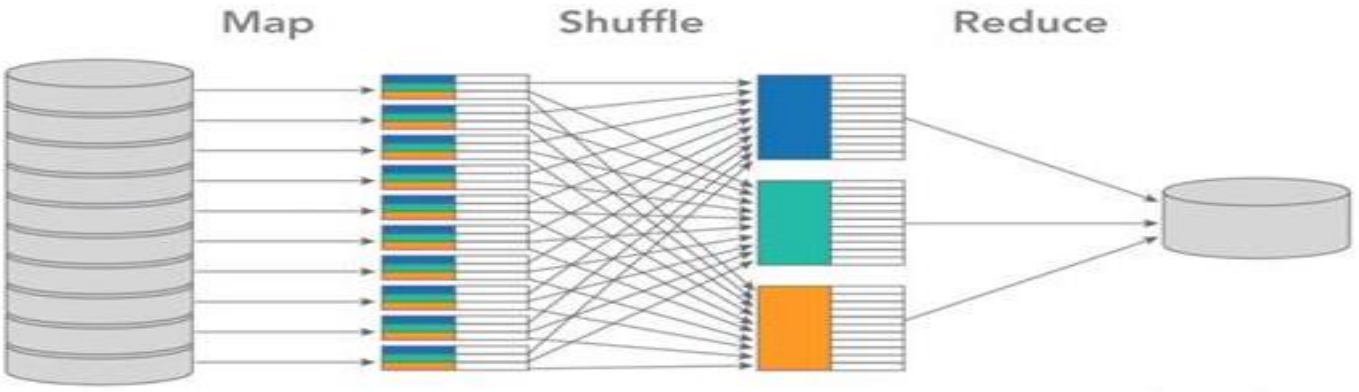
(10) 대기열에 있는 패킷에 대한 수신 확인을 파이프 라인의 모든 DataNode로 부터 수신하게 되면 'Ack 대기열'에서 제거된다. DataNode가 실패 할 경우 이 큐의 패킷을 사용하여 작업을 다시 시작한다.

(11) 클라이언트가 데이터 쓰기를 완료하면 close() 메서드를 호출하여 남아있는 데이터 패킷을 파이프 라인으로 플러시하고 최종 승인을 기다린다.

(12) 최종 승인을 받으면 NameNode에 연락하여 파일 쓰기 작업이 완료되었음을 알린다.

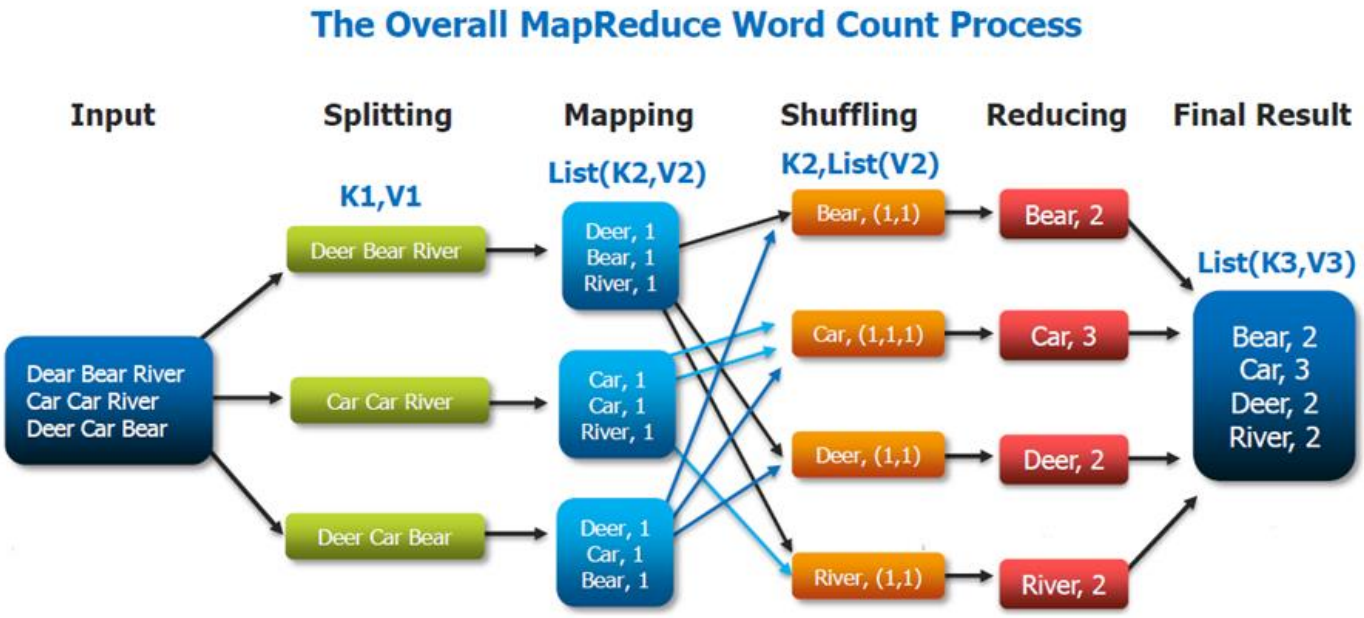
[Hadoop MapReduce]

MapReduce는 구글에서 대용량 데이터 처리를 분산 병렬 컴퓨팅에서 처리하기 위한 목적으로 제작하여 2004년 발표한 소프트웨어 프레임워크다. 이 프레임워크는 페타바이트 이상의 대용량 데이터를 신뢰도가 낮은 컴퓨터로 구성된 클러스터 환경에서 병렬 처리를 지원하기 위해서 개발되었다.

