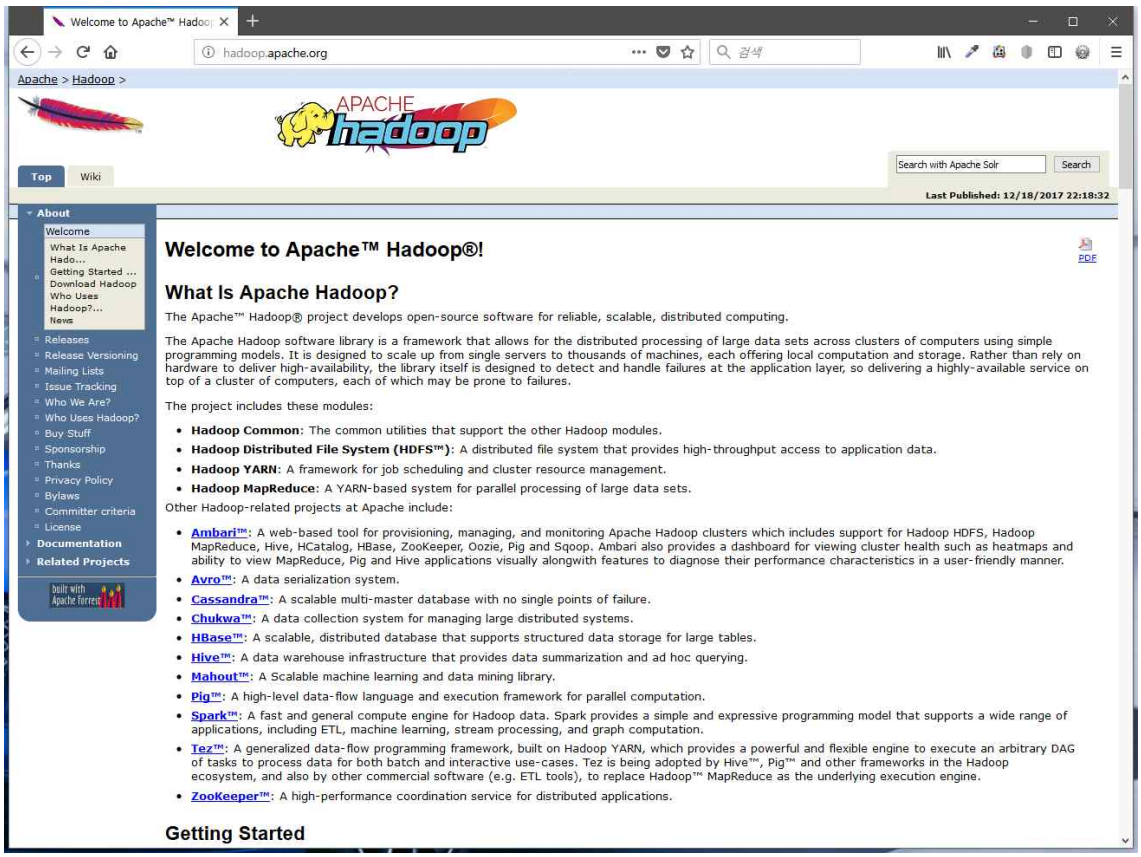
■ Hadoop

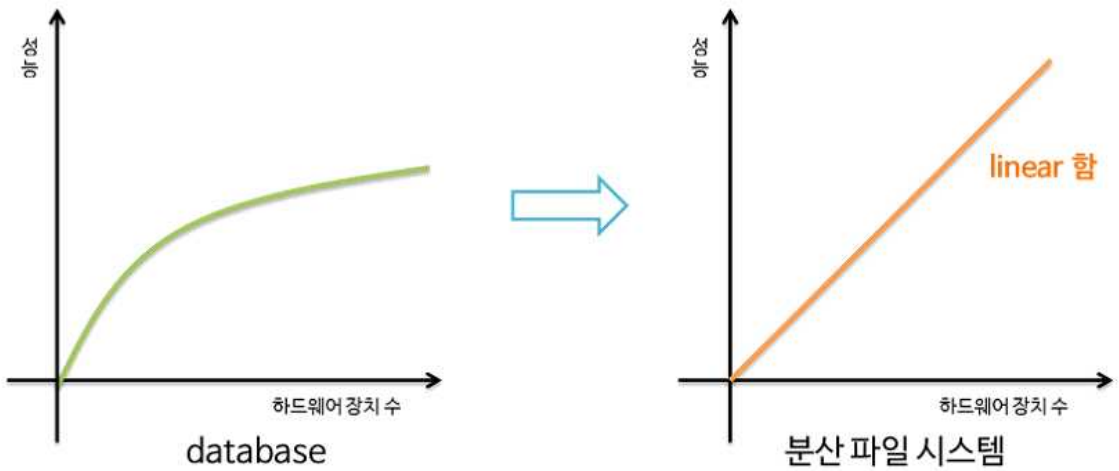


1) 하둡(Hadoop)은 대용량 데이터를 분산처리 할 수 있는 자바 기반의 오픈소스 프레임워크

2) 하둡은 구글이 논문으로 발표한 GFS(Google File System)와 맵리듀스(MapReduce)를 2005년에 더그 커팅(Doug Cutting)이 구현한 결과물

3) 하둡은 분산 파일 시스템인, HDFS(Hadoop Distributed File System)에 데이터를 저장하고 분산 처리 시스템인 맵리듀스(MapReduce)를 이용해 데이터를 처리

4) 큰 데이터를 처리하기에 기존의 슈퍼컴퓨터 등을 이용하는 방식은 큰 비용이 들었으며, 데이터베이스는 하드웨어 장치의 갯수와 성능이 비례하지 않았음

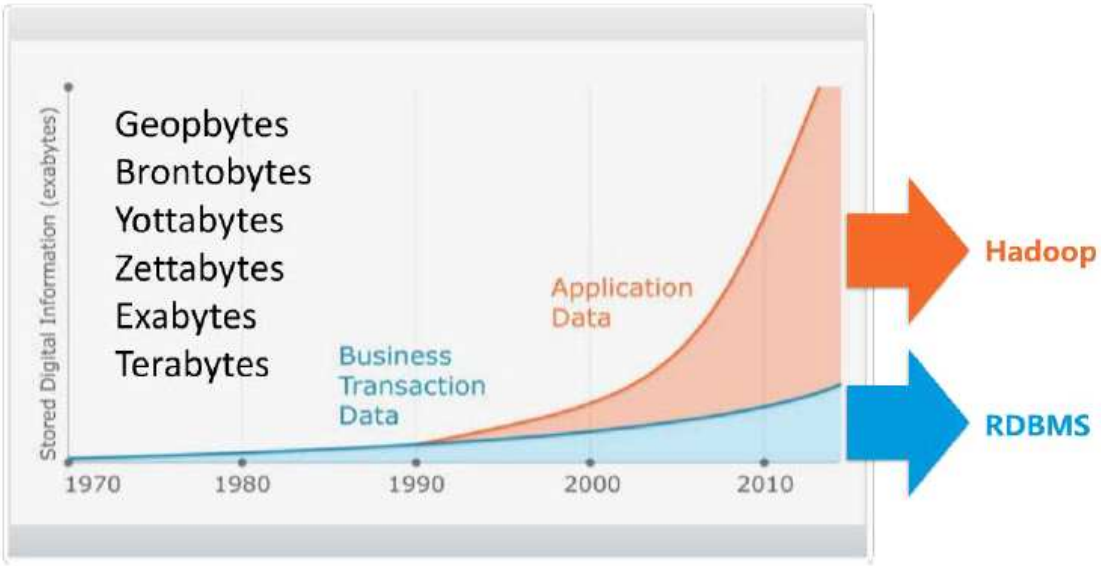


5) 하지만 분산파일 시스템은 하드웨어가 늘어날수록 성능도 비례함

6) 즉, 하둡은 데이터를 값싸게 처리하면서 고성능을 발휘함

■ 하둡을 왜 빅데이터에서 사용하는가?

(정형 데이터의 경우 기존의 관계형 데이터베이스(RDBMS)에 저장할 수 있지만 웹 로그 같은 비정형 데이터를 RDBMS에 저장하기에는 데이터 크기가 너무 크다. 상용 RDBMS가 설치되는 곳은 대부분 고가의 장비인데, 데이터를 감당하기 위해 무한정 저장공간을 늘리기는 힘들다.)



