Hadoop Distributed File System

ISTD, SUTD

Feb 21, 2023

Plan of Attack

- Week 10: Hadoop HDFS, Data Parallelism, MapReduce
- Week 11: Spark RDD, Spark Dataframe
- Week 12: Spark Scheduling, YARN

Big Data

- It is a technology?
  - Or a business problem?
- Core of the modern technology stack, because of business analytics, AI, cloud computing
- 3 Vs definition : (Volume, Velocity, Variety)
  - 2 more Vs
    - V _ _ _ _
    - V _ _ _ _ _ _ _ _

Hadoop Distributed File System

By the end of this lesson, you are able to

- Explain the file system model of HDFS
- Explain the architecture of HDFS
- Explain the replication placement strategy
- Explain the operation of HDFS client
- Explain the use of erasure coding

History of Hadoop

- Created in 2005
- in Yahoo! to support Nutch seach feature
- Became the industry standard for Big Data System

High level of Architecture of Hadoop

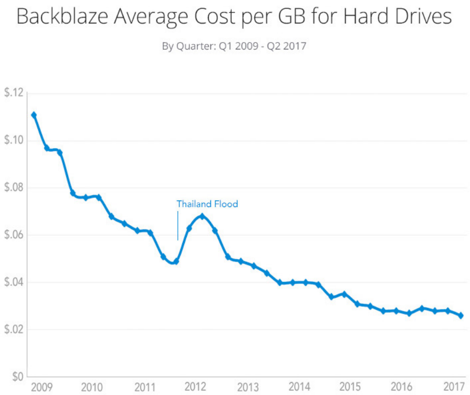
- MapReduce - Data Processing Layer
- Hadoop Distributed File System - Data Storage Layer
- YARN - Resource Management Layer

Hadoop Distributed File System

- An open source implementation of the the Google File System
  - The Google file system, *Sanjay Ghemawat, Howard Bradley Gobioff and Shun-Tak Leung*, ACM SIGOPS 2005

Why a Distributed File System

- Challenges Google faced (then, in 2003)
  - Databases were expensive
  - They have a lot of non-table data (one of the V of Big Data)
  - 10s/PB
  - slow disk throughput 100-200MB/s
- But disks were cheap



Why Google FS

- Network File System has the following limitation
  - A file must reside on one and only one machine
  - No reliability guarantee
  - Not scalable!
  - 1 disk has 0.001 fail rate, what if we have 1000 disks in the NFS, which has no replica?
  - Network I/O will be high

Why Google FS

The list of features we want

1. Support many clients
2. Support many disks
3. Support many PBs
4. Fault tolerant
5. Read/write like normal files

No system can achieve all. Something must be let go.

Why Google FS

The list of features we want

1. Support many clients
2. Support many disks
3. Support many PBs
4. Fault tolerant
5. **Read/write like normal files**

No system can achieve all. Something must be let go.

How Google FS differs from normal files?

Normal file

- Read randomly
- Write/update randomly

Google FS

- Read sequentially
- Append only

Can you think of some applications operating like this?

A quick summary

- Google FS is designed to simplify distributed file system by *throwing* away random read and write
- HDFS is strongly influenced by Google FS

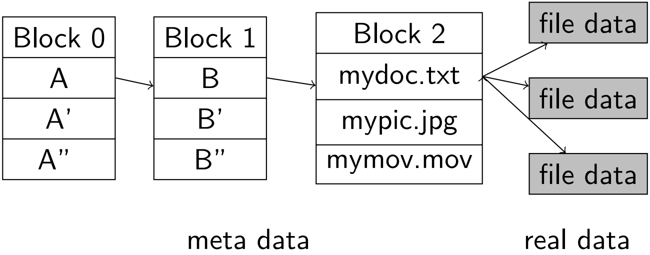
HDFS File System Model

Most of the file systems adopt a model with a hierarchical name space,

```
|- A
|  |- B
|     |- mydoc.txt
|  |- B'
|  |- B''
|
|- A'
|- A''
```

File System Model

e.g. `/A/B/mydoc.txt`



File System Model

A file consists of blocks of data

- Good design, simple abstraction
- A file might be larger than a physical disk
- We can distribute blocks belonging to a file into multiple disks/hosts (distributed)