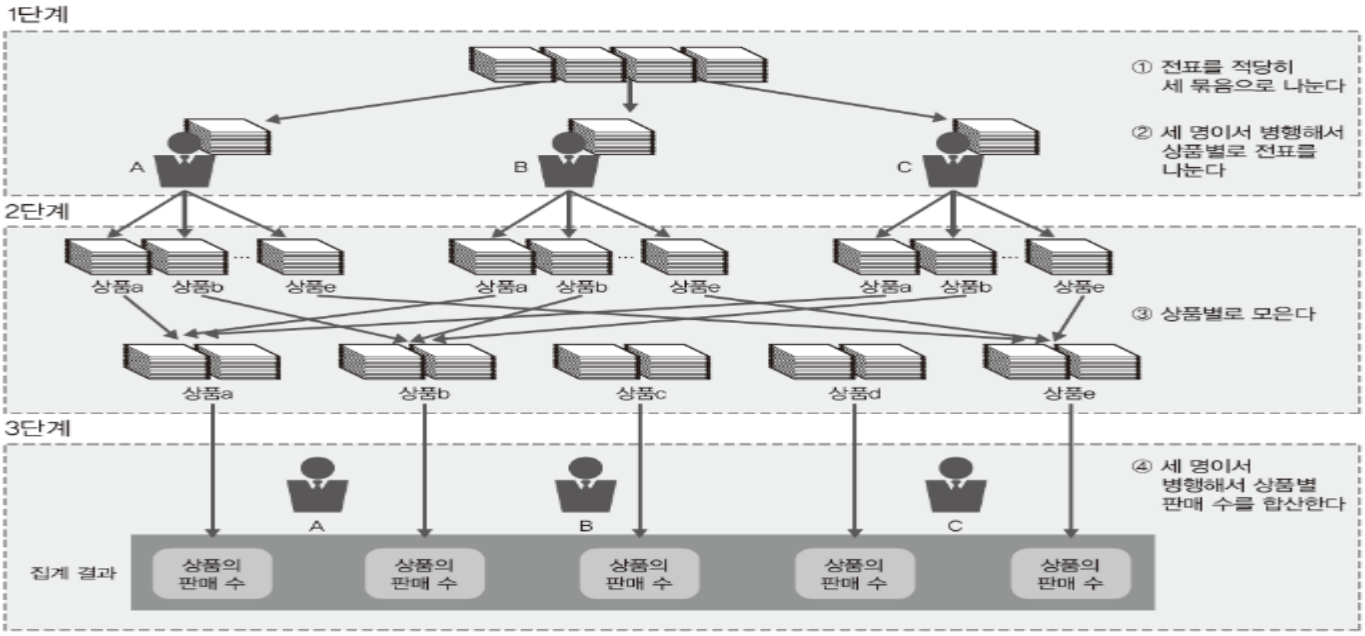


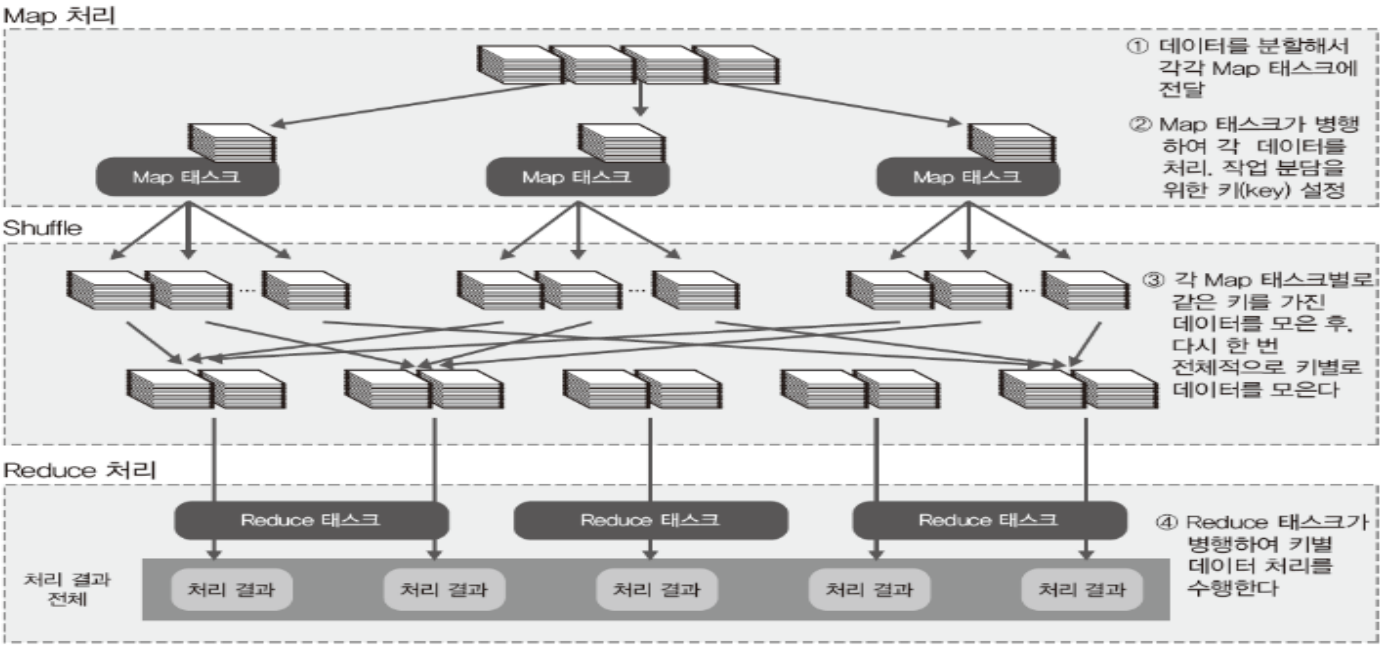
MapReduce는 Hadoop 클러스터의 데이터를 처리하기 위한 시스템으로 총 2개(Map, Reduce)의 단계로 구성된다. Map과 Reduce 사이에는 shuffle과 sort라는 단계가 존재한다. 각 Map task는 전체 데이터 셋에 대해서 별개의 부분에 대한 작업을 수행하게 되는데, 기본적으로 하나의 HDFS block을 대상으로 수행하게 된다. 모든 Map 태스크가 종료되면, MapReduce 시스템은 중간(intermediate) 데이터를 Reduce 단계를 수행할 노드로 분산하여 전송한다.