특 성	RDBMS	Hadoop
데이터 크기	Gigabyte	Petabyte (RDBMS의 100배)
접근	Interactive and batch 데이터 추가 발생시 업데이트 하는 경우가 대다수. 그러나 특정 경우에 일정 시간마다 주기적으로 업데이트 하는 경우도 존재	Batch 일정 시간마다 주기적으로 업데이트
업데이트	읽고 쓰기를 빈번하게 함	한번 쓰고 여러 번 읽어 들인다.
구조	Static schema 테이블을 만들고 데이터를 집어넣음 한번 테이블을 만들면 테이블을 잘 바꾸지 않음. (정적인 스키마를 가짐)	Dynamic schema 정형 반정형 비정형 모두 다루기 때문에 동적 스키마를 가짐. 자유롭다.
통합 정도	높음	낮음
성능이 늘어나는 방식	Non-linear	Linear

하둡은 오픈소스 프로젝트이기에 라이선스 비용이 없음

:x86 CPU에 리눅스 서버이면 얼마든지 하둡을 설치해서 운영할 수 있음(데이터 저장 용량이 부족할 경우, 필요한 만큼 리눅스 서버를 추가), 하둡은 데이터의 복제본을 저장하기 때문에 데이터의 유실이나 장애가 발생했을 때 복구가 가능

- 하둡은 여러 대의 서버에 데이터를 저장하고 데이터가 저장된 서버에서 동시에 데이터를 처리

(2008년 뉴욕 타임즈는 130년 분량의 신문기사 1,100만 페이지를 아마존 EC2, s3, 하둡을 이용

해 하루 만에 PDF로 변환함. 이때 소요된 비용은 200만원. 만약 이 변환 작업을 그 당시 일반 서버로 진행할 경우, 약 14년이 소요될 정도로 엄청난 작업량이었음)

- 저렴한 구축 비용과 비용 대비 빠른 데이터 처리, 그리고 장애를 대비한 특성은 하둡의 장점. 이 특성은 빅데이터의 3V인, 데이터의 크기(Volume), 속도(Velocity), 다양성(Velocity)을 만족시킴. 하둡은 초기에 야후에서만 주도적으로 사용됐지만 현재는 아마존, 이베이, 페이스북, 구글 등 글로벌 서비스 업체에서 주로 이용하고 있으며, 국내에서는 네이버나 다음카카오 같은 포털 기업과 KT나 SKT 같은 이동통신사에서 사용됨. 현재 야후에서는 약 5만 대, 페이스북에서 1만 대 이상의 하둡 클러스터를 운영하고 있음

■ HDFS(Hadoop Distributed File System)

- 하둡 분산 파일 시스템

- 대용량 파일을 분산된 서버에 저장하고, 저장된 데이터를 빠르게 처리할 수 있게 설계된 파일 시스템

■ HDFS의 목표

① 대용량 데이터 저장

- HDFS는 하나의 파일이 기가바이트에서 테라바이트 이상의 크기로 저장될 수 있음

② 데이터 무결성

- 데이터베이스에서 데이터 무결성이란 데이터베이스에 저장되는 데이터의 일관성을 의미
- 데이터의 입력이나 변경 등을 막아 데이터의 안정성을 유지
- HDFS에서는 한 번 저장된 데이터는 수정할 수 없고 읽기만 가능해서 데이터 무결성을 유지
-데이터 수정은 불가능하지만 파일 이동, 삭제, 복사는 가능

③ 장애 복구

- HDFS는 장애를 빠른 시간에 감지하고 대처 가능
- 데이터를 저장하면 복제 데이터도 함께 저장되어 데이터 유실을 방지하고 분산 서버간에 주기적으로 상태를 체크해 빠른 시간에 장애를 인지
(기본 블록당 3개를 복제하여 다른 데이터 노드에 저장)

④ 스트리밍 방식의 데이터 접근

- HDFS는 클라이언트의 요청을 빠른 시간 내에 처리하는 것보다 동일한 시간 내에 더 많은 데이터를 처리
- 클라이언트는 끊김 없이 연속된 흐름으로 데이터에 접근

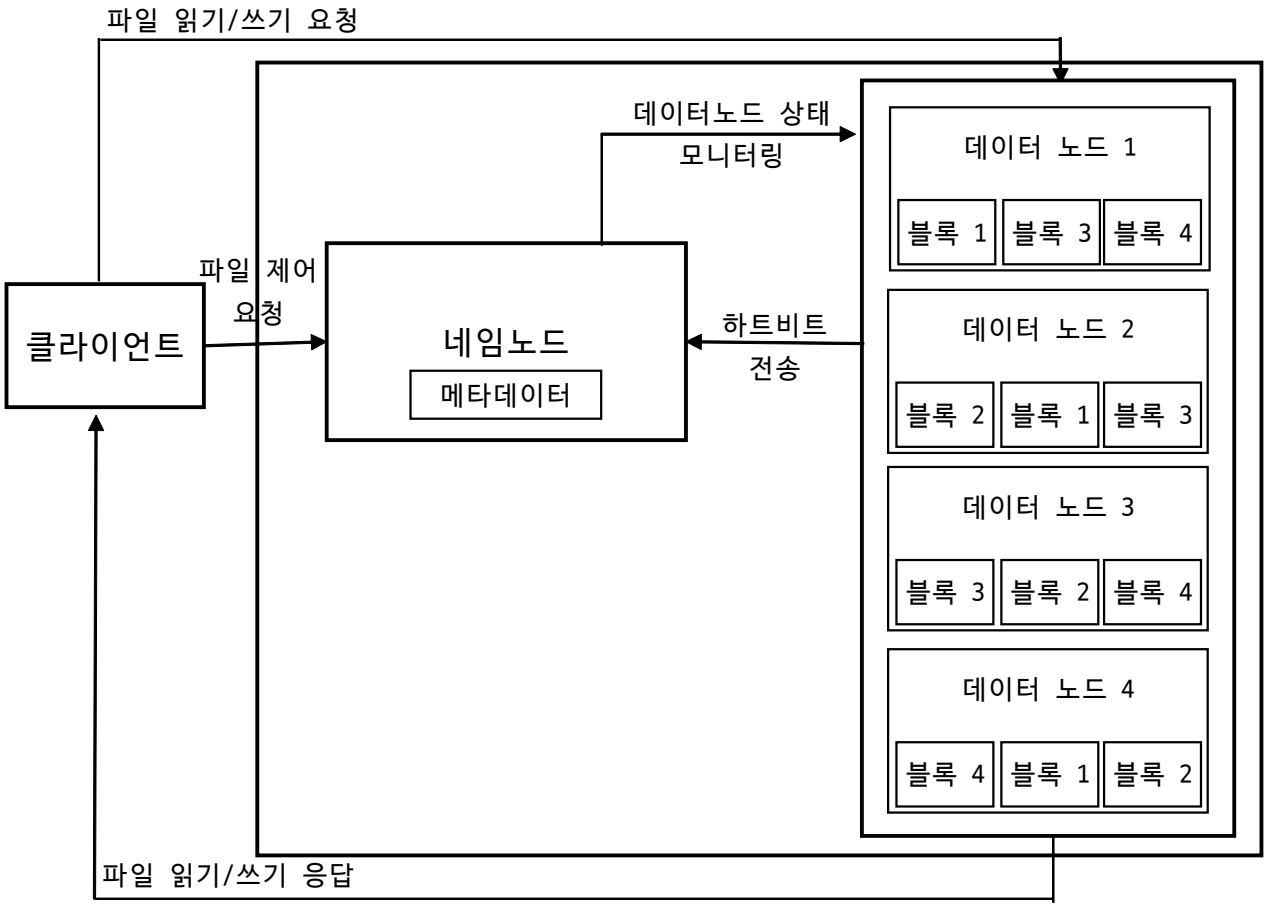
■ HDFS 아키텍처

1) 블록 구조 파일 시스템

- 파일은 블록(block) 단위로 분할
- 각 블록은 기본적으로 64MB(하둡 1.0 기준) 또는 128MB(하둡 2.0 기준) 크기
- 디스크 시크타임(Disk seek time)의 감소
- 네임노드가 유지하는 메타데이터(블록 위치, 파일명, 디렉터리 구조, 권한 정보 가 있는 데이터)의 크기 감소
- HDFS는 블록을 저장할 때 기본적으로 3개씩 블록의 복제본을 저장(복제본 수는 변경 가능)
- 특정 서버의 하드디스크에 오류가 생겨도 복제된 블록을 이용해 데이터를 조회

2) 네임노드와 데이터노드

- HDFS는 마스터(Master)와 슬레이브(Slave) 구조
- 1개의 하드웨어(마스터)가 1개 이상인 다른 기기(슬레이브)를 제어하는 구조
- 마스터 역할을 수행하는 네임노드와 슬레이브 역할을 수행하는 데이터 노드들로 구성



3) 네임노드 역할

① 메타 데이터(파일 시스템이미지+파일 블록매핑정보) 관리

- 파일 시스템 이미지: 파일명, 디렉터리, 크기, 권한
- 메모리에 저장된 메타데이터의 파일 시스템 이미지를 저장한 파일(스냅샷)
- 파일에 대한 블록 매핑 정보: 어떤 블록이 어느 데이터노드에 저장되어있는 지
- 메모리 전체에 메타데이터를 로딩

② 데이터 노드 모니터링

- 데이터 노드가 3초마다 보내는 하트비트(heartbeat)를 네임노드가 받음
- 하트비트로 네임노드는 데이터 노드의 실행상태와 용량 체크
- 하트비트 전송 X -> 장애 감지

③ 블록관리

- 장애가 발생한 데이터 노드의 블록들을 새로운 데이터 노드로 복제
- 용량이 부족한 데이터 노드가 있으면 여유있는 데이터 노드로 블록 이동
- 정해진 복제본 수와 일치하지 않는 블록의 복제본 삭제 혹은 복제

④ 클라이언트 요청 접수

4) DataNode 역할

① 클라이언트가 HDFS에 저장하려는 데이터를 로컬디스크에 유지(DISK)

- 메타데이터: 체크섬, 파일 생성 일자
- 로우데이터 : 실제 데이터가 저장

5) 네임노드의 약점

- 네임노드는 메모리에 '파일 시스템 메타 데이터'를 관리
- 서버가 꺼지면 '파일 시스템 메타 데이터'는 사라짐
- 서버를 다시 켜도 '파일 시스템 메타 데이터'를 관리하기 위해 네임노드에는 editslog와 fsimage라는 파일을 저장
- 에디트 로그(editslog): HDFS의 모든 변경이력이 담겨 있음 HDFS에 저장된 파일을 수정하면 네임노드에 에디트 로그 만들어짐
- Fsimage : '파일 시스템 메타 데이터'의 스냅샷
- 네임노드가 시작이 되면 Fsimage를 메모리로 로딩
- Fsimage는 에디트로그를 반영함

