03. HDFS

2020년 7월 11일 토요일 오후 10:04

1. HDFS
   - DAS, NAS, SAN에 비해 저사양 서버를 이용하여 스토리지를 구성할 수 있음
   - 트랜잭션이 중요한 경우 HDFS는 적합하지 않음
   - 대규모 데이터를 저장, 배치로 처리하는 경우 적합

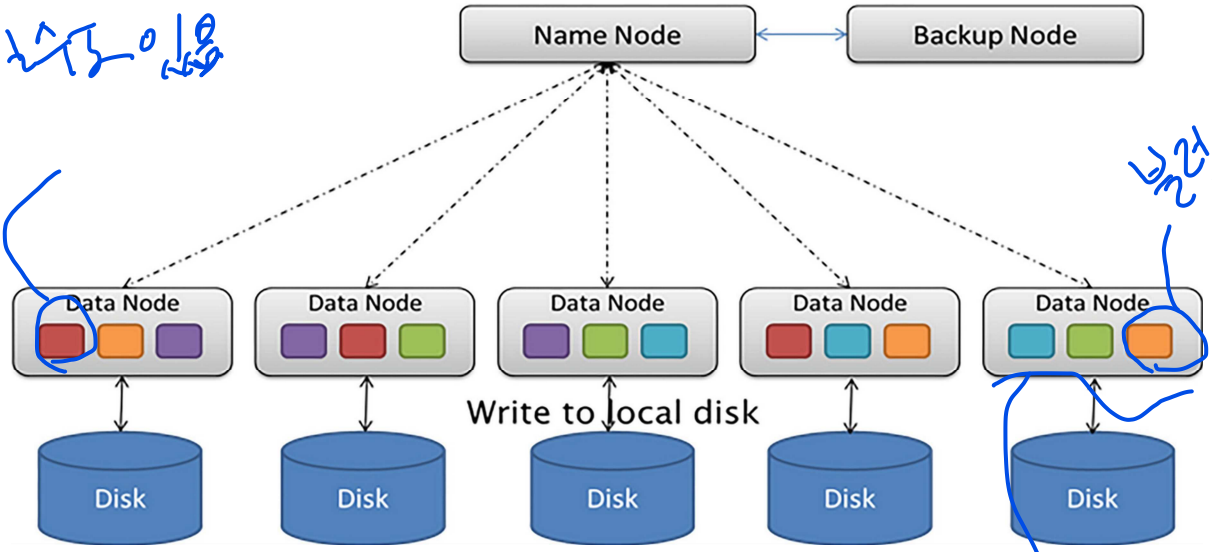
   - HDFS의 목표

1) 장애 복구
   a. 빠른 시간내에 장애 감지 및 대처
   b. 복제 데이터를 이용한 데이터 유실 방지
2) 스트리밍 방식의 데이터 접근
   a. 클라이언트의 요청을 빠른 시간내에 처리 보다는 동일한 시간 내에 더 많은 데이터를 처리
   b. 이를 위해 랜덤 방식의 데이터 접근을 고려하지 않기 때문에 기존의 인터넷 뱅킹, 쇼핑몰 같은 서비스에 적용하기 부적합
   c. 스트리밍 방식으로 데이터에 접근하도록 설계, 클라이언트는 끊김 없이 연속된 흐름으로 데이터에 접근
3) 대용량 데이터 저장
   a. 높은 데이터 전송 대역폭과 하나의 클러스터에서 수백 대의 노드를 지원할 수 있음
   b. 하나의 인스턴스에서 수백만개 이상의 파일을 지원
4) 데이터 무결성
   a. 한 번 저장한 데이터는 수정할 수 없고 읽기만 가능
   b. 이동, 삭제, 복사 가능
   c. 하둡 2.0 부터 저장된 파일에 append 기능 지원

