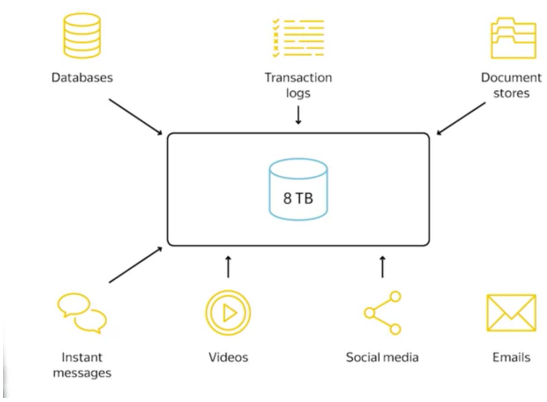
DFS, HDFS Architecture and Scalability Problems @Sieun Bae

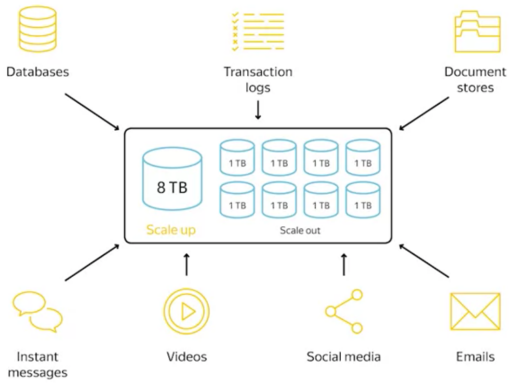
coursera lecture (link)

Scaling Distributed File System

Storage layer and the data processing paradigms and technologies

DFS(Distributed File System)

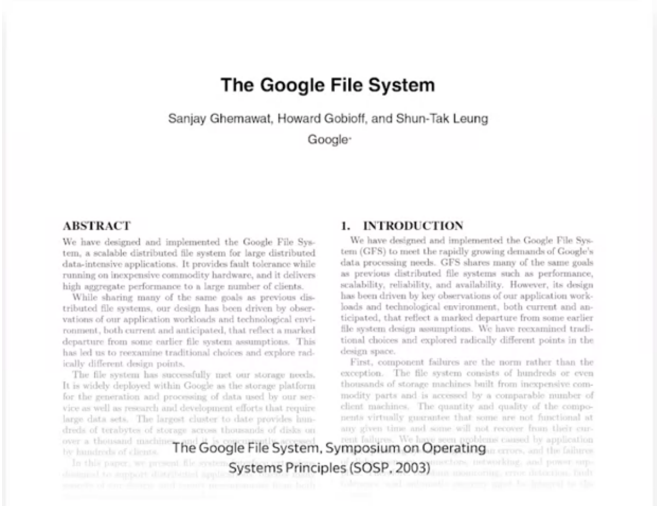




- Scale up / Vertical scaling
- Get a big capacity node
- lower latency

- Scale out / Horizontal scaling
- higher latency, bigger storage

GFS (Google File System)



The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google·

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require large data sets. The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients.

1. INTRODUCTION

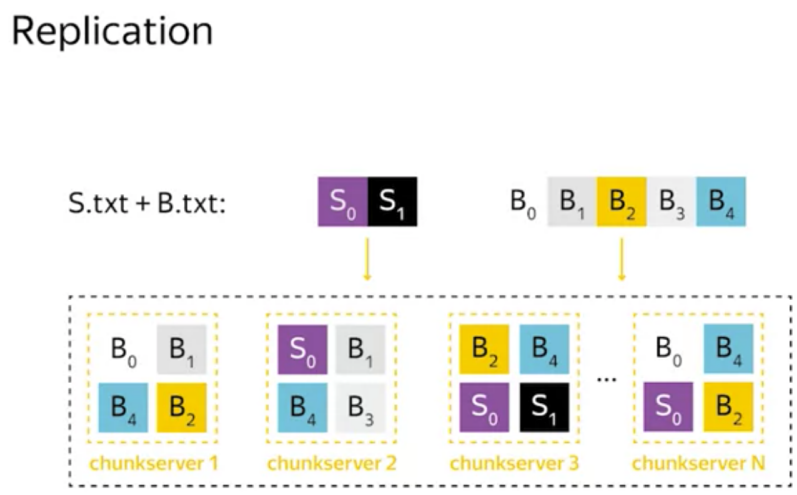
We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines. The quantity and quality of the components virtually guarantee that some are not functional at any given time and some will not recover from their current failures. We have seen problems caused by application

The Google File System, Symposium on Operating Systems Principles (SOSP, 2003)

- components failures are a norm (→ replication) 데이터는 기술적으로 복제됨
- even space utilisation
- write-once-read-many data usage pattern

  is the primary reason to simplify the DFS API.. (can append, cannot modify)

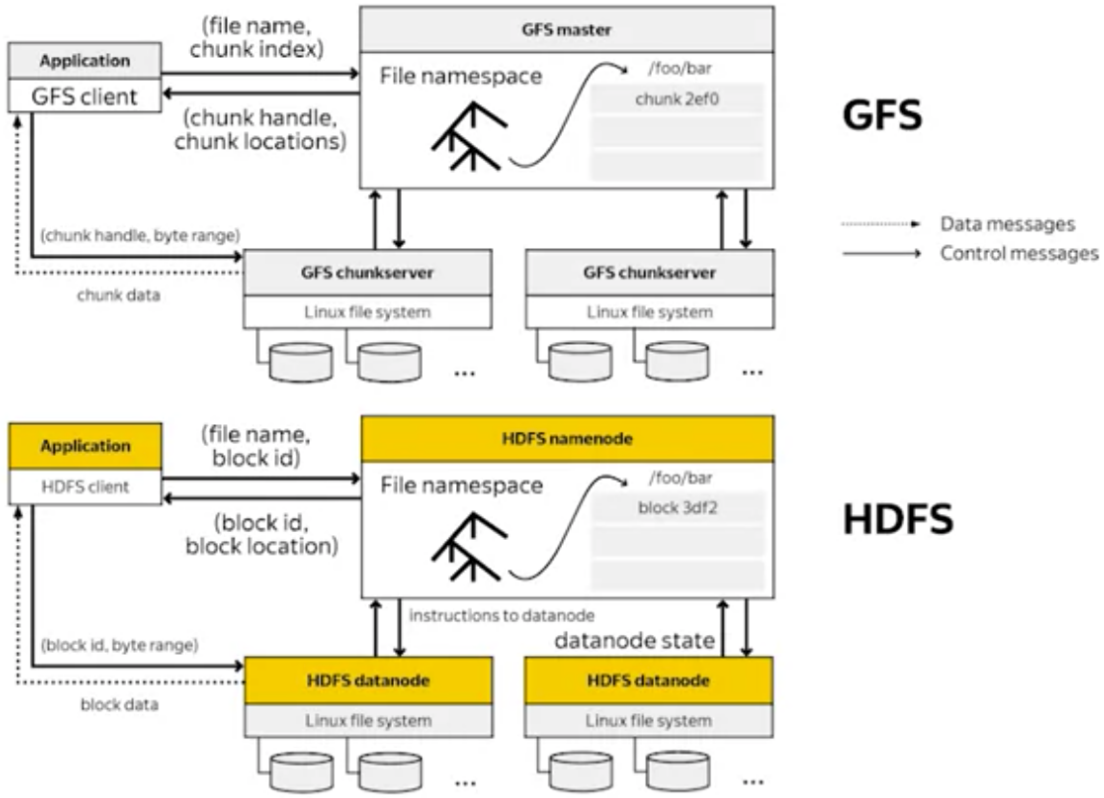


Storage System = Chunkserver = Data Node

HDFS (Hadoop Distributed File System)