Block Size

- Normal file system 4KB
- RDBMS 4-32KB
- HDFS 64MB (configurable)

HDFS Architecture

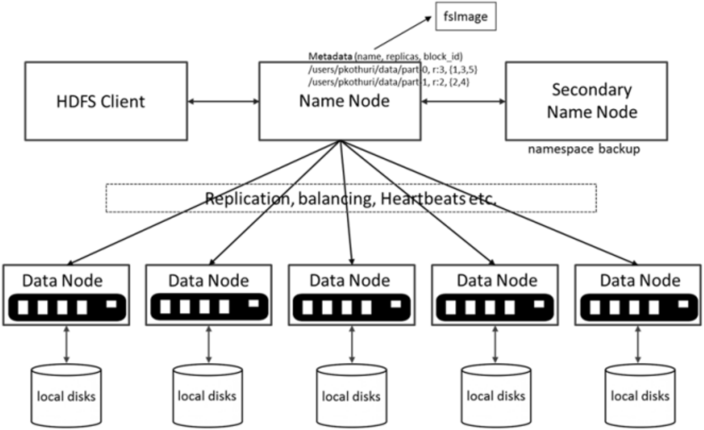


Figure 2: HDFS Architecture

- A Master-worker architecture
- Why not a peer to peer?

HDFS Architecture

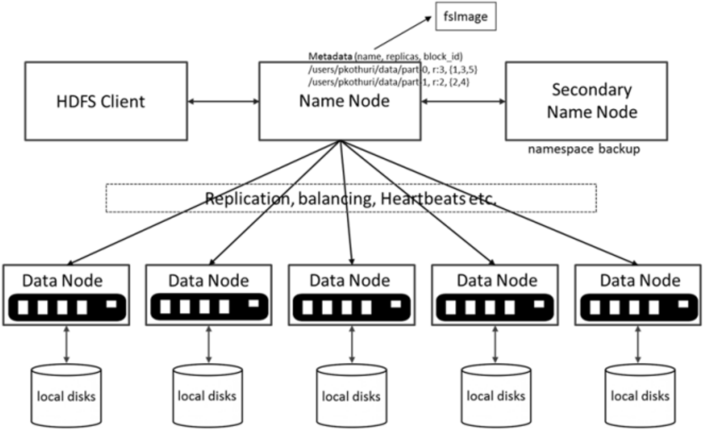
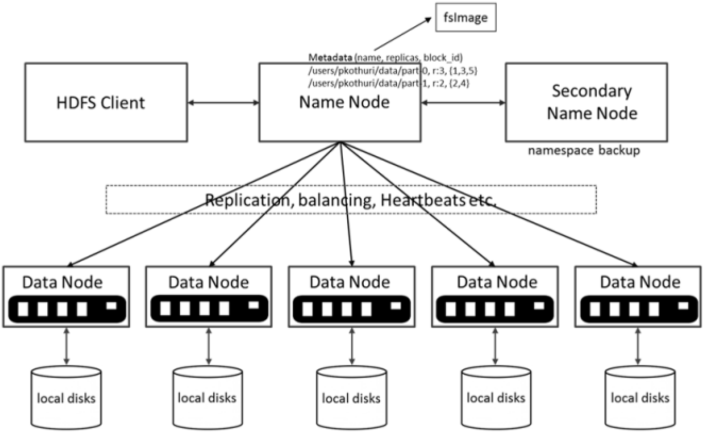


Figure 3: HDFS Architecture

- Block Size (default 64MB v1, 128MB v2+)
- Each block is replicated (Recommended 3 or any odd number > 3). But why?