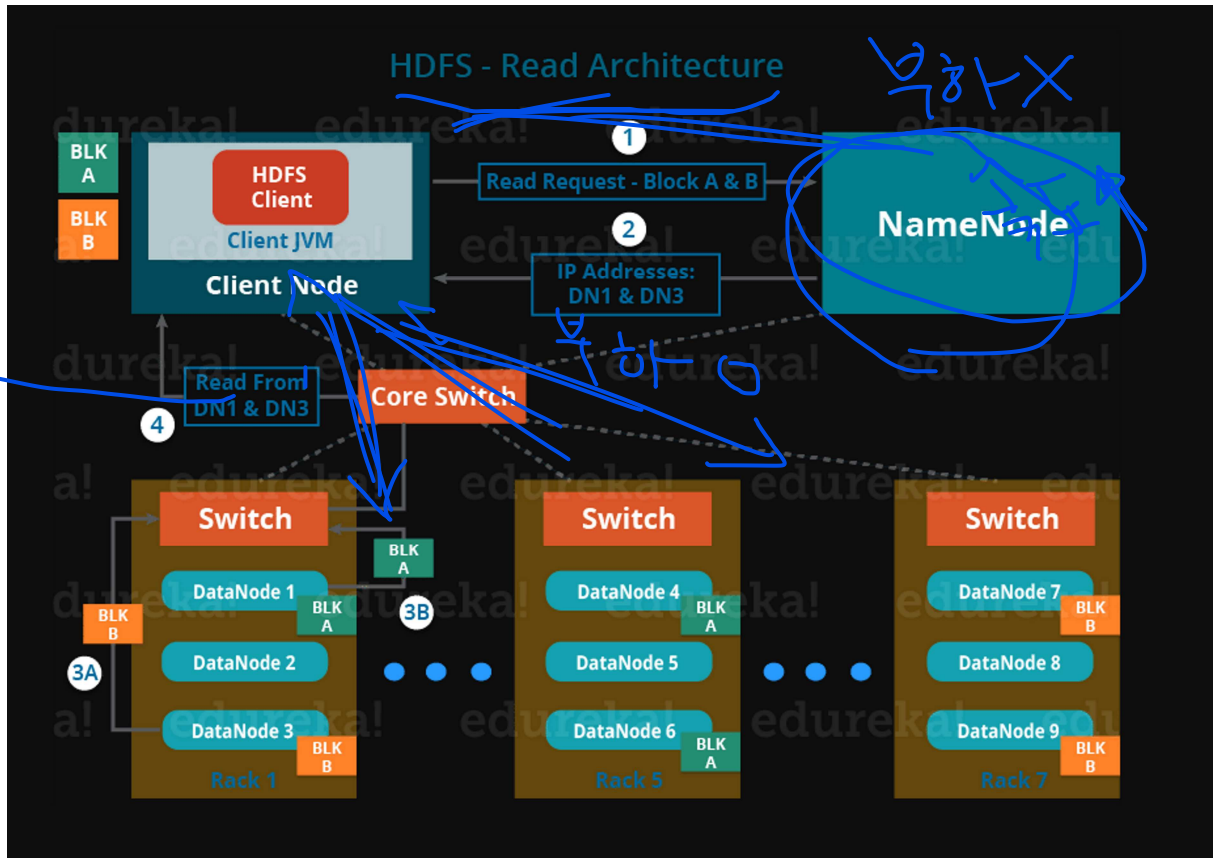
2. HDFS Architecture

1) 블록 구조 파일 시스템
   a. HDFS에 저장하는 파일은 특정 크기의 블럭으로 나뉘어 분산된 서버에 저장
   b. 블록 크기는 기본적으로 128MB로 설정되어 있으며 변경 가능
   c. 128MB 보다 작은 데이터는 그 크기에 맞게 블럭이 생성됨

HDFS Architecture



1) 네임 노드, 데이터 노드
   a. Master - Slave 아키텍쳐 (네임 노드 - 데이터 노드)



2) 네임 노드
   ● 메타데이터 관리
   파일 시스템을 유지하기 위한 메타데이터 관리
   메타데이터는 파일 시스템 이미지(파일명, 디렉터리, 크기, 권한 등)와 파일에 대한 블록 매핑 정보로 구성

클라이언트에게 빠르게 응답할 수 있게 메모리에 전체 메타데이터를 로딩해서 관리

● 데이터 노드 모니터링

데이터 노드는 네임 노드에게 매 3초 마다 heartbeat 메시지를 전송
heartbeat는 데이터 노드 상태 정보와 저장되어 있는 블럭의 목록으로 구성
네임노드는 heartbeat 를 이용해 데이터 노드의 실행 상태와 용량을 모니터
일정 시간 동안 heartbeat 가 도착하지 않으면 네임 노드는 해당 데이터 노드를 장애로 판단