- 에디트 로그가 너무 커져버리면 Fsimage 파일을 메모리로 로딩하는 시간이 느려짐

6) 보조 네임노드 역할

- 하둡 1.0 버전에서 주로 사용됨(네임노드가 오로지 하나만 작동 될 때 사용)
- 주기적으로(한시간 마다) Fsimage를 갱신하는 역할을 함(체크포인트라고도 함)

■ 네임노드 HA(High Availability:고가용성)

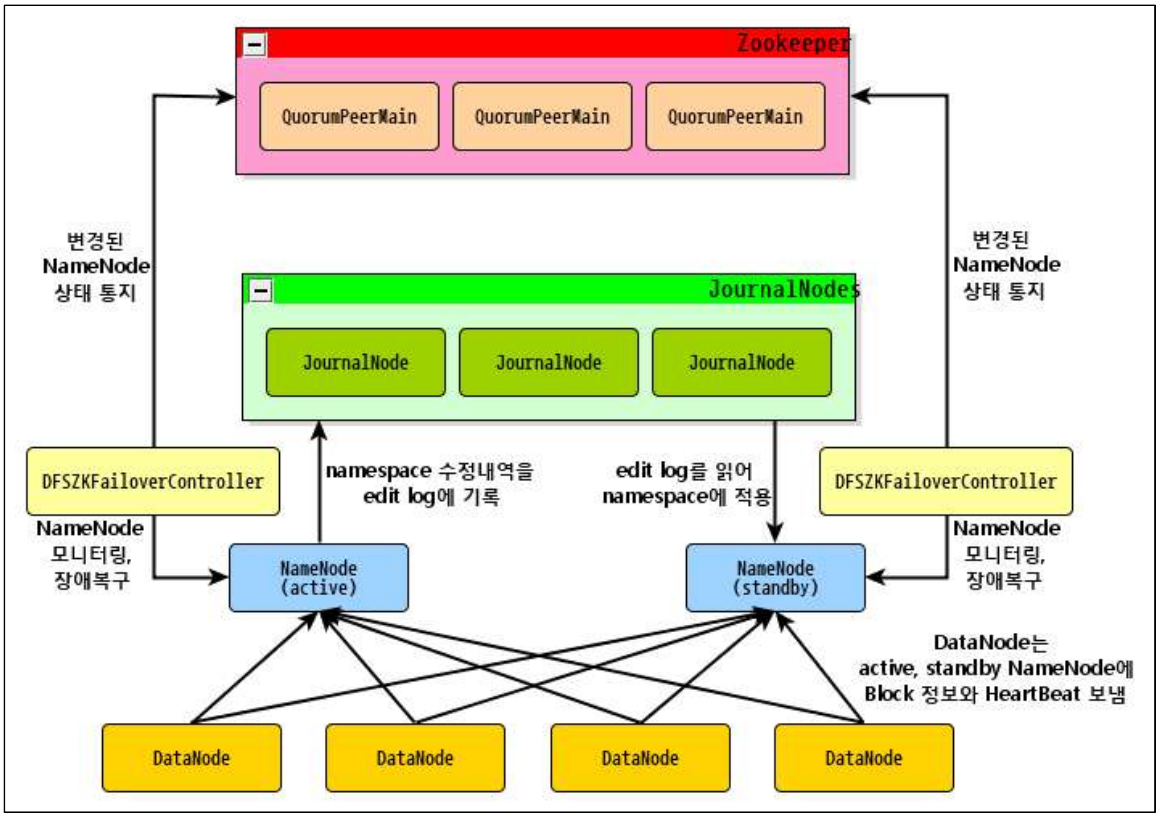
1) 필요성

- 하둡 1.0에서는 네임노드 한대 여러대의 데이터노드들로 구성이 되었음
- SPOF(Single Point Of Failure, 단일 고장점): 네임노드가 정상적으로 작동하지 않으면 모든 클라이언트가 HDFS에 접근 불가
- 네임노드 파일 시스템 이미지에 문제가 생겨도 HDFS에 저장된 데이터에 조회 불가능
- 파일 시스템 이미지에 HDFS의 디렉토리 구조와 파일 위치가 저장되어 있기 때문에 네임노드에 문제가 생기면 블록에 접근할 수 있는 통로가 사라짐
- 네임노드의 에디트로그에 문제가 생겨도 데이터 유실이 될 수 있음
- 에디트 로그(editslog): HDFS의 모든 변경이력이 담겨 있음 HDFS에 저장된 파일을 수정하면 네임노드에 에디트로그 만들어짐
- 에디트 로그는 파일시스템이미지에 반영이 되기 때문에 문제가 생기면 HDFS가 변경된 이력이 적용이 안될 수 있음
- 보조네임노드는 체크포인팅 작업을 통해 에디트로그를 파일시스템이미지에 갱신하지만, 체크포인트가 만들어지기 전에 에디트로그가 손상되고 네임노드가 재시작되면 손상된 에디트로그는 Fsimage에 반영이 안된 채 네임노드 구동

○ 하둡 2.0

- 네임노드를 이중화(Active 네임노드, Standby 네임노드)를 통해 네임노드의 SPOF(Single Point Of Failure) 문제 해결
- 에디트로그의 저장을 담당하는 저널노드를 생성해 네임노드의 에디트로그 대한 부담을 줄임

2) 네임노드 HA 주요 컴포넌트



① 저널노드

- 하둡 1.0에서는 네임노드에서만 에디트 로그 저장 -> 하둡 2.0에서는 여러 서버에 에디트 로그를 복제해서 저장
- 저널노드(데몬)는 에디트 로그를 자신이 실행되는 서버의 로컬디스크에 저장
- 네임노드는 클라이언트가 돼서 저널노드에게 접근해 저장을 요청
- 단 액티브 네임노드만 에디트로그를 저장할 권한이 있음
- 스탠바이 네임노드는 조회만 요청 가능
- 저널노드는 3대 이상의 서버에서 실행되어야하고 홀수 단위로만 실행가능
- 네임노드가 저널노드의 장애에 영향을 받지 않으려면 "(전체 저널노드 설치대수/2) / +1" 만큼 실행
- 예) 5대에 저널 노드를 설치했으면 최소 3대 이상의 저널노드가 실행
- 리소스를 적게 사용해서 "네임노드", "잡트래커", "리소스매니저"와 같은 데몬이 돌아가는 서버에서 함께 실행가능

② 주키퍼

- 네임노드 HA 상태 정보를 저장하는 장소
- 어떤 서버가 액티브(active) 네임노드인지 스탠바이(standby) 네임노드인지 저장
- 분산시스템 코디네이터

- 애플리케이션들이 잘 돌아가도록 중재
- 분산된 서버 간의 정보 공유
- 새로운 서버를 열거나 제거했을 때 다른 노드들에게 알려줌
- 서버 모니터링 : 연결이 끊어진(장애가 발생했을) 노드 삭제
- 시스템관리, 분산 락(Lock) 처리, 네이밍서비스 등등
- 주키퍼 마스터(주키퍼 앙상블에 구성된 주키퍼 한대)는 홀수 단위로 실행
- 주키퍼 클라이언트는 주키퍼 마스터가 응답을 안할 경우 다른 주키퍼 마스터 에게 요청(2대중 한대가 고장나면 실행 안됨)
- 주키퍼 마스터는 동일한 주키퍼 데이터를 복제하고 있기 때문에 클라이언트로부터 조회요청이 들어오면 자신이 보관한 데이터를 이용해 응답
- 쓰기 요청은은 오로지 리더로 설정된 주키퍼 마스터에게 보내짐
- ZNode: 주키퍼에서 저장되는 파일 하나하나를 ZNode라고 함

③ ZKFC(ZookeeperFailoverController)

- 로컬 네임노드(자신이 위치한 네임노드)의 상태를 모니터링
- 주키퍼 세션 관리
- 액티브 네임노드의 상태가 정상이면 주키퍼 마스터에 대한 세션 유지
- 자동 장애 처리(failover) : 액티브 네임 노드에 장애가 발생하면 감지해서 자동으로 zkfc와 주키퍼 마스터 간의 세션 종료 -> 스탠바이 네임 노드를 액티브 네임 노드로 전환 -> 기존의 액티브 네임 노드 제거