HDFS Architecture



Given a file request, Namenode

1. file (full path name) -> block IDs
2. block IDs -> actual data

Replica Placement Strategy

- HDFS is a logical cluster
- Some physical location info will help, e.g. Rack
- Goals:
  - Maximize chance of survival
  - Maximize load balance

Replica Placement Strategy

- Max 1 replica per datanode
- Max 2 replicas per rack
- Num of racks for replication $<$ RF

Replica Placement Strategy

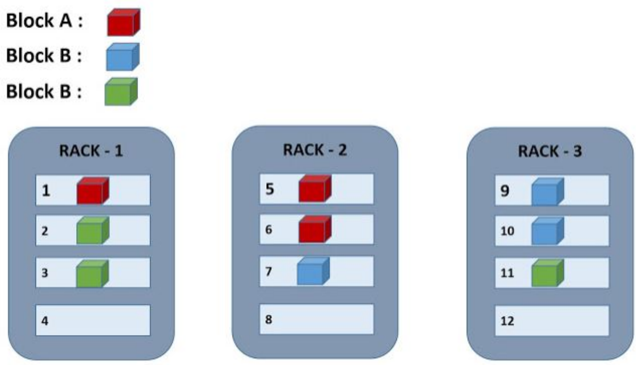
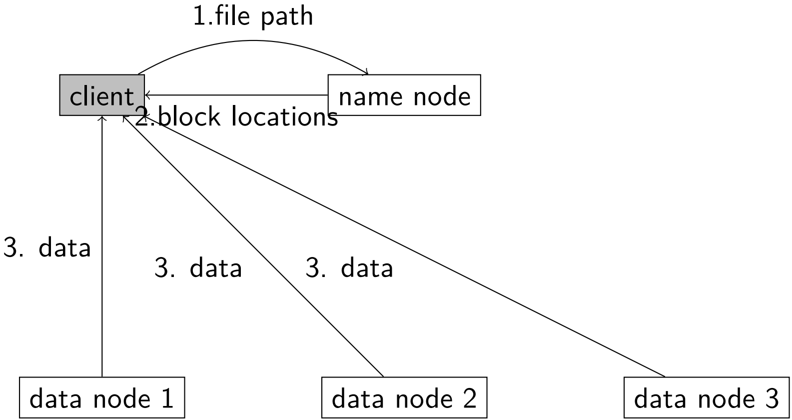


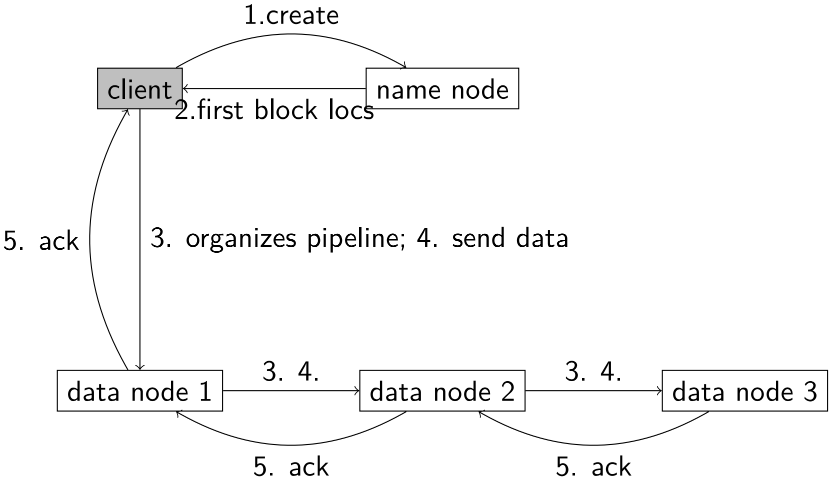
Figure 4: Replace placement strategy

- Replica 1: rack 1 (first datanode contacted during write)
- Replica 2: different rack than rack 1, let's say rack 2
- Replica 3: rack 2, min
- Replica >=4: random

