HDFS

Anurag Nagar

File System (FS)

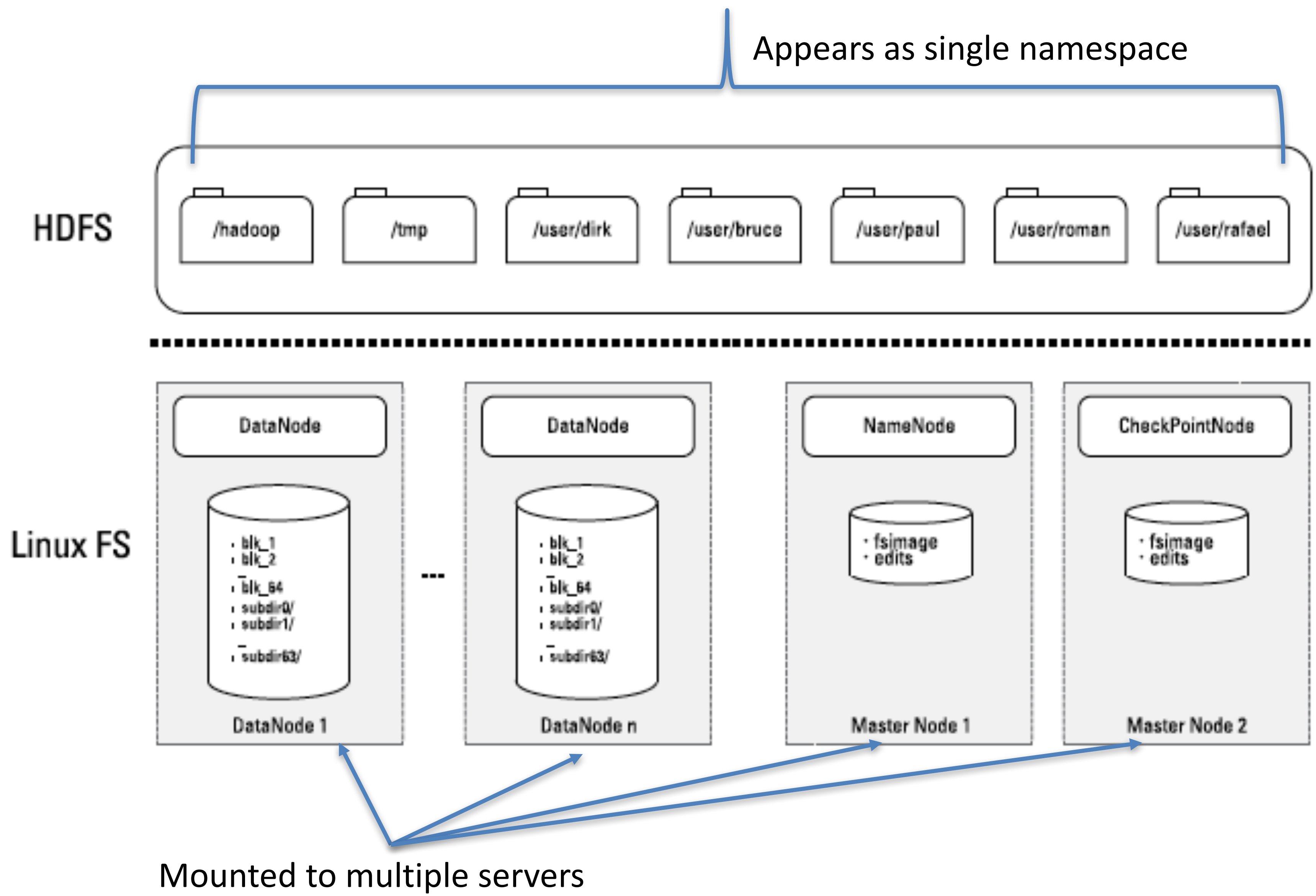
- For Big Data storage, and analysis, we need a FS that has the <u>following qualities</u>:

- Distributed
- Fault and Error tolerant
- Scalable
- High performance
- High availability
- Able to store and manipulate large files
- Low (or medium) cost

Distributed FS

Important features

- Mounted on multiple servers
  - transparent to client
- Consistent namespace across all servers
  - clients can think of it as one big system, without worrying about location
- Concurrency
- Failure tolerant
- Should allow remote access