● 블록 관리

장애가 발생한 데이터 노드를 발견하면, 해당 데이터 노드의 블럭을 새로운 데이터 노드로 복제
용량이 부족한 데이터 노드가 있다면 상대적으로 여유가 있는 데이터 노드로 블럭을 이동
블럭의 복제 본 수 관리(복제 본 수와 일치하지 않는 블럭이 발견되면 추가로 블럭을 복제 또는 삭제)

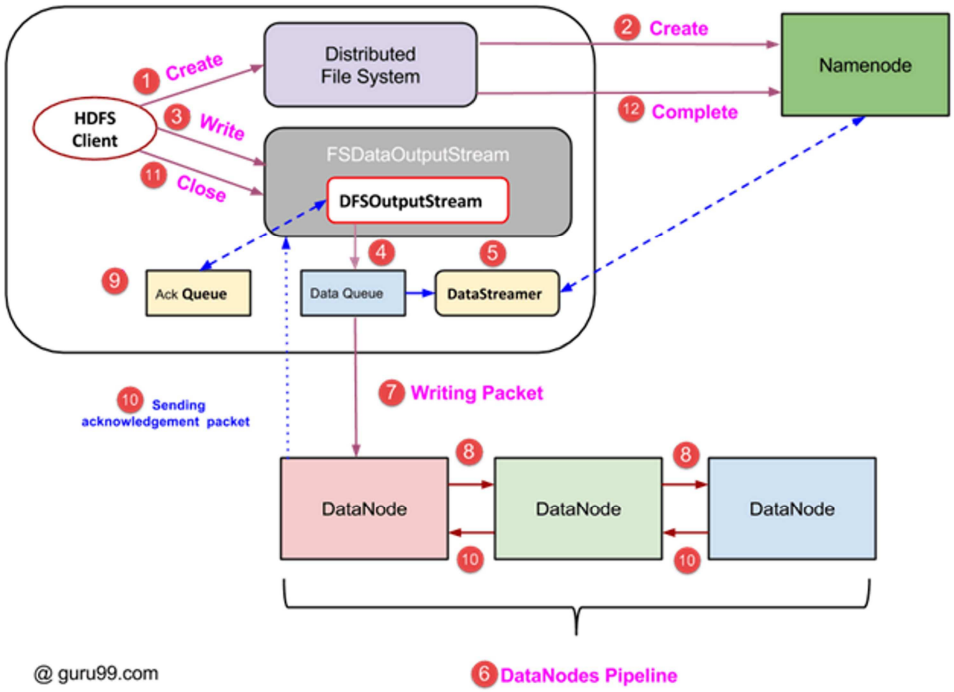
● 클라이언트의 요청 접수

클라이언트가 HDFS에 접근하려면 반드시 네임 노드에 먼저 접속
HDFS에 파일을 저장하는 경우 기존 파일의 저장 여부와 권한 확인 절차를 거쳐 승인 처리
HDFS에 저장되 파일을 조회하는 경우 블럭의 위치 정보를 반환

3) 데이터 노드

● 클라이언트가 HDFS에 저장하는 파일을 로컬 디스크에 유지
● 로컬 디스크에 저장되는 파일은 실제 데이터가 저장되어 있는 raw 데이터 파일과 checksum 이나 파일 생성 일자와 같은 메타 데이터가 설정된 파일

3. HDFS 의 파일 저장



1) 파일 저장 요청

a. 하둡은 FileSystem 이라는 추상클래스에 일반적인 파일 시스템을 관리하기 위한 메서드를 정의, 이 추상클래스를 상속받아 각 파일 시스템에 맞게 구현된 다양한 파일 시스템 클래스를 제공, HDFS에 파일을 저장하는 경우에 파일 시스템 클래스 중 DistributedFileSystem 클래스를 사용, 클라이언트는 DistributedFileSystem 클래스의 create 메서드를 호출해 스트림 객체를 생성

b. DistributedFileSystem 은 클라이언트에게 반환할 스트림 객체로 FSDataOutputStream객체를 생성, FSDataOutputStream은 데이터노드와 네임노드간 통신을 관리하는 DFSOutputStream 클래스를 wrapping 하는 클래스임, DistributedFileSystem은 DFSOutputStream을 생성하기 위해 FSDataOutputStream의 create 메서드를 호출

c. FSDataOutputStream은 DFSOutputStream을 생성, 이 때 DFSOutputStream은 RPC통신으로 네임노드의 create 메서드 호출, 네임노드는 클라이언트의 요청이 유효한지 검사를 진행, 이미 생성된 파일이거나 권한에 문제가 있거나, 현재 파일 시스템의 용량을 초과한다면 오류가 발생, 네임노드는 파일 유효성 검사 결과가 정상일 경우 파일 시스템 이미지에 해당 파일의 엔트리를 추가, 네임노드는 클라이언트에게 해당 파일을 저장할 수 있는 제어권을 부여

d. 네임노드의 유효성 검사를 통과했다면 DFSOutputStream객체가 정상적으로 생성, DistributedFileSystem은 DFSOutputStream을 래핑한 FSDataOutputStream을 클라이언트에게 반환