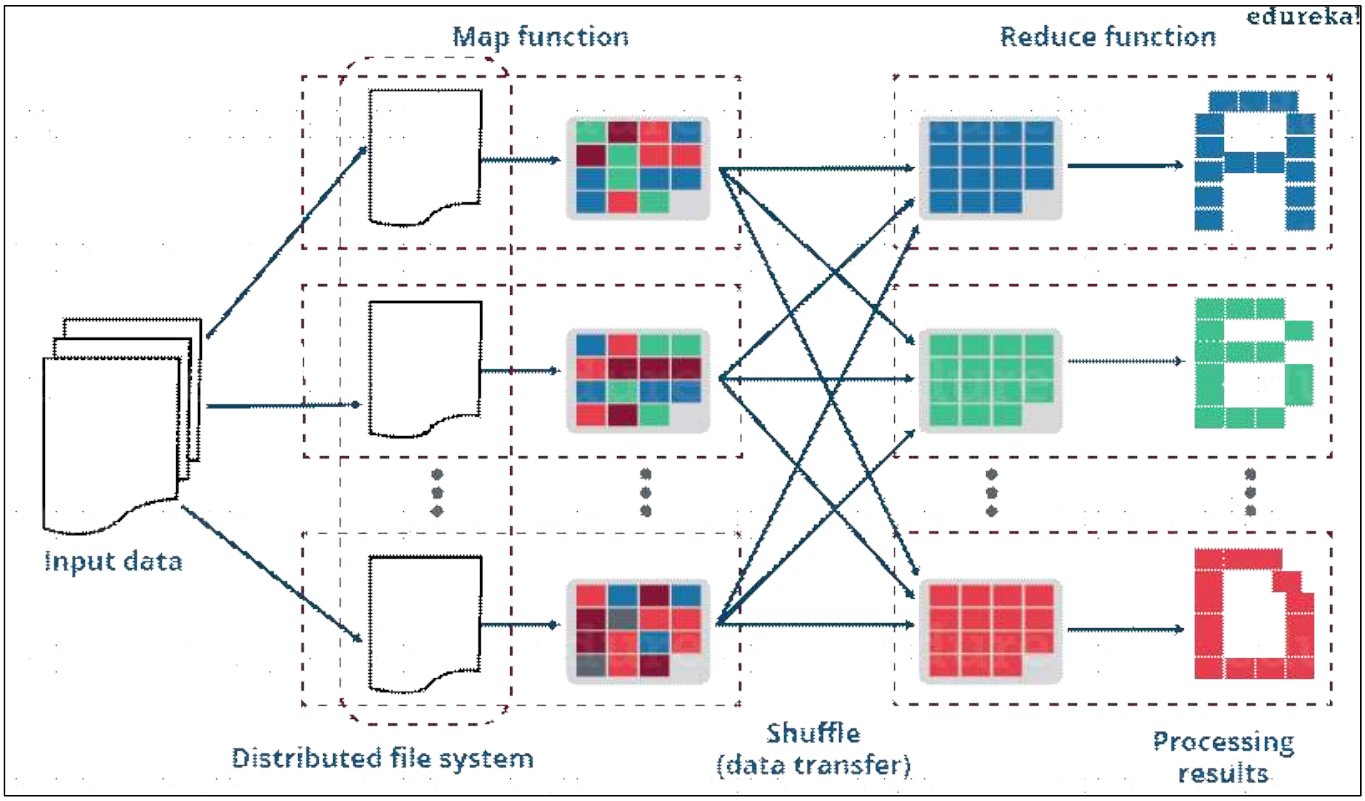
④ 네임노드

- 네임노드 내부에 있는 QJM(QuorumJournalManager)이 저널노드에 에디트 로그 출력
- 반드시 절반이상의 저널노드가 실행되고 있어야 에디트 로그를 fsimage에 반영함 (3대중 2대는 최소 작동)
- 액티브 네임노드만 저널노드에 에디트로그 쓸수 있음
- 스탠바이 네임노드는 저널노드에서 에디트로그 조회하고 fsimage를 갱신 -> 그래서 보조 네임노드를 실행할 필요가 없음

⑤ 데이터 노드

- 액티브 네임노드, 스탠바이 네임노드 모두 블록 리포트 전송

■ 맵리듀스(MapReduce)



1) 구글에서 대용량 데이터 처리를 분산 병렬 컴퓨팅에서 처리하기 위한 목적으로 제작하여 2004년 발표한 소프트웨어 프레임워크

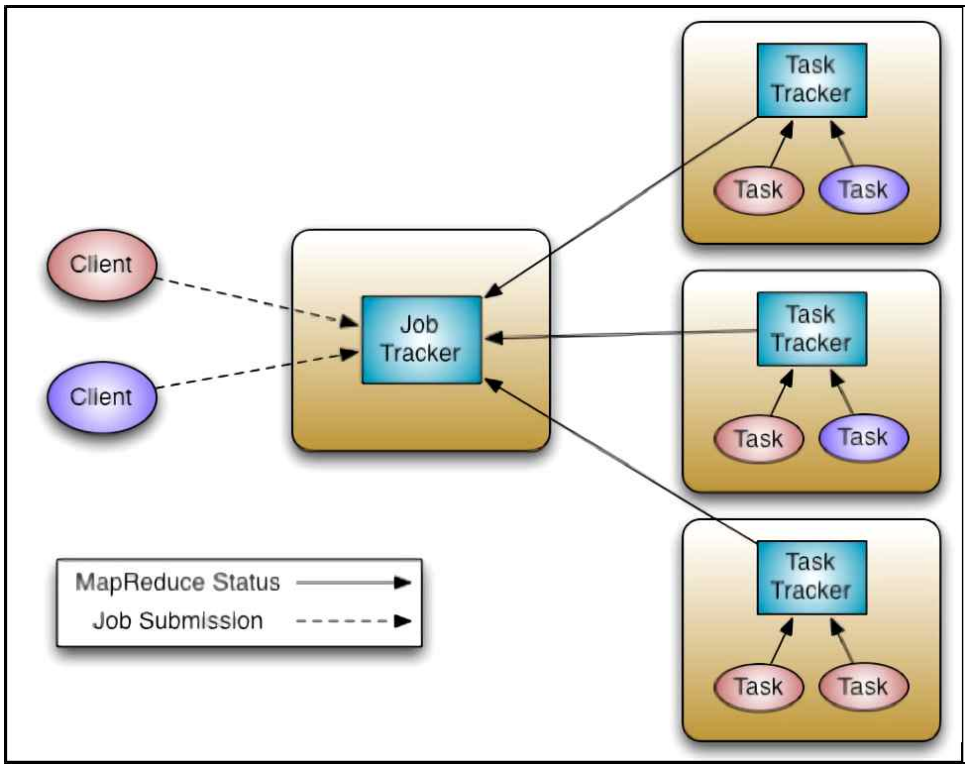
2) 페타바이트 이상의 대용량 데이터를 신뢰도가 낮은 컴퓨터로 구성된 클러스터 환경에서 병렬 처리를 지원하기 위해서 개발

3) HDFS에 저장된 파일을 분산 배치 분석을 할 수 있게 도와주는 프레임워크

4) 맵(Map) : 입력파일을 한 줄씩 읽어서 데이터를 변형

5) 리듀스(Reduce) : 맵의 결과 데이터를 집계(즉, 모은다)

6) 맵리듀스 아키텍쳐(하둡 1.0 기준)



① 클라이언트

- 사용자가 실행한 맵리듀스 프로그램과 하둡에서 제공하는 맵리듀스 API
- 사용자는 맵리듀스 API로 맵리듀스 프로그램 개발, 하둡에서 실행

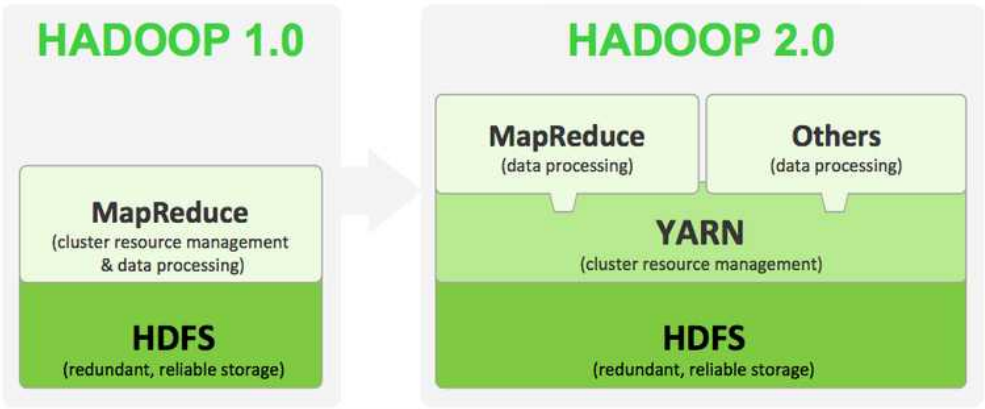
② 잡트래커

- 잡(job): 클라이언트가 하둡에 실행을 요청할 때 발생되는 맵리듀스 프로그램은 job이라는 단위로 관리
- 하둡 클러스터에 등록된 전체 잡의 스케줄링을 관리/모니터링
- 스케줄링 관리
- 잡을 처리하기 위해 몇 개의 맵과 리듀스를 실행할지 계산
- 계산된 맵과 리듀스는 어떤 태스크트래커에서 실행할지 결정한 뒤 잡을 나눔
- 스케줄링 모니터링
- 하트비트로 태스크트래커와 통신하면서 상태과 작업 실행정보를 받음
- 태스크트래커에 장애 생기면 다른 대기중인 태스크 트래커에게 태스크 실행
- 전체 하둡 클러스터에서 하나만 실행됨
- 보통 네임노드 서버에 설치하지만 따로 설치해도 됨

③ 태스크트래커

- 하둡의 데이터노드 서버에서 실행되는 데몬(데몬: 백그라운드에서 여러 작업을 수행하는 프로그램, 사용자가 직접 제어하지 않음)
- 사용자가 설정한 맵리듀스 프로그램 실행
- 잡트래커가 명령을 하면 요청 받은 맵과 리듀스 수만큼의 맵 태스크와 리듀스 태스크를 생성
- 맵 태스크, 리듀스 태스크: 사용자가 만든 맵과 리듀스 프로그램