Appears as single namespace
HDFS
/hadoop
/tmp
/user/dirk
/user/bruce
/user/paul
/user/roman
/user/rafael
Linux FS
DataNode
blk_1
blk_2
blk_64
subdir0/
subdir1/
subdir63/
DataNode 1
DataNode
blk_1
blk_2
blk_64
subdir0/
subdir1/
subdir63/
DataNode n
NameNode
fsimage
edits
Master Node 1
CheckPointNode
fsimage
edits
Master Node 2
Mounted to multiple servers

DFS

- Should allow remote access
- Replication of data
  - What would happen if one server becomes unresponsive?
- Scalable

Google FS

- It is the precursor to HDFS.
- The secret was revealed in a paper published in 2003:

Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." *ACM SIGOPS operating systems review*. Vol. 37. No. 5. ACM, 2003.

It's a very interesting paper. Please read.

Evolution of Hadoop DFS

- Created by Doug Cutting, part of Apache project
- 2004: Google publishes GFS paper
- 2005: Nutch (open source web search) uses MapReduce
- 2008: Becomes Apache top-level project, was Lucene sub-project before.
- 2009: Yahoo used Hadoop to sort 1TB in 62 sec.
- 2013: Hadoop is used by hundreds of the companies

Hadoop DFS (HDFS)

Some key features

- Distributed
  - built using a number of nodes (generally Linux machines)
- Fault tolerant
  - can recover quickly and automatically from failures
- Streaming data access
  - batch processing, high throughput, NOT low latency
- Larger file sizes

HDFS Concepts

- What is low latency and high throughput?
  - Latency measures speed of response.
  e.g. If you hit a key, how fast do you get the data back
  - Throughput measures how much data you are able to process in a unit of time.
- HDFS is designed for large batch jobs, not smaller interactive jobs.
- Also known as "streaming data access"

Hadoop DFS (HDFS)

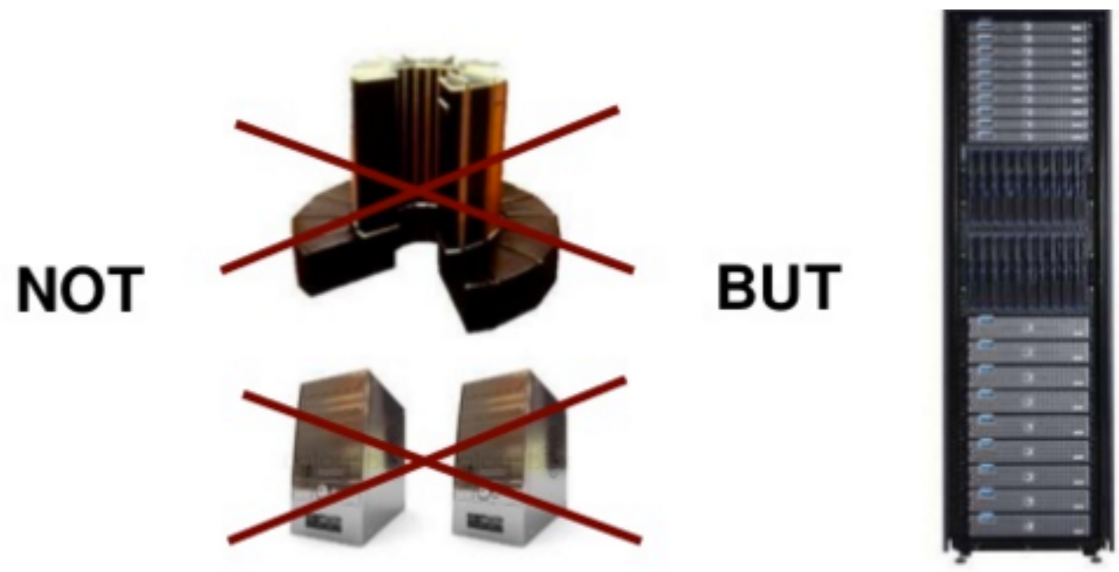
Some key features

- Write-once, read-many
- Moving computation rather than moving data
  - data processed in the node where it is.
  - saves bandwidth
- Two types of nodes:
  - Namenode - master node
  - Datanode – worker nodes, store and manipulate data

Hadoop DFS (HDFS)

Some key features

- Uses commodity hardware.
  - Not expensive single supercomputers



HDFS not suitable if..