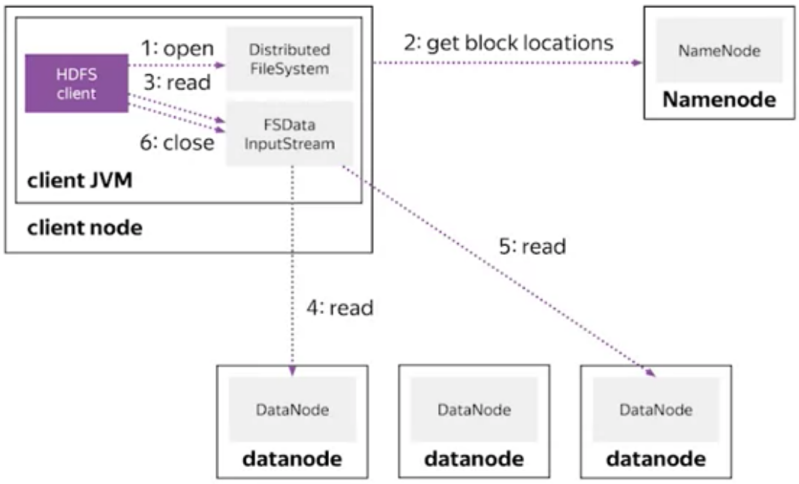
- **open source implementations of GFS**
- along with local FS, DFS need metadata (administrative info. about creation time, access properties..)
- master node stores all metadata in memory
- namenode vs masternode, (HDFS)Java vs (GFS)C++

- HDFS client provides CLI : no need to write in PL, binary RPC protocol, HTTP protocol



- How to read files from HDFS.

1. request name node to get info. about file blocks' locations (blocks are distributed over different machines, but all of complexity is hidden behind HDFS API)
2. if at some point a datanode you retrieve data from died, you get this data from another replica. get data from closest (difficult to measure... physical distance, unpredictable system load.. metric overutilization...). → data center topology

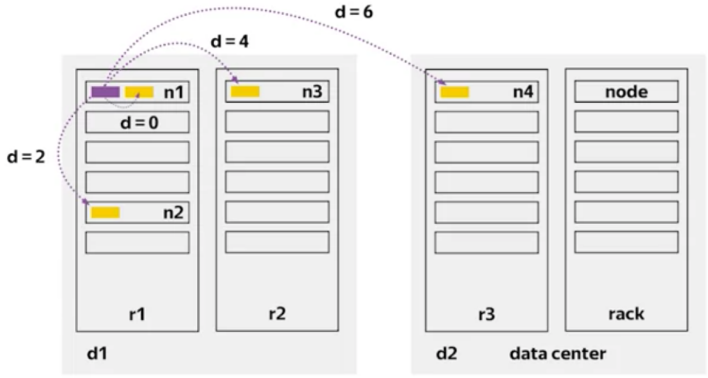


d=0 : 동일한 시스템에서 사용할 경우 데이터 인접성(data locality)을 사용하여 추가 RPC 코드 없이 하드드라이브에서 데이터 읽을 수 있음

d=2 : same rack

d=4 : another rack

d=6 : another data center

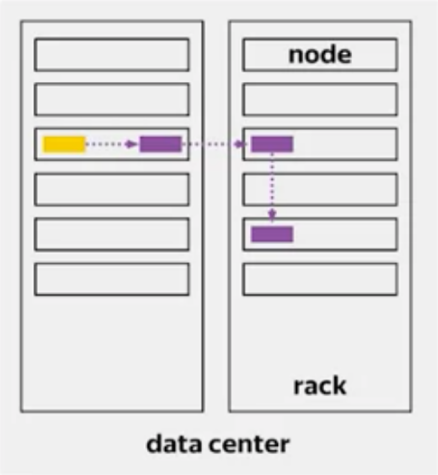


- Redundancy Model

HDFS에 데이터 블록 작성

→ Hadoop storage에 복제본(R1) 배포 (일반적으로 DataNode 시스템에서 쓰는 경우 동일 노드에 위치하나, 그렇지 않은 경우 첫 번째 DataNode가 임의로 선택됨)

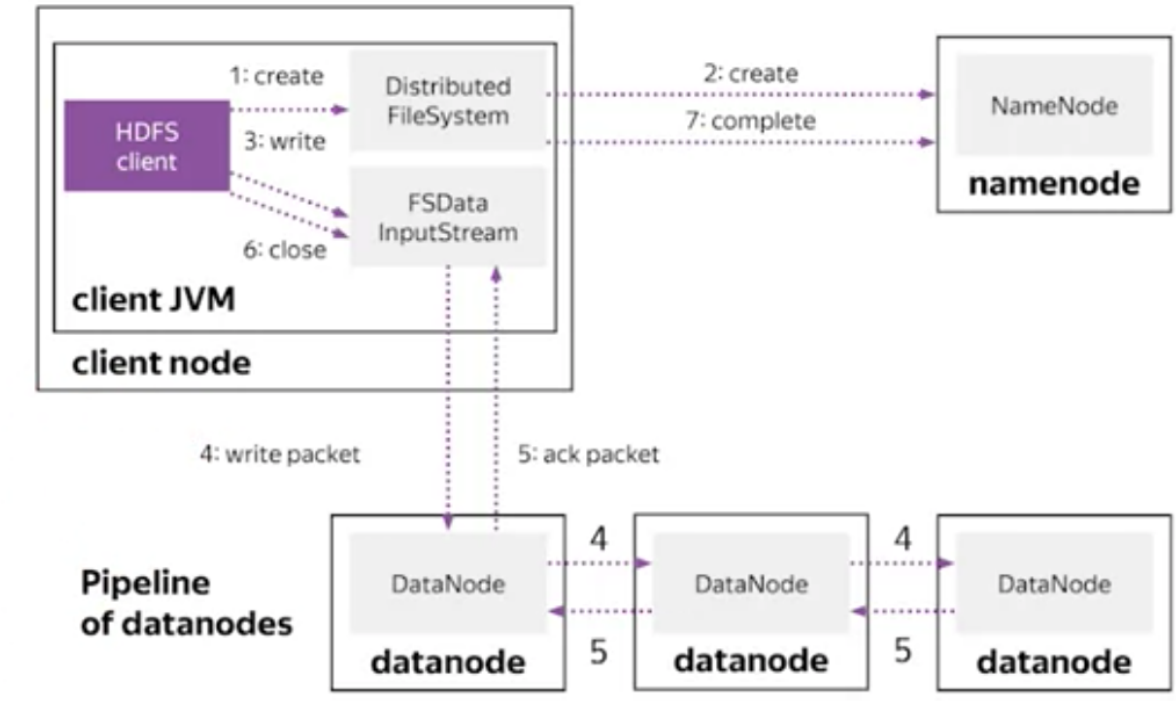
→ 두 번째 복제본(R2) 다른 랙에 배치



→ 세 번째 복제본(R3) 두 번째 복제본(R2)과 동일한 랙의 다른 컴퓨터에 위치 (R2-R3 간 rack network utilization 간 추가 비용 없음)