Distributed File System에서 수행되는 MapReduce 작업이 끝나면 HDFS에 파일이 써지고, MapReduce 작업이 시작할 때는 HDFS로 부터 파일을 가져오는 작업이 수행된다.



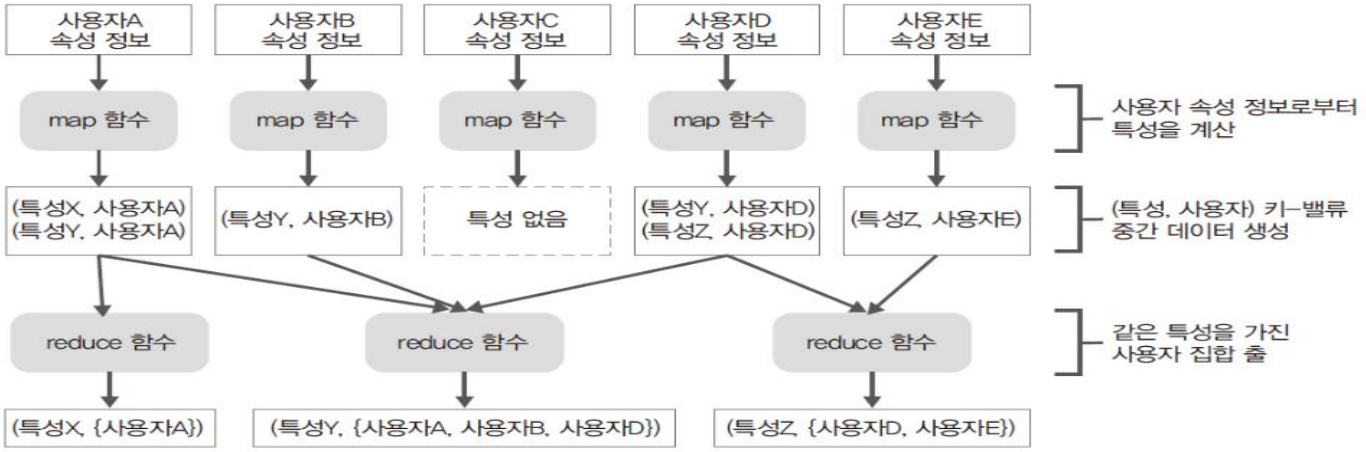
[MapReduce의 처리 개념 : 현실에서도 일어나는 일]





[MapReduce 응용]

1. 비슷한 특성의 사람 찾기



2. 키워드를 추출한 문서 인덱스 작성

문서1에 포함된 키워드와 문서 쌍

키워드	문서
Hadoop	문서1
Pig	문서1
Hive	문서1
Mahout	문서1

문서2에 포함된 키워드와 문서 쌍

키워드	문서
Hadoop	문서2
Java	문서2
Pig	문서2

문서3에 포함된 키워드와 문서 쌍

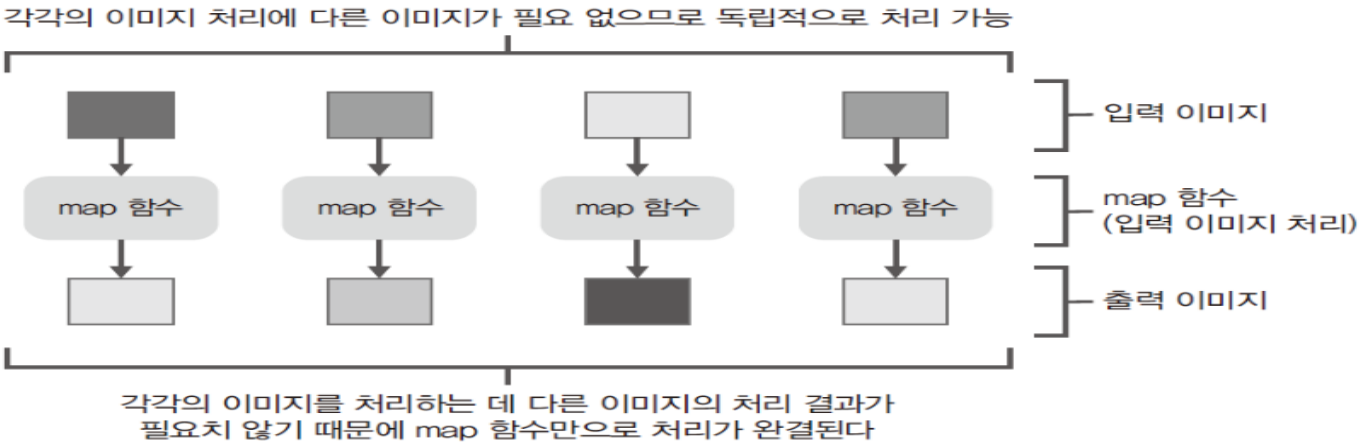
키워드	문서
Hadoop	문서3
Java	문서3
Scala	문서3
Clojure	문서3

키워드별로 병합

키워드	키워드를 포함하는 문서
Hadoop	문서1, 문서2, 문서3
Java	문서2, 문서3
Pig	문서1, 문서2
Hive	문서1
Mahout	문서1
Scala	문서3
Clojure	문서3