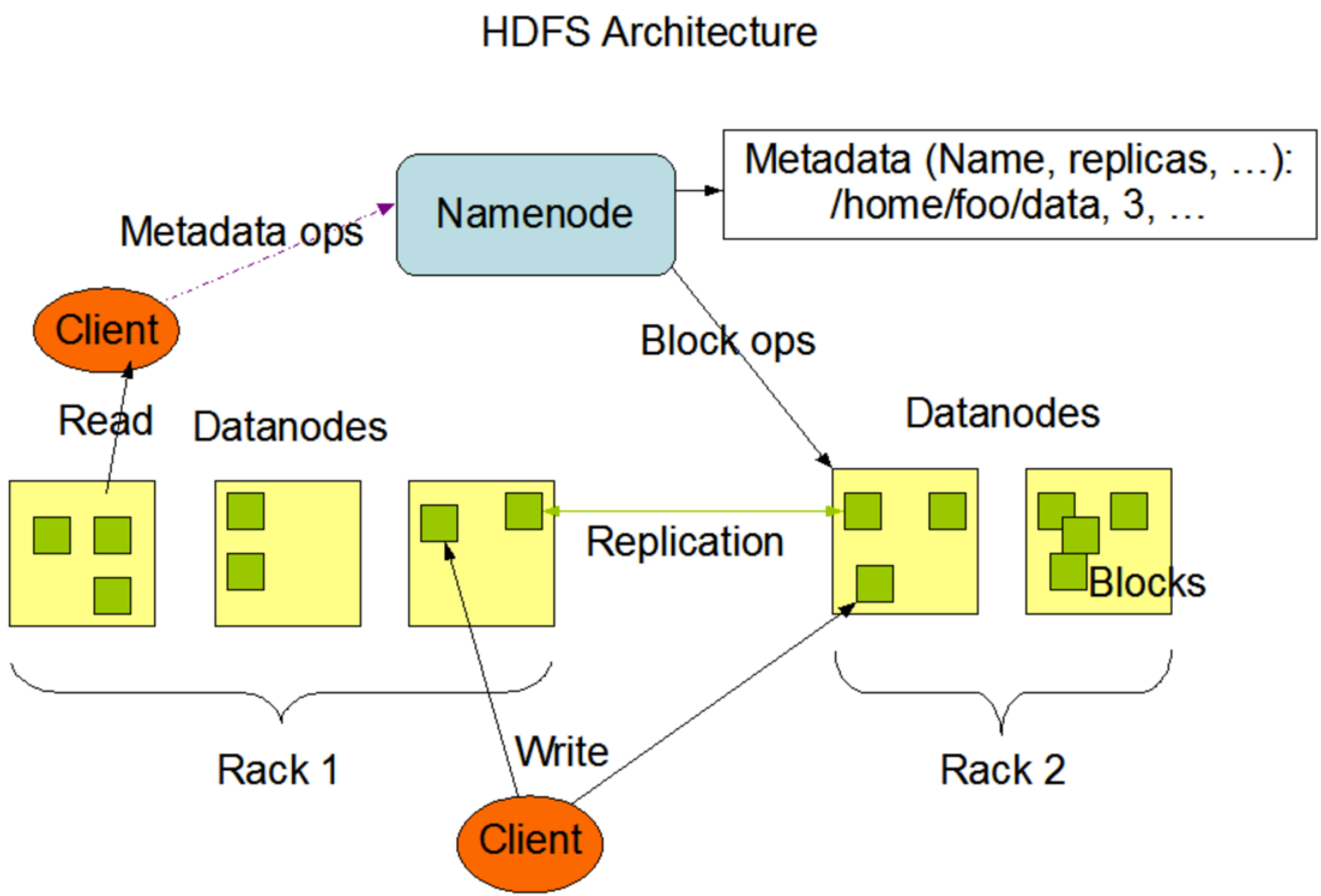
HDFS is not the best route for:

- Low latency reads- if you are interested in fast response times only
- Large number of smaller files
- Large number of writes and writers
  - Our assumption was “write-once, read-many”