2) 패킷 전송

a. 클라이언트는 스트림 객체의 write 메서드를 호출하여 파일 저장을 시작, DFSOutputStream은 클라이언트가 저장하는 파일을 64K 크기의 패킷으로 분할

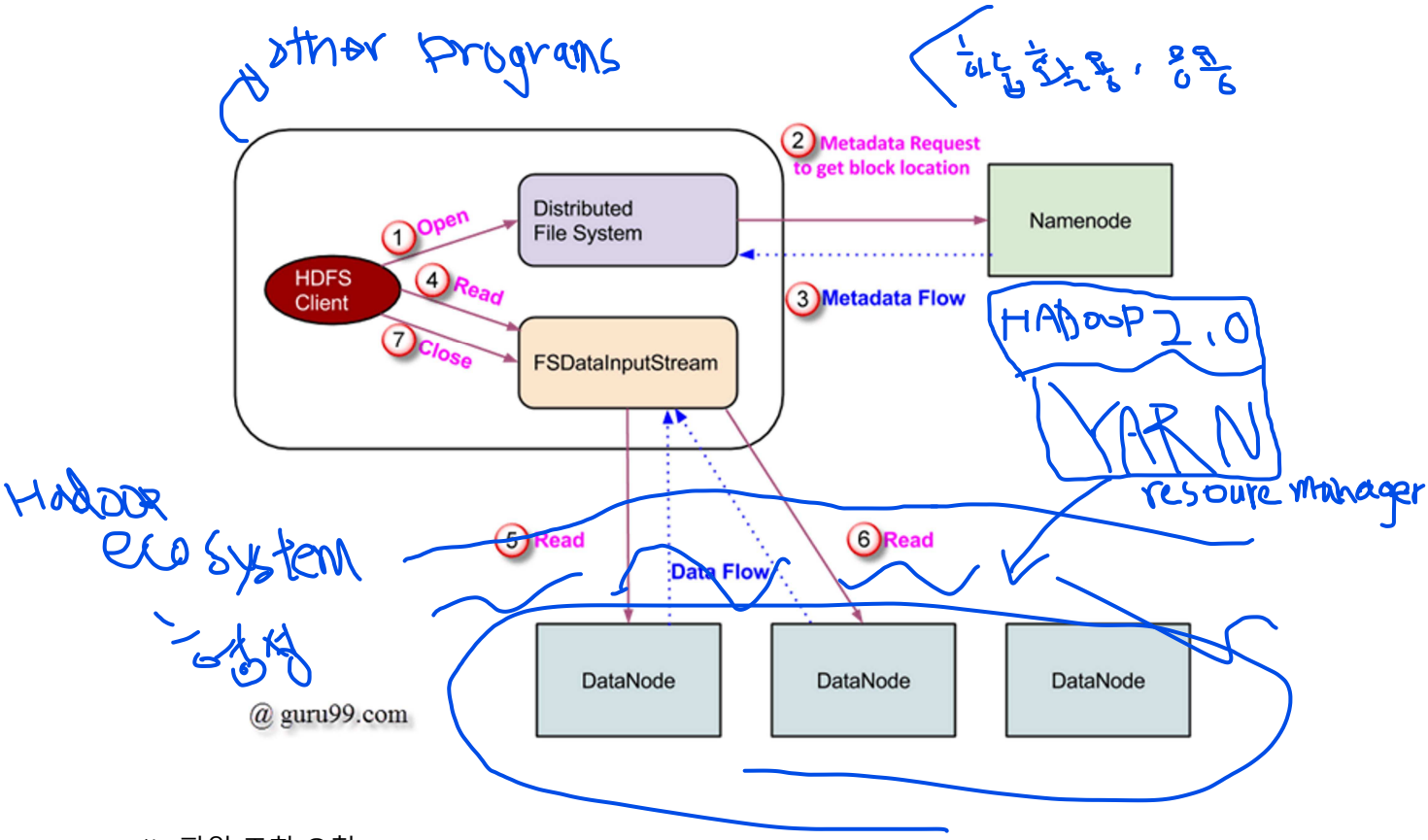
b. DFSOutputStream은 전송할 패킷을 내부 큐인 dataQueue 에 등록,