키워드를 포함하는 문서 리스트

3. 이미지 데이터 분산처리(개인 정보 모자이크 처리)



[MapReduce 정리]

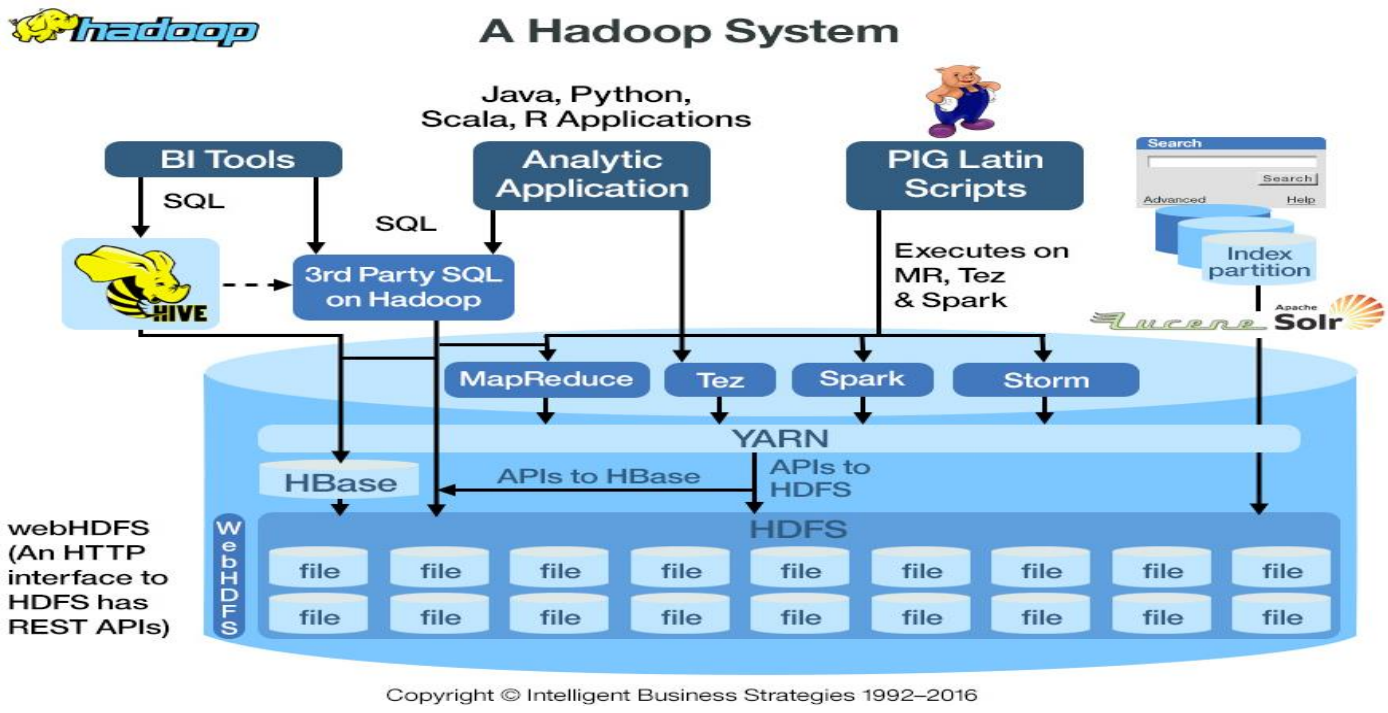
MapReduce는 데이터를 개별로 가공 및 필터링하거나, 어떤 키값에 기반해 데이터를 분류하거나, 분류한 데이터로 통계치를 계산하는 등, 수많은 데이터 처리에서 사용되고 있는 기법들을 일반화 하고 있다. map() 함수와 reduce() 함수는 한 번에 처리할 수 있는 데이터와 데이터 전달 방법 등이 다르다.

map() 함수는 처리 대상 데이터 전체를 하나씩, 하나씩 처리한다. 처리대상 데이터간에 의존관계가 없고 독립적으로 실행 가능하며 처리나 순서를 고려하지 않아도 되는 처리에 적합하다.(전처리)

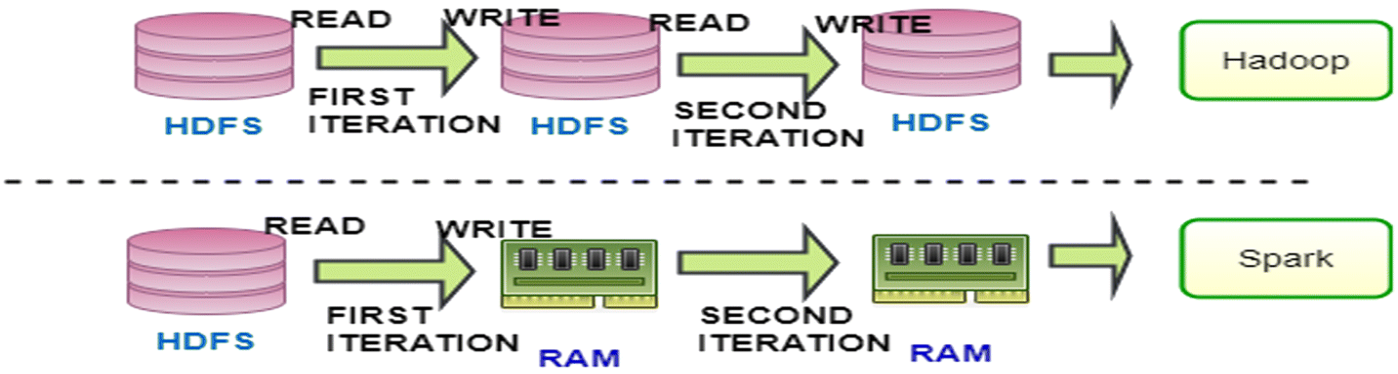
reduce() 함수에는 키와 연관된 복수의 데이터가 전달된다. 또한 reduce() 함수에 전달되는 데이터는 키값으로 정렬되어 있다. 그룹화된 복수의 데이터를 필요로 하는 처리 또는 순서를 고려해야 하는 처리에 적합하다.(그룹별 합계)