HDFS Design & Architecture

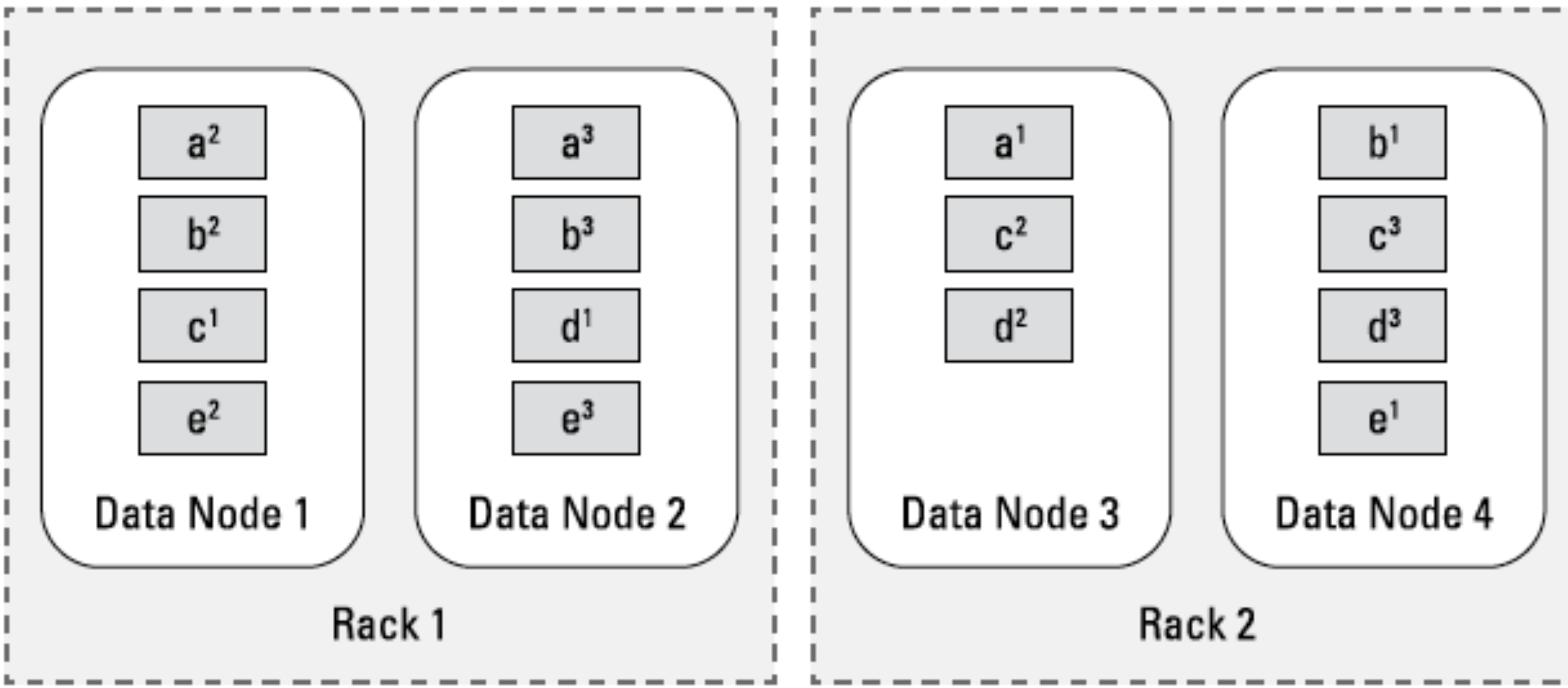
Architecture

- Based on master-slave architecture
- Two types of nodes:
  - Master -> Namenode
  - Slave -> Datanode
- Only one Namenode and multiple DataNodes


HDFS Architecture
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Namenode
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