DFSOutputStream의 내부 쓰레드가 데이터큐에 패킷이 등록된 것을 확인하면 DFSOutputStream의 inner 클래스인 DataStreamer는 네임노드의 addBlock 메서드를 호출

c. 네임노드는 DataStreamer에게 블럭을 저장할 데이터노드 목록을 반환, 이 목록으로 복제본 수와 동일한 수의 데이터노드를 연결한 파이프라인을 형성
d. DataStreamer는 파이프라인의 첫 번째 데이터노드 부터 패킷전송을 시작, 데이터노드는 클라이언트와 다른 데이터노드로부터 패킷을 주고 받기 위해 DataXceiverServer 데몬을 실행, DataXceiverServer는 클라이언트 및 다른 데이터노드와 패킷 교환 기능을 제공
e. 첫 번째 데이터노드는 패킷을 저장하면서 두 번째 데이터노드에게 패킷 저장을 요청, 두 번째 데이터노드도 패킷을 저장하면서 세 번째 데이터노드에게 패킷 저장을 요청, 마지막으로 세 번째 데이터노드가 패킷을 저장
f. 첫 번째 데이터노드에 패킷을 저장할 때 DFSOutputStream은 내부 큐인 승인큐(ackQueue)에 패킷을 등록, 승인큐는 패킷 전송이 완료되었다는 응답을 기다리는 패킷이 등록되어 있으며 모든 데이터노드로부터 응답을 받았을 때 해당 패킷이 제거됨
g. 각 데이터노드는 패킷이 정상적으로 저장되면 자신에게 패킷을 전송한 데이터노드에게 ACK 메시지를 전송, ACK메시지는 패킷 수신이 정상적으로 완료되었다는 승인 메시지, 승인 메시지는 파이프라인을 통해 DFSOutputStream 에게까지 전달됨
h. 각 데이터노드는 패킷 저장이 완료되면 네임노드의 blockReceived 메서드를 호출, 이를 통해 네임노드는 해당 블럭이 정상적으로 저장되었다는 것을 인지
i. DFSOutputStream의 내부 쓰레드인 ResponseProcessor는 파이프라인에 있는 모든 데이터노드로부터 승인 메시지를 받게 되면 해당 패킷을 승인큐에서 제거, 만약 패킷을 전송하는 중에 장애가 발생하면 승인큐에 있는 모든 패킷을 데이터큐로 이동, 네임노드에게서 장애가 발생한 데이터노드가 제거된 새로운 데이터노드 목록을 내려받음, 마지막으로 새로운 파이프라인을 생성한 후 다시 패킷 전송작업을 시작

3) 파일 닫기
   a. 클라이언트는 DistributedFileSystem 의 close 메서드를 호출
   b. DistributedFileSystem 은 DFSOutputStream 의 close 메서드를 호출, 이 메서드는 DFSOutputStream에 남아 있는 모든 패킷을 파이프라인으로 flush 함
   c. DFSOutputStream 은 네임노드의 complete 메서드를 호출해 패킷이 정상적으로 저장되었는지 확인, 네임노드의 최소 블록 복제본 수만 저장했다면 complete 메서드는 true 를 반환, DFSOutputStream은 true를 반환 받으면 파일 저장이 완료된 것으로 설정