같은 랙에 많은 복사본 피하려는 경향 있음

Detailed data flow of a right apparition

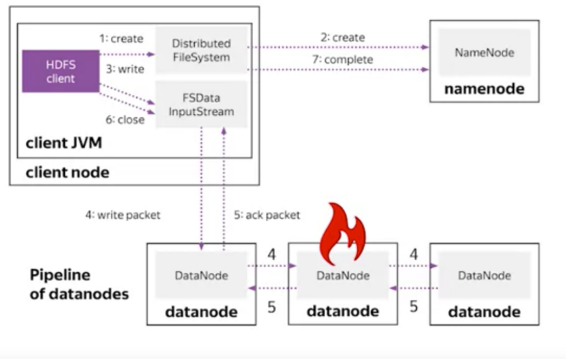


1. request and NameNode via RPC protocol
2. NameNode validates if you have rights to create a file and no naming conflicts
3. HDFS client requests a list of datanodes to put a fraction of blocks of the file
    1. form a pipeline as your first client sends packets of data to the closest datanode

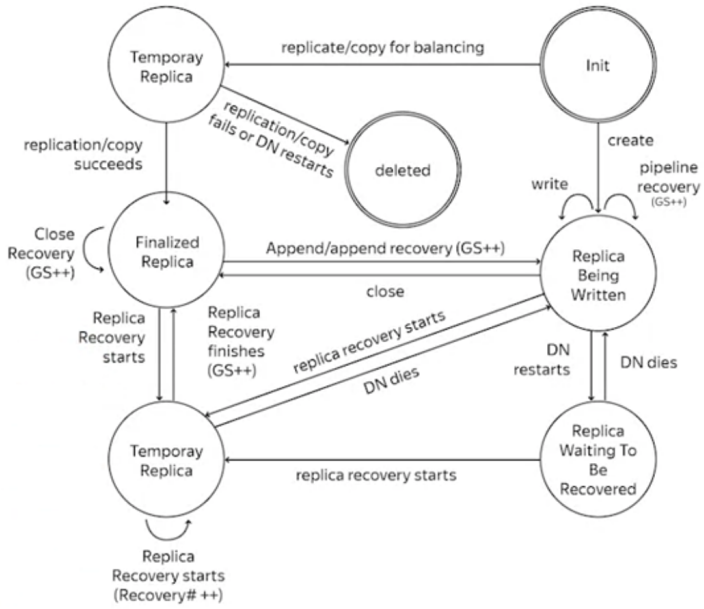
2. the later one transfers copies of packets through a datanode pipeline
3. store all, send ack packets back

4. if something goes wrong...

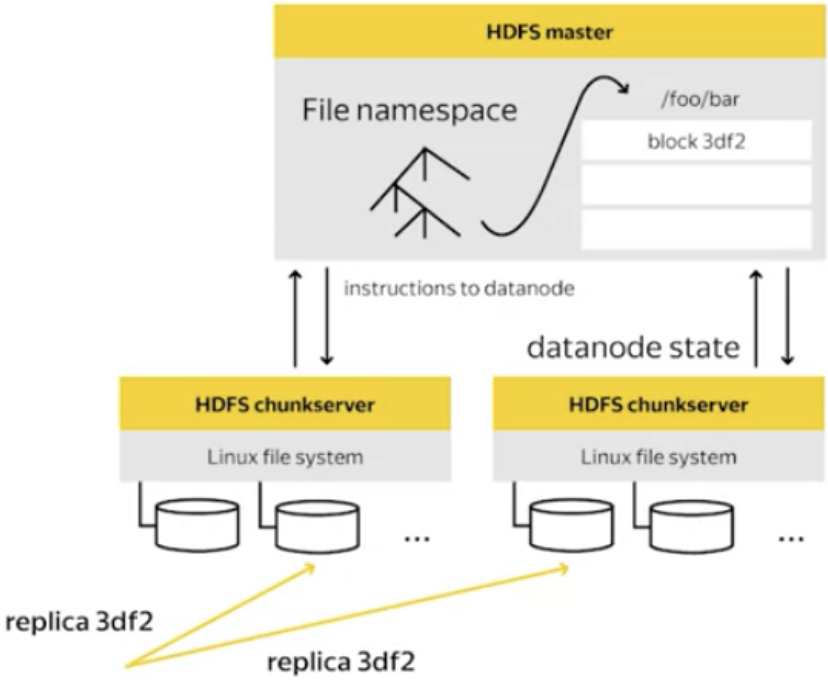
   client closes the datanode pipeline → marks the misbehaving datanode 'bad' → request a replacement for the 'bad' datanodes from a NameNode. → NEW datanode pipeline will be organized



⇒ All these happens transparently to a user, HDFS hides all this complexity for you.