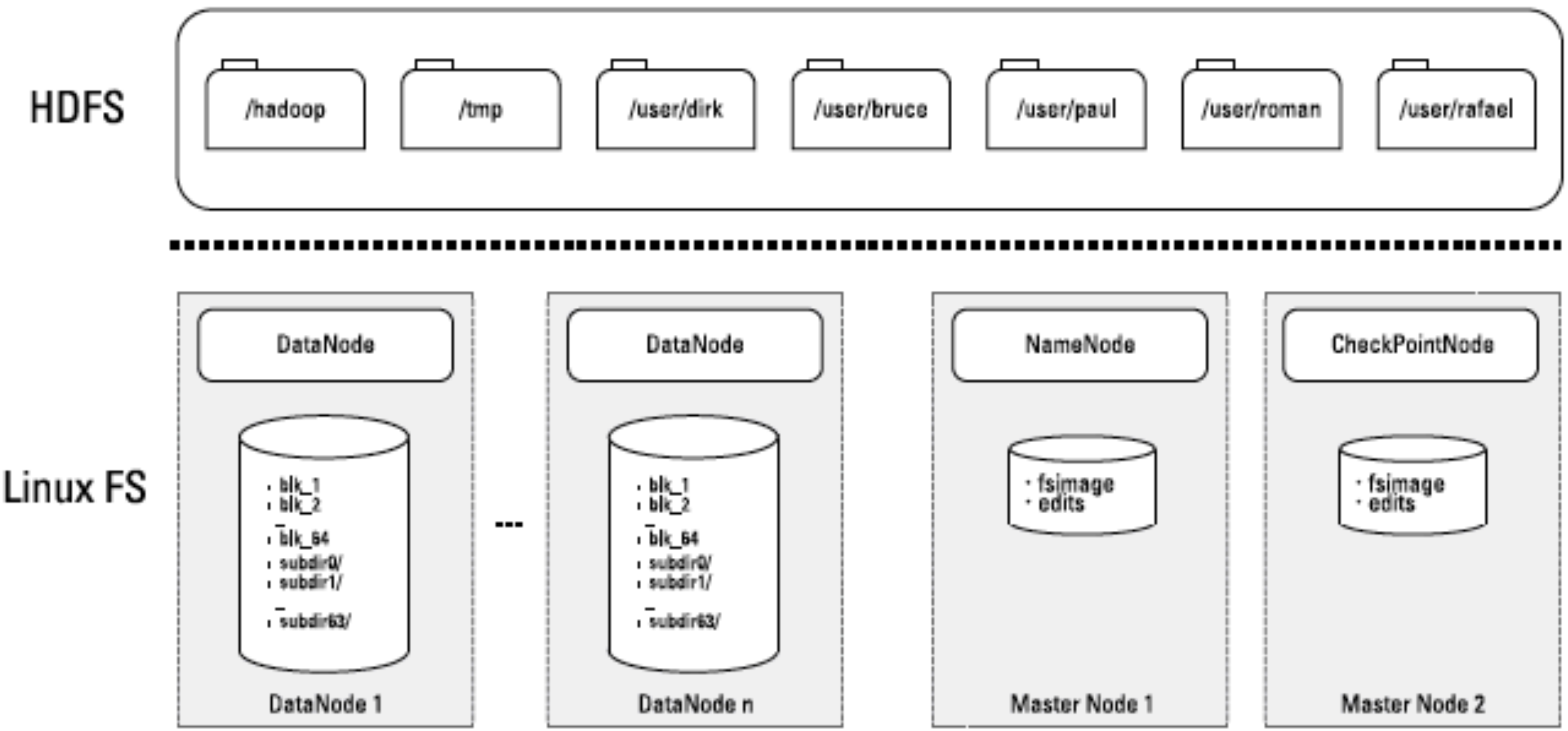
DataNodes

- Worker nodes
- Store blocks of data in a replicated way



DataNodes

- Inexpensive hardware
- Provides consistent namespace
  - user gets an impression that there is one single namespace



DataNodes

- Provide & receive data from clients
- Data doesn’t flow through the Namenode

NameNode

- Master node, only one instance
- Keeps track of metadata
  - block – node mapping
- For faster processing, NameNode keeps all metadata in memory.
  => You can't use commodity hardware for NN
- Checks periodically on the DataNode state

NameNode

- NameNode does not directly read/write data.
  => This is one of the reasons of scalability of HDFS
- Client interacts with NN only to update namespace (file names) or get block locations.
- Client interacts with DN to get data.
  (HDFS is single-write, multiple-read)

NN Operation

How does NN persist metadata?

Two different constructs:

- fsimage

-> <u>point-in-time</u> snapshot of file system's metadata

Think of it like Mac's time machine

-> It's good for writing bulk changes,

but not very suitable for incremental changes

e.g. if you rename a file, you don't want to go and write it to the fsimage

NN Operation

How does NN persist metadata?

Two different constructs:

- edits file

-> smaller incremental changes are recorded.

Thus, the major changes (checkpoints) are recorded in the fsimage file and incremental updates are in the edits file.

NN Startup

What happens when NN starts up

1. The NameNode loads the `fsimage` file into memory.
2. The NameNode loads the `edits` file and re-plays the journaled changes to update the block metadata that's already in memory.
3. The DataNode daemons send the NameNode block reports.