■ YARN(Yet Another Resource Negotiator, 또다른 리소스 협상가)



1) 맵리듀스의 문제

- 하둡 1.0의 맵리듀스 프레임워크는 무조건 맵리듀스 API로 만든 프로그램만 실행
- 맵리듀스 SPOF(Single Point Of Failure, 단일 고장점) 문제: 잡트래커에 문제가 생기면 테스크트래커가 작동중이라도 맵리듀스를 사용 못함
- 잡트래커에 메모리가 부족하면 잡의 상태 모니터링을 못하고 새로운 잡도 실행 요청 못함
- 맵리듀스는 슬롯(한번에 실행되는 태스크들의 묶음)이란 개념으로 클러스터에서 실행할 수 있는 태스크의 개수를 관리 즉, 태스크 수행 단위
- 맵 슬롯과 리듀스 슬롯으로 구분되는데 실행중인 잡이 맵 슬롯만 사용하고 있거나 리듀스 슬롯만 사용한다면 다른 슬롯은 잉여자원이 되서 리소스가 낭비
- 맵리듀스 리소스는 맵리듀스 기반의 프레임워크만 자원을 공유해서 다른 에코시스템은 자원을 공유할 수 가 없음
- 맵과 리듀스 로 정해진 구조 외에 다른 알고리즘을 지원하는데 한계가 있음
- 맵리듀스 잡을 실행하는 클라이언트와 맵리듀스 클러스터 버전이 반드시 동일해야함

2) YARN의 목표

- 잡트래커의 주요기능 추상화
- 잡트래커의 주요 기능인 클러스터 자원 관리와 애플리케이션 라이프 사이클 관리를 분리
- 다양한 데이터 처리 애플리케이션 수용
- 기존 맵리듀스는 반드시 맵리듀스 API로 구현된 프로그램만 실행
- 얀에서 맵리듀스는 실행되는 애플리케이션들 중 하나일 뿐임

3) YARN의 장점

① 확장성

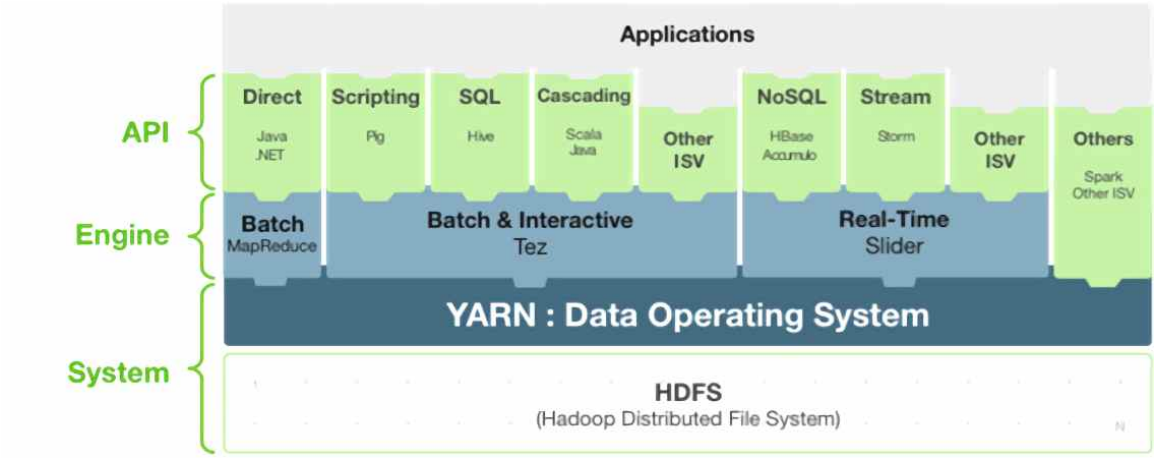
- 수용가능한 단일 클러스터 규모가 10000 노드까지 확대됨 (전보다 두배 이상)
- 데이터 처리 작업의 개수도 증가

② 클러스터 활용 개선

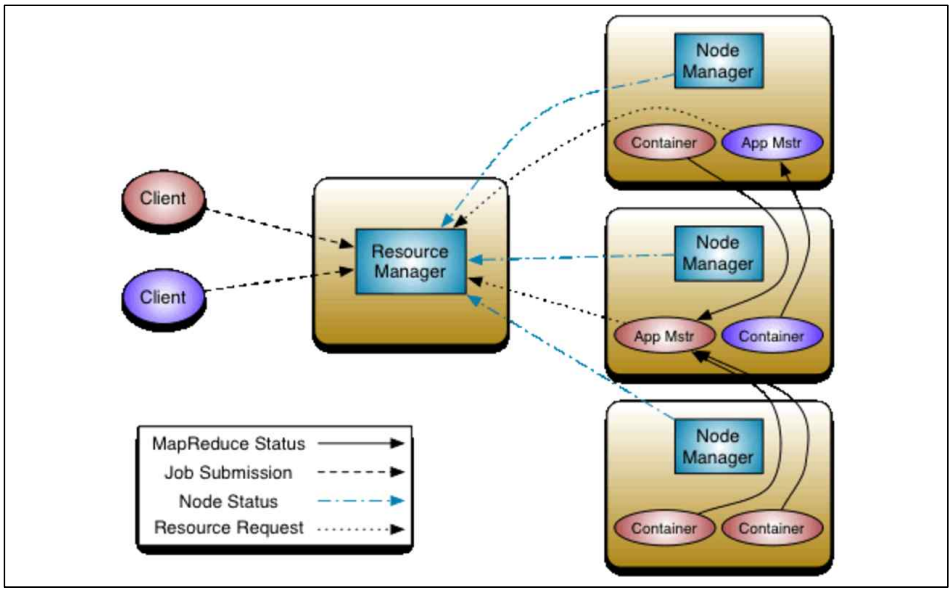
- 자원관리를 위해 리소스 매니저라는 새로운 컴포넌트 개발
- 리소스 매니저는 실제로 사용 가능한 단위로 자원을 관리(CPU, 메모리, 디스크, 네트워크 등)하는데 얀에서 실행되는 애플리케이션에 이 자원을 배분함
- 개별 에코시스템의 자원 배분을 고민하지 않아도 됨

③ 워크로드(발생하는 모든 작업) 확장

- 얀은 맵리듀스, 인터랙티브 질의, 실시간 처리, 그래프 알고리즘 등 다양한 형태의 워크로드 확장이 가능



4) YARN 아키텍쳐



① 리소스 매니저

- 얀의 마스터 서버로 하나 또는 이중화를 위해 두개의 서버에만 실행됨
- 글로벌 스케줄러: 클러스터 자원관리
- 최적의 데이터 노드를 찾아준다
- 전체 클러스터에서 사용할 수 있는 모든 시스템 자원들(CPU, 메모리, 디스크, 네트워크 등)을 관리
- 얀 클러스터에서 실행되는 프로그램이 리소스를 요청하면 적절하게 분배하고 리소스 사용 상태를 모니터링

② 노드매니저

- 얀의 worker 서버
- 맵리듀스의 태스크트래커 기능 담당
- 리소스 매니저를 제외한 모든 서버에서 실행
- 컨테이너를 실행시키고, 컨테이너의 라이프 사이클을 모니터링
- 컨테이너 장애상황 모니터링
- 컨테이너가 요청한 리소스 보다 많이 사용하고 있는 지 감시

③ 컨테이너

- 노드매니저가 실행되는 서버의 자원을 표현(CPU, Disk, Memory, Network..)
- 리소스 매니저가 요청에 의해 컨테이너가 실행
- 하나의 서버에 여러 컨테이너 있을 수 있음
- 맵리듀스의 태스크트래커가 태스크 단위로 잡을 실행한것 처럼 노드매니저는 컨테이너를 단위

로 애플리케이션을 실행하고 각 상태를 스케줄링

④ 애플리케이션 마스터

- 하나의 애플리케이션(프로그램)을 관리하는 마스터서버
- 클라이언트가 얀에 애플리케이션 실행을 요청하면 얀은 하나의 애플리케이션에 하나의 애플리케이션 마스터를 할당
- 예를 들어, 얀 클러스터에 '맵리듀스 잡'과 '스톰 애플리케이션' 실행을 요청하면 두개의 애플리케이션 마스터가 실행
- 애플리케이션에 필요한 리소스를 스케줄링
- 노드매니저에게 애플리케이션에 필요한 컨테이너를 실행시킬 것을 요청
- 애플리케이션 마스터는 컨네이터에서 실행됨

■ HDFS 기본 명령어

○ ./bin/hdfs dfs -cmd [args]

- hdfs dfs : hdfs에만 한정되어 사용할 수 있는 dfs쉘명령어

- cmd : 사용자가 설정한 명령어

- args : 해당 명령어를 실행할 때 필요한 파라미터

○ 파일 목록 보기

◆ ls

• 지정한 디렉터리에 있는 파일 정보를 출력

• 또는 특정 파일을 지정해 정보를 출력

• 파일 및 디렉터리의 권한 정보, 소유자, 소유 그룹, 생성일자, 바이트 수등 확인 가능

```
$ ./bin/hdfs dfs -ls [디렉터리/파일]
```