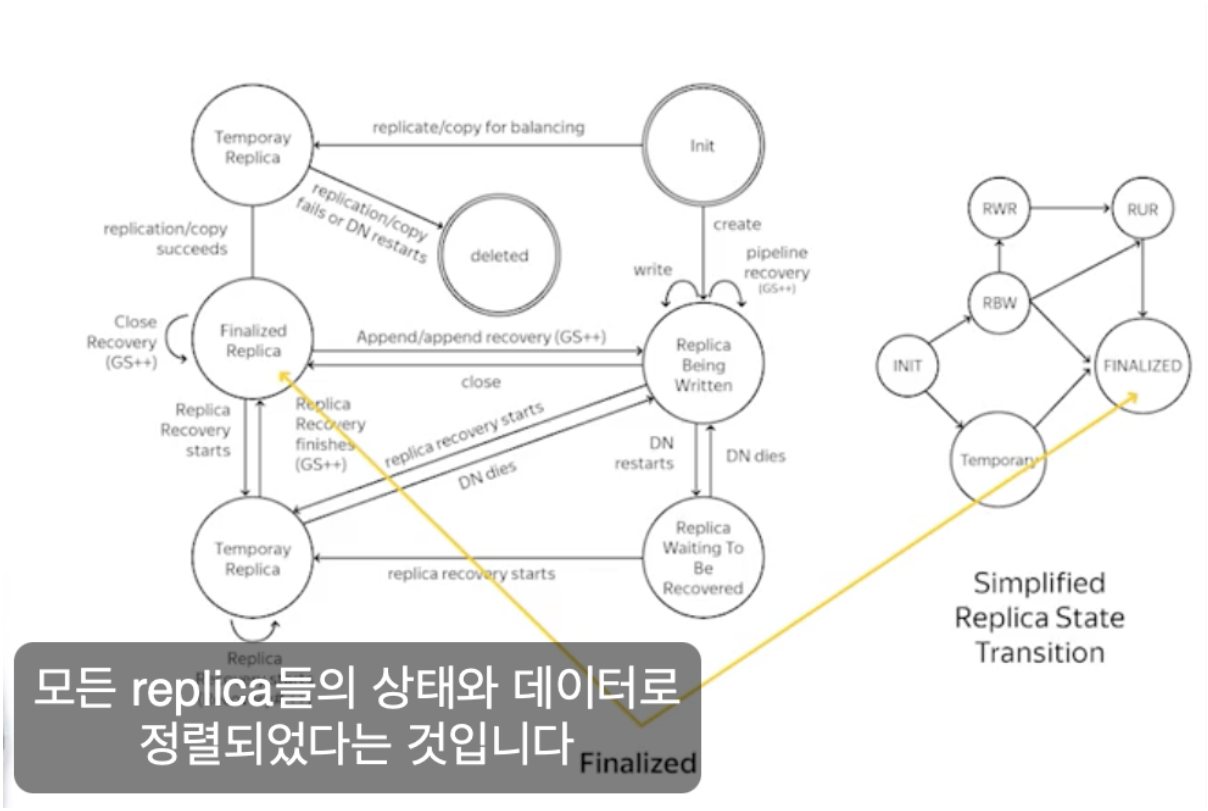


Block and Replica States, Recovery Process

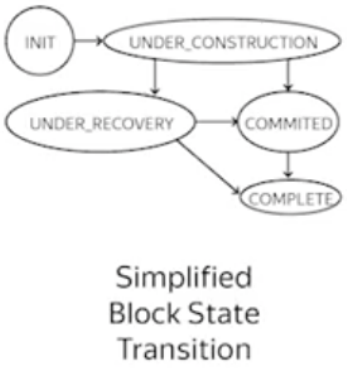


replica(복제): 데이터노드에 있는 물리적 데이터 저장공간

block(블록): 네임노드에 있는 위치정보(meta-info) 저장공간, replica의 위치, 상태에 대한 정보

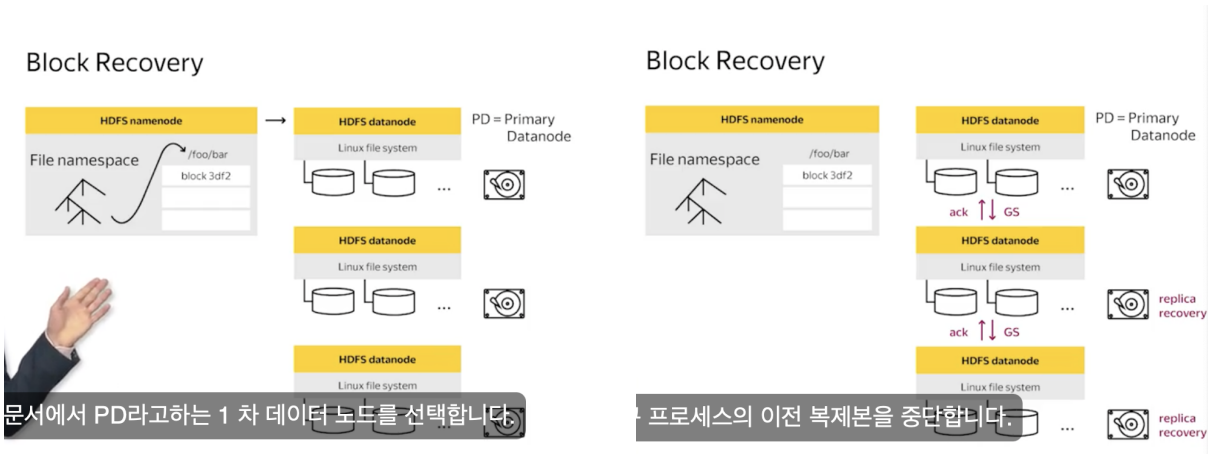


frozen, 읽기 일관성, GS번호, 데이터 내구성, RWR, RUR, RBW

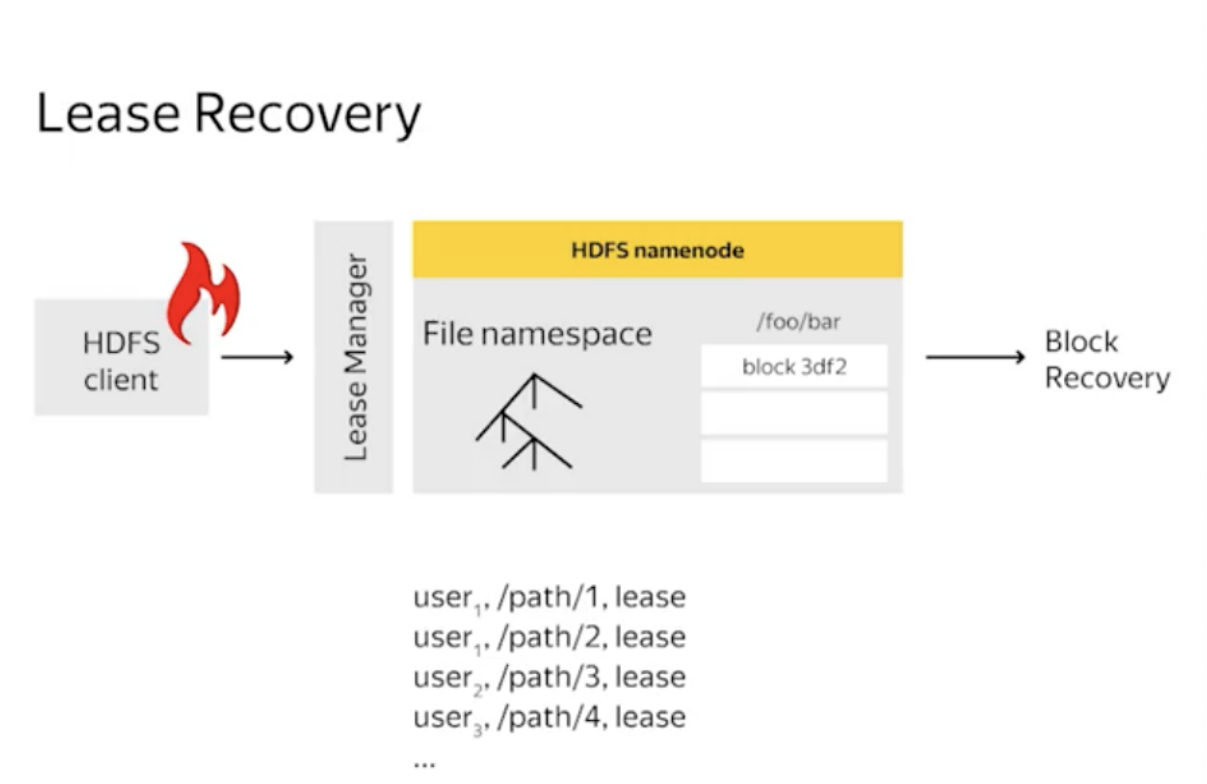


Simplified Block State Transition

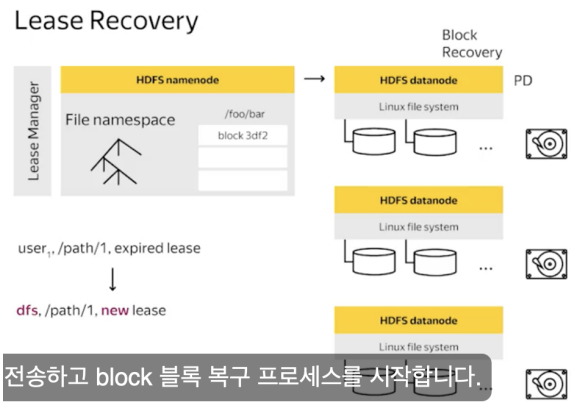
- 블록상태는 메모리에 저장, 디스크는 x, under_construction상태 노드 생성
- could we have "finalized" replicas with different visible lengths or generation stamps? NO
- during failures "temporary" replicas are just removed. it is impossible for a replica to transition from the Temporary to RUR state.