Checkpointing

Checkpointing is the process that takes an fsimage and edit log and compacts them into a new fsimage. This prevents the edit log from becoming too large.

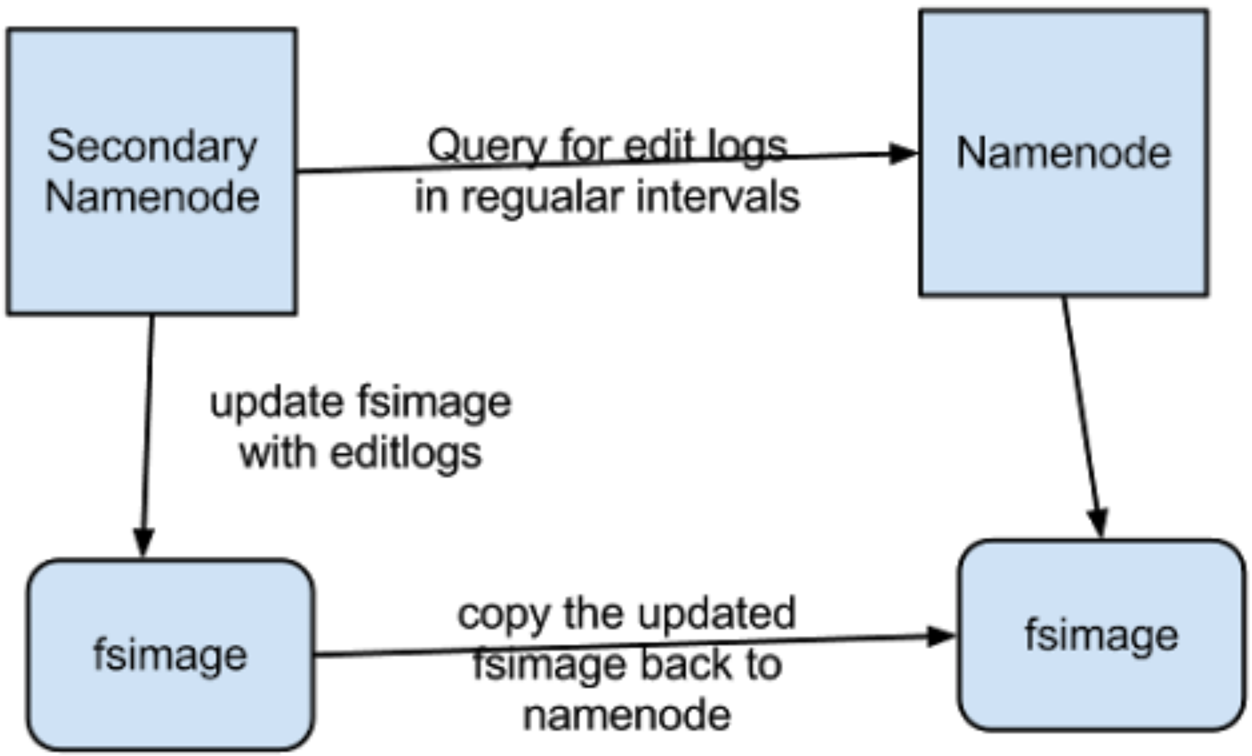
Checkpointing creates a new fsimage from an old fsimage and edit log.

When does it happen ?

- When NN restarts, it merges the two and cleans up the edit file
- However, NN restart are rare in production environment.
- So we need a way so that the edits file doesn't become very large and there is periodic merging of the two log files.

Secondary NN to the rescue

- Secondary NN takes over this responsibility from NN



https://hadoop.apache.org/docs/r1.2.1/hdfs_user_guide.html#Secondary+NameNode

Namenode failure

- Namenode is a single point of failure.
- What happens if it crashes??