4. HDFS 파일의 읽기

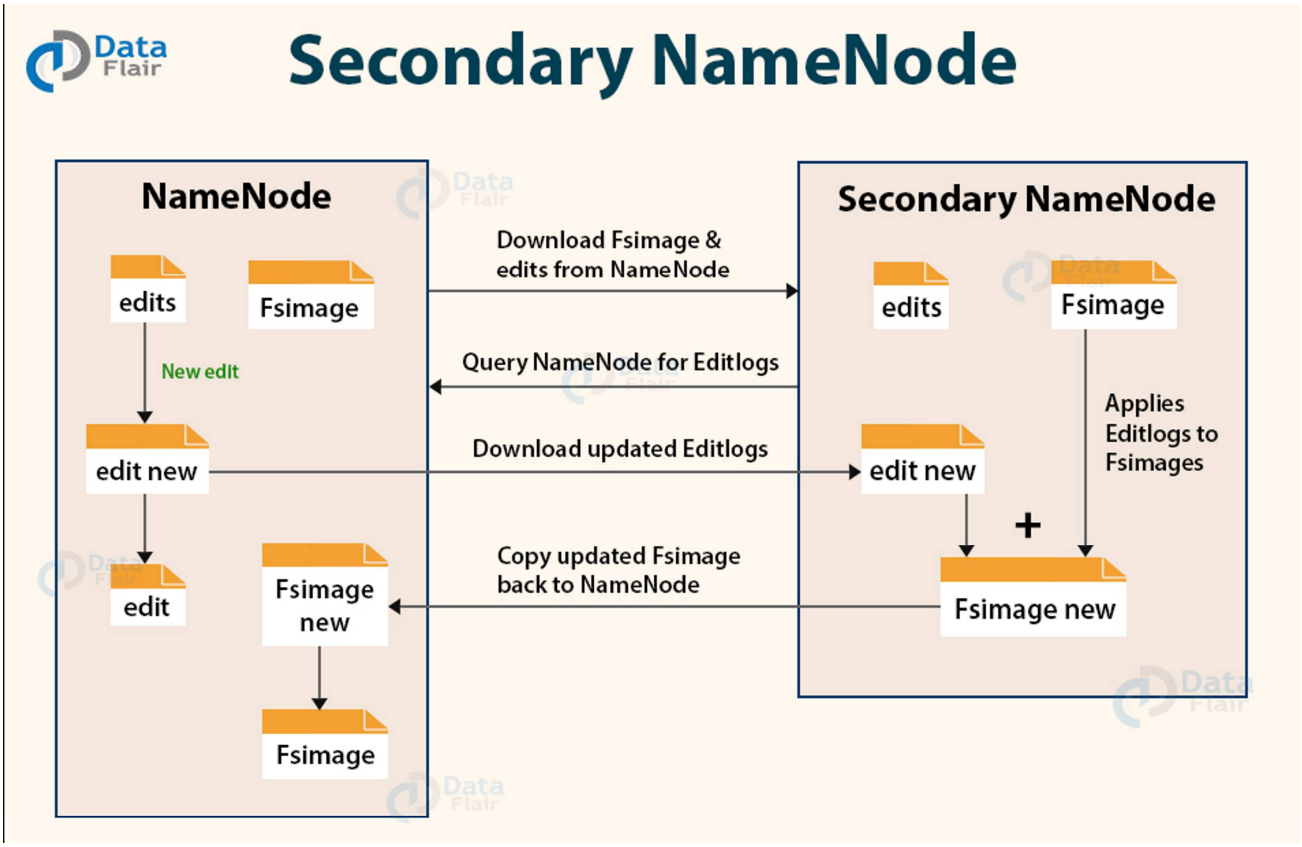


1) 파일 조회 요청
   a. 클라이언트는 DistributedFileSystem 의 open 메서드를 호출하여 스트림 객체 생성을 요청
   b. DistributedFileSystem 은 FSDataInputStream 객체를 생성, 이 때 FSDataInputStream 은 DFSDataInputStream 과 DFSInputStream 을 차례대로 래핑, DistributedFileSystem 은 마지막에 래핑이 되는 DFSInputStream을 생성하기 위해 FSDataInputStream 을 생성하기 위해 FSDataInputStream의 open 메서드를 호출
   c. FSDataInputStream 은 DFSInputStream 을 생성, 이 때 DFSInputStream 은 네임노드의 getBlockLocations 메서드를 호출해 대상 파일의 블록 위치 정보를 조회
   d. 네임노드는 조회 대상 파일의 블록 위치 목록을 생성한 후 목록을 클라이언트에 가까운 순으로 정렬, 정렬이 완료되면 DFSInputStream에 정렬된 블록 위치 목록을 반환, DistributedFileSystem 은 FSDataInputStream 로부터 전달 받은 DFSInputStream을 이용해 FSDataInputStream 객체로 생성해서 클라이언트에게 반환
2) 블록 조회
   a. 클라이언트는 입력 스트림 객체의 read 메서드를 호출해 스트림 조회를 요청
   b. DFSInputStream 은 첫 번째 블럭과 가장 가까운 데이터노드를 조회한 후 해당 블럭을 조회하기 위한 리더를 생, 클라이언트와 블럭이 저장된 데이터노드가 같은 서버에 있다면 로컬 블록 리더인 BlockReaderLocal 을 생성하고, 그렇지 않다면 RemoteBlockReader를 생성
   c. DFSInputStream 은 리더의 read 메서드를 호출하여 블럭을 조회, BlockReaderLocal 은 로컬 파일 시스템에 저장된 블럭을 DFSInputStream 에게 반환, RemoteBlockReader 은 원격에 있는 데이터노등게 블럭을 요청하며 데이터노드의 DataXceiverServer가 블럭을 DFSInputStream 에게 반환
   d. DFSInputStream 은 파일을 모두 읽을 때까지 계속해서 블럭을 조회, 만약 DFSInputStream 이 저장하고 있던 블럭을 모두 읽었는데도 파일을 모두 읽지 못