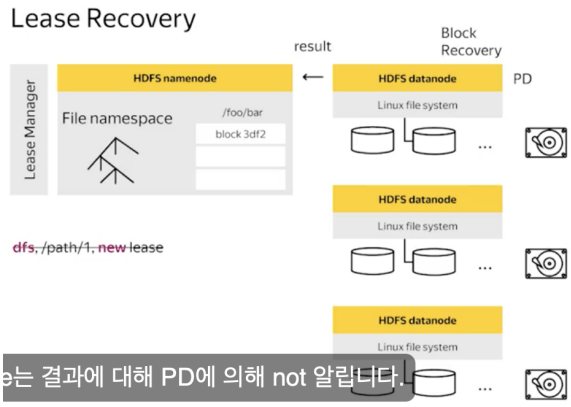


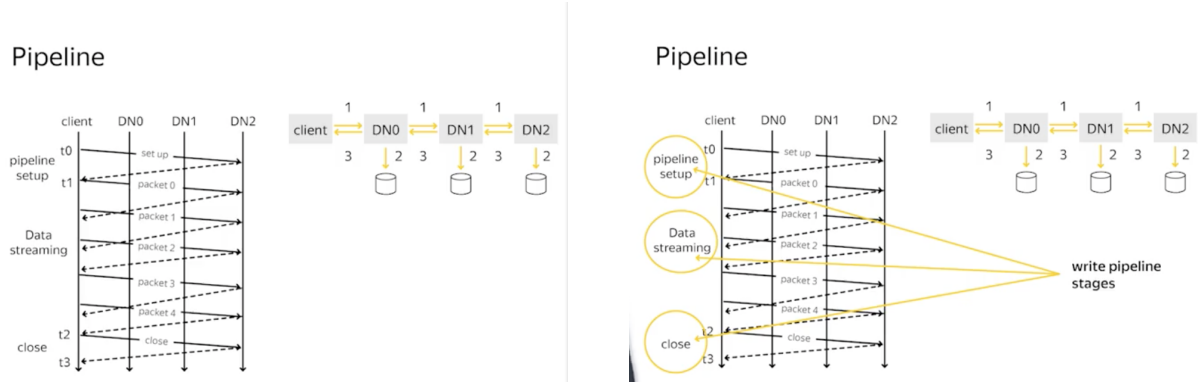
It is not possible to transition a replica from RWR replica's state to under_recovery. under_recovery is a block's state.



soft, hard expiry... 2가지 보장: concurrency control, consistency guarantee!







pipeline setup → data streaming (t1, t2) 전에 패킷이 완료되지 않아도 다음 패킷을 전송받을 수 있음 (RBW replica: the process is already started, but not finalized, so other options are not possible) → close (replica to finalized state)

Flush 패킷: synclonized..

- if recovery needed at pipeline setup stage, (new file, append mode): 파이프라인 포기
- if recovery needed at data streaming stage: 모든 연결 닫고 dataNode 파이프라인 할당, 새 패킷 전송 중지, 기존 파이프라인, nameNode에서 새로운 생성 스탬프 요청, 좋은 dataNode에서 재구축 요청, 일부 패킷 재전송될 수 있으나 여분의 디스크 IO는 안됨 (바이트_
- if close stage: 좋은 dataNode에서 재구축 요청, bump generation stamp와 finalize replica

Summary

- you can **draw** State block and replica transition tables
- you can **identify** write pipeline stages and associated recovery process
- you can **list** 4 recovery processes and **explain** their purpose (lease recovery → block recovery → replica recovery; pipeline recovery)

블록 상태 logical, physical 디스크 내용 같아야함. nameNode 설계문서에서 PD 1 선택 (nameNode로부터의 PD(Primary dataNode) 요청, 새로운 GS, 복구 프로세스를 위한 replica의 위치

Replica recovery 활성 클라이언트를 복제본으로 바로 중단하는 작업 포함

HDFS Client

how to use HDFS client: LocalFile vs DistributedFile
https://en.wikipedia.org/wiki/Command-line_interface

man

```
hdfs dfs -help
```

info

```
hdfs dfs -usage <unility_name>
```

ls

```
hdfs dfs -ls -R -h /data/wiki
hdfs dfs -du -h /data/wiki //-du 옵션으로 replica의 총합 메모리를 -h 플래그로 사람이 읽을 수 있도록 출력
```

mkdir

```
hdfs dfs -ls
hdfs dfs -mkdir deep/nested/path //error
hdfs dfs -mkdir -p deep/nested/path //no error
hdfs dfs -ls -R deep //하위 디렉토리 확인
```

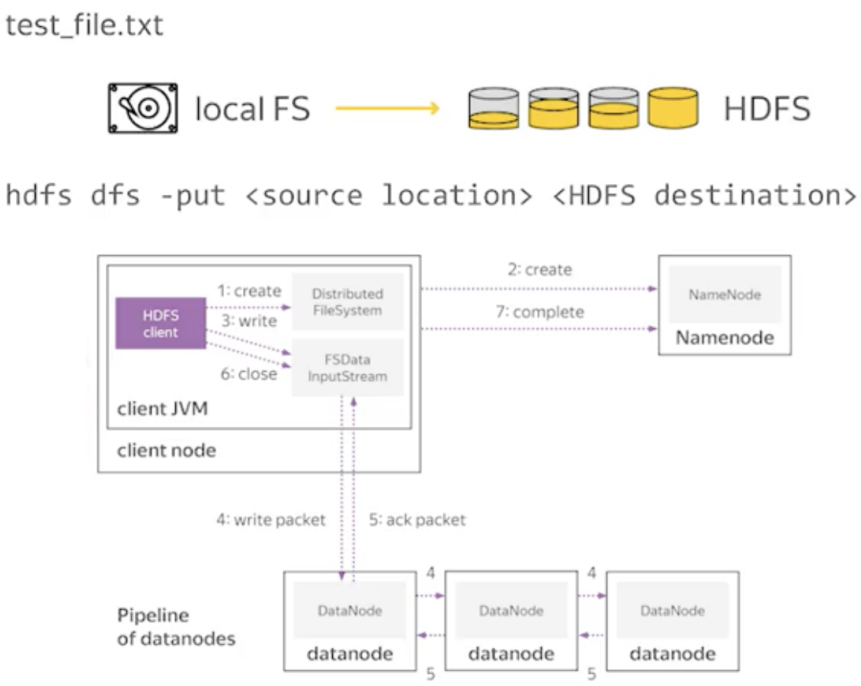
rm 로컬은 변화 x

```
hdfs dfs -rm deep
hdfs dfs -rm -r deep //일부 하둡 클러스터에서는 trash 생성 못할수도
hdfs dfs -mkdir -p deep/nested/path && hdfs dfs -rm -r -skipTrash deep //shift+delete
```