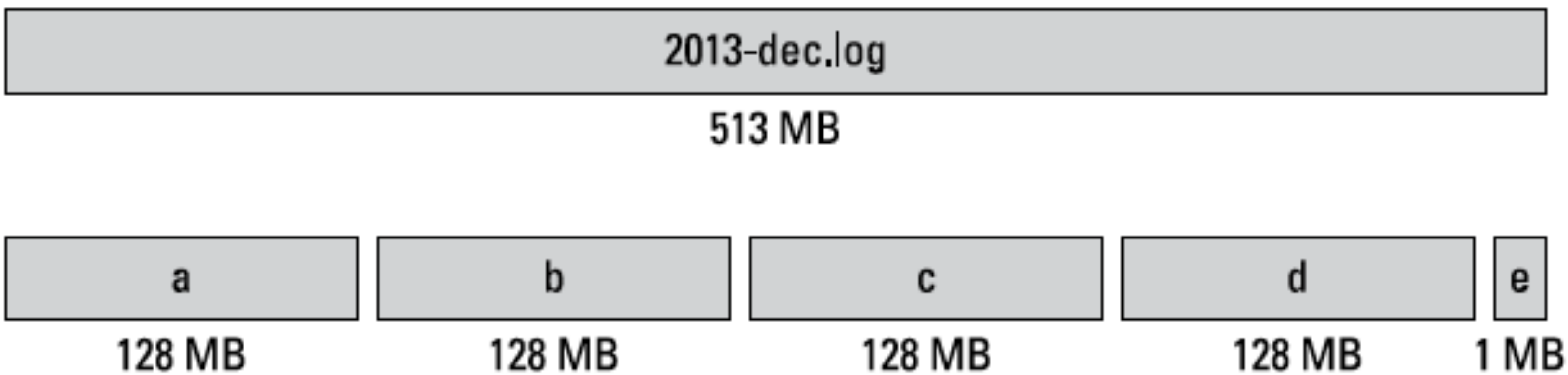
2 solutions:

- Backup the files that make up the persistent state to multiple filesystems
- Run a secondary namenode.
  It periodically merges the latest namespace image to the edit log.
  Creates checkpoints

Design Decisions- Replication

- Files broken down into chunks
  - Default chunk size = 64 MB (Hadoop1)
  - Default chunk size = 128 MB (Hadoop2)
- To ensure reliability and fault tolerance, each chunk is replicated.
  - Default replication factor = 3
- There are various nodes (generally Linux machines) in the Hadoop cluster.
- The replicated chunks are placed on different Datanodes. If one node goes down, chunk can be recovered.

A file of size 513 MB needs to be stored in HDFS. Assuming block size=128 MB, find the number of blocks needed.



Some helpful conversions:

IEC prefix		Representations				Customary prefix	
Name	Symbol	Base 2	Base 1024	Value	Base 10	Name	Symbol
kibi	Ki	2^{10}	1024^1	1024	$\approx 1.02 \times 10^3$	kilo	k[13] or K
mebi	Mi	2^{20}	1024^2	1 048 576	$\approx 1.05 \times 10^6$	mega	M
gibi	Gi	2^{30}	1024^3	1 073 741 824	$\approx 1.07 \times 10^9$	giga	G
tebi	Ti	2^{40}	1024^4	1 099 511 627 776	$\approx 1.10 \times 10^{12}$	tera	T
pebi	Pi	2^{50}	1024^5	1 125 899 906 842 624	$\approx 1.13 \times 10^{15}$	peta	P
exbi	Ei	2^{60}	1024^6	1 152 921 504 606 846 976	$\approx 1.15 \times 10^{18}$	exa	E
zebi	Zi	2^{70}	1024^7	1 180 591 620 717 411 303 424	$\approx 1.18 \times 10^{21}$	zetta	Z
yobi	Yi	2^{80}	1024^8	1 208 925 819 614 629 174 706 176	$\approx 1.21 \times 10^{24}$	yotta	Y

A file of size 8 PB (petabytes) needs to be stored in HDFS. Assuming block size=128 MB, find the number of blocks needed.

Answer = $8 \times 2^{50} / 128 \times 2^{20}$

$= 2^{26}$

Design Decisions

- What would happen if a cluster needs more processing power?

- HDFS is based on concept of “scale-out” and not “scale-up”.

- That means, you can add more commodity hardware easily.