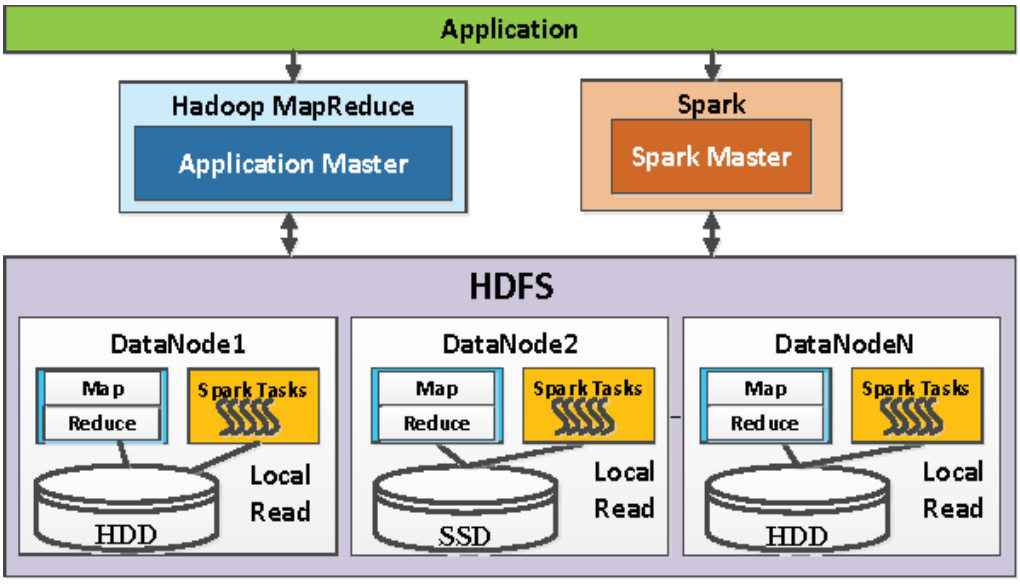
map : (K1, V1) -> list(K2, V2)

reduce : (K2, list(V2)) -> list(K3, V3)



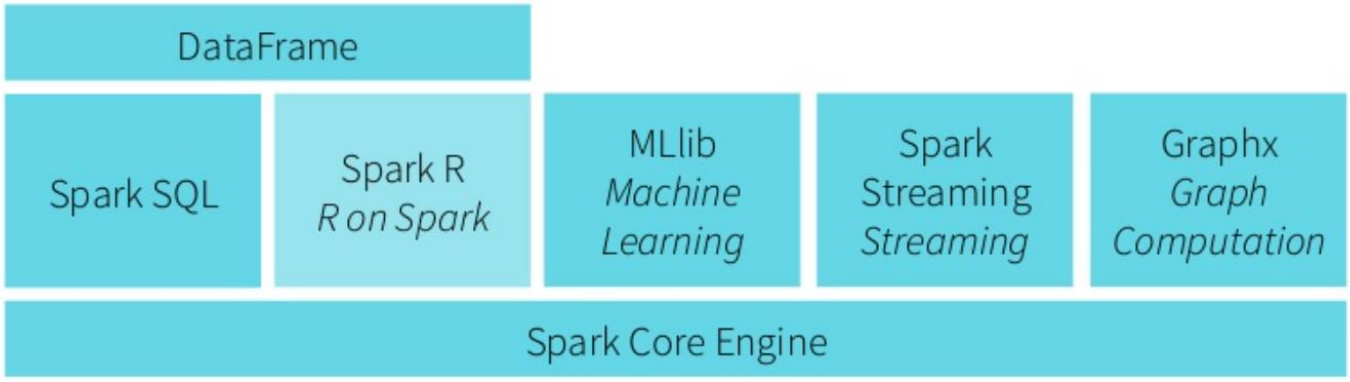
Apache Hadoop MapReduce와 Apache Spark



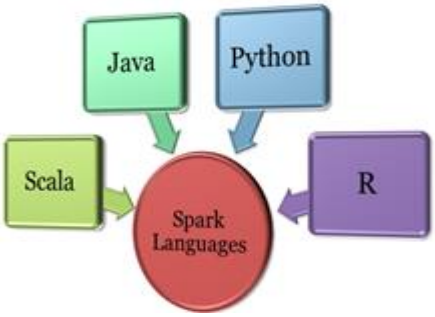


Apache Spark

Apache Spark는 메모리 내 처리를 지원하여 빅데이터를 분석하는 애플리케이션의 성능을 향상시키는 오픈 소스 병렬 처리 프레임워크이다. 빅데이터 솔루션은 기존 데이터베이스에 비해 너무 크거나 복잡한 데이터를 처리하도록 설계되었다.



Spark는 대량의 데이터를 고속 병렬 분산 처리한다. 데이터소스로부터 데이터를 읽어들인 뒤 스토리지 I/O와 네트워크 I/O를 최소화하도록 처리한다. 따라서 Spark는 동일한 데이터에 대한 변환처리가 연속으로 이루어지는 경우와 머신러닝처럼 결과셋을 여러 번 반복해 처리하는 경우에 적합하다.



[Spark의 데이터 처리 단위 : RDD]

데이터셋을 추상적으로 다루기 위한 RDD (Resilient Distributed Dataset: 내결함성 분산 데이터셋)이라는 데이터셋이 있으며, RDD가 제공하는 API로 변환과 액션 기능을 구현한다.