- 경로를 지정하지 않으면 해당 개정의 HDFS 홈디렉터리 조회

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls
```

```
Found 2 items
drwxr-xr-x   - bigdata supergroup          0 2018-05-30 13:38 conf
drwxr-xr-x   - bigdata supergroup          0 2018-05-30 13:35 output
```

- 디렉터리를 지정할 경우 해당 디렉터리에 저장된 파일 목록 출력

(/ 는 HDFS 최상위 디렉터리)

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls /
```

```
Found 2 items
drwxrwx---   - bigdata supergroup          0 2018-05-30 13:24 /tmp
drwxr-xr-x   - bigdata supergroup          0 2018-05-30 13:15 /user
```

◆ ls -R

• 현재 디렉터리의 하위 디렉터리 및 파일 정보까지 출력

• -r 옵션은 하위 디렉터리만 출력

```
$ ./bin/hdfs dfs -ls -R [디렉터리/파일]
```

- 홈디렉터리 안의 모든 디렉터리, 파일 출력

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls -R
```

```
drwxr-xr-x   - bigdata supergroup          0 2018-05-30 13:38 conf
-rw-r--r--   3 bigdata supergroup       4986 2018-05-30 13:38 conf/hadoop-env.sh
drwxr-xr-x   - bigdata supergroup          0 2018-05-30 13:35 output
-rw-r--r--   3 bigdata supergroup          0 2018-05-30 13:35 output/_SUCCESS
-rw-r--r--   3 bigdata supergroup          0 2018-05-30 13:35 output/part-r-00000
```

○ 디렉터리 생성: mkdir

◆ mkdir

• 디렉터리를 생성

```
$ ./bin/hdfs dfs -mkdir [디렉터리 명]
```

- HDFS 홈디렉터리(/user/bigdata)에 "testDir"이라는 디렉터리 생성

따로 경로를 설정하지 않으면 홈디렉터리에 생성(/user/bigdata)

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -mkdir testDir
```

```
drwxr-xr-x   - bigdata supergroup          0 2018-05-30 13:38 conf
drwxr-xr-x   - bigdata supergroup          0 2018-05-30 13:35 output
drwxr-xr-x   - bigdata supergroup          0 2018-05-30 16:20 testDir
```

○ 파일 복사

◆ put

• 지정한 로컬 파일 및 디렉토리를 시스템을 HDFS의 목적지 경로로 복사

• copyFromLocal 명령어도 같은 역할을 함

```
$ ./bin/hdfs dfs -put [로컬 디렉터리] [HDFS 목적지 디렉터리/파일]
```

- 하둡 홈디렉터리 안에 있는 README.txt 파일을 HDFS의 testDir로 복사

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -put README.txt testDir
```

- testDir안에 README.txt 파일 확인

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls testDir
```

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls testDir
Found 1 items
-rw-r--r--   3 bigdata supergroup       1366 2018-06-01 08:47 testDir/README.txt
```

◆ get

• 지정한 HDFS에 저장된 데이터를 로컬 파일 시스템으로 복사

• copyToLocal 명령어도 같은 역할을 함

```
$ ./bin/hdfs dfs -get [HDFS에 저장된 데이터] [로컬 디렉토리/파일]
```

- testDir 디렉터리를 bigdata 홈디렉터리로 복사

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -get testDir /home/bigdata
```

- /home/bigdata/testDir 경로로 이동후 README.txt 확인

```
[bigdata@server01 hadoop-2.9.1]$ cd /home/bigdata/testDir
[bigdata@server01 testDir]$ ls
```

```
[bigdata@server01 testDir]$ ls
README.txt
```

◆ cp

• HDFS 안의 파일들을 목적지 경로로 복사

```
$ ./bin/hdfs dfs -cp [HDFS에 저장된 데이터] [HDFS 목적지 디렉토리/파일]
```

- 하둡 홈디렉터리로 이동 후 testDir 디렉터리 안의 README.txt 파일을 readme.txt로 복사

```
[bigdata@server01 testDir]$ cd /home/bigdata/hadoop-2.9.1
[bigdata@server01 hadoop-2.9.1]$
./bin/hdfs dfs -cp testDir/README.txt testDir/readme.txt
```

- readme.txt 파일 확인

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls testDir
```

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls testDir
Found 2 items
-rw-r--r--   3 bigdata supergroup       1366 2018-06-01 08:47 testDir/README.txt
-rw-r--r--   3 bigdata supergroup       1366 2018-06-01 09:16 testDir/readme.txt
```

○ 파일 이동

◆ mv

• HDFS안의 파일들을 목적지 경로로 이동

```
$ ./bin/hdfs dfs -mv [HDFS에 저장된 데이터] [HDFS 목적지 디렉토리/파일]
```

- conf 디렉터리를 testDir 디렉터리로 이동

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -mv conf testDir
```

- testDir 확인

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls -R testDir
```

```
drwxr-xr-x   - bigdata supergroup          0 2018-05-25 11:27 testDir/conf
-rw-r--r--   3 bigdata supergroup       1366 2018-06-01 08:47 testDir/README.txt
```

- conf 디렉터리 확인

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls conf
```

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls conf
ls: `conf': No such file or directory
```

◆ moveFromLocal

• 지정한 로컬 파일 및 디렉토리를 시스템을 HDFS의 목적지 경로로 이동

```
$ ./bin/hdfs dfs -moveFromLocal [로컬 디렉토리/파일] [HDFS 목적지 디렉토리/파일]
```

- 로컬 /home/bigdata/testDir 디렉터리를 HDFS의 testDir로 이동

```
[bigdata@server01 hadoop-2.9.1]$
./bin/hdfs dfs -moveFromLocal /home/bigdata/testDir/ testDir
```

- testDir 확인

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls -R testDir
```

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls -R testDir
-rw-r--r--   3 bigdata supergroup       1366 2018-06-01 08:47 testDir/README.txt
drwxr-xr-x   - bigdata supergroup          0 2018-05-25 11:27 testDir/conf
-rw-r--r--   1 bigdata supergroup       5003 2018-05-25 11:27 testDir/conf/hadoop-env.sh
drwxr-xr-x   - bigdata supergroup          0 2018-06-01 09:42 testDir/testDir
-rw-r--r--   3 bigdata supergroup       1366 2018-06-01 09:42 testDir/testDir/README.txt
```

- /home/bigdata/testDir 확인

```
[bigdata@server01 hadoop-2.9.1]$ ls /home/bigdata/testDir
```

```
[bigdata@server01 hadoop-2.9.1]$ ls /home/bigdata/testDir
ls: cannot access /home/bigdata/testDir: 그런 파일이나 디렉터리가 없습니다
```

○ 파일 내용 보기

◆ cat

• HDFS 파일의 내용 출력

```
$ ./bin/hdfs dfs -cat  [HDFS에 저장된 데이터]
```

- testDir/readme.txt 파일 안의 내용 출력

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -cat testDir/README.txt
```

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -cat testDir/README.txt
For the latest information about Hadoop, please visit our website at:

   http://hadoop.apache.org/core/

and our wiki, at:

   http://wiki.apache.org/hadoop/

This distribution includes cryptographic software.  The country in
which you currently reside may have restrictions on the import,
possession, use, and/or re-export to another country, of
encryption software.  BEFORE using any encryption software, please
check your country's laws, regulations and policies concerning the
import, possession, or use, and re-export of encryption software, to
see if this is permitted.  See <http://www.wassenaar.org/> for more
information.
```

◆ tail

• HDFS 파일의 마지막 1KB에 해당하는 내용 화면 출력

• -f 옵션 사용 시, 해당 파일에 내용이 추가 될 때 출력 내용도 갱신

```
$ ./bin/hdfs dfs -tail <-f>  [HDFS에 저장된 데이터]
```

- testDir/README.txt 파일의 마지막 1KB 출력

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -tail testDir/README.txt
```

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -tail testDir/README.txt
try, of
encryption software.  BEFORE using any encryption software, please
check your country's laws, regulations and policies concerning the
import, possession, or use, and re-export of encryption software, to
see if this is permitted.  See <http://www.wassenaar.org/> for more
information.

The U.S. Government Department of Commerce, Bureau of Industry and
Security (BIS), has classified this software as Export Commodity
Control Number (ECCN) 5D002.C.1, which includes information security
software using or performing cryptographic functions with asymmetric
algorithms.  The form and manner of this Apache Software Foundation
distribution makes it eligible for export under the License Exception
ENC Technology Software Unrestricted (TSU) exception (see the BIS
Export Administration Regulations, Section 740.13) for both object
code and source code.

The following provides more details on the included cryptographic
software:
  Hadoop Core uses the SSL libraries from the Jetty project written
by mortbay.org.
```

○ 삭제

◆ rm

• 저장한 디렉터리나 파일 삭제

• 디렉터리는 안이 비어있는 경우만 삭제 가능

```
$ ./bin/hdfs dfs -rm  [HDFS에 저장된 파일/디렉터리]
```

- testDir/README.txt 파일 삭제

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -rm testDir/README.txt
```

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -rm testDir/README.txt
Deleted testDir/README.txt
```

- testDir/README.txt 파일 삭제 확인

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls testDir/README.txt
```

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls testDir/README.txt
ls: `testDir/README.txt': No such file or directory
```

◆ rm -R

• 저장한 디렉터리나 파일 삭제

• 디렉터리가 비어져 있지 않아도 삭제 가능

```
$ ./bin/hdfs dfs -rm -r [HDFS에 저장된 파일/디렉터리]
```

- testDir 디렉터리 삭제

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -rm -r testDir
```

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -rm -r testDir
Deleted testDir
```