HDFS client operation - Read

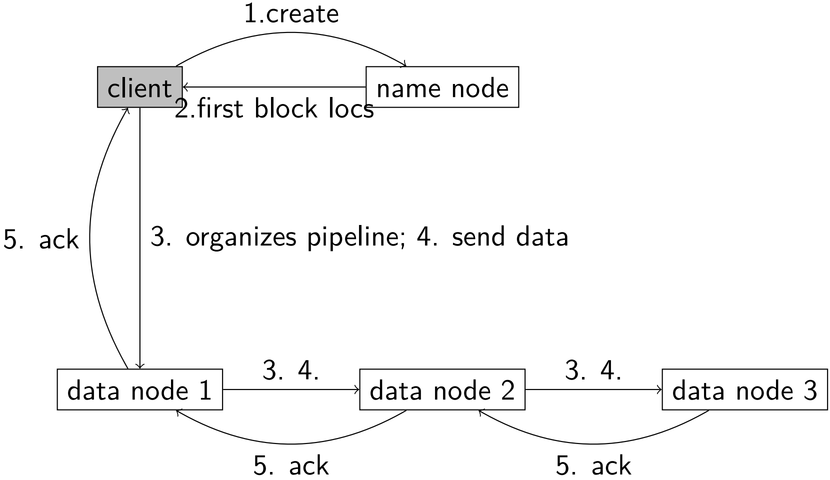


HDFS client operation - Write



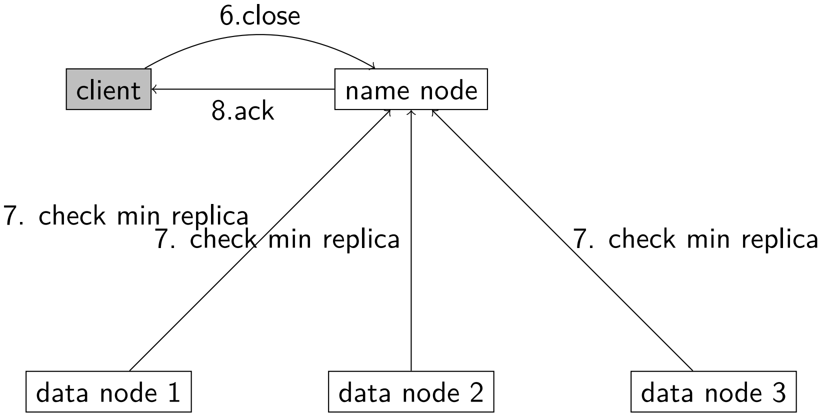
- If it is an append, the last block loc is returned.
- Repeat steps 2 - 5 if there are more blocks
- ~~Retry steps 3 - 5 if fail~~

HDFS client operation - Write



If some data node fails during the write of a data block (a) The written ones are retained; (b) The namenode will be informed that the data block is under replication; (c)The pipeline will re-initialized for the next data block

HDFS client operation - Write



HDFS Erasure Coding

One issue with the replication is that given Replication Factor = N,

- we have (N - 1)*100% storage overhead
- (1/N) storage efficiency
- N - 1 as fault tolerance.

In Hadoop v3, besides replicas, we have another option - Erasure Coding.

HDFS Erasure Coding

Recall that XOR operation $\bigoplus$ on bits

IN	IN	XOR
0	0	0
1	0	1
0	1	1
1	1	0

having some nice properties

$$X \bigoplus Y = Y \bigoplus X$$

$$(X \bigoplus Y) \bigoplus Z = X \bigoplus (Y \bigoplus Z)$$

$$X \bigoplus Y = Z \Rightarrow X \bigoplus Z = Y$$

We can use the result of XOR to recover if one of the inputs is lost.

HDFS Erasure Coding

What if we lose more than one input?

- Reed-Solomon EC

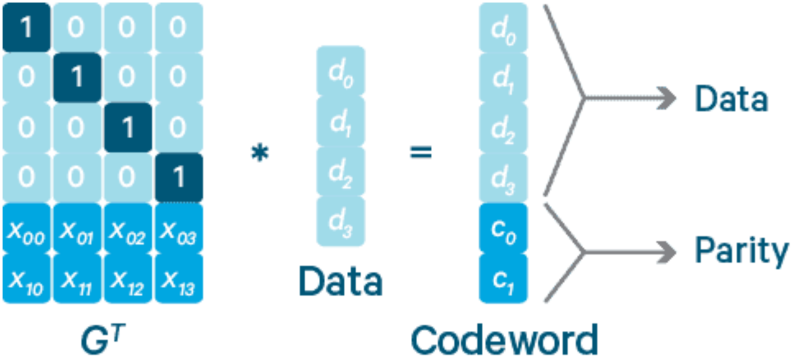


Figure 5: Erasure Coding

G^T is called a *Generator Matrix*.