touchz 로컬에 약간의 변화

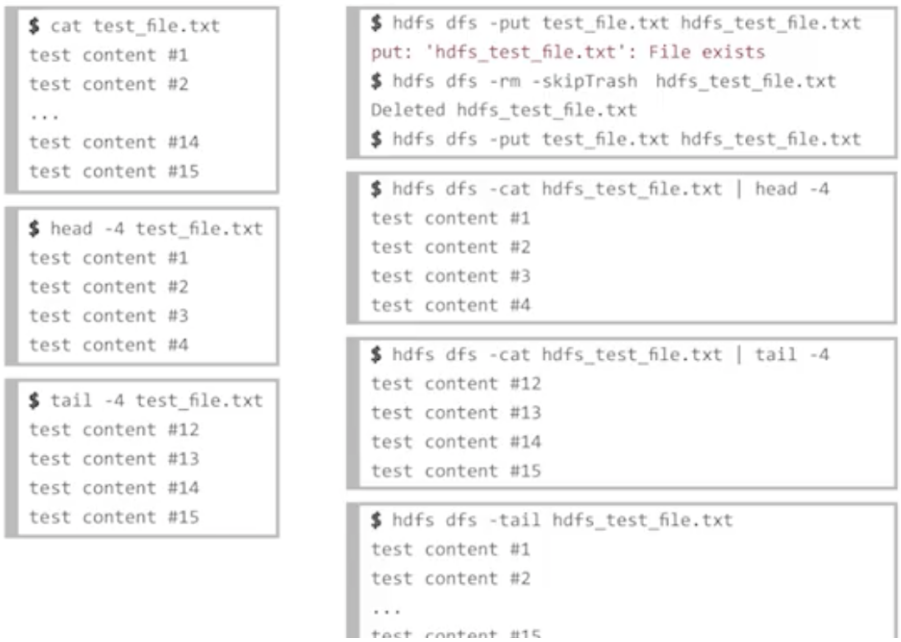
```
hdfs dfs -touchz file.txt
hdfs dfs -ls
```

put local to hdfs



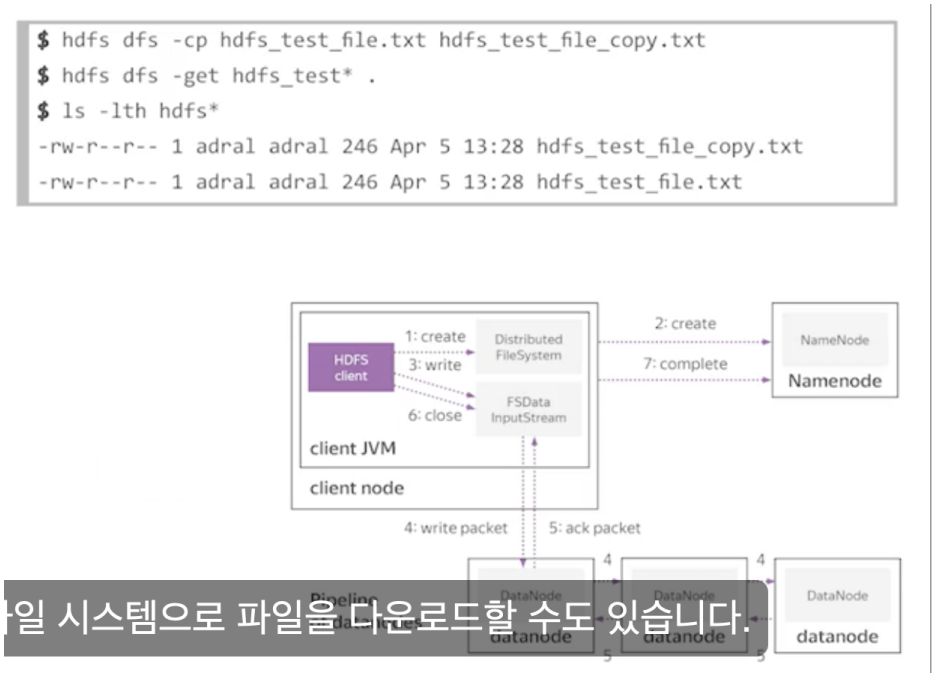

```
hdfs dfs -put <source location> <HDFS destination>
```

cat, head, tail text(local) vs binary(hdfs)



tail 명령어의 경우 1킬로바이트의 하위 데이터 출력

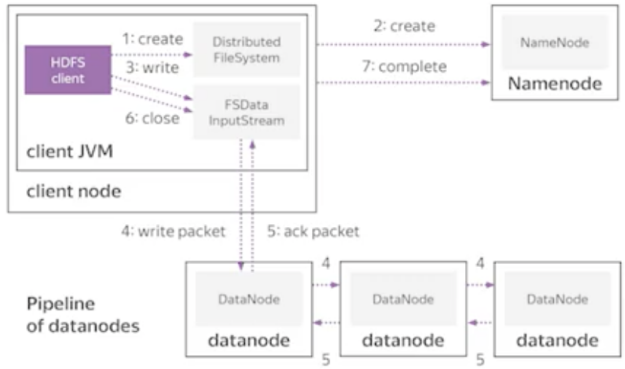
파일 다운로드(접두사를 통해 로컬로 다운로드)



merge

```
$ hdfs dfs -cp hdfs_test_file.txt hdfs_test_file_copy.txt
$ hdfs dfs -get hdfs_test* .
$ ls -lth hdfs*
-rw-r--r-- 1 adral adral 246 Apr 5 13:28 hdfs_test_file_copy.txt
-rw-r--r-- 1 adral adral 246 Apr 5 13:28 hdfs_test_file.txt
$ hdfs dfs -getmerge hdfs_test* hdfs_meged.txt
$ ls -lth hdfs*
-rw-r--r-- 1 adral adral 492 Apr 5 13:32 hdfs_meged.txt
-rw-r--r-- 1 adral adral 246 Apr 5 13:28 hdfs_test_file_copy.txt
-rw-r--r-- 1 adral adral 246 Apr 5 13:28 hdfs_test_file.txt
```

```
$ cat hdfs_meged.txt
test content #1
test content #2
...
test content #15
test content #1
...
test content #14
test content #15
```



chown, hdfs groups,

setrep, hdfs fsck

```
$ time hdfs dfs -setrep -w 1 hdfs_test_file.txt
Replication 1 set: hdfs_test_file.txt
Waiting for hdfs_test_file.txt ...
WARNING: the waiting time may be long for DECREASING the number of
replications...done
real 0m13.148s
user 0m4.232s
sys 0m0.156s
```

```
$ hdfs dfs -ls
Found 3 items
drwx------    - adral adral 0   2017-04-05 13:00 .Trash
-rw-r--r--    1   adral adral 246 2017-04-05 13:03 hdfs_test_file.txt
-rw-r--r--    3   adral adral 246 2017-04-05 13:28 hdfs_test_file_copy.txt
```

```
$ time hdfs dfs -setrep -w 2 hdfs_test_file.txt
Replication 2 set: hdfs_test_file.txt
Waiting for hdfs_test_file.txt .... done
real 0m13.168s
user 0m4.092s
sys 0m0.148s
```

find

```
$ hdfs dfs -find /data/wiki -name ''*part*''
/data/wiki/en_articles_part
/data/wiki/en_articles_part/articles-part
```

```
$ hdfs dfs -find /data -name ''*part*'' - iname ''*Article*''
/data/wiki/en_articles_part
/data/wiki/en_articles_part/articles-part
```