-testDir 디렉터리 확인

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls testDir
```

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls testDir
ls: `testDir': No such file or directory
```

■ HDFS 운영

- 네임노드에게 클라이언트는 권한을 받으면 데이터노드에 접근해 데이터 저장
- 데이터는 블록 단위 (기본 64MB 또는 128MB)로 나눠져 저장
- HDFS의 블록 복제수는 기본적으로 3개
- 네임노드는 데이터 노드에 장애가 나면 그 데이터노드의 블록들을 다른 서버로 복제

Q. 블록이 어떻게 복제되어 저장되나

A. 블록은 생성이 되면 다른 데이터노드들과 파이프라인을 형성해 복제

○ server01을 bigdata 계정으로 로그인 example_data 폴더 생성

- hadoop 실행

```
[bigdata@server01 ~]$ mkdir example_data
```

○ FileZilar를 이용해 HDFS에 저장할 데이터를 example_data에 옮기기

○ /home/bigdata/example_data에 있는 call_chicken_04month.csv 데이터를 hdfs에 저장

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -put
/home/bigdata/example_data/call_chicken_04month.csv testDir
```

○ 데이터 확인

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs dfs -ls testDir
```

```
Found 1 items
-rw-r--r--   3 bigdata supergroup    2463189 2018-05-30 16:25 testDir/call_chicken_04month.csv
```

○ HDFS 파일 시스템 상태 확인 : fsck

-files -blocks 옵션 : 블록 보고서 출력

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs fsck testDir/call_chicken_04month.csv
-files -blocks
```

- 블록 아이디: blk_1073741835_1011

- 블록 크기 : 2463189 B

- 블록 복제 수: 3

```
/user/bigdata/testDir/call_chicken_04month.csv 2463189 bytes, 1 block(s):  OK
0. BP-995797781-192.168.56.101-1527652230556:blk_1073741835_1011 len=2463189 Live_repl=3

Status: HEALTHY
 Total size:    2463189 B
 Total dirs:    0
 Total files:   1
 Total symlinks:                0
 Total blocks (validated):      1 (avg. block size 2463189 B)
 Minimally replicated blocks:   1 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
 Average block replication:     3.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          4
 Number of racks:               1
FSCK ended at Wed May 30 16:27:43 KST 2018 in 8 milliseconds


The filesystem under path '/user/bigdata/testDir/call_chicken_04month.csv' is HEALTHY
```

○ server01의 데이터노드 로그를 확인하고 블록이 어떤 서버로 복제되었는 지 확인

```
[bigdata@server01 hadoop-2.9.1]$ cd /home/bigdata/hadoop-2.9.1/logs
[bigdata@server01 logs]$ vi hadoop-bigdata-datanode-server01.log
```

- vi편집기에서 명령 모드에서 'G'를 입력하면 맨 밑줄로 이동함

server01, server02, server04에 복제 된 것을 확인 가능

```
2018-05-30 16:25:10,353 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /192.168.56.101:39326, dest: /1
92.168.56.101:50010, bytes: 2463189, op: HDFS_WRITE, cliID: DFSClient_NONMAPREDUCE_775291377_1, offset: 0, srvID: b8fac73b-79a
a-4a57-8b41-3e9739583742, blockid: BP-995797781-192.168.56.101-1527652230556:blk_1073741835_1011, duration(ns): 55081179
```

```
2018-05-30 16:25:10,353 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder: BP-995797781-192.168.56.101-152
7652230556:blk_1073741835_1011, type=HAS_DOWNSTREAM_IN_PIPELINE, downstreams=2 [192.168.56.102:50010, 192.168.56.104:50010] te
rminating
```

- 데이터노드 블록들이 저장되는 디렉토리는 hdfs-site.xml에서 설정한 /home/bigdata/data/dfs/datanode 안에 있음
- 아래 경로로 이동하여 블록확인

```
$ cd
/home/bigdata/data/dfs/datanode/current/BP-995797781-192.168.56.101-1527652230556/current/finalized/subd
ir0/subdir0
```

```
[bigdata@server02 subdir0]$ ls
blk_1073741834            blk_1073741835
blk_1073741834_1010.meta  blk_1073741835_1011.meta
```

```
[bigdata@server01 subdir0]$ ls
blk_1073741832            blk_1073741833_1009.meta  blk_1073741835
blk_1073741832_1008.meta  blk_1073741834            blk_1073741835_1011.meta
blk_1073741833            blk_1073741834_1010.meta
```

```
[bigdata@server04 subdir0]$ ls
blk_1073741832            blk_1073741833_1009.meta  blk_1073741835
blk_1073741832_1008.meta  blk_1073741834            blk_1073741835_1011.meta
blk_1073741833            blk_1073741834_1010.meta
```

○ FileZillar를 이용해 HDFS에 저장할 데이터를 example_data에 옮기기

○ /home/bigdata/example_data에 있는 SmartCarStatus.txt 데이터를 hdfs에 저장

- 약 100mb -> 2개의 블록으로 쪼개짐

```
[bigdata@server01 hadoop-2.9.1]$
 ./bin/hdfs dfs -put /home/bigdata/example_data/SmartCarStatus.txt testDir
```

○ HDFS에 적재한 SmartCarStatus.txt 파일 상태 및 블록 확인

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs fsck testDir/SmartCarStatus.txt -files
-blocks
```

```
/user/bigdata/testDir/SmartCarStatus.txt 104074314 bytes, 2 block(s):  OK
0. BP-995797781-192.168.56.101-1527652230556:blk_1073741836_1012 len=67108864 Live_repl=3
1. BP-995797781-192.168.56.101-1527652230556:blk_1073741837_1013 len=36965450 Live_repl=3

Status: HEALTHY
 Total size:    104074314 B
 Total dirs:    0
 Total files:   1
 Total symlinks:                0
 Total blocks (validated):      2 (avg. block size 52037157 B)
 Minimally replicated blocks:   2 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
 Average block replication:     3.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          4
 Number of racks:               1
FSCK ended at Wed May 30 17:21:01 KST 2018 in 1 milliseconds
```

○ server01의 데이터노드 로그를 확인하고 블록이 어떤 서버로 복제됐는지 확인

```
[bigdata@server01 hadoop-2.9.1]$ cd /home/bigdata/hadoop-2.9.1/logs
[bigdata@server01 logs]$ vi hadoop-bigdata-datanode-server01.log
```

```
2018-05-30 17:17:22,421 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketRes
ponder: BP-995797781-192.168.56.101-1527652230556:blk_1073741836_1012, type=HAS_DOWNSTR
EAM_IN_PIPELINE, downstreams=2:[192.168.56.103:50010, 192.168.56.104:50010] terminating
```

```
2018-05-30 17:17:23,048 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketRes
ponder: BP-995797781-192.168.56.101-1527652230556:blk_1073741837_1013, type=HAS_DOWNSTR
EAM_IN_PIPELINE, downstreams=2:[192.168.56.104:50010, 192.168.56.102:50010] terminating
```

```
[bigdata@server01 subdir0]$ ls -l
합계 105104
-rw-rw-r-- 1 bigdata bigdata    23295  5월 30 13:35 blk_1073741832
-rw-rw-r-- 1 bigdata bigdata      191  5월 30 13:35 blk_1073741832_1008.meta
-rw-rw-r-- 1 bigdata bigdata   197097  5월 30 13:35 blk_1073741833
-rw-rw-r-- 1 bigdata bigdata     1547  5월 30 13:35 blk_1073741833_1009.meta
-rw-rw-r-- 1 bigdata bigdata     4986  5월 30 13:38 blk_1073741834
-rw-rw-r-- 1 bigdata bigdata       47  5월 30 13:38 blk_1073741834_1010.meta
-rw-rw-r-- 1 bigdata bigdata  2463189  5월 30 16:25 blk_1073741835
-rw-rw-r-- 1 bigdata bigdata    19251  5월 30 16:25 blk_1073741835_1011.meta
-rw-rw-r-- 1 bigdata bigdata 67108864  5월 30 17:17 blk_1073741836
-rw-rw-r-- 1 bigdata bigdata   524295  5월 30 17:17 blk_1073741836_1012.meta
-rw-rw-r-- 1 bigdata bigdata 36965450  5월 30 17:17 blk_1073741837
-rw-rw-r-- 1 bigdata bigdata   288803  5월 30 17:17 blk_1073741837_1013.meta
```

```
[bigdata@server02 subdir0]$ ls -l
합계 141260
-rw-rw-r-- 1 bigdata bigdata     4986  5월 30 13:38 blk_1073741834
-rw-rw-r-- 1 bigdata bigdata       47  5월 30 13:38 blk_1073741834_1010.meta
-rw-rw-r-- 1 bigdata bigdata  2463189  5월 30 16:25 blk_1073741835
-rw-rw-r-- 1 bigdata bigdata    19251  5월 30 16:25 blk_1073741835_1011.meta
-rw-rw-r-- 1 bigdata bigdata 36965450  5월 30 17:17 blk_1073741837
-rw-rw-r-- 1 bigdata bigdata   288803  5월 30 17:17 blk_1073741837_1013.meta
-rw-rw-r-- 1 bigdata bigdata 67108864  5월 30 18:06 blk_1073741838
-rw-rw-r-- 1 bigdata bigdata   524295  5월 30 18:06 blk_1073741838_1014.meta
-rw-rw-r-- 1 bigdata bigdata 36965450  5월 30 18:06 blk_1073741839
-rw-rw-r-- 1 bigdata bigdata   288803  5월 30 18:06 blk_1073741839_1015.meta
```

```
[bigdata@server03 subdir0]$ ls -l
합계 168716
-rw-rw-r-- 1 bigdata bigdata    23295  5월 30 13:35 blk_1073741832
-rw-rw-r-- 1 bigdata bigdata      191  5월 30 13:35 blk_1073741832_1008.meta
-rw-rw-r-- 1 bigdata bigdata   197097  5월 30 13:35 blk_1073741833
-rw-rw-r-- 1 bigdata bigdata     1547  5월 30 13:35 blk_1073741833_1009.meta
-rw-rw-r-- 1 bigdata bigdata 67108864  5월 30 17:17 blk_1073741836
-rw-rw-r-- 1 bigdata bigdata   524295  5월 30 17:17 blk_1073741836_1012.meta
-rw-rw-r-- 1 bigdata bigdata 67108864  5월 30 18:06 blk_1073741838
-rw-rw-r-- 1 bigdata bigdata   524295  5월 30 18:06 blk_1073741838_1014.meta
-rw-rw-r-- 1 bigdata bigdata 36965450  5월 30 18:06 blk_1073741839
-rw-rw-r-- 1 bigdata bigdata   288803  5월 30 18:06 blk_1073741839_1015.meta
```

```
[bigdata@server04 subdir0]$ ls -l
합계 105104
-rw-rw-r-- 1 bigdata bigdata    23295  5월 30 13:35 blk_1073741832
-rw-rw-r-- 1 bigdata bigdata      191  5월 30 13:35 blk_1073741832_1008.meta
-rw-rw-r-- 1 bigdata bigdata   197097  5월 30 13:35 blk_1073741833
-rw-rw-r-- 1 bigdata bigdata     1547  5월 30 13:35 blk_1073741833_1009.meta
-rw-rw-r-- 1 bigdata bigdata     4986  5월 30 13:38 blk_1073741834
-rw-rw-r-- 1 bigdata bigdata       47  5월 30 13:38 blk_1073741834_1010.meta
-rw-rw-r-- 1 bigdata bigdata  2463189  5월 30 16:25 blk_1073741835
-rw-rw-r-- 1 bigdata bigdata    19251  5월 30 16:25 blk_1073741835_1011.meta
-rw-rw-r-- 1 bigdata bigdata 67108864  5월 30 17:17 blk_1073741836
-rw-rw-r-- 1 bigdata bigdata   524295  5월 30 17:17 blk_1073741836_1012.meta
-rw-rw-r-- 1 bigdata bigdata 36965450  5월 30 17:17 blk_1073741837
-rw-rw-r-- 1 bigdata bigdata   288803  5월 30 17:17 blk_1073741837_1013.meta
```