HDFS Erasure Coding - A Concreate example

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

G^T Data Codeword

- A property of G^T: all $k \times k$ sub matrices must be non-singular (an inverse exists), where k is the size of the data.
- Note it is a bad idea to use binary data here. However for demonstration purposes, we stick with binary data and hand pick a sub-matrix that is non-singular

HDFS Erasure Coding - A Concreate example

Let's say we lose the 2nd and 4th cells in the Codeword, and want to recover the data, We remove the correspondent rows from the G^T, the following equation still holds

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

$G^T_{\neg(1,3)}$ Data Codeword

HDFS Erasure Coding - A Concreate example

We find the inverse of $G^T_{\neg(1,3)}$ and multiply it to both sides

$$\underbrace{\begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 1 & 0\\0 & 1 & 0 & 0\\1 & 1 & 0 & 1\end{bmatrix}}_{G^{T^{-1}}_{\neg(1,3)}} \times \underbrace{\begin{bmatrix}1 & 0 & 0 & 0\\0 & 0 & 1 & 0\\0 & 1 & 1 & 0\\1 & 0 & 1 & 1\end{bmatrix}}_{G^{T}_{\neg(1,3)}} \times \underbrace{\begin{bmatrix}1\\0\\1\\0\end{bmatrix}}_{\text{Data}} = \underbrace{\begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 1 & 0\\0 & 1 & 0 & 0\\1 & 1 & 0 & 1\end{bmatrix}}_{G^{T^{-1}}_{\neg(1,3)}} \times \underbrace{\begin{bmatrix}1\\1\\1\\0\end{bmatrix}}_{\text{Codeword}}$$

HDFS Erasure Coding - A Concrete example

We cancel ${G_{\neg(1,3)}^{T}}^{-1} \times G_{\neg(1,3)}^{T}$ from the LHS

$$\begin{bmatrix}1\\0\\1\\0\end{bmatrix} = \begin{bmatrix}1&0&0&0\\0&1&1&0\\0&1&0&0\\1&1&0&1\end{bmatrix} \times \begin{bmatrix}1\\1\\1\\0\end{bmatrix}$$

Data ${G_{\neg(1,3)}^{T}}^{-1}$ Codeword

HDFS Erasure Coding

- Note that G^T is fixed for all data.
- For RS(k,m) we have m/k storage overhead and k / (k + m) storage efficiency, where k is the data size and m is the parity size.
- Some popular choices of k and m for HDFS are (6,3) and (10,4)
  - RS(6,3) we have
    - 3 / 6 = 50% storage overhead
    - 6 / 9 = 66.6% storage efficiency
    - Fault tolerance (can afford losing 3 rows from the codeword out of 9).
  - RS(10,4) we have
    - 4 / 10 = 40% storage overhead
    - 10 / 14 = 71.4% storage efficiency
    - Fault tolerance (can afford losing 4 rows from the codeword out of 14).

HDFS Erasure Coding references

- https://blog.cloudera.com/introduction-to-hdfs-erasure-coding-in-apache-hadoop/
- https://www.backblaze.com/blog/reed-solomon/

Summary

- Google File System hugely influential
  - Scalable, fault-tolerant
  - Designed for specific workloads
- HDFS implements GFS
- HDFS de-facto distributed file system in the cloud
  - All cloud-based data analytics systems support reading from HDFS

In class discussion 1

1. Consider HDFS append operation, it doesn't provide correctness! Give an example of how incorrect append could happen.
2. Why do you think it's difficult to guarantee correctness for append?

In class discussion 2

In an hadoop setup, the erasure coding configuration is RS(12,6)

1. What is the storage overhead?
2. What is the storage efficiency?
3. What is the fault tolerance level?
4. What is the dimension of the Generator Matrix?

In class discussion 3

Suppose you are engaged by a client to setup a HDFS for data computation. Here is the user requirement

- Existing active data size 5TB
- Estimated year-over-year data growth rate 80%
- 50% buffer space for intermediate/temp data file
- HDFS replication factor 3

1. What is the projected disk space requirement for HDFS in 3 years time?
2. What is the projected disk space requirement for HDFS in 3 years time, if we replace RF=3 by RS(10,4)?