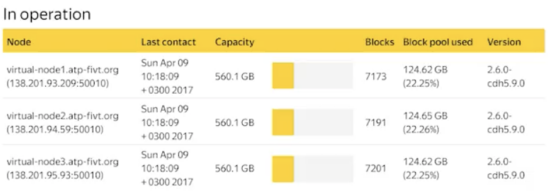
Summary

- you can **request meta-information** from Namenode and change its structure (ls, mkdir, rm, rm -r (-skipTrash), touch, mv)
- you can **read and write** data from and to Datanodes in HDFS (put, cat, head, tail, get, getmerge)
- you can **change replication factor** of files and **get detailed information** about data in HDFS (chown, hdfs groups; setrep; hdfs fsck; find)

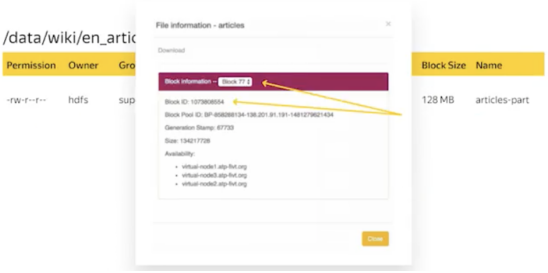
Gentle Introduction into "curl"

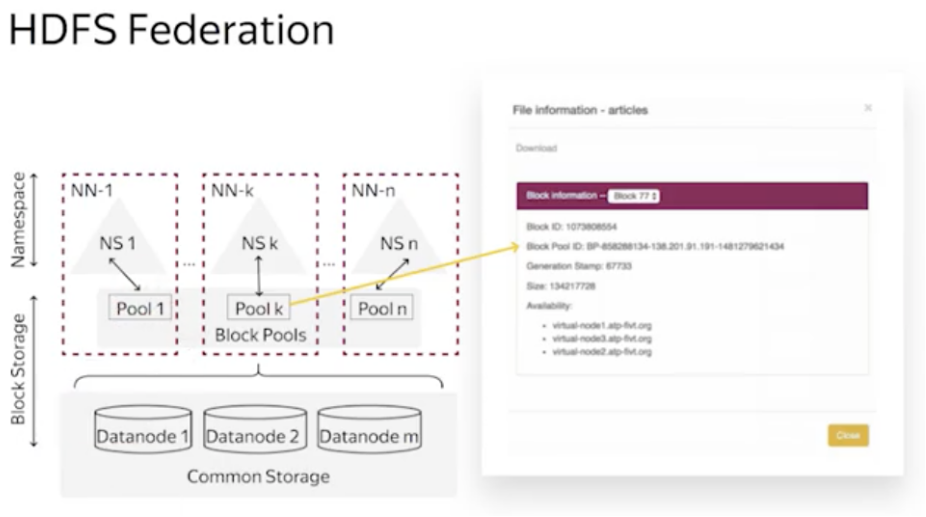
Web UI, REST API

Datanode Information



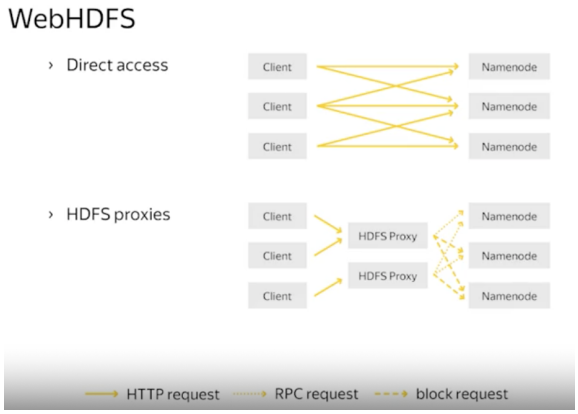
Browse Directory

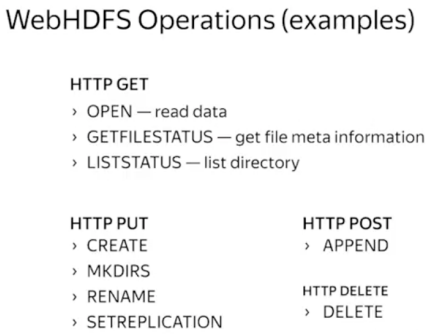




How is HDFS Federation scaled? there is no coordination necessary between Namenodes, so you don't need to provide performance host machines. It is called horizontal scaling (or "scale out").

Web UI: 읽기전용, 쓰기까지? → **WebHDFS**





```
$ curl -i http://virtual-master.atp-fivt.org:50070/webhdfs/v1/data/access_logs/big_log/
access.log.2015-12-10\?op=OPEN
HTTP/1.1 307 TEMPORARY_REDIRECT
Cache-Control: no-cache
Expires: Sun, 09 Apr 2017 08:32:02 GMT
Date: Sun, 09 Apr 2017 08:32:02 GMT
Pragma: no-cache
Expires: Sun, 09 Apr 2017 08:32:02 GMT
Date: Sun, 09 Apr 2017 08:32:02 GMT
Pragma: no-cache
Content-Type: application/octet-stream
X-FRAME-OPTIONS: SAMEORIGIN
Location: http://virtual-node1.atp-fivt.org:50075/webhdfs/v1/data/access_logs/
big_log/access.log.2015-12-10?op=OPEN&namenoderpcaddress=virtual-master.atp-fivt.
org:8020&offset=0
Content-Length: 0
Server: Jetty(6.1.26.cloudera.4)
```

```
$ curl -i http://virtual-node1.atp-fivt.org:50075/webhdfs/v1/data/access_logs/big_log/
access.log.2015-12-10\?op\=OPEN\&namenoderpcaddress\=virtual-master.atp-fivt.org:8020\&of
fset\=0\&length\=100
HTTP/1.1 200 OK
Access-Control-Allow-Methods: GET
Access-Control-Allow-Origin: *
Content-Type: application/octet-stream
Connection: close
Content-Length: 100
41.190.60.121 - - [10/Dec/2015:00:00:00 +0400] ''GET /cgi-bin/commandit.cgi HTTP/1.1''
404 0 ''-'' ''Mozi%
```