Q. 데이터노드에 장애가 생기면 네임노드는 어떻게 대처하는가?

A. 장애가 생긴 데이터 노드의 블록이 다른 데이터노드로 복제됨

```
[bigdata@server01 hadoop-2.9.1]$ ./bin/hdfs fsck test3 -files -blocks
Connecting to namenode via http://server01:50070/fsck?ugi=bigdata&files=1&blocks=1&path=%2Fuser%2Fbigdata%2Ftest3
FSCK started by bigdata (auth:SIMPLE) from /192.168.56.101 for path /user/bigdata/test3 at Fri Jun 01 12:49:49 KST 2018
/user/bigdata/test3 289 bytes, 1 block(s):  OK
0. BP-1269912599-192.168.56.101-1527159601259:blk_1073742127_1303 len=289 Live_repl=3

Status: HEALTHY
 Total size:    289 B
 Total dirs:    0
 Total files:   1
 Total symlinks:                0
 Total blocks (validated):      1 (avg. block size 289 B)
 Minimally replicated blocks:   1 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
 Average block replication:     3.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          4
 Number of racks:               1
FSCK ended at Fri Jun 01 12:49:49 KST 2018 in 1 milliseconds


The filesystem under path '/user/bigdata/test3' is HEALTHY
```

- blk_1073742127_1303 블록은 현재 server01, server03, server04에 저장됨

```
[bigdata@server01 subdir1]$ ls blk_1073742127_1303.meta
blk_1073742127_1303.meta
```

```
[bigdata@server03 subdir1]$ ls blk_1073742127_1303.meta
blk_1073742127_1303.meta
```

```
[bigdata@server04 subdir1]$ ls blk_1073742127_1303.meta
blk_1073742127_1303.meta
```

- server02에는 블록이 없음

```
[bigdata@server02 subdir1]$ ls blk_1073742127_1303.meta
ls: cannot access blk_1073742127_1303.meta: 그런 파일이나 디렉터리가 없습니다
```

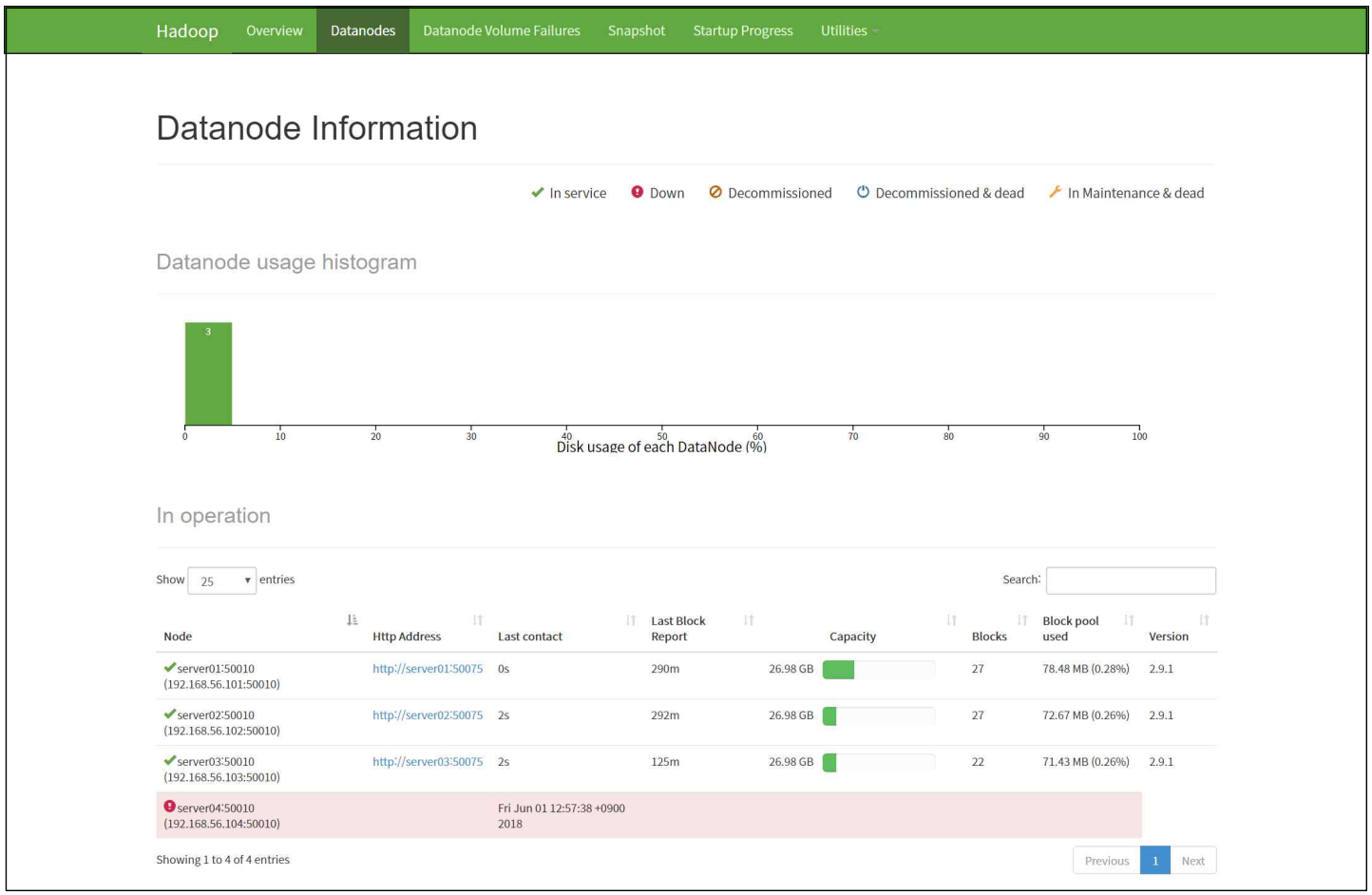
- jps로 데이터노드의 프로세스 아이디를 확인 후 server04의 데이터노드를 끔

```
[bigdata@server04 ~]$ jps
```

```
[bigdata@server04 subdir1]$ jps
1634 NodeManager
1512 DataNode
2061 Jps
```

```
[bigdata@server04 ~]$ kill -9 1512
```

- server01:50070에 접속해 server04 데이터노드 상태 확인



-server02에 blk_1073742127_1303 블록이 복제됨

```
2018-06-01 13:10:16,285 INFO org.apache.hadoop.hdfs.server.datanode.DataNode:
 DataTransfer, at server01:50010: Transmitted BP-1269912599-192.168.56.101-15
27159601259:blk_1073742127_1303 (numBytes=289) to /192.168.56.102:50010
```

```
[bigdata@server02 subdir1]$ ls blk_1073742127_1303.meta
blk_1073742127_1303.meta
```