했다면 네임모드의 getBlockLocations 메서드를 호출하여 필요한 블록 위치 정보를 다시 요청, 위와 같이 파일을 끊김 없이 연속적으로 읽기 때문에 클라이언트는 스트리밍 데이터를 읽는 것처럼 처리할 수 있음

3) 입력 스트림 닫기
   a. 클라이언트는 입력 스트림 객체의 close 메서드를 호출, 스트림 닫기를 요청
   b. DFSInputStream 은 데이터노드와 연결되어 있는 컨넥션을 종료, 블록 조회용으로 사용했던 리더도 close 함

5. 보조네임노드



● 네임노드는 메타데이터를 메모리에서 처리
● 서버 재 부팅 시 모든 메타데이터 유실가능성 때문에 HDFS는 editslog 와 fsimage 라는 두 개의 파일을 생성
● editslog 는 HDFS의 모든 변경 이력을 저장
● HDFS는 클라이언트가 파일을 저장, 삭제, 이동하는 경우 editslog와 메모리에 로딩되어 있는 메타데이터에 기록
● fsimage 는 메모리에 저장된 메타데이터의 파일 시스템 이미지를 저장한 파일

1) 네임노드 구동 시 동작 절차
   a. 네임노드가 구동되면 로컬에 저장된 fsimage와 editslog를 조회
   b. 메모리에 fsimage를 로딩, 파일 시스템 이미지를 생성
   c. 메모리에 로딩된 파일 시스템 이미지에 editslog 에 기록된 변경 이력을 적용
   d. 메모리에 로딩된 파일 시스템 이미지를 이용해 fsimage 파일을 갱신
   e. editslog를 초기화

f. 데이터노드가 전송한 블럭리포트를 메모리에 로딩된 파일 시스템 이미지에 적용

● editslog 파일은 별도의 크기 제한이 없기 때문에 무한대로 커질 수 있음
● editslog 가 너무 크면 위 세 번재 단계를 진행할 때 많은 시간이 소요
● 이러한 문제점을 해결하기 위해 HFDS는 보조네임노드(Secondary Name Node)를 사용
● 보조네임노드는 주기적으로 네임노드의 fsimage를 갱신, 이러한 작업을 체크포인트 라고 함

2) 보조네임노드의 체크포인팅 단계
   a. 보조네임노드는 네임노드에게 editslog를 롤링(현재 로그 파일의 이름을 변경하고 원래 이름으로 새 로그 파일을 만드는 작업)할 것을 요청
   b. 네임노드는 기존 editslog를 롤링한 후 editslog.new 를 생성
   c. 보조네임노드는 네임노드에 저장된 롤링된 editslog 와 fsimage 를 다운로드
   d. 보조네임노드는 다운받은 fsimage 를 메로리에 로딩, editslog 에 있는 변경 이력을 메모리에 로딩된 파일 시스템 이미지에 적용
   e. 메모리 갱신이 완료되면 새로운 fsimage 를 생성, 이 파일(fsimage.ckpt)을 체크포인팅할 때 사용
   f. 보조네임노드는 fsimage.ckpt 를 네임노드에게 전송
   g. 네임노드는 로컬에 저장되어 있는 fsimage를 보조네임노드가 전송한 fsimage.ckpt로 갱신하고 editslog.new 파일을 editslog 로 변경

● 체크포인팅을 1시간 마다 일어나며 환경설정에서 변경 가능