RDD(Resilient Distributed DataSet) :

Spark 에서 처리되는 데이터는 RDD 객체로 생성하여 처리한다.

RDD 객체는 두 가지 방법으로 생성 가능하다.

(1) Collection 객체를 만들어서

(2) HDFS 의 파일을 읽어서

RDD 객체의 특징 : Read Only(immutable)

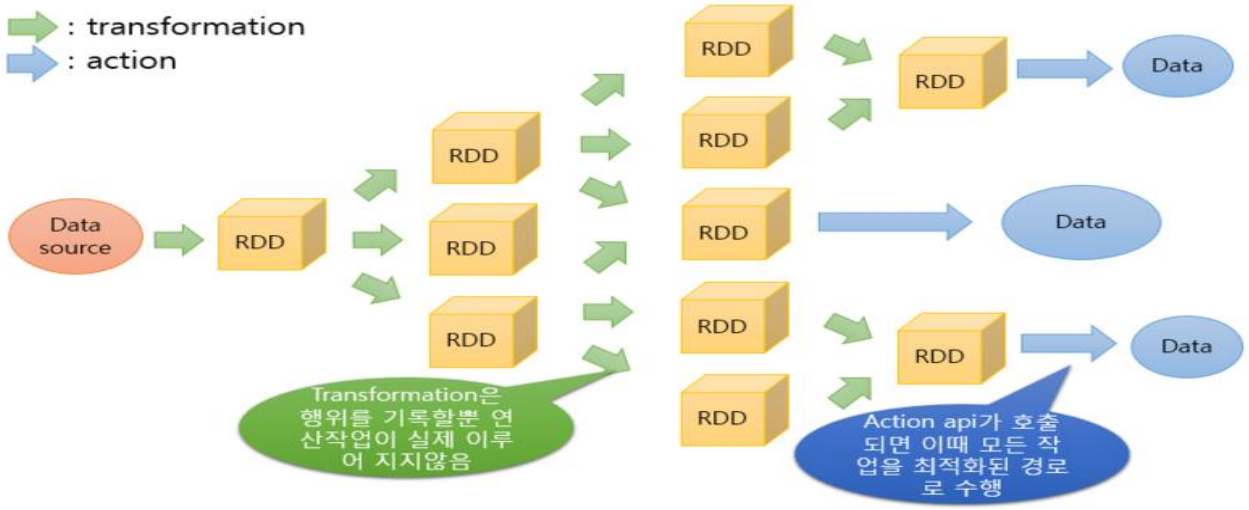
1~n 개의 partition 으로 구성 가능

병렬적(분산) 처리가 가능하다.

RDD의 연산은 Transformation 연산과 Action 연산으로 나뉜다.

Transformation은 Lazy-execution을 지원한다.

lineage를 통해서 fault tolerant(결함 내성)를 확보한다.



Transformations	Actions
map (func)	reduce(func)
flatMap(func)	collect()
filter(func)	count()
groupByKey()	first()
reduceByKey(func)	take(n)
mapValues(func)	saveAsTextFile(path)
sample(...)	countByKey()
union(other)	foreach(func)
distinct()	...
sortByKey()	
...	

[Transformation과 Action]

- 연산의 수행 결과가 RDD 이면 Transformation
- Transformation 은 기존 RDD를 이용해 새로운 RDD를 생성하는 연산이다.

(변환, 필터링 등의 작업들 : 맵, 그룹화, 필터, 정렬...)

Lineage를 만들어 가는 과정이다.

- 연산의 수행 결과가 정수, 리스트, 맵 등 RDD 가 아닌 다른 타입이면 Action

(first(), count(), collect(), reduce()...)

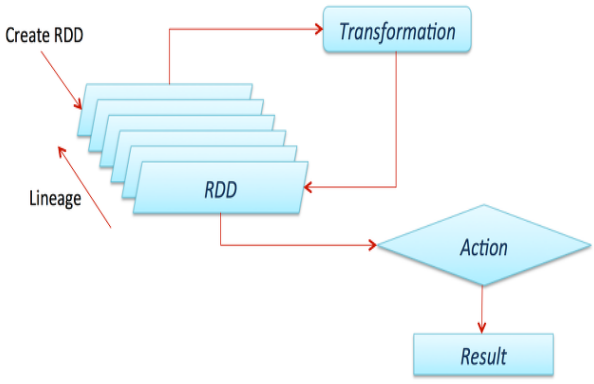
Lineage를 실행하고 결과를 만든다.

Lazy-execution 이란 Action 연산이 실행되기 전에는 Tranformation 연산이 처리되지 않는 것을 의미한다. Transformation 연산은 관련 메서드를 호출하여 연산을 요청해도 실제 수행은 되지 않고 연산 정보만 보관한다. 이렇게 Transformation 연산 정보를 보관한 것을 Lineage(리니지)라고 한다. 보관만 하다가 첫 번째 Action 연산이 수행될 때 모든 Lineage 에 보관된 Transformation 연산을 한 번에 처리한다.

Lazy-execution과 Lineage를 활용함으로써 처리 효율을 높이고 클러스터 중 일부 고장으로 작업이 실패해도 Lineage를 통해 데이터를 복구한다.