actual data

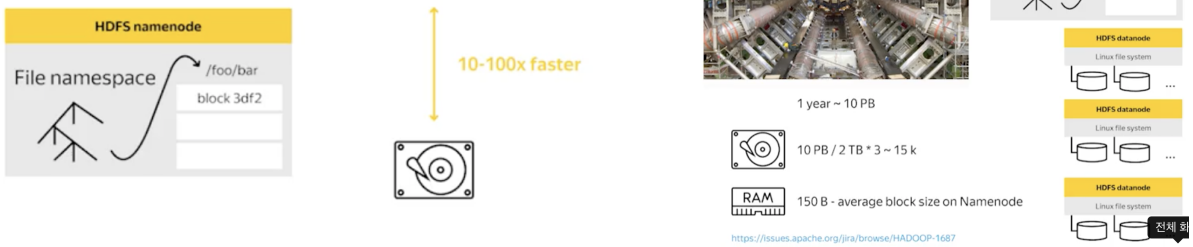
```
$ curl -i -L http://virtual-master.atp-fivt.org:50070/webhdfs/v1/data/access_logs/big_log/
access.log.2015-12-10\?op=OPEN\&length\=100
HTTP/1.1 307 TEMPORARY_REDIRECT
…
Content-Type: application/octet-stream
X-FRAME-OPTIONS: SAMEORIGIN
Location: http://virtual-node2.atp-fivt.org:50075/webhdfs/v1/data/access_logs/big_
log/access.log.2015-12-10?op=OPEN&namenoderpcaddress=virtual-master.atp-fivt.org:
8020&length=100&offset=0
Content-Length: 0
Server: Jetty(6.1.26.cloudera.4)

HTTP/1.1 200 OK
Access-Control-Allow-Methods: GET
Access-Control-Allow-Origin: *
Content-Type: application/octet-stream
Connection: close
Content-Length: 100
41.190.60.121 - - [10/Dec/2015:00:00:00 +0400] ''GET /cgi-bin/commandit.cgi HTTP/1.1''
404 0 ''-'' ''Mozi%
```

Summary

- you can use HDFS Web UI to list folders and find detailed information about file blocks
- you can use curl to move files through WebHDFS REST API, read / write files, ...

http://namenode-server:50070/dfshealth.html#tab-datanode

Namenode Architecture

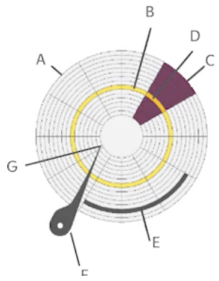


NameNode is a service responsible for keeping hierarchy of folders and files. NameNode stores all of this data in memory.

RAM speed of reading and writing data is fostered by order of magnitude compared to a disk.

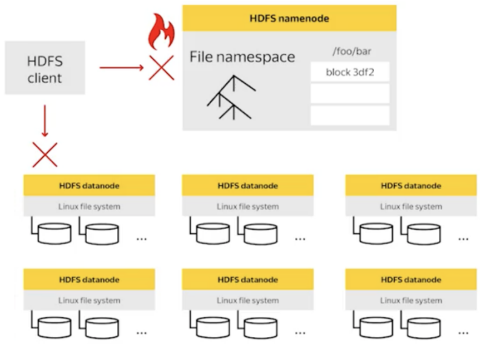
small file problems : smaller than the HDFS block size.. (default 64MB)

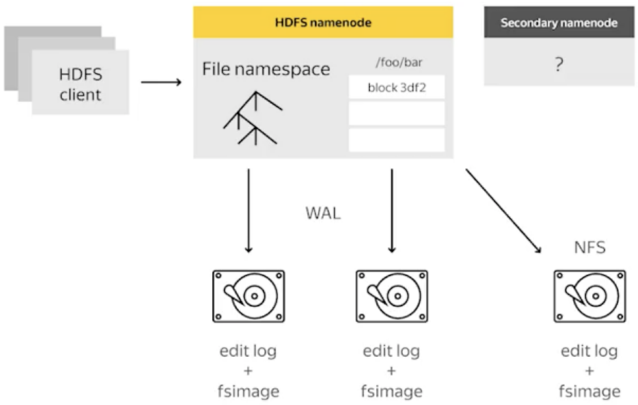
Default Block Size



Samsung 940 PRO SSD:
* reading speed - 3.5 GB/sec
* 128 MB - 30-40 ms

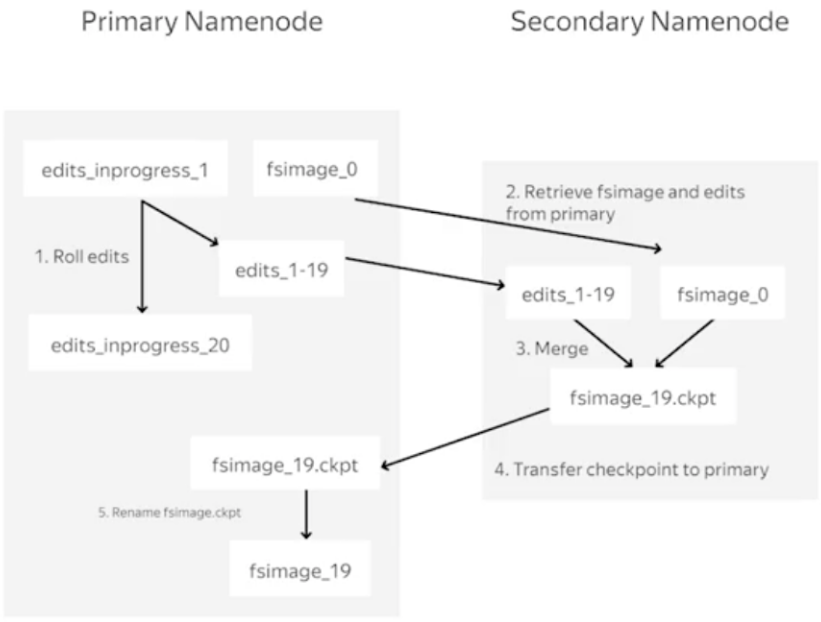
average block size = **128 MB** 하드드라이브 공간 활용도 쉽게 유지, 1%의 오버헤드, 작은 크기로 유지 가능한 최적의 사이즈





WAL(Write Ahead Log) 데이터베이스의 데이터 파일에 대한 모든 조작 기록을 보관, WAL 아카이브 저장 시 레코드 정보를 다른 곳 (NFS(Network File System)마운트 위치 등)에 저장할 수 있음

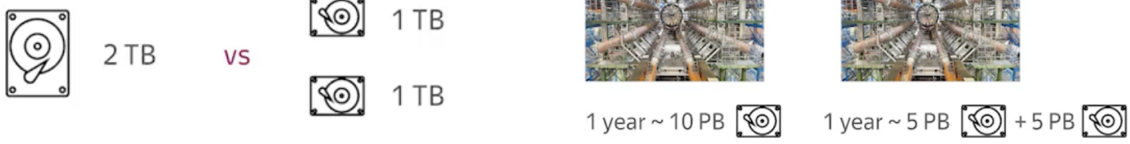
It is a normal scenario when replay of one week of transactions from edit log will take several hours to boot a NameNode. As you can guess, it is not appropriate for a high demand service. For this reason, Hadoop developers invented secondary NameNode.



robust and poorly asynchronous process.

secondary NameNode consumes the same amount of RAM to build a new fsimage.

SecondaryNode = Checkpoint Namenode **≠ Backup Node**



35 days vs 17.5 days

Summary

- you can **explain and reason about** HDFS Namenode architecture (RAM; fsimage + edit log; block size)
- you can **estimate** required resources for a Hadoop cluster
- you can **explain** what small files problem is and where a bottleneck is
- you can **list differences** between different types of Namenodes (Secondary / Checkpoint / Backup)

Distributed File System (Quiz)