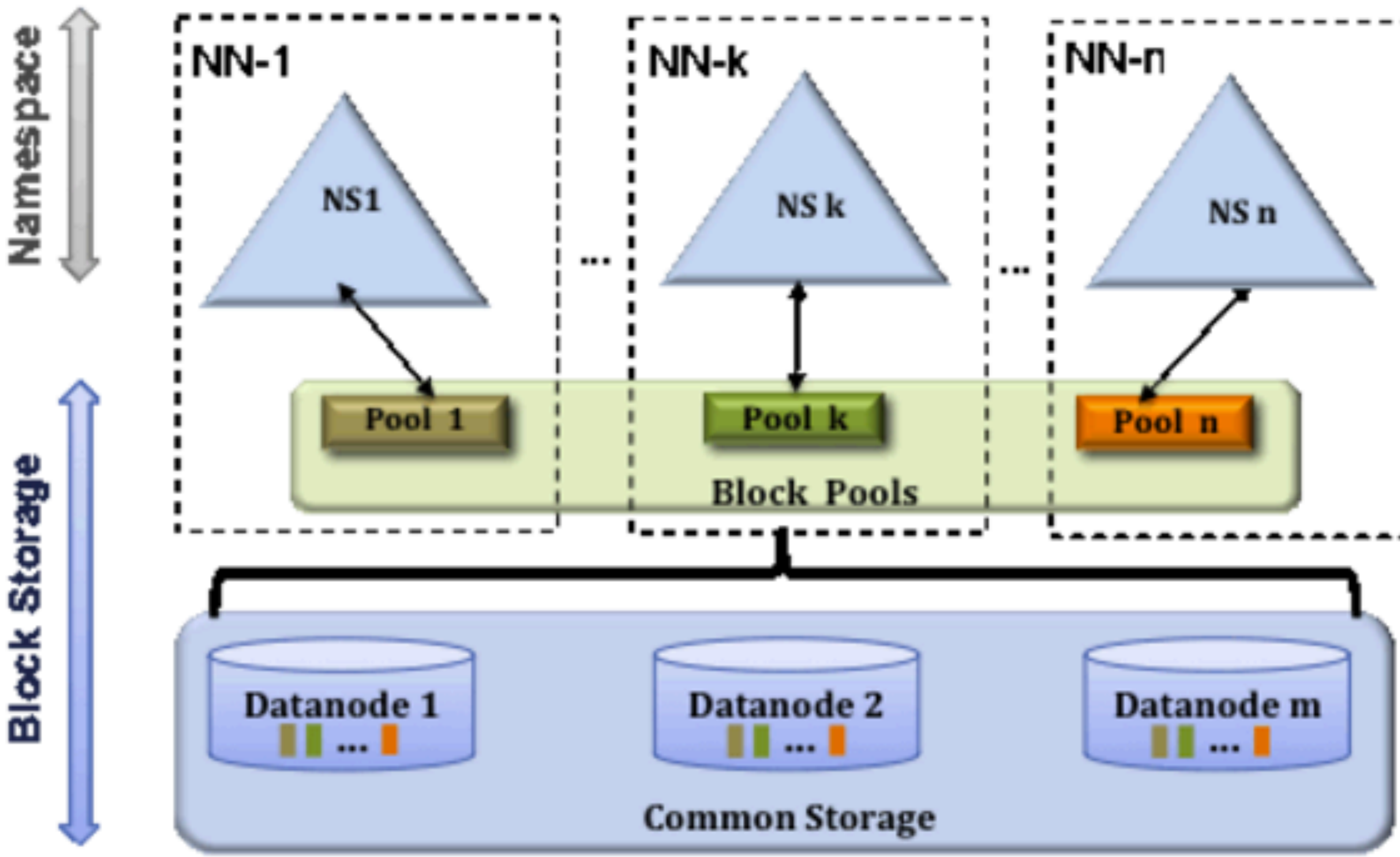
- You don’t need to upgrade to expensive hardware

- Another advantage- there is no single point of failure.

HDFS Federation

- In Hadoop 2.x, concept of HDFS federation is introduced.
- Allows a cluster to scale by adding namenodes, each of which manages a portion of the filesystem namespace.

Namenode failure Hadoop 2.x



Read more here:
https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/Federation.html

Some interesting questions

What is the difference between RDBMS and HDFS processing?

- RDBMS is for structured data, HDFS can work with unstructured, larger datasets.
- RDBMS is transactional, has ACID properties; HDFS is batch oriented, failure tolerant.
- RDBMS supports read-write operations, HDFS has write-once, read-many property.

Some interesting questions

What is high throughput? How is it achieved?

- Throughput measures the amount of processing done in a unit of time.
- By performing computation in a distributed, parallel way, HDFS achieves high throughput.
- HDFS provides streaming access to data, which ensures the entire data stream can be accessed and processed in the most efficient manner

Some interesting questions

What is commodity hardware? Would you use it for the namenode also?

Please think about this question

Some interesting questions

What is secondary namenode? Is it a backup node to the main namenode?

Please think about this question