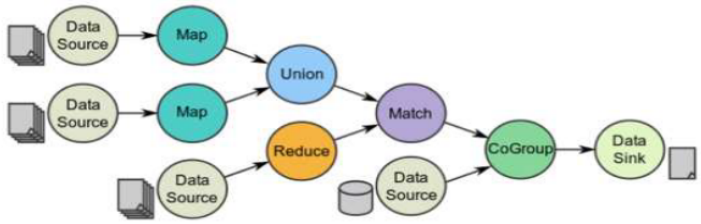
RDD를 제어하는 연산은 크게 2개의 타입

- Transformation : RDD에서 데이터를 조작하여 새로운 RDD를 생성하는 함수
- Action : RDD에서 RDD가 아닌 타입의 data로 변환하는 하는 함수



operation의 순서를 기록해 DAG로 표현한 것을 Lineage라 부름

Lineage의 예시



1. fault-tolerant 확보
   - **계보(lineage)**만 기록해두면 동일한 RDD를 생성할 수 있음
   - RDD의 copy를 보관하기 보다, Lineage만 보관해도 복구가 가능
   - 일부 계산 코스트가 큰 RDD는 디스크에 Check pointing

   reliablity 확보

2. Lazy-execution이 가능해짐
   - 인터프리터에서 Transformation 명령어를 읽어 들일 때는 단순히 lineage만 생성
   - Action 명령어가 읽히면, 쌓여있던 lineage를 실행

Transformations	*map*(*f* : T ⇒ U)	:	RDD[T] ⇒ RDD[U]
	filter(*f* : T ⇒ Bool)	:	RDD[T] ⇒ RDD[T]
	flatMap(*f* : T ⇒ Seq[U])	:	RDD[T] ⇒ RDD[U]
	sample(*fraction* : Float)	:	RDD[T] ⇒ RDD[T] (Deterministic sampling)
	groupByKey()	:	RDD[(K, V)] ⇒ RDD[(K, Seq[V])]
	reduceByKey(*f* : (V, V) ⇒ V)	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	union()	:	(RDD[T], RDD[T]) ⇒ RDD[T]
	join()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (V, W))]
	cogroup()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]
	crossProduct()	:	(RDD[T], RDD[U]) ⇒ RDD[(T, U)]
	mapValues(*f* : V ⇒ W)	:	RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)
	sort(*c* : Comparator[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	partitionBy(*p* : Partitioner[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
Actions	*count*()	:	RDD[T] ⇒ Long
	collect()	:	RDD[T] ⇒ Seq[T]
	reduce(*f* : (T, T) ⇒ T)	:	RDD[T] ⇒ T
	lookup(*k* : K)	:	RDD[(K, V)] ⇒ Seq[V] (On hash/range partitioned RDDs)
	save(*path* : String)	:	Outputs RDD to a storage system, *e.g.*, HDFS

Table 2: Transformations and actions available on RDDs in Spark. Seq[T] denotes a sequence of elements of type T.

parallelize() : 스칼라컬렉션 객체를 이용해서 RDD 객체를 생성한다.

count() : RDD의 요소 개수를 리턴한다.

first() : RDD 객체의 첫 번째 요소를 리턴한다.

collect() : RDD 객체의 모든 요소를 배열로 리턴한다

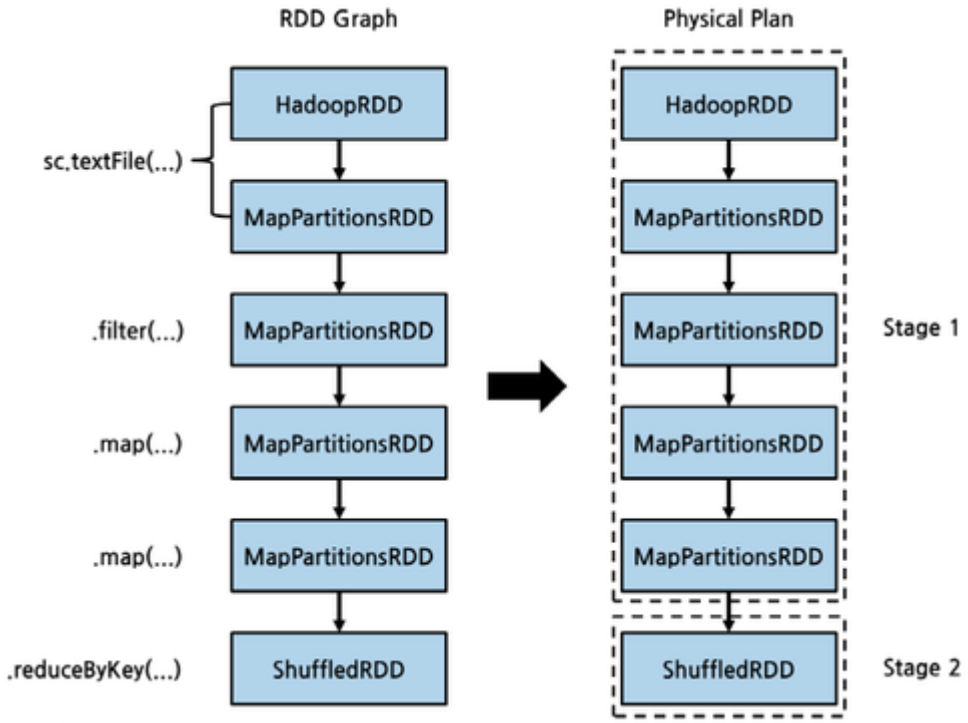
map() : 입력 RDD의 모든 요소에 f를 적용한 결과를 저장한 RDD를 반환한다.

flatMap() : 입력 RDD의 모든 요소에 f를 적용하고 모든 요소들을 하나로 묶어서 반환한다.

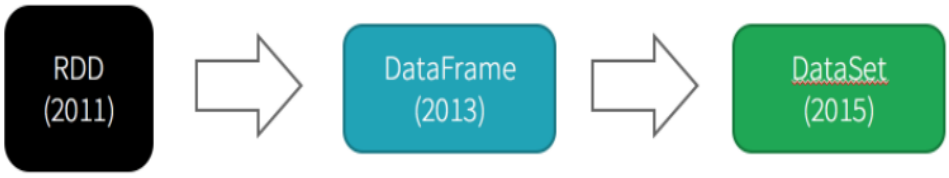
filter() : 입력 RDD의 모든 요소에 f를 수행하고 참을 리턴하는 결과만 저장한 RDD를 반환한다.

reduce() : 지정된 f를 수행시켜 입력 RDD의 개수를 축소시켜서 생성된 RDD를 반환한다.

[Spark이 작업을 처리하는 흐름 예]



[RDD 와 DataFrame 과 DataSet]



Apache Spark 는 특정한 데이터셋에 대하여 반복처리와 연속적으로 이루어지는 변환처리를 고속화할 목적으로 개발되었다.

- 반복처리 + 연속으로 이루어지는 변환처리의 고속화

- 시행착오에 적합한 환경 제공

- 서로 다른 처리를 통합해 이용할 수 있는 환경

PySpark

PySpark는 Apache Spark 기능을 사용하여 Python 애플리케이션을 실행하기 위해 Python으로 작성된 Spark 라이브러리이다. PySpark을 사용하면 분산 클러스터 (다중 노드)에서 애플리케이션을 병렬로 실행할 수 있다.

Spark은 Scala라는 언어로 작성되었으며 Spark 프로그래밍에는 Scala가 가장 많이 사용되고 있다. 또한 데이터 처리나 분석 프로그래밍에 많이 사용되는 Python 으로도 개발할 수 있게 Py4J를 사용하여 Python 용으로 출시 된 API 가 바로 PySpark 이다. Py4JPySpark 내에 통합 된 Java 라이브러리이며 Python이 JVM 개체와 동적으로 인터페이스 할 수 있도록 하므로 PySpark를 실행하려면 Python 뿐만 아니라 Java 개발 환경(JDK)도 설치되어 있어야 한다.

또한 개발을 위해 Spyder IDE , Jupyter 노트북(랩)과 같은 많은 유용한 도구와 함께 제공되는 Anaconda 배포 (머신 러닝 커뮤니티에서 널리 사용됨)를 사용하여 PySpark 애플리케이션을 실행할 수 있다.

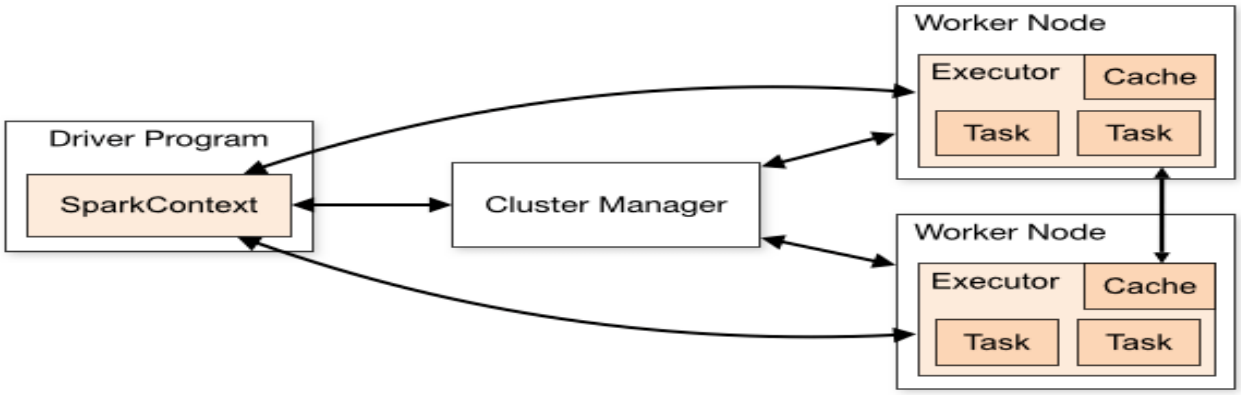
PySpark는 대규모 데이터 세트를 효율적으로 처리하기 때문에 NumPy, Pandas, TensorFlow를 포함하여 Python으로 작성된 데이터 터 과학 라이브러리와 함께 많이 사용됩니다.

[PySpark의 장점]

PySpark는 분산 방식으로 데이터를 효율적으로 처리 할 수 있는 범용 인 메모리 분산 처리 엔진이다.
PySpark에서 실행되는 애플리케이션은 기존 시스템보다 100 배 빠르다.
데이터 수집 파이프 라인에 PySpark를 사용하면 큰 이점을 얻을 수 있다.
PySpark를 사용하여 Hadoop HDFS, AWS S3 및 여러 파일 시스템의 데이터를 처리 할 수 있다.
PySpark는 Streaming 및 Kafka를 사용하여 실시간 데이터를 처리하는데도 사용된다.
PySpark 스트리밍을 사용하면 파일 시스템에서 파일을 스트리밍하고 소켓에서 스트리밍 할 수도 있다.
PySpark에는 기본적으로 기계 학습 및 그래프 라이브러리가 있다.

[PySpark 아키텍처]

Apache Spark는 마스터를 "드라이버"라 하고 슬레이브를 "작업자"라고 하는 마스터-슬레이브 아키텍처에서 작동한다. Spark 애플리케이션을 실행하면 Spark Driver가 애플리케이션의 진입점인 컨텍스트를 생성하고 모든 작업 (변환 및 작업)이 작업자 노드에서 실행되고 리소스는 Cluster Manager에서 관리된다.



[PySpark 모듈 및 패키지]

PySpark RDD (pyspark.RDD)

PySpark DataFrame 및 SQL (pyspark.sql)

PySpark 스트리밍 (pyspark.streaming)

PySpark MLib (pyspark.ml , pyspark.mllib)

PySpark GraphFrames (GraphFrames)

PySpark 리소스 (pyspark.resource) PySpark 3.0의 새